Intel PROSet For Windows* Device Manager WMI Provider User's Guide

White Paper

Revision 1.8

Contents

Introduction

Intel® PROSet for Windows* Device Manager deploys Network Configuration Services version 2.0, an easy to use solution for deploying and managing all Intel end-station networking technologies using industry standard methods. The NCS2 architecture works closely with the Windows Management Instrumentation (WMI) service to provide remote management of Intel network devices. This document describes the WMI classes and providers supplied by Intel® PROSet for Windows* Device Manager.

This document is divided into several sections

- Technology overview – an overview of WMI technology.
- Class summaries – the class and namespace details for the NCS2 architecture.
- Working examples – how to use the NCS2 architecture to manage Intel® network devices.
- Errata – additional information specific to some environments.

Intel® PROSet for Windows*Device Manager WMI providers offer the following features.

Category	Features
Adapter	Enumerate all supported physical network adapters Update settings for an adapter Obtain an adapter's physical device information Monitor adapter link Uninstall an adapter driver Query IPv4 and IPv6 adapter addresses
Boot	Change an adapter's boot agent settings View and modify adapter iSCSI settings
Diagnostics	Enumerate diagnostic tests, settings, and results Run or stop a diagnostic test on an installed adapter
Team	Enumerate supported team types Create or remove a team of adapters Update team settings Add or remove team member adapters Change team member priorities Obtain the IPv4 protocol settings for a team
VLAN	Create, discover, or remove Virtual LANs on an adapter or team Update VLAN settings Obtain the IPv4 protocol settings for a VLAN

Technology Overview

This section offers an overview of Windows Management Instrumentation in Microsoft operating systems and is recommended for anyone not familiar with the architecture. Further reading on this topic is encouraged and additional are links are provided at the end of this section.

Web-based Enterprise Management

Web-based Enterprise Management (WBEM) is a Distributed Management Task Force (DMTF) initiative providing enterprise system managers with a standardized, cost-effective method for end station management. The WBEM initiative encompasses a multitude of tasks, ranging from simple workstation configuration to full-scale enterprise management across multiple platforms. Central to the initiative is the Common Information Model (CIM), an extensible data model representing objects in typical management environments, and the Managed Object Format (MOF) language for defining and storing modeled data.

Windows Management Instrumentation

Windows Management Instrumentation (WMI) is the Microsoft implementation of WBEM for Windows* operating systems. It exposes a programmable interface to view and interact with management objects. Running as a system service, this operating system component offers many powerful capabilities.

WMI consists of the following components:

- Management applications
- Managed objects
- Providers
- Management infrastructure
- A COM API to allow access to management information.

Management applications process or display data from managed objects, which are logical or physical enterprise components. These components are modeled using CIM and accessed by applications through Windows Management Services. Providers supply Windows Management with data from managed objects, handle requests from applications and notification of events. The providers for Intel® PROSet for Windows* Device Manager play a central role in network card configuration management.

Windows management consists of the CIM Object Manager (for handling the communication between management applications and providers) and a central storage area (CIMOM object repository). Data is placed in the repository using either the MOF language compiler or the Windows Management API.

The following diagram shows the interrelationship of these components:

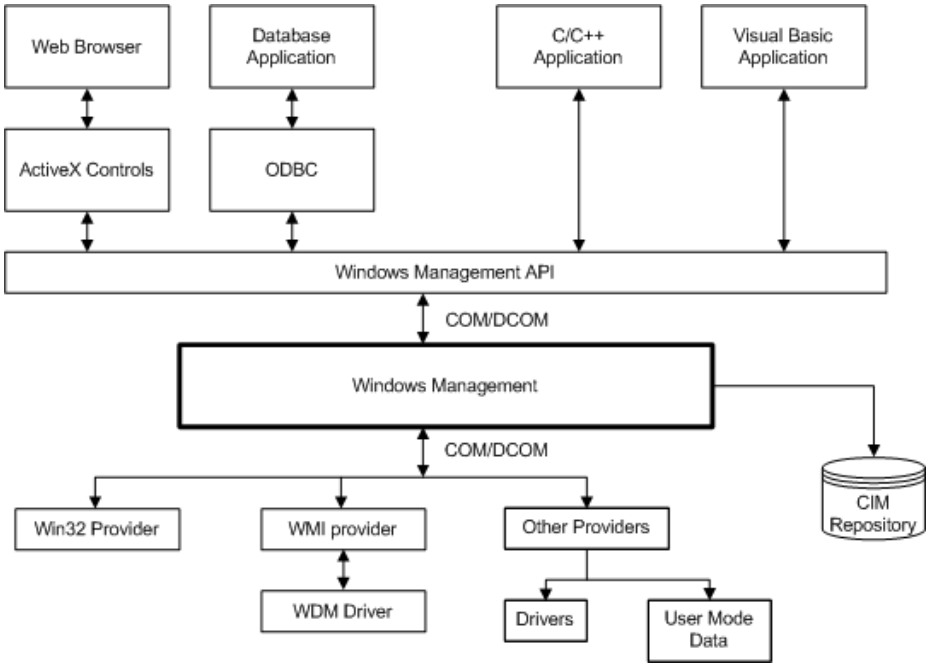


Figure 1 – Windows Management Architecture

Common Information Model

The Common Information Model (CIM) presents a consistent and unified view of all types of logical and physical objects in a managed environment. Managed objects are represented as classes. CIM was designed by the DMTF to be operating system and platform independent, but the Microsoft implementation pre-dominates the specification. WBEM technology includes an extension of CIM for Microsoft Windows* operating system platforms. Please refer to the DMTF CIM schema on the DMTF web site for more information. Intel® PROSet for Windows* Device Manager is based on CIM Schema version 2.6.

CIM defines three levels of classes:

- Classes representing managed objects that apply to all areas of management. These classes provide a basic vocabulary for analyzing and describing managed systems and are part of what is referred to as the "core model."
- Classes representing managed objects that apply to a specific management area but are independent of a particular implementation or technology. These classes are part of what is referred to as the common model - an extension of the core model.
- Classes representing managed objects that are technology-specific additions to the common model. These classes typically apply to specific platforms such as UNIX or the Microsoft Win32 environment.

Inheritance Relationships

Classes can be related by inheritance, where a child class includes data and methods from its parent. Inheritance relationships are not typically visible to the management application using them, nor are the applications required to know the inheritance hierarchy. Class hierarchies can be viewed with CIM repository viewers. Since the NCS2 architecture uses inheritance, it is important to understand the limits and capabilities of these relationships.

Association Classes

Windows Management also supports association classes. Association classes link two different classes to model a user-defined relationship, and are visible to management applications. Third-party developers can also define association classes

for their management environment. Associations represent a relationship between two WMI objects (classes). The properties of the association class include two pointers or references, each linking to a different instance. The relationships are maintained by path only; the association class does not have the capability to modify the instances it links. For additional information on CIM, visit http://www.dmtf.org

CIM Tools

There are many ways to interact with a CIM repository depending on which operating system the user has installed. These tools are best used to view CIM information; scripting and programming languages are recommended for configuration changes.

Tool	Explanation
Wbemtest	Native support on any Windows* operating system where WMI has been installed.
CIM Studio	Browser based implementation of WBEMTest.exe and much easier to use. However, it requires download and install an additional program. To locate this tool, search for "WMI Administrative Tools" on Microsoft's web site (www.microsoft.com).
Windows Powershell	Optional shell environment which offers powerful scripting capabilities and provides easy access to WMI namespaces through simple queries.
WMIC	Windows Management Instrumentation Command-line. A command line interface to WMI namespaces.

Installed Files

Executables

When information is requested about Intel® PROSet for Windows* Device Manager through a WMI service call, the NCS2 provider will be launched. This will be visible as a running process in the operating system. Start and shutdown of WMI providers is completely transparent to user; there is no need to directly manipulate them. After a period of inactivity, the NCS2 provider will unload itself (usually a few minutes).

Filename	Description
Ncs2Prov.exe	Intel® PROSet for Windows* Device Manager WMI provider.

Dynamically Linked Libraries

The following dynamically linked libraries are used by Intel® PROSet for Windows* Device Manager.

Filename	Description
Ncs2Core.dll	Implements the Ethernet Adapter Schema.
Ncs2Diag.dll	Implements the Diagnostics Schema.
Ncs2Boot.dll	Implements the Boot Agent Schema.
Ncs2Team.dll	Implements the Team Schema.
Ncs2VLAN.dll	Implements the VLAN Schema.

MOF Files

A "MOF" file is a Managed Object Format file which contains information about WMI classes. A set of basic MOF files are included on distribution media for reference only. There are separate MOF files for language neutral and language specific data, which become available upon installation. The following are .mof files for the 'root\IntelNCS2' namespace.

Filename	Description
ICmLn.mof	CIM base classes on which the NCS2 classes depend.
ICmEnu.mfl	US English version of the CIM base classes.
ICoreLn.mof	Classes for the IEEE 802.3 adapters.

ICoreEnu.mfl	US English textual amendments to the adapter classes.
IBootLn.mof	Classes for the IEEE 802.3 boot service
IBootEnu.mfl	US English textual amendments to the 802.3 boot service classes.
IDiagLn.mof	Classes for the CDM (Common Diagnostic Model).
IDiagEnu.mfl	US English textual amendments to the CDM classes.
ITeamLn.mof	Classes for the IEEE 802.3 teams.
ITeamEnu.mfl	US English textual amendments to the team classes.
IVLANLn.mof	Classes for the IEEE 802.3 VLANs.
IVLANEnu.mfl	US English textual amendments to the VLAN classes.

Security

The NCS2 WMI provider uses client impersonation to manage the security; every call will be made in the client's own security context. This context is passed down to the lower layers. An operation may fail if the user does not have suitable administrative rights on the target machine. Please see Permissions for more information.

Namespaces

CIM classes are organized into namespaces, a logical partitioning of the CIM object management repository. Installation of Intel® PROSet for Windows* Device Manager will create the namespace "root\IntelNCS2" The NCS2 architecture uses this namespace to organize management information and make it available to clients. This namespace is only visible when PROSet has been installed and will be removed upon product uninstall.

root\IntelNCS2

The root\IntelNCS2 namespace contains information about Intel® PROSet for Windows* Device Manager and is based on CIM version 2.6. The root\CIMv2 namespace was not used as a primary because it is based on CIM version 2.2 and has object key differences. Classes in this namespace have been extended through class inheritance to contain information specific to the NCS2 architecture. All operations regarding adapters, teams, VLANs, boot agent settings, and diagnostics must interact with this namespace.

Locales and Localization

Localized MOF files

All the MOF files used by the NCS2 WMI Provider are localized according to the Microsoft Windows Management Instrumentation globalization model. To accomplish this, each class definition is separated into the following:

- a language-neutral version that contains only the basic class definition in the .mof file.
- a language-specific version that contains localized information, such as property descriptions that are specific to a locale in the corresponding .mfl file.

Class Storage

The language-specific class definitions are stored in a child sub-namespace beneath the namespace that contains a language-neutral basic class definition. For example, for the NCS2 WMI Provider, a child namespace ms_409 will exist beneath the root/IntelNCS2 namespace for the English locale. Similarly, there exists a child sub-namespace for each supported language beneath the root/IntelNCS2 namespace.

Runtime Support

To retrieve localized data, a WMI application can specify the locale using strLocale parameter in SWbemLocator::ConnectServer and IWbemLocator::ConnectServer calls. If the locale is not specified, the default locale for that system will be used. (e.g. MS_409 for US English). This locale is used to select the correct namespace when adding in the English strings. In addition, IWbemServices::GetObject, SWbemServices.GetObject, IWbemServices:: ExecQuery, and SWbemServices.ExecQuery must specify the WBEM_FLAG_USER_AMENDED_ QUALIFIERS flag to request localized data stored in the localized namespace, along with the basic definition. This is required in all functions that produce displayable values using value maps or display descriptions or other amended qualifiers from the MOF files.

WBEM Context

IWbemContext is a WMI programming interface which allows users to optionally communicate additional parameters to providers when submitting function calls. If you plan on making any changes to the NCS2 configuration through a WMI call, then you must pass a WbemContext parameter. These optional parameters are constructed by the user and passed as part of a WbemServices call. Interaction with NCS2 is dependent upon WbemContext objects when modify operations are requested. Thus, any request to NCS2 for a configuration change requires a WbemContext object to be constructed by the user and passed in the WbemServices function call. The following table contains the context qualifiers (named values) used by the NCS2 Provider.

Context Qualifier	Variant Type	Description
ClientSetId	VT_BSTR	A client handle allows the NCS2 software to manage single access to the configuration. The application cannot make any changes to classes without first establishing this; see the section on the IANet_NetService class to see how to establish and use a client handle.
MachineName	VT_BSTR	The name of the machine that is connecting to the IntelNCS2 provider. This is required for logging.

Use Cases

A session handle is required to change a configuration and is managed through the root\IntelNCS2 namespace classes. This identification number allows the NCS2 software to manage single access to the configuration, thereby preventing changes from more than one source at a time. Understanding the role of these client handles is crucial for successful management changes.

Getting a Client Handle

The client must get the object path of the single instance of IANet_NetService before accessing the client handle. There will only be one instance of this class. Before making any changes to the configuration, the client must get a client handle provided by this class through the BeginApply ( ) method. Use this method to obtain a numeric lock ID which will authenticate access requests. Client handles are random numbers generated new each time they are requested. This lock will remain in place until the Apply ( ) method is called or the provider unloads itself from inactivity. Client handles are only required during operation which make a configuration change.

Using a Client Handle in the IWbemContext Object

After the client handle is obtained, a WbemContext object has to be created. Store the client handle in the ClientSetId qualifier of this object. A pointer to this COM object should be passed to every call into IWbemServices. The client handle is not required when making calls to access the IANet_NetService object as this takes the handle as an explicit argument. By passing the client handle as an argument with the method, the software stack can identify the source of the request. Since the client handle is a number, it can be treated as such for assignment purposes.

Finishing with a Client Handle

After changing the configuration, call the IANet_NetService::Apply ( ) method to commit the changes. The client handle ID is passed as an argument to the Apply ( ) method. This may return a follow-up action code (e.g., reboot the system before the

changes can take effect). If any devices became disabled during change operations, committing an Apply ( ) method will enable them.

Error Reporting

This section details how to handle errors generated by the NCS2 provider. How and when an error object is returned depends on whether a call is synchronous, semi-synchronous or asynchronous. In most cases, the HRESULT is set to WBEM_E_FAILED when an error occurs. At this point, however, it is unknown whether WMI or a NCS2 Provider generated the error.

Getting the Error Object

Synchronous Calls

Use GetErrorInfo ( ) to get the IErrorInfo object. Use QueryInterface ( ) to get the IWbemClassObject that contains the error information.

Asynchronous Calls

The IWbemClassObject is passed back as the last item in the last SetStatus ( ) call. After you get the error object instance, you can check the __Class property to determine the origin of the error. WMI creates an instance of __ExtendedStatus, and the NCS2 WMI Provider creates an instance of IANet_ExtendedStatus for errors relating to IANet_ classes and NCS2 WMI Provider. IANet_ExtendedStatus is derived from __ExtendedStatus and contains the following attributes:

Error Codes

For all error codes, the NCS2 provider gives a description customized to the locale. Error codes are in the form of HRESULT with severity set to one (1) and facility set to ITF. An application may use these codes as a basis for a recovery action. See IANet_ExtendedStatus for a list of error codes.

Classes

The following classes are used by Intel® PROSet for Windows* Device Manager and are located in the *root\IntelNCS2* namespace.

Class List

IANet_802dot1QVLANService
IANet_AdapterSetting
IANet_AdapterSettingEnum
IANet_AdapterSettingInt
IANet_AdapterSettingMultiSelection
IANet_AdapterSettingMultiString
IANet_AdapterSettingSlider
IANet_AdapterSettingString
IANet_AdapterToSettingAssoc
IANet_BootAgent
IANet_BootAgent_iSCSI_Adapters
IANet_BootAgentSetting
IANet_BootAgentSettingEnum
IANet_BootAgentSettingInt
IANet_BootAgentSettingString
IANet_BootAgentToBootAgentSettingAssoc
IANet_Device802dot1QVLANServiceImplementation
IANet_DeviceBootServiceImplementation
IANet_DiagConnectionResultStrings
IANet_DiagResult
IANet_DiagResultForMSE
IANet_DiagResultForTest
IANet_DiagResultInPackage
IANet_DiagSetting
IANet_DiagSettingForTest
IANet_DiagTest
IANet_DiagTestForMSE
IANet_EthernetAdapter
IANet_ExtendedStatus
IANet_LogicalEthernetAdapter
IANet_NetService
IANet_NetworkVirtualAdapter
IANet_PhysicalEthernetAdapter
IANet_Setting
IANet_TeamedMemberAdapter
IANet_TeamOfAdapters
IANet_TeamSetting
IANet_TeamSettingEnum
IANet_TeamSettingInt
IANet_TeamSettingMultiSelection
IANet_TeamSettingSlider
IANet_TeamSettingString
IANet_TeamToTeamSettingAssoc
IANet_VLAN
IANet_VLANFor
IANet_VLANSetting
IANet_VLANSettingEnum
IANet_VLANSettingInt
IANet_VLANSettingMultiSelection
IANet_VLANSettingSlider
IANet_VLANSettingString
IANet_VLANToVLANSettingAssoc

IANet_802dot1QVLANService

This class is used to hold the IEEE 802.1Q properties of a network adapter. This class implements the CIM class CIM_802dot1QVLANService.

Instances

An instance of this class exists for each adapter or team that supports IEEE 802.1Q. Each adapter or team can have just one IANet_802dot1QVLANService. Some teams, such as multi-vendor fault tolerant teams do not support this service. The user cannot create instances of this class If the adapter does not have an instance associated with it, then the adapter does not support this service. The user cannot delete instances of this class.

Properties

There are no supported or modifiable properties.

Methods

Method	Returns	Parameters	Detail
CreateVLAN	uint16	[in] uint32 VLANNumber [in] string Name [out] ref:IANet_VLAN	Used to create a VLAN on the adapter or team. The client must supply the VLAN number and the VLAN name, and will get the object path of the newly created VLAN.

Associations

IANet_Device802dot1QVLANServiceImplementation

IANet_AdapterSetting

This abstract class is used to describe a settable property in a configuration. The class is derived from IANet_Setting. Instances of this class will exist for each setting on each adapter. There are several sub-classes for IANet_AdapterSetting. The sub-classes correspond to the different types and ranges of values that settings can take. Each sub-class corresponds to a different style of GUI that may be used to display or change the settings.

Instances

There will be one instance for every class which inherits this one; a single instance for every type of adapter setting.

Properties

See class IANet_Setting for supported properties.

Methods

There are no supported methods.

Associations

Association Class	Association Partner
IANet_AdapterToSettingAssoc	IANet_PhysicalEthernetAdapter

IANet_AdapterSettingInt

The class models a setting that takes an integer value. There are several IANet setting classes used to model integers. The differences between these classes concerns how the integer is displayed and modified by the user interface and how validation is done by the NCS2 WMI Provider. For IANet_AdapterSettingInt, it is expected that the user interface will display an edit box with a spin control.

Instances

An instance of this class exists for each setting that should be displayed as an integer edit box. Users can neither create nor remove instances.

Properties

In addition to the properties supported by IANet_Setting, this class supports:

Name	Type	Description
base	uint64	Base is the root from which an integer value may take values.

max	sint64	The maximum value the integer can take.
min	sint64	The minimum value the integer can take.
Scale	sint64	The unit of measurement to set or estimate series of marks or points at known intervals to measure value of the parameter.
step	sint64	Granularity of the integer value.

Modifiable properties: CurrentValue must be within the range of .min and .max.

Methods

There are no supported methods.

Associations

Inherits an association with IANet_PhysicalEthernetAdapter through IANet_AdapterToSettingAssoc.

IANet_AdapterSettingEnum

The class models an enumeration setting value. For IANet_AdapterSettingEnum, it is expected that the user interface will display a list of strings which map onto a small number of enumerated values. (e.g., a drop list, combo box).

Instances

An instance of this class exists for each setting that will be displayed as an enumeration.

Properties

In addition to the properties supported by IANet_Setting, this class supports:

Name	Type	Description
DescriptionMap	[ ] string	Contains what each value means
PossibleValues	[ ] sint64	An array of possible values allowed for the enum.

Modifiable properties: CurrentValue $\in$ PossibleValues[]

Methods

There are no supported methods.

Associations

Inherits an association with IANet_PhysicalEthernetAdapter through IANet_AdapterToSettingAssoc.

IANet_AdapterSettingMultiString

The class objectifies adapter related driver and network device settings; specifically, it handles multi-string settings.

Instances

An instance of this class exists for each setting that will be as a list of string values. Users can neither create nor remove instances.

Properties

In addition to the properties supported by IANet_Setting, this class supports:

Name	Type	Description
MaxLength	uint32	The maximum length of the string.

Modifiable properties: CurrentValue

Methods

There are no supported methods.

Associations

Inherits an association with IANet_PhysicalEthernetAdapter through IANet_AdapterToSettingAssoc.

IANet_AdapterSettingMultiSelection

This class models a setting whereby the user can select several options from a list of options. For IANet_AdapterSettingMultiSelection, it is expected that the GUI will display multi-selection list box which will allow the user to choose any (or no) option(s).

Instances

An instance of this class exists for each setting that should be displayed as a list of options.

Properties

In addition to the properties supported by IANet_Setting, this class supports:

Name	Type	Description
FirstLabel	string	The label that should be displayed on the left side of the slider.
LastLabel	string	The label that should be displayed on the right side of the slider.
PossibleValues	[ ] sint64	The initial value of the parameter.

Modifiable properties: CurrentValue $\in$ PossibleValues[]

Methods

There are no supported methods.

Associations

Inherits an association with IANet_PhysicalEthernetAdapter through IANet_AdapterToSettingAssoc.

IANet_AdapterSettingString

This class models a setting whereby the user can enter a free-form string value. For IANet_AdapterSettingString, it is expected that the user interface will display an edit box.

Instances

An instance of this class exists for each setting that should be displayed as a string.

Properties

In addition to the properties supported by IANet_Setting, this class supports:

Name	Type	Description
MaxLength	uint32	The maximum length of the string.

Modifiable properties: CurrentValue

Methods

There are no supported methods.

Associations

Inherits an association with IANet_PhysicalEthernetAdapter through IANet_AdapterToSettingAssoc.

IANet_AdapterSettingSlider

The class models a setting that specifically handles Slider settings. For IANet_AdapterSettingSlider, it is expected that the user interface will display a slider which will allow the user to choose the value in a graphical manner – the actual value chosen need not be displayed.

Instances

An instance of this class exists for each setting that will be displayed as a slider. Users can neither create nor remove instances.

Properties

In addition to the properties supported by IANet_Setting, this class supports:

Name	Type	Description
FirstLabel	string	The label that should be displayed on the left side of the slider.
LastLabel	string	The label that should be displayed on the right side of the slider.

PossibleValues	[ ] sint64	The initial value of the parameter.

Modifiable properties: CurrentValue ∈ PossibleValues[]

Methods

There are no supported methods.

Associations

Inherits an association with IANet_PhysicalEthernetAdapter through IANet_AdapterToSettingAssoc.

IANet_AdapterToSettingAssoc

This is an association class between an instance of IANet_PhysicalEthernetAdapter and IANet_AdapterSetting.

Instances

There will be one instance of this class for every adapter setting on an adapter.

Properties

Name	Type	Description
Element	ref	Reference to IANet_PhysicalEthernetAdapter
Setting	ref	Reference to IANet_AdapterSetting

IANet_BootAgent

This class is used to capture information about the network boot capabilities of an adapter (e.g., settings for the PXE Boot Agent supported by some Intel adapters).

Instances

An instance exists for each adapter that supports boot agent capabilities, even if the boot agent is not currently installed. Users can neither create nor remove instances.

Properties

Name	Type	Description	Values	
FlashImageType	uint32	Boot Agent Flash Image type.	0 1 3 4 5 6 7 255	PXE PXE_EFI EFI DISABLED BLANK MISSING iSCSI Unknown
InstalledFlashImageTypes	uint32	Boot Agent flash image types that are currently installed in the ROM.	1 2 4 255	PXE EFI ISCSI Unknown
InvalidImageSignature	boolean	Will be set to true if the boot agent has a corrupted flash image.		
iSCSI_Status	uint32	Boot Agent iSCSI status.	0 1 2 255	iSCSI_PRIMARY iSCSI_SECONDARY iSCSI_DISABLED Unknown
UpdateAvailable	boolean	Indicates if install or upgrade to boot agent software is available.		
Version	string	String describing boot agent version.		
VersionNumber	uint32	Boot agent version in the format x.x.x		

There are no other supported properties.

Modifiable properties: none

Methods

There are two methods on this class that can be used to update the Flash ROM on the NIC:

Method	Returns	Parameters	Detail
ProgramFlash	uint32	[IN] uint32 Action [IN] array of uint8 NewFlashData [OUT] uint32 FlashRetCode	This method is used to update the Flash ROM on the NIC. This will cause the NIC to stop communicating with the network while the flash is updated.
ReadFlash	uint32	[OUT] array of uint8 FlashData	This method reads the Flash ROM on the NIC.

Associations

Association Class	Association Partner
IANet_BootAgentToBootAgentSettingAssoc	IANet_BootAgentSetting
IANet_DeviceBootServiceImplementation	IANet_PhysicalEthernetAdapter

IANet_BootAgent_iSCSI_Adapters

This class is used to capture information about iSCSI supported adapters installed in the system.

Instances

There will be one instance of each adapter which supports iSCSI boot. Users can neither create nor remove instances.

Properties

Name	Type	Description	Values	
AdapterName	string	Friendly name of the adapter.		
Caption	This is an inherited property; refer to parent class CIM definition.			
iSCSI_Status	uint32	The boot agent iSCSI status.	0 1 2 255	iSCSI_PRIMARY iSCSI_SECONDARY iSCSI_DISABLED Unknown
Name	This is an inherited property; refer to parent class CIM definition.			

There are no other supported properties.

Modifiable properties: none

Methods

There is one method of this class which can be used to set the iSCSI priority of adapters:

Method	Returns	Parameters	Detail
SetiSCSI_Status	uint32	[IN] uint32 iSCSI_State [OUT] uint32 RetCode	This method will update the status of adapters that support iSCSI Boot. The function only takes the primary and secondary adapter IDs and sets them accordingly. The remaining adapters are set to disabled. iSCSI_State 0 Set adapter to Primary 1 Set adapter to Secondary 2 Set adapter to Disabled RetCode 0 The state change was successful 1 The state change failed

Associations

There are no associations.

IANet_BootAgentSetting

This abstract class is used to describe a settable property in a configuration. The class is derived from IANet_Setting. Instances will exist for each Boot Agent setting. There are several sub-classes for IANet_BootAgentSetting which correspond to the different types and ranges of values that settings can take.

Instances

There will be one instance for every class which inherits this one; a single instance for every type of adapter boot setting.

Properties

See class IANet_Setting for supported properties.

Methods

There are no supported methods.

Associations

Association Class	Association Partner
IANet_BootAgentToBootAgentSettingAssoc	IANet_BootAgent

IANet_BootAgentSettingEnum

The class models an enumeration setting value.

Instances

An instance of this class exists for each setting that will be displayed as an enumeration.

Properties

In addition to the properties supported by IANet_Setting, this class supports:

Name	Type	Description
DescriptionMap	[ ] string	Contains what each value means.
PossibleValues	[ ] sint64	An array of possible values allowed for the Enum.

Modifiable properties: CurrentValue $\in$ PossibleValues[]

Methods

There are no supported methods.

Associations

Inherits an association with IANet_BootAgent through IANet_BootAgentToBootAgentSettingAssoc

IANet_BootAgentSettingInt

This class objectifies Boot Agent related driver and network device settings. IANet_BootAgentSettingInt specifically handles Integer settings.

Instances

An instance of this class exists for each setting that should be displayed as an integer edit box. Users can neither create nor remove instances.

Properties

In addition to the properties supported by IANet_Setting, this class supports:

Name	Type	Description
base	uint64	Base is the root from which an integer value may take values.
max	sint64	The maximum value the integer can take.
min	sint64	The minimum value the integer can take.
scale	sint64	The unit of measurement to set or estimate series of marks or points at known intervals to measure value of the parameter.
step	sint64	Granularity of the integer value.

Modifiable properties: CurrentValue. Must be within the range of .min and .max.

Methods

There are no supported methods.

Associations

Inherits an association with IANet_BootAgent through IANet_BootAgentToBootAgentSettingAssoc

IANet_BootAgentSettingString

This class objectifies Boot Agent related driver and network device settings. IANet_BootAgentSettingString specifically handles Integer settings

Instances

An instance of this class exists for each boot agent setting that should be displayed as string. Users can neither create nor remove instances.

Properties

In addition to the properties supported by IANet_Setting, this class supports:

Name	Type	Description
MaxLength	uint32	The maximum length of the string.

Modifiable properties: CurrentValue

Methods

There are no supported methods.

Associations

Inherits an association with IANet_BootAgent through IANet_BootAgentToBootAgentSettingAssoc

IANet_BootAgentToBootAgentSettingAssoc

This is an association class between an instance of IANet_BootAgent and IANet_BootAgentSetting.

Instances

There will be one instance of this class for every adapter which supports boot agent configuration.

Properties

Name	Type	Description
Element	ref	Reference to IANet_BootAgent
Setting	ref	Reference to IANet_BootAgentSetting

IANet_Device802dot1QVLANServiceImplementation

This is an association class between an instance of IANet_PhysicalEthernetAdapter and IANet_802dot1QVLANService.

Instances

There will be one instance of this class for every VLAN attached to an adapter. Users can neither create nor remove instances.

Properties

Name	Type	Description
Antecedent	ref	Reference to IANet_PhysicalEthernetAdapter
Dependent	ref	Reference to IANet_802dot1QVLANService

IANet_DeviceBootServiceImplementation

This is an association class between an instance of IANet_PhysicalEthernetAdapter and IANet_BootAgent.

Instances

There will be once instance of this class for every adapter which has a boot agent.

Properties

Name	Type	Description
Antecedent	ref	Reference to IANet_PhysicalEthernetAdapter
Dependent	ref	Reference to IANet_BootAgent

IANet_DiagConnectionResultStrings

This is a class used internally to store strings related to diagnostic connection results. These strings are storied in this class so they can be localized. There are no instances of this class.

IANet_DiagResult

Instances of IANet_DiagResult display result data for a particular test run on a particular Adapter. Instances of this class correspond identically to instances of IANet_DiagTest and IANet_DiagSetting.

Instances

When a diagnostic test is executed, instances of this class are created to hold the results. If the same diagnostic is executed again, previous instances will be replaced. Instances will persist as long as the provider is actively running; after the provider is shut down, all instances of this class will be cleared. The user cannot create instances or delete instances of this class.

For some diagnostic tests, the results are 'packaged' as a set of results. In these cases, there will be a single parent result class instance associated to each packaged result through the IANet_DiagResultInPackage association. Thus, it is possible to execute a single diagnostic test but have multiple results displayed.

Properties

Name	Type	Description	Values
Grouped	boolean	Some of the tests are grouped under specific categories. Grouped is true if this is the case.	
GroupId	uint16	Some of the tests are grouped under specific categories. This parameter specifies the ID of the group under which this test belongs.	
Description	string	Description of the test and its purpose. . Not all diagnostic tests have support for this parameter.	
Name	string	This is an inherited property; refer to parent class CIM definition.	
PackageName	string	Denotes the name of the parent package, if it exists.	
Result	string	A description of the result. Not all diagnostic tests have support for this parameter.	
ResultCode	uint16	An additional code used to describe the type of result. Not all diagnostic tests have support for this parameter.	
TestCompletionTime	datetime	This is an inherited property; refer to parent class CIM definition.	
TimeStamp	datetime	This is an inherited property; refer to parent class CIM definition.	
TestState	uint16	This is an inherited property; refer to parent class CIM definition.	
Title	string	Name of the test. Not all diagnostic tests have support for this parameter.	

There are no other supported properties.

Methods

There are no supported methods.

Associations

Association Class	Association Partner
IANet_DiagResultForTest	IANet_DiagTest

IANet_DiagResultForMSE	IANet_PhysicalEthernetAdapter

IANet_DiagResultInPackage

This is an association class between an instance of an IANet_DiagResult and another IANet_DiagResult. It is used to correlate a single diagnostic result with a parent result, creating a packaged grouping between a single diagnostic result and additional results for that test.

Instances

There will be on instance of this class for every result which is packaged within a parent IANet_DiagResult instance.

Properties

Name	Type	Description
PackageResult	ref	Reference to IANet_DiagResult
Result	ref	Reference to IANet_DiagResult

IANet_DiagResultForMSE

This class relates diagnostic test results to the ManagedSystemElement that was tested.

Instances

There will be one instance of this class for every diagnostic result. Diagnostic tests must be executed before instances of this class will exist.

Properties

Name	Type	Description
Antecedent	ref	Reference to IANet_DiagTest
Dependent	ref	Reference to IANet_PhysicalEthernetAdapter

IANet_DiagResultForTest

This is an association class between an instance of IANet_DiagResult and IANet_DiagTest

Instances

There will be once instance of this class for every diagnostic which has been executed.

Properties

Name	Type	Description
DiagnosticResult	ref	Reference to IANet_DiagResult
DiagnosticTest	ref	Reference to IANet_DiagTest

IANet_DiagSetting

Instances of IANet_DiagSetting provide specific run time diagnostic test directives. Directives used are in common to all tests and are bound to the super class CIM_DiagnosticSetting. These include properties such as ReportSoftErrors and HaltOnError. There are no additional properties added to the subclass IANet_DiagSetting.

Instances

The user cannot create instances or delete instances of this class. There will be one instance for each adapter and test combination.

Properties

Name	Type	Description
HaltOnError	boolean	This is an inherited property; refer to parent class CIM definition.
PercentOfTestCoverage	unit8	This is an inherited property; refer to parent class CIM definition.
QuickMode	boolean	This is an inherited property; refer to parent class CIM definition.

ReportSoftErrors	boolean	This is an inherited property; refer to parent class CIM definition.
ReportStatusMessages	boolean	This is an inherited property; refer to parent class CIM definition.
TestWarningLevel	uint16	This is an inherited property; refer to parent class CIM definition.

There are no other supported properties.

Methods

There are no supported methods.

Associations

Association Class	Association Partner
IANet_DiagSettingForTest	IANet_DiagTest

IANet_DiagSettingForTest

This is an association class between an instance of IANet_DiagTest and IANet_DiagSetting.

Instances

There will be once instance of this class for every diagnostic test and setting for that test.

Properties

Name	Type	Description
Element	ref	Reference to IANet_DiagTest
Setting	ref	Reference to IANet_DiagSetting

IANet_DiagTest

IANet_DiagTest is sub classed from CIM_DiagnosticTest. The class provides a generic vehicle to run and control Diagnostic tests for a supported Ethernet adapter. The super class, CIM_DiagnosticTest, is designed to generically support the testing of any computer hardware on a CIM enabled system. Properties of the class are descriptive in nature and the mechanics of the testing are provided by the exposed methods.

Instances

There is a one to one relationship between available diagnostic tests and instances of this class. Each test is distinguished by a key, which is the concatenation of a diagnostic ID number, the "@" symbol, and the GUID of the referenced adapter (e.g. 1@{12345678-9ABC-DEF0-1234-123456789012}). These unique strings will appear in the "Name" parameter of these class instances. This key value is, in one sense, redundant information, as all information to reference an adapter and test is passed as object parameters to the RunTest and other methods. Still, the instance must be consistent with parameters to the method or the NCS2 WMI Providers will reject the command. Other properties provide other description and run time information. The user cannot create or delete instances of this class.

The following table contains the diagnostic IDs which comprise the "<ID>@" part of the string. You can select which test to run on an adapter by choosing an ID from the table below and pairing it with the GUID of an adapter.

Diagnostic ID	Test Type
1	EEPROM
2	FIFO
3	REGISTER
4	INTERRUPT
17	LOOPBACK
18	EXTENDED LOOPBACK

Diagnostic ID	Test Type
32	LINK & DUPLEX ONLINE
33	LINK & DUPLEX OFFLINE
34	CABLE ONLINE
35	CABLE OFFLINE
36	PING
37	CONNECTION*

The Connection test (37) encompasses both the Ping test (36) and the Online Link & Duplex test (32).

Properties

Name	Type	Description	Values
Characteristics	[ ] uint16	This is an inherited property; refer to parent class CIM definition.	
Grouped	boolean	Some of the tests are grouped under specific categories. Grouped is true if this is the case.	
GroupId	uint16	Some of the tests are grouped under specific categories. This parameter specifies the ID of the group under which this test belongs.	
Name	string	This is an inherited property; refer to parent class CIM definition.	
TestId	uint16	The test ID of the diagnostic test.	

No other properties are supported.

Methods

Method	Returns	Parameters	Detail
RunTest	uint32	[IN] ref : CIM_ManagedSystemElement SystemElement [IN] ref : CIM_DiagnosticSetting Setting [OUT] ref : CIM_DiagnosticResult Result	Runs a test as defined by three parameters referencing: SystemElement defines the adapter, which we are to run the test on by referring to an instance of SystemElement, which will always be the subclass IANet_EthernetAdapter. Setting defines the test to be run, and the manner in which it is run by referring to an instance of CIM_DiagnosticSetting, which will always be the subclass IANet_DiagSetting. Result defines an instance of the class CIM_DiagnosticResult, which will always be the class IANet_DiagResult.
DiscontinueTest		[IN] ref : CIM_ManagedSystemElement SystemElement [IN] ref : CIM_DiagnosticResult Result [OUT] Boolean TestingStopped	Attempts to stop a diagnostic test in progress as defined by two parameters referencing SystemElement and Result. These parameters function the same as RunTest. A third parameter TestingStopped returns a BOOLEAN value, which indicates if the command was successful in stopping the test.
ClearResults		[IN] ref : CIM_ManagedSystemElement SystemElement [OUT] [ ] String ResultsNotCleared	The referenced parameter ManagedSystemElement, combined with this object's object path combine to reference instances of DiagnosticResultForMSE, which will be deleted. Also, all references of DiagnosticResult objects referenced by DiagnosticResultForMSE will be deleted. Also, all instances of Diagnostic-ResultForTest, which refer to the deleted DiagnosticResult objects, will be deleted. Finally, the string array Output parameter ResultsNotCleared will list the keys of the DiagnosticResults, which could not be cleared.

There are no other supported methods

Associations

Association Class	Association Partner
IANet_DiagTestForTest	IANet_DiagResult

IANet_DiagSettingForTest	IANet_DiagSetting
IANet_DiagTestForMSE	IANet_PhysicalEthernetAdapter

IANet_DiagTestForMSE

This is an association class between an instance of a CIM_DiagnosticTest and CIM_ManagedSystemElement.

Instances

There will be once instance of this class for every diagnostic test.

Properties

Name	Type	Description
Antecedent	ref	Reference to IANet_DiagTest
Dependent	ref	Reference to IANet_PhysicalEthernetAdapter

IANet_EthernetAdapter

This is an abstract base class which objectifies network characteristics of an Intel network card. The IANet_EthernetAdapter class is inherited by IANet_LogicalEthernetAdapter and contains properties common to both virtual and physical network devices. If you need information on teaming classes, reference IANet_LogicalEthernetAdapter.

Instances

There will be one instance for every physical Ethernet adapter and team.

Properties

Both child classes, IANet_LogicalEthernetAdapter and IANet_PhysicalEthernetAdapter support different sets of properties. There is very little in common between them. For an accurate list of supported properties, look to these particular class definitions.

Methods

There are no supported methods.

Associations

Association Class	Association Partner
IANet_Device802dot1QVLANServiceImplementation	IANet_802dot1QVLANService

IANet_ExtendedStatus

The NCS2 WMI Provider will return additional information about errors to the user through this class.

Instances

Once an internal error has occurred, an instance of this class will be present.

Properties

Name	Type	Description
ClientSetHandle	uint32	Client lock ID in use at time of exception.
Description	string	Description of the error tailored to the current locale.
File	string	Code file where the error was generated.
Line	uint32	Line number in the code file with the error.
Operation	string	Operation being attempted when the error occurred.
ProviderName	string	Name of the provider that caused the error.
ParameterInfo	string	Class or attribute that was being utilized when the error occurred.
RuleFailureReasons	[ ] string	Reason for operation failure. An operation can fail because a technical rule has failed. (e.g., you must have a management adapter in certain teams).
StatusCode	uint32	Code returned from the internal call that failed: 0x80040901 "WMI: Put property failed"

0x80040902	"WMI: No class object"
0x80040903	"WMI: Failed to create class"
0x80040904	"WMI: Failed to spawn instance of class"
0x80040905	"WMI: Failed to create safe array"
0x80040906	"WMI: Failed to put safe array"
0x80040907	"WMI: Failed to return object to WMI"
0x80040908	"WMI: Get property failed"
0x80040909	"WMI: Unexpected type while getting property"
0x8004090A	"WMI: Class not implemented by this provider"
0x8004090B	"WMI: Unable to parse WQL statement"
0x8004090C	"WMI: Provider only supports WQL"
0x8004090D	"WMI: Parameter in context has the wrong type"
0x8004090E	"WMI: Error formatting debug log"
0x8004090F	"WMI: bad object path"
0x80040910	"WMI: Failed to update setting"
0x80040911	"WMI:[Null parameter passed to method"
0x80040912	"Setting value too small"
0x80040913	"Setting value too big"
0x80040914	"Setting not in step"
0x80040915	"String setting is too long"
0x80040916	"Setting is not one of the allowed values"
0x80040917	"WMI: Qualifier not found"
0x80040918	"WMI: Qualifier set not found"
0x80040919	"WMI: Safe array access failed"
0x8004091A	"WMI: Unhandled exception"
0x8004091B	"WMI: Operation is not supported for this class"
0x8004091C	"WMI: Unexpected event class"
0x8004091D	"WMI: Bad event data"
0x8004091E	"WMI: Operation succeeded with warnings"
0x8004081F	"WMI: The NCS2 Service has been stopped"

Methods

There are no supported methods.

Associations

Association Class	Association Partner
IANet_Device802dot1QVLANServiceImplementation	IANet_802dot1QVLANService

IANet_LogicalEthernetAdapter

This class objectifies the general network characteristics of an Intel ANS team portrayed as a logical device.

Instances

For every team instance there will be one instance of this class. This class implements CIM_EthernetAdapter for a virtual team interface.

Properties

Name	Type	Description
Caption	string	This is an inherited property; refer to parent class CIM definition.
Description	string	This is an inherited property; refer to parent class CIM definition.
DeviceID	string	This is an inherited property; refer to parent class CIM definition.
MiniPortInstance	string	This is an inherited property; refer to parent class CIM definition.
MiniPortName	string	This is an inherited property; refer to parent class CIM definition.
Name	string	This is an inherited property; refer to parent class CIM definition.
StatusInfo	string	This is an inherited property; refer to parent class CIM definition.
Caption	string	This is an inherited property; refer to parent class CIM definition.

All other properties are not supported.

Methods

There are no supported methods

Associations

Association Class	Association Partner
IANet_NetworkVirtualAdapter	IANet_TeamOfAdapters
IANet_TeamToTeamSettingAssoc	IANet_TeamSetting

IANet_NetService

This class enables the client to establish active sessions where changes can be made to the configuration. When requesting or applying a client lock handle, this class must be used : it exposes two methods for performing these operations.

Instances

There is one instance of this object. The client should not rely on the key used for this class. Instead, the client should get the instance of the class by enumerating all instances of IANet_NetService. The user cannot create or delete instances of this class.

Properties

Name	Description
Version	Contains the current version of the core provider

All other properties are not supported.

Methods

Method	Returns	Parameters	Detail
BeginApply	void	[OUT] uint32 ClientSetHandle	Used to get a Client session handle , which should be placed in the context object in the ClientSetId qualifier. Once called, this will, in effect, lock the software stack until an Apply( ) is called.
Apply	void	[IN] uint32 ClientSetHandle [OUT] uint32 FollowupAction	Applies changes made with a particular session handle and releases the session handle after it has been used. The uint32 argument returned is used by the provider to tell the application the server must be rebooted before the changes will take effect. FollowupAction 1 (system reboot required) 0 (no reboot required)

All other methods are not supported.

IANet_NetworkVirtualAdapter

This is an association class between an instance of IANet_TeamOfAdapters and IANet_LogicalEthernetAdapter.

Instances

There will be once instance of this for every team.

Properties

Name	Type	Description
SameElement	ref	Reference to IANet_TeamOfAdapters
SystemElement	ref	Reference to IANet_LogicalEthernetAdapter

IANet_PhysicalEthernetAdapter

IANet_PhysicalEthernetAdapter defines the capabilities and status of all the installed Intel adapters.

Instances

Instances of this class will exist for all installed network adapters. Non-Intel network cards will be represented by an instance of this class, although only a subset of the properties will have values; they do not support some properties specific to Intel network drivers. The user cannot create instances of IANet_PhysicalEthernetAdapter. Each physical port will have an instance of this class. Thus, an adapter with 4 ports will have 4 instances, each representing a distinct port on a single adapter. Deleting an instance of IANet_PhysicalEthernetAdapter will uninstall a physical adapter; a client handle is required for this operation.

Properties

Name	Type	Description	Values
AdapterStatus	uint32	Adapter status specifies the current status of the adapter. This value is the sum of any of the values which apply. Example: 51 = 1 + 2 + 16 + 32	1 Installed 2 DriverLoaded 4 HardwareMissing 16 HasDiag 32 HasLink 1024 HasTCOEnabled 2048 DeviceError
AdditionalAvailability	uint16[ ]	This is an inherited property; refer to parent class CIM definition.	
Availability	uint16	This is an inherited property; refer to parent class CIM definition.	
BusType	uint16	Bus Type indicates the bus type.	0 Unknown 1 ISA 2 EISA 3 PCMCIA 4 Cardbus 5 PCI 6 PCI-X 7 PCI Express
Capabilities	uint16[ ]	Capabilities of the Ethernet adapter. Some capabilities are dependent upon feature discovery in the operating system. Therefore, a capability may not be present because operating system requirements have not been met. Capability IDs 0 Unknown 1 Other 2 AlertOnLan 3 WakeOnLan 4 Adapter Fault Tolerance 5 Adaptive Load Balancing 6 IPSec Offload 7 ASF 8 GEC/802.3ad Static Link Aggregation 9 Static Link Aggregation 10 IEEE 802.3ad Dynamic Link Aggregation 11 Checksum Offload 12 Switch Fault Tolerance 13 Basic AlertOnLan 14 AlertOnLan 2 15 Security Offload AH 16 Security Offload ESP 17 Security Payload Tunnel 18 Security Payload Transport 19 Security IPV4 Packets 20 Authentication Algorithm MD5 21 Authentication Algorithm SHA1 22 Encryption Algorithm EAS 23 Encryption Algorithm DES	28 ESP Receive Checksum Authentication 29 TCO Capability 30 Wake Up Capabilities 31 IP Checksum Offload 32 10 Mbps 33 100 Mbps 34 1000 Mbps 35 10000 Mbps 36 Teaming 37 VLAN 38 IEEE VLAN 39 ISL VLAN 40 Uninstallable 41 Identify Adapter Support 42 Cable Test Support 43 Diagnostic Support 44 Flash support 45 ICH Support 46 Usage Server 47 Vendor Intel 48 Phoneline PHY 49 Mobile 50 PowerManagement Support 51 Feature Not Supported 52 MFO 53 Pass Through 54 Quad-Port Support 55 Dedicated MAC Address 56 Jumbo Frame Support 57 Feature Not Supported 58 Signal Quality Test

		24 Encryption Algorithm 3DES 25 ESP Xmit Checksum Encryption 26 ESP Xmit Checksum Authentication 27 ESP Receive Checksum Encryption 59 Cable Offline Test 60 Adapter is LOM 61 Scalable Networking Pack Capability 62 CB Platform Capability 63 iSCSI Capability 64 LinkSEC Support
Caption	string	This is an inherited property; refer to parent class
ControllerID	uint32	The Controller ID identifies the Ethernet controller that the adapter uses. Adapters with different DeviceIDs can have the same Controller ID. 0 Unknown 1 Intel 82542 3 Intel 82543 6 Intel 82544 7 Intel 82540 8 Intel 82545 11 Intel 82541 13 Intel 82547 20 Intel 82571 30 Intel 82573 31 Intel 82574 40 Intel ESB2LAN 50 Intel ICH8 51 Intel ICH9 52 Intel ICH10 60 Intel 82575 62 Intel 82576 63 Intel ADORAM_VIRTUAL 65537 Intel D100_A_STEP 65538 Intel D100_B_STEP 65539 Intel D100_C_STEP 65540 Intel D101_A_STEP 65541 Intel D101_B0_STEP 65542 Intel D101M_A_STEP 65543 Intel D101S_A_STEP 65544 Intel D102_A_STEP 65545 Intel D102_B_STEP 65546 Intel D102_C_STEP 65547 Intel D102_D_STEP 65548 Intel D102_E_STEP 65549 Intel D102_F_STEP 65550 Intel 82562_G 65551 Intel 82562_GZ 65552 Intel 82562_GX_GT 65553 Intel 82562 131073 Intel 82597 EX 196609 Intel 82598 196610 Intel 82599
Description	string	This is an inherited property; refer to parent class CIM definition.
DeviceID	string	This is an inherited property; refer to parent class CIM definition.
EEPROMVersion	string	EEPROM version of the device.
HardwareStatus	uint32	Hardware status specifies the current status of the hardware. 0 Unknown 1 Ready 2 Initializing 3 Reset 4 Closing 5 Not Ready
MaxSpeed	uint16	This is an inherited property; refer to parent class CIM definition.
MediaType	uint16	MediaType indicates the media which interfaces to this PHY. 0 Unknown 1 Copper 2 Fiber 3 Phone Line 4 CX4 Copper 5 SFP+ Direct Attach 6 SR Fiber 7 LR Fiber 8 KX/KX4 Backplane
MiniPortInstance	string	This is an inherited property; refer to parent class CIM definition.
MiniPortName	string	This is an inherited property; refer to parent class CIM definition.
Name	string	This is an inherited property; refer to parent class CIM definition.
NegotiatedLinkWidth	uint16	Negotiated Link Width specifies the negotiated link width of the bus. Only PCI-Express adapters will have a non zero value. 0 Unknown 1 x1 2 x2 4 x4

NetworkAddresses	string[ ]	This is an inherited property; refer to parent class CIM definition.
OriginalDisplayName	string	If teaming is enabled on this adapter OriginalDisplayName will contain the original display name of the adapter.
PartNumber	string	PartNumber is the NIC's PBA manufacturing part number.
PCIDeviceID	string	PCI device Id of the device.
PermanentAddress	string	This is an inherited property; refer to parent class CIM definition.
PortNumber	uint16	PortNumber indicates the port number on PCIe Quad port adapters. Any other value indicates this field is not applicable to the adapter. 0 A 1 B 2 C 3 D
SlotID	string	SlotID field of the System Slot structure provides a mechanism to correlate the physical attributes of the slot to its logical access method.
Speed	uint64	This is an inherited property; refer to parent class CIM definition.
Status	string	This is an inherited property; refer to parent class CIM definition.
StatusInfo	uint16	This is an inherited property; refer to parent class CIM definition.

No other properties are supported. There are no user modifiable properties.

Methods

Method	Returns	Parameters	Detail
GetAdapterFanStatus	uint32	[OUT] uint32 dwAdapterStatus	Returns status of adapter fans. Not supported on all adapters.
GetNDISVersion	uint32	[OUT] uint32 dwMajorVersion dwMinorVersion	This method can be used to get the NDIS version
GetPowerUsageOptions	uint32	[OUT] uint32 AutoPowerSaveEnabled ReduceSpeedOnPowerDown SmartPowerDown SavePowerNowEnabled EnhancedASPMPowerSaver ACBSMode LinkSpeedBatterySaver	Detects any optional power usage settings (e.g., power usage for standby, battery operation, etc.). For all return uint32 values: 0 = Off 1 = On
GetWakeOnLanPowerOptions	uint32	[IN] uint32 WakeFromPoweroff WakeOnLink WakeOnMagicPacket WakeOnDirectedPacket	GetWakeOnLanPowerOptions returns WakeOnLan power settings. For example, information about wakeonlink, wakeonmagicpacket etc.. If an adapter does not support this feature, the returned structure will be empty. Values to pass in: 0 = Off 1 = On
IdentifyAdapter	uint32	[IN] uint16 nSeconds	Identifies adapter by flashing the light on the adapter for a few seconds. This method will only work for physical adapters.
IsISCSIEnabled	uint32	[OUT] uint32 iSCSIStatus	This method can be used to check if iSCSI is enabled on that adapter. iSCSIStatus 0 – Unavailable 1 – Disabled

			2 – Primary 3 - Secondary
IsiSCSISupported	uint32	[OUT] boolean bIsiSCSIOS bIsISCSIPatch bIsISCSIHotFix	This method can be used to check if iSCSI is supported by the OS and iSCSI patch and hot fix are installed. The "hot fix" is also known as the Microsoft iSCSI initiator.
IsSetPowerMgmtCapabilitiesReq	uint32	[OUT] boolean bIsSetRequired	This method can be used to check if SetPowerMgmt-Capabilities() needs to be called.
SetPowerMgmtCapabilities	uint32	This method is used to makes changes to the Power management capabilities during NCS2 install so that any upgrade scenarios from earlier releases will have the right options for all the WakeOnLan options and NCS2 will not have reinterpret them dynamically.	
SetPowerUsageOptions	uint32	[IN] uint32 AutoPowerSaveModeEnabled ReduceSpeedOnPowerDown SmartPowerDown SavePowerNowEnabled EnchancedASPM-PowerSaver ACBSMode LinkBatterySaver	Changes power usage options (e.g., method can be used to reduce power usage for standby, battery operation, etc.) Note: Power usage settings are stored and used for subsequent reboots. Values to pass in: 0 = Off 1 = On
SetWakeOnLanPowerOptions	uint32	[IN] uint32 WakeFromPoweroff WakeOnLink WakeOnMagicPacket WakeOnDirected-Packet	This method can be used to makes changes to the WakeOnLan options. For example, this method could be used to set options like wakefromPoweroff, wakeOnlink, WakeOn-MagicPacket, WakeOn-DirectedPacket etc. Note WakeOnLan settings are stored and used for every boot. 0 = Off 1 = On
ValidateSettingOnNewTeam	Internal use only.		

There are no other supported methods.

Associations

Association Class	Association Partner
IANet_Device802dot1QVLANServiceImplementation	IANet_802dot1QVLANService
IANet_DiagTestForMSE	IANet_DiagTest
IANet_DiagResultForMSE	IANet_DiagResult
IANet_DeviceBootServiceImplementation	IANet_BootAgent
IANet_AdapterToSettingAssoc	IANet_AdapterSetting
IANet_TeamedMemberAdapter	IANet_TeamOfAdapters

IANet_Setting

This is an abstract super class for a set of concrete classes of different types. This set of classes allows open ended usage of a variable number of settings. These will be different between adapters, teams, or VLANs and it may not always be possible to predict what parameters are required. Between the setting categories, this class groups the most common parameters for inheritance.

Instances

There will be one instance for every setting.

Properties

Name	Type	Description
Caption	string	This is an inherited property; refer to parent class CIM definition.
Description	string	This is an inherited property; refer to parent class CIM definition.
ExposeLevel	uint32	Internal use only
Grouped	boolean	Internal use only
GroupId	uint16	Internal use only
MiniHelp	string	Description of the setting
ParentId	string	The unique identifier (GUID) of the parent device
ParentType	string	Name of the parent (NIC, Team, VLAN)
Writable	boolean	Whether the value can be changed

All other properties are not supported.

Methods

There are no supported methods.

Associations

There are no associations.

IANet_TeamedMemberAdapter

This class is used to associate the adapter with the team, determine the function of the adapter in the team, and establish that the adapter is currently active in the team. To add an adapter to a team, create an instance of IANet_TeamedMemberAdapter to associate the adapter with the team. To remove an adapter from the team, remove the instance of IANet_ TeamedMemberAdapter. The adapter will no longer be part of the team and may be bound to an IP protocol endpoint after the Apply() function is called.

Instances

An instance of this class exists for each adapter that is a member of a team. The user cannot create instances or delete instances of this class.

Properties

Name	Type	Description	Values
AdapterFunction	uint32	Describes how the adapter is used in the team. The AdapterFunction property of this class may be modified to describe how the adapter is used.	0 Unknown 1 Primary Adapter 2 Secondary Adapter 3 Other
AdapterStatus	uint32	Describes the adapter's status within the team.	0 Unknown 1 Active 2 Standby 3 InActive
GroupComponent	ref	Reference to IANet_TeamOfAdapters	
PartComponent	ref	Reference to IANet_PhysicalEthernetAdapter	

All other properties are not supported.

IANet_TeamOfAdapters

This class has members that describe the type of the team, the number of adapters in the team, and the maximum number of adapters that can be in the team.

Instances

There is an instance of this class for each Intel adapter team. To remove a team the user should delete the instance of IANet_TeamOfAdapters. The NCS2 WMI Provider will delete the associations to the team members, and will also delete the virtual adapter and settings for the team.

Properties

Name	Type	Description	Values
AdapterCount	uint32	The number of adapters currently in the team.	
Caption	string	This is an inherited property; refer to parent class CIM definition.	
Description	string	This is an inherited property; refer to parent class CIM definition.	
LoadBalancedGroup	boolean	This is an inherited property; refer to parent class CIM definition.	
MaxAdapterCount	uint32	The maximum number of adapters that can be placed in this team.	
MFOEnabled	boolean	The MFO status in the current team.	
Name	string	This is an inherited property; refer to parent class CIM definition.	
RedundancyStatus	uint16	This is an inherited property; refer to parent class CIM definition.	
StaticIPAddress	string	The static IP address assigned to the team, otherwise this is 0.0.0.0	
Status	string	This is an inherited property; refer to parent class CIM definition.	
SubnetMask	string	The subnet mask assigned to the team, otherwise this is 0.0.0.0	
TeamingMode *	uint32	The type of the current team.	0 AFT 1 ALB 2 SLA 4 IEEE 802.3ad 5 SFT 255 Unknown
TeamMACAddress	string	The configured MAC address of this team.	

There are no other supported properties

Methods

Method	Returns	Parameters	Detail
CreateTeam	uint32	[IN] array of ref Adapters [IN] uint32 TeamingMode [IN] string TeamName [IN] boolean MFOEnable [OUT] ref TeamPath	CreateTeam adds a new Intel NIC Team to the system. The 1st input parameter Adapter is a reference to an array of IANet_PhysicalEthernetAdapter which will be added to this team. TeamingMode is the desired mode of the team to be created and TeamName is the unique name to be given to the new team. TeamingMode: 0 AFT 1 ALB 2 SLA 4 IEEE 802.3ad 5 SFT * the array of Adapter references must contain strings representing paths to an instance of IANet_PhysicalEthernetAdapter.

RenameTeam	uint32	[IN] string TeamName	Changes the name of an existing Intel team in the system.
TestSwitchConfiguration	uint32	[out] uint16 [ ] CauseMessageId [out] string [ ] strCause [out] uint16 [ ] SolutionMessageId [out]string [ ] strSolution	Tests the switch configuration to ensure that the team is functioning correctly with the switch. This test can be used to check that link partners i.e., a device that an adapter links to, such as another adapter, hub, switch, etc., support the chosen adapter teaming mode. For example, if the adapter is a member of a Link Aggregration team, then this test can verify that link partners connected to the adapter support Link Aggregation
ValidateAddAdapters	uint32	[in] [ ] ref: IANet_PhysicalEthernetAdapter Adapters [out] uint16 ValResult	Validates the adapters which will be added to this team. The function will return 0 in .ValResult if the adapter can be added.
ValidateSetting	uint32	[in] ref: IANet_PhysicalEthernetAdapter Adapter [in] string SettingName [in] sint64 Value [out] uint16 ValResult	Validates the member adapter setting value before the setting is actually changed.

Associations

Association Class	Association Partner
IANet_VirtualNetworkAdapter	IANet_LogicalEthernetAdapter
IANet_TeamedMemberAdapter	IANet_PhysicalEthernetAdapter

IANet_TeamSetting

This abstract class is used to describe a settable property in a configuration and contains an important association between an instance of a team and an instance of a particular setting. There are several sub-classes for IANet_TeamSetting. The sub-classes correspond to the different types and ranges of values that settings can take.

Instances

Instances of this class will exist for each setting on each Team.

Properties

See class IANet_Setting for supported properties.

Methods

There are no supported methods.

Associations

Association Class	Association Partner
IANet_TeamToTeamSettingAssoc	IANet_PhysicalEthernetAdapter

IANet_TeamSettingInt

The class models a setting that takes an integer value. There are several IANet setting classes used to model integers. The differences between these classes concerns how the integer is displayed and modified by the GUI, and how validation is done by the NCS2 WMI Provider. For IANet_TeamSettingInt, it is expected that the GUI will display an edit box with a spin control.

Instances

An instance of this class exists for each setting that should contain an integer value.

Properties

In addition to the properties supported by IANet_Setting, this class supports:

Name	Type	Description
base	uint64	Base is the root from which an integer value may take values.
max	sint64	The maximum value the integer can take.
min	sint64	The minimum value the integer can take.
Scale	sint64	The unit of measurement to set or estimate series of marks or points at known intervals to measure value of the parameter.
step	sint64	Granularity of the integer value.

Unsupported properties: refer to IANet_Setting

Modifiable properties : CurrentValue. Must be within the range of .min and .max.

Methods

There are no supported methods.

Associations

Inherits an association with IANet_LogicalEthernetAdapter through IANet_TeamToTeamSettingAssoc.

IANet_TeamSettingEnum

The class models an enumeration setting value. For IANet_TeamSettingEnum, it is expected that the GUI will display a list of strings which map onto a small number of enumerated values. (e.g., a drop list combo box).

Instances

An instance of this class exists for each setting that will be displayed as an enumeration.

Properties

In addition to the properties supported by IANet_Setting, this class supports:

Name	Type	Description
DescriptionMap	[ ] string	Contains what each value means
PossibleValues	[ ] sint64	An array of possible values allowed for the Enum.

Unsupported properties: refer to IANet_Setting

Modifiable properties : CurrentValue $\in$ PossibleValues[]

Methods

There are no supported methods.

Associations

Inherits an association with IANet_LogicalEthernetAdapter through IANet_TeamToTeamSettingAssoc.

IANet_TeamSettingSlider

The class models a setting that specifically handles Slider settings. For IANet_AdapterSettingSlider, it is expected that the GUI will display a slider which will allow the user to choose the value in a graphical manner – the actual value chosen need not be displayed.

Instances

An instance of this class exists for each setting that will be displayed as a slider. Users can neither create nor remove instances.

Properties

In addition to the properties supported by IANet_Setting, this class supports:

Name	Type	Description
FirstLabel	string	The label that should be displayed on the left side of the slider.
LastLabel	string	The label that should be displayed on the right side of the slider.
PossibleValues	[ ] sint64	The initial value of the parameter.

Unsupported properties: refer to IANet_Setting

Modifiable properties: CurrentValue ∈ PossibleValues[]

Methods

There are no supported methods.

Associations

Inherits an association with IANet_LogicalEthernetAdapter through IANet_TeamToTeamSettingAssoc.

IANet_TeamSettingMultiSelection

This class models a setting whereby the user can select several options from a list of options. For IANet_AdapterSettingMultiSelection, it is expected that the GUI will display multi-selection list box which will allow the user to choose any (or no) option(s).

Instances

An instance of this class exists for each setting that should be displayed as a list of options.

Properties

In addition to the properties supported by IANet_Setting, this class supports:

Name	Type	Description
FirstLabel	string	The label that should be displayed on the left side of the slider.
LastLabel	string	The label that should be displayed on the right side of the slider.
PossibleValues	[ ] sint64	The initial value of the parameter.

Unsupported properties: refer to IANet_Setting

Modifiable properties: CurrentValue ∈ PossibleValues[]

Methods

There are no supported methods.

Associations

Inherits an association with IANet_LogicalEthernetAdapter through IANet_TeamToTeamSettingAssoc.

IANet_TeamSettingString

This class models a setting whereby the user can enter a free-form string value. For IANet_AdapterSettingString, it is expected that the GUI will display an edit box.

Instances

An instance of this class exists for each setting that should be displayed as a string.

Properties

In addition to the properties supported by IANet_Setting, this class supports:

Name	Type	Description
MaxLength	uint32	The maximum length of the string.

Unsupported properties: refer to IANet_Setting

Modifiable properties: CurrentValue

Methods

There are no supported methods.

Associations

Inherits an association with IANet_LogicalEthernetAdapter through IANet_TeamToTeamSettingAssoc.

IANet_TeamToTeamSettingAssoc

This is an association class between an instance of a team and a setting on that team.

Instances

There will be one instance of this class for every setting on a team.

Properties

Name	Type	Description
Element	ref	Reference to IANet_LogicalEthernetAdapter
Setting	ref	Reference to IANet_TeamSetting

IANet_VLAN

This class holds the information for each Intel VLAN. This class implements CIM_VLAN.

Instances

An instance of this class will exist of each Intel VLAN. To create a VLAN, call CreateVLAN from the appropriate instance of IANet_802dot1QVLANService.The user can remove an instance of this class to remove the corresponding VLAN.

Properties

Name	Type	Description	Values
Caption	string	This is an inherited property; refer to parent class CIM definition.	
Description	string	This is an inherited property; refer to parent class CIM definition.	
Name	string	This is an inherited property; refer to parent class CIM definition.	
ParentID	uint16	Contains the VLAN's parent device ID.	
ParentType	uint16	Contains the VLAN's parent device type.	0 Adapter 1 Team 2 Unknown
StaticIPAddress	string	This field has a value if the VLAN is configured to have a static IP address. Otherwise, it will be set to 0.0.0.0	
StatusInfo	uint16	This is an inherited property; refer to parent class CIM definition.	
SubnetMask	string	This field has a value if the VLAN is configured to have a subnet mask. Otherwise, it will be set to 0.0.0.0	
VLANName	string	This is the name of the VLAN chosen by the user.	
VLANNumber	uint32	This is the VLAN's identifying number.	

No other properties are supported.

The user is able to modify the VLANNumber and Caption attribute.

Methods

There are no supported methods.

Associations

Association Class	Association Partner
IANet_VLANToVLANSettingAssoc	IANet_VLANSetting

IANet_VLANFor

This is an association class between an instance of CIM_VLAN and CIM_VLANService.

Instances

There will be one instance of this class for every VLAN.

Properties

Name	Type	Description
Antecedent	ref	Reference to CIM_VLAN

Dependent	ref	Reference to CIM_VLANService

IANet_VLANSetting

This abstract class is used to describe a settable property in a configuration and contains an important association between an instance of a VLAN and an instance of a particular setting.. The class is derived from IANet_Setting. Instances of this class will exist for each setting on each VLAN. There are several sub-classes for IANet_VLANSetting. The sub-classes correspond to the different types and ranges of values that settings can take.

Instances

There will be one instance for every class which inherits this one; a single instance for every type of VLAN setting.

Properties

See class IANet_Setting for supported properties.

Methods

There are no supported methods.

Associations

Association Class	Association Partner
IANet_AdapterToSettingAssoc	IANet_PhysicalEthernetAdapter

IANet_VLANSettingInt

The class models a setting that takes an integer value. There are several IANet setting classes used to model integers. The differences between these classes concerns how the integer is displayed and modified by the GUI, and how validation is done by the NCS2 WMI Provider. For IANet_AdapterSettingInt, it is expected that the GUI will display an edit box with a spin control.

Instances

An instance of this class exists for each setting that should be displayed as an integer edit box. Users can neither create nor remove instances.

Properties

In addition to the properties supported by IANet_Setting, this class supports:

Name	Type	Description
base	uint64	Base is the root from which an integer value may take values.
max	sint64	The maximum value the integer can take.
min	sint64	The minimum value the integer can take.
Scale	sint64	The unit of measurement to set or estimate series of marks or points at known intervals to measure value of the parameter.
step	sint64	Granularity of the integer value.

Unsupported properties: refer to IANet_Setting

Modifiable properties : CurrentValue. Must be within the range of .min and .max.

Methods

There are no supported methods.

Associations

Inherits an association with IANet_VLAN through IANet_VLANToVLANSettingAssoc.

IANet_VLANSettingEnum

The class models a enumeration setting value. For IANet_AdapterSettingEnum, it is expected that the GUI will display a list of strings which map onto a small number of enumerated values. (e.g., a drop list combo box).

Instances

An instance of this class exists for each setting that will be displayed as an enumeration.

Properties

In addition to the properties supported by IANet_Setting, this class supports:

Name	Type	Description
DescriptionMap	[ ] string	Contains what each value means.
PossibleValues	[ ] sint64	An array of possible values allowed for the Enum.

Unsupported properties: refer to IANet_Setting

Modifiable properties: CurrentValue ∈ PossibleValues[]

Methods

There are no supported methods.

Associations

Inherits an association with IANet_VLAN through IANet_VLANToVLANSettingAssoc.

IANet_VLANSettingSlider

The class models a setting that specifically handles Slider settings. For IANet_AdapterSettingSlider, it is expected that the GUI will display a slider which will allow the user to choose the value in a graphical manner – the actual value chosen need not be displayed.

Instances

An instance of this class exists for each setting that will be displayed as a slider. Users can neither create nor remove instances.

Properties

In addition to the properties supported by IANet_Setting, this class supports:

Name	Type	Description
FirstLabel	string	The label that should be displayed on the left side of the slider.
LastLabel	string	The label that should be displayed on the right side of the slider.
PossibleValues	[ ] sint64	The initial value of the parameter.

Unsupported properties: refer to IANet_Setting

Modifiable properties : CurrentValue ∈ PossibleValues[]

Methods

There are no supported methods.

Associations

Inherits an association with IANet_VLAN through IANet_VLANToVLANSettingAssoc.

IANet_VLANSettingMultiSelection

This class models a setting whereby the user can select several options from a list of options. For IANet_AdapterSettingMultiSelection, it is expected that the GUI will display multi-selection list box which will allow the user to choose any (or no) option(s).

Instances

An instance of this class exists for each setting that should be displayed as a list of options.

Properties

In addition to the properties supported by IANet_Setting, this class supports:

Name	Type	Description
FirstLabel	string	The label that should be displayed on the left side of the slider.
LastLabel	string	The label that should be displayed on the right side of the slider.
PossibleValues	[ ] sint64	The initial value of the parameter.

Unsupported properties: refer to IANet_Setting

Modifiable properties : CurrentValue

Methods

There are no supported methods.

Associations

Inherits an association with IANet_VLAN through IANet_VLANToVLANSettingAssoc.

IANet_VLANSettingString

This class models a setting whereby the user can enter a free-form string value. For IANet_VLANSettingString, it is expected that the GUI will display an edit box.

Instances

An instance of this class exists for each setting that should be displayed as a string.

Properties

In addition to the properties supported by IANet_Setting, this class supports:

Name	Type	Description
MaxLength	uint32	The maximum length of the string

Methods

There are no supported methods.

Associations

Inherits an association with IANet_VLAN through IANet_VLANToVLANSettingAssoc.

IANet_VLANtoVLANSettingAssoc

This is an association class between an instance of a VLAN and a setting on the VLAN.

Instances

There will be one instance of this class for every setting on a VLAN

Properties

Name	Type	Description
Element	ref	Reference to IANet_VLAN
Setting	ref	Reference to IANet_VLANSetting

Appendix

This section contains specific information to help users working with the NCS2 architecture.

Related Documents

- CIM schema version 2.0, 2.2 published by Distributed Management Task Force (DMTF), http://www.dmtf.org.
- Microsoft Windows Management Instrumentation (and other manageability information) http://www.microsoft.com/hwdev/WMI/.
- Web-based Enterprise Management (WBEM) initiative by DMTF http://www.dmtf.org/wbem/index.html.

Terminology

Term	Explanation
ANS	Advanced Networking Services (ANS) teaming is a feature of the Intel® Advanced Networking Services component that lets you group multiple adapters in a system into a team.
API	An Application Programming Interface exposed by a library or system for service requests.
CIM	Common Information Model; a standard for describing computers and services.
CIMOM	CIM Object Manager; part of Windows Management.
COM	Component Object Model; a Microsoft platform for inter-process communication and object creation.
DMiX	Acronym for Intel® PROSet for Windows* Device Manager
DMTF	Distributed Management Task Force; a standards organization for the IT industry.
GUI	Graphical User Interface; refers to the user interface layer of Intel® PROSet for Windows* Device Manager
MOF	Managed Object Format; a file extension of a special file format used in Windows management.
NCS2	Network Configuration Services 2.0 - the architecture used in Intel® PROSet for Windows* Device Manager
VLAN	Virtual LAN; a method for creating logical networks within a physical network.
WBEM	Web Based Enterprise Management; technologies to unify distributed computing environments.
WMI	Windows Management Instrumentation; Microsoft's implementation of the CIM standard for Windows.

Working Examples

Getting Current Configuration

The client does not need to get a client handle to read the current configuration. Clients can use a NULL context, however, any error messages will be returned in the default language for the managed machine. In the following tables, items enclosed in { } are object paths. These paths are assumed to have been obtained from previous WQL queries. The client should never need to construct an object path without doing a query. The __PATH attribute of every object contains the object path for that object. In all the following use cases, the methods IWbemServices::ExecQuery or IWbemServices::ExecQueryAsync are used to execute WQL queries.

Physical Adapters

The main class for adapters is IANet_PhysicalEthernetAdapter. This class is used for both physical and virtual adapters, and the client needs to know how to distinguish between them.

Task	WQL Query	Result Class	Comment
Enumerate all adapters	SELECT * FROM IANet_EthernetAdapter	IANet_EthernetAdapter	Returns all IANet_EthernetAdapter instances. This is equivalent to IWbemServices::CreateInstanceEnumAsync.
Determine if adapter is	ASSOCIATORS OF {adapter path} WHERE AssocClass =	IANet_TeamOfAdapters	If the query results in no classes then the adapter is a real adapter.

virtual	IANet_NetworkVirtualAdapter

Team Configuration

The main classes in the teaming schema are IANet_LogicalEthernetAdapter, IANet_TeamOfAdapters, IANet_NetworkVirtualAdapter and IANet_TeamedMemberAdapter in the root\IntelNCS2 namespace. The association class IANet_NetworkVirtualAdapter contains no useful data – clients are really only interested in the endpoints of this association. IANet_TeamedMemberAdapter does contain useful data about how the member adapter is used within the team.

Task	WQL Queries	Result Class	Comments
Enumerate all teams	SELECT * FROM IANet_TeamOfAdapters	IANet_TeamOfAdapters	There is one instance of IANet_TeamOfAdapters for each team.
Get the virtual adapter for a team	ASSOCIATORS OF {IANet_TeamOfAdapters Path} WHERE AssocClass = IANet_NetworkVirtualAdapter	IANet_LogicalEthernetAdapter	Returns only the adapter object for the virtual adapter in the team. Apply must be called before this instance will exist.
Enumerate the team's member adapters	ASSOCIATORS OF {IANet_TeamOfAdapters path} WHERE AssocClass = IANet_TeamedMemberAdapter	IANet_PhysicalEthernetAdapter	Returns the adapters which are in the team, but does not describe what role the adapter plays.
Determine an adapter's role in a team	REFERENCES OF {IANet_PhysicalEthernetAdapter path} WHERE ResultClass = IANet_TeamedMemberAdapter	IANet_TeamedMemberAdapter	The class contains information about how the member adapter relates to the team and its current status within the team.

VLAN Configuration

Any adapter or team supporting VLANs has an IANet_802dot1QVLANService associated with it, using the association class IANet_Device802do1QVVLANServiceImplementation. If an adapter or team does not have an instance of this class associated with it, then it does not support VLANs. Each VLAN is represented by an instance of IANet_VLAN in the root\IntelNCS2 namespace. IANet_VLAN does not have a direct association – it is associated with the corresponding IANet_802dot1QVLANService for the adapter or team. The association class IANet_VLANFor is used to associate each VLAN instance with the correct ANet_802dot1QVLANService.

Task	WQL Queries	Result Class	Comments
Get the 802.1q VLAN service object associated with an adapter	ASSOCIATORS OF {IANet_EthernetAdapter path} WHERE ResultClass = IANet_802dot1QVLANService	IANet_802dot1QVLANService	Returns one or no object(s).
Get the VLANs on an adapter	ASSOCIATORS OF {IANet_802dot1QVLANService path} WHERE ResultClass = IANet_VLAN	IANet_VLAN	This can return no objects if there are no VLANs installed.

Boot Agent Information

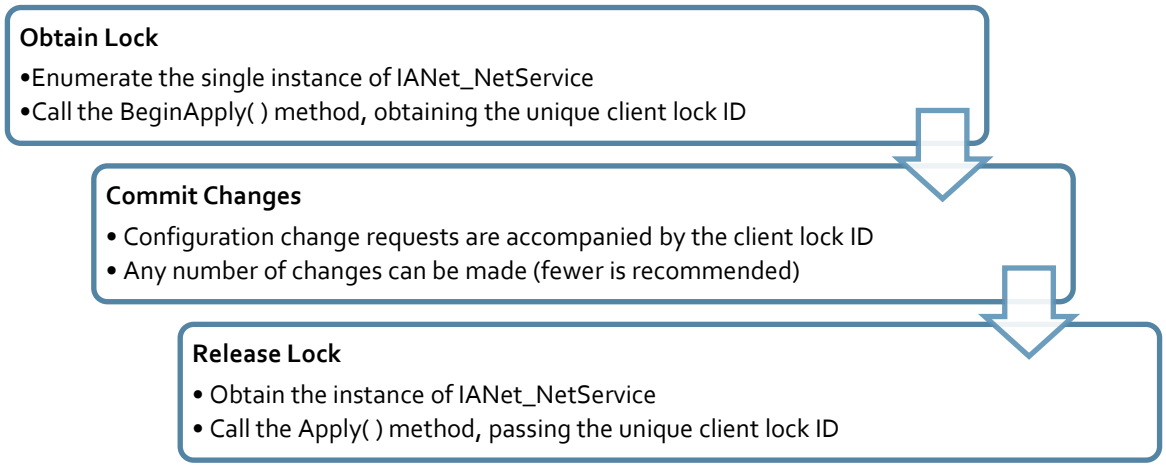
Each adapter that can support a boot agent in flash ROM will have an IANet_BootAgent instance associated with it using the IANet_DeviceBootServiceImplementation association class.

Task	WQL Queries	Result Class	Comments
Get the Boot Agent associated with an adapter	ASSOCIATORS OF {path of IANet_EthernetAdapter} WHERE ResultClass = IANet_BootAgent	IANet_BootAgent	The following read only properties provide information on the boot ROM image for this adapter: InvalidImageSignature, Version, UpdateAvailable, FlashImageType

Updating the Configuration

Client Locks

A client lock ID is used to authenticate the source of changes and to make sure two sources are not trying to make changes at the same time. When submitting configuration requests, a sequence of smaller changes is recommended rather than submitting all changes within a single session. For example, creating a team and then adding VLANs to the team should be accomplished in two separate lock and apply sequences.



Obtain Lock

In most cases, to update the configuration, the client application will need to get a client handle from the IANet_NetService class and store this handle in a WbemContext context object. The handle is simply an integer and is retrieved from the IANet_NetService BeginApply ( ) function. Changes to the configuration will finalized when the "Apply" method on the IANet_NetService is called. WbemContext is a user created object which has to be customized for use with the NCS2 provider. Discussion of the WbemContext objects is located at WBEM Context. The following code sample shows how this might be accomplished:

VBScript

```
Set colNetServiceObjects = mWbemServices.ExecQuery("Select * from IANet_NetService", , 16)
For Each NetServiceObject In colNetServiceObjects
        Set objReturn = NetServiceObject.ExecMethod_("BeginApply")
        If objReturn.ReturnValue = 0 Then
                iClientID = CInt(objReturn.ClientSetHandle)
        End If
Next
```

Commit Changes

The client ID lock cannot be simply passed into functions as an integer argument. It has to be contained within a context qualifier. Context qualifiers are optional additional information which can be passed to WMI providers; use of this qualifier is mandatory with the NCS2 provider. The SwbemNamedValueSet interface is used to create named value pairs; the client lock ID is paired with the string "ClientSetId". The following code demonstrates:

VBScript

```
Set oValueSet = CreateObject("WbemScripting.SWbemNamedValueSet")
oValueSet.Add "ClientSetId", iClientID
```

This named value set is then passed into functions which require a client ID lock. In this example, it is being used with a request to create a team:

VBScript

```
Set OutParameterObj = IANet_TeamOfAdapters_Def.ExecMethod_ ("CreateTeam",  InParameterObj, 0,
oValueSet)
```

Release Lock

Once configuration change requests are submitted, they need to be applied. The changes are not complete until the IANet_NetService function ApplyDone ( ) has been called. This function also requires users pass in the client lock ID but it uses it differently than the previous example. In this case, the ClientSetHandle parameter of the Apply IN parameters is assigned the number which was obtained from the earlier call to BeginApply ( ).

VBScript

```
Set objInParams = NetServiceObject.Methods_. Item("Apply").InParameters.SpawnInstance_()
objInParams.ClientSetHandle = iClientID
NetServiceObject.ExecMethod_("Apply", objInParams)
```

Troubleshooting

Disabled Drivers

Some device drivers may become disabled until the IANet_NetService.Apply( ) function is called. This is expected; Apply( ) will re-enable the drivers. Typically, this is only seen during teaming changes.

Restarting the Provider

It may be necessary to restart the NCS2 provider if a mistake is made in locking and unlocking the software stack. Normally, the provider will time out in a few minutes, allowing users to resume troubleshooting of scripts and programs. To avoid this timeout, the NCS2 provider can be directly restarted by ending the NCS2Prov.exe process then restarting the WMI Service in the operating system. Use of this option should always be a last resort and unexpected consequences in the software may occur if the provider is shut down during an operation.

Changing Settings

To change an adapter, VLAN or team setting, the client must first get the object path of the setting that it will change. This is best done by enumerating the settings on the object and storing the __PATH attribute of the setting.

How to change a setting:

- Get an instance of the setting to be modified
- Obtain a client software ID lock by calling IANet_NetService.BeginApply( ).
- Create a context qualifier and set the client ID lock value. See the section above on Client Locks.
- Modify the 'CurrentValue' parameter of the setting object to the new value for the setting.
- Call IWbemServices::PutInstance() to pass the modified instance back to the NCS2 WMI provider. PutInstance must be called with the flag WBEM_FLAG_UPDATE_ONLY. The context qualifier must be passed in as well.
- The NCS2 provider will validate CurrentValue and return 0 if the operation succeeded. An instance of IANet_ExtendedStatus will be available if there were any errors.
- Release the client software ID lock by calling IANet_NetService. Apply( )
-

Troubleshooting Setting Changes

Failed setting changes can occur for several reasons. There are certain rules about what values a setting can have, often enforced by other parameters in the class. A few of the reasons for typical failures are:

- The integer setting value was less than the minimum allowed
- The integer setting value was greater than the maximum allowed
- The integer setting value is not one of the allowable steps
- The length of the string setting is bigger than the maximum allowed
- The length of the string is smaller than the minimum allowed
- The setting value is not one of the allowable values
- A string requires special formatting, such as that seen in IP addresses ("x.x.x.x")
- A string may contain invalid characters
- An IP address is invalid

Working with Teams

Adapter teams can be created by utilizing classes and methods in the root\IntelNCS2 namespace: Teams can be created through the IANet_TeamOfAdapters.CreateTeam( ) function. Like all configuration changes, team creation requires a client ID lock to be in place.

Steps to create a team

- Create an instance of the IN parameters for the CreateTeam( ) function. This can be accomplished by obtaining a definition of the IANet_TeamOfAdapters class and calling using IWbemServices::SpawnInstance().
- The IN parameters object will contain 4 fields which need to be filled out:
  - Set .*Adapters* to an array of IANet_PhysicalEthernetAdapter paths; these are the adapters which will be teamed.
  - Set .*TeamingMode* to an integer corresponding to the type of team which will be created.
  - Set .*TeamName* to the name of the team
  - Set .*MFOEnable* to *false*. MFO is manageability failover and only needs to be set *true* with manageability adapters and environments supporting this feature.
- Obtain a client software ID lock by calling IANet_NetService.BeginApply( )
- Create a context qualifier and set the client ID lock value. See the section above on Client Locks.
- Invoke the CreateTeam( ) method of the IANet_TeamOfAdapters class, passing in the context qualifier.
- The NCS2 Provider will validate the candidate adapters and team mode. It will return 0 if the operation succeeded. An instance of IANet_ExtendedStatus will be available if there were any errors.
- Release the client software ID lock by calling IANet_NetService. Apply( )

Troubleshooting Team Creation

Teams have certain rules around team types and membership which are enforced by all team creation requests. Documentation for Intel® PROSet for Windows* Device Manager contains these restrictions.

- The adapter may already be part of another team
- Teaming membership rules have not been followed

Modifying Teams

Teams and their membership can be modified through the WMI interface of the NCS2 provider. There isn't a method available for performing these actions; the user can directly manipulate instances to effect configuration changes. The required steps are outlined below.

Adding an adapter to a team

- Create an instance of IANet_TeamedMemberAdapter (i.e., use IWbemServices::GetObject() to get a class object for IANet_TeamedMemberAdapter, and then use IWbemServices::SpawnInstance() to create an instance of this object).
- The following properties in the object must be set:
  - GroupComponent must be set to be the full object path of the IANet_TeamOfAdapter which the adapter is to be added
  - PartComponent must be set to be the full object path of the IANet_EthernetAdapter that is to be added to the team.
- Set priority of the adapter in the team (optional)
- Obtain a client software ID lock by calling IANet_NetService.BeginApply( )
- Create a context qualifier and set the client ID lock value. See the section above on Client Locks.
- Finally, call IWbemServices::PutInstance() to add the adapter to the team, passing the flag WBEM_FLAG_CREATE_ONLY. If this action fails, check IANet_ExtendedStatus for the error code.
- Release the client software ID lock by calling IANet_NetService. Apply( )

Removing an adapter from a team *(abbreviated steps)*

- Obtain a client software ID lock by calling IANet_NetService.BeginApply( )
- Delete the IANet_TeamedMemberAdapter instance that associates the adapter to the team using IWbemServices::DeleteInstance(). If this action fails, check IANet_ExtendedStatus for the error code.
- Release the client software ID lock by calling IANet_NetService. Apply( )

Deleting a team *(abbreviated steps)*

- Obtain a client software ID lock by calling IANet_NetService.BeginApply( )
- To delete a team, delete the IANet_TeamOfAdapters instance using IWbemServices::DeleteInstance(). If this action fails, check IANet_ExtendedStatus to get the error code.
- Release the client software ID lock by calling IANet_NetService. Apply( )

Changing team type *(abbreviated steps)*

- Manipulate the IANet_TeamOfAdapters.TeamMode parameter. This will require obtaining and releasing a client software ID lock.

Changing adapter priority *(abbreviated steps)*

- Manipulate the IANet_TeamedMemberAdapter.AdapterFunction parameter. This will require obtaining and releasing a client software ID lock.

Working with VLANs

VLANs can be added, removed, and modified through the NCS2 provider. The steps for performing these actions are detailed below. VLANs can be added and removed from teams and adapters.

Steps to create a VLAN

- Create an instance of the IN parameters for the CreateVLAN ( ) function. This can be accomplished by obtaining a definition of the IANet_802dot1QVLANService class and calling using IWbemServices::SpawnInstance(). For this step, it is easier to work with the instance of IANet_802dot1QVLANService that associates to the device to receive the VLAN.

- The IN parameters object will contain two fields which need to be filled out:
  - Set *.VLANNumber* to number of the VLAN. (Range 1- 4094) Untagged VLANs use the number 0.
  - Set *.Name* to user defined name to identify the VLAN.
- Obtain a client software ID lock by calling IANet_NetService.BeginApply( )
- Create a context qualifier and set the client ID lock value. See the section above on Client Locks.
- Call the CreateVLAN method on the IANet_802dot1QVLANService for the device (adapter or team) to which the VLAN is to be added.
- The function will return the object path of the newly created VLAN in the out parameter. If this action fails, check IANet_ExtendedStatus for the error code.
- Release the client software ID lock by calling IANet_NetService. Apply( )

Steps to delete a VLAN *(abbreviated steps)*

- Obtain a client software ID lock by calling IANet_NetService.BeginApply( )
- Call IWbemServices::DeleteInstance passing the object path of the VLAN to delete.
- Release the client software ID lock by calling IANet_NetService. Apply( )

Troubleshooting VLANs

VLANs also have certain rules around how they are created and where they can exist. Fortunately, they are very simple:

- The VLAN ID must be in the range of 1-4094
- When creating an untagged VLAN (ID 0), a tagged VLAN must first exist.
- An untagged VLAN cannot exist without a tagged VLAN on a device. Removing the last tagged VLAN will also remove the untagged VLAN.

Running Diagnostics

Diagnostics can be executed through the NCS2 provider and the results retrieved after execution. The class IANet_DiagTest is used for this purpose and its method, RunTest ( ) facilitates diagnostic requests. Client software ID locks are not required for diagnostic execution – these steps have intentionally been left out of the following instructions.

Steps to run a diagnostic

- Identify an instance of IANet_DiagTest which represents both the diagnostic to be executed and the adapter on which it is to run.
- Create an instance of the IN parameters for the RunTest( ) function.
- The IN parameters must be populated with the following:
  - The *.Setting* receives a reference to an instance of IANet_DiagSetting whose .Setting ID represents the desired <ID>@<GUID> test.
  - The *.SystemElement* receives a reference to an instance of IANet_PhysicalEthernetAdapter which corresponds to the particular diagnostic under test. Assign the object's path to this parameter.
- Execute the RunTest( ) method.
- Examine instances of IANet_DiagResult to find results.

Troubleshooting Diagnostics

Getting the correct object paths for each diagnostic can be complicated. They key piece of information that ties them all together is the {GUID}, a long number that uniquely identifies each device. When working with diagnostics, this number is very important to getting the correct object instances. The following table provides some hints on how this number can be used across classes.

Hint	Value	Use
IANet_DiagTest.Name	1@{1F770A4C-B9DB-4174-9FD3-0BF520C3CB73}	Use the .RunTest( ) of this instance
IANet_PhysicalEthernetAdapter.DeviceID	{1F770A4C-B9DB-4174- 9FD3-0BF520C3CB73}	Assign the .Path of this object to the RunTest( ) IN parameter .SystemElement.
IANet_DiagSetting.SettingID	1@{1F770A4C-B9DB-4174-9FD3-0BF520C3CB73}	Assign the .Path of this object to the RunTest( ) IN parameter .Setting

The association classes for IANet_DiagTest can also help locate the correct instances of the classes above. IANet_DiagTestForMSE associates to an instance of IANet_PhysicalEthernetAdapter and IANet_DiagSettingForTest associates to an instance of IANet_DiagSetting. Using this association can take a lot of the guesswork out of obtaining the correct references.

Getting Diagnostic Results

Diagnostic results can be enumerated any time a diagnostic test has been executed. Since the NCS2 provider does not store results permanently, the results are only available as long as the provider is running. If it has become idle for too long, it will unload and erase any existing results.

Diagnostic Names

The name of a diagnostic result will be contained in the *.DiagnosticName* parameter for a class instance of IANet_DiagResult. It is a concatenation of the diagnostic ID, the "@" symbol, and the unique identifier of the adapter for which it was executed. The IANet_DiagResultForMSE association class can also be used to find all results for a particular adapter. For most instances, the results of the test will be contained in the *.Result* parameter.

Packages

The results of some diagnostic tests may be contained in a *package*. This is a logical grouping of a parent test result and any additional result instances which apply to it. Thus it is possible to have one main result for a test and a one or more results which provide additional results. Packages can be recognized in a few ways:

- Results in a package will have a sub-index in the diagnostics ID for the *DiagnosticName* parameter of the class instance.
- The parent result will have its *IsPackage* parameter set to "Yes" and will have a IANet_DiagResultInPackage association to any packaged results.

Type	*DiagnosticName*	*IsPackage*
Parent Result	**37**@{66DFCDDC-A535-4265-99CB-21B03561497A}	**Yes**
Packaged Result	**37.1**@{66DFCDDC-A535-4265-99CB-21B03561497A}	
Packaged Result	**37.2**@{66DFCDDC-A535-4265-99CB-21B03561497A}	
Packaged Result	**37.3**@{66DFCDDC-A535-4265-99CB-21B03561497A}	
Packaged Result	**37.4**@{66DFCDDC-A535-4265-99CB-21B03561497A}	
Packaged Result	**37.5**@{66DFCDDC-A535-4265-99CB-21B03561497A}	

iSCSI Settings

The iSCSI settings of an Intel network card can be manipulated through the WMI interface. Only operating systems which support iSCSI will have these settings available. Working with iSCSI configurations requires additional information which is not readily available in the class descriptions.

Getting iSCSI Status

To retrieve the iSCSI status of a network card, enumerate instances of the IANet_BootAgent_iSCSI_Adapters class. In this class, look at the iSCSI_Status parameter. This will indicate the current state of an iSCSI enabled adapter.

Setting iSCSI Status

The iSCSI state of the adapter is manipulated through the SetiSCSI_Status method of the IANet_BootAgent_iSCSI_Adapters class. This only controls whether the adapter can be set to a Primary, Secondary or Disabled state.

Manipulating iSCSI Parameters

There are several settings applicable to iSCSI which can be manipulated through WMI. To locate these parameters, use the table below to find the name of the class, the type of parameter, and guidelines for setting them. Each iSCSI enabled adapter will have its own settings.

Setting Caption	Class	Notes
Authentication	IANet_BootAgentSettingEnum	0 (Disable CHAP) or 1 (Enable CHAP)
BootLUN	IANet_BootAgentSettingInt	The iSCSI Boot LUN
ChapPassword	IANet_BootAgentSettingString	A string value no longer than 16 characters
ChapUserName	IANet_BootAgentSettingString	A string value no longer than 16 characters
CrashDump	IANet_BootAgentSettingEnum	0 (Disable) or 1 (Enable)
InitiatorDHCP	IANet_BootAgentSettingEnum	This requires an string formatted as an IP address
InitiatorGateway	IANet_BootAgentSettingString	This requires an string formatted as an IP address
InitiatorIPAddress	IANet_BootAgentSettingString	This requires an string formatted as an IP address
InitiatorName	IANet_BootAgentSettingString	A string value no longer than 255 characters
InitiatorSubnetMask	IANet_BootAgentSettingString	This requires an string formatted as an IP address
TargetDHCP	IANet_BootAgentSettingEnum	This requires an string formatted as an IP address
TargetIPAddress	IANet_BootAgentSettingString	This requires an string formatted as an IP address
TargetName	IANet_BootAgentSettingString	A string value no longer than 255 characters
TargetPort	IANet_BootAgentSettingInt	An integer between 0 and 65535
TargetSecret	IANet_BootAgentSettingString	A string between 12 – 255 characters.

Some parameters which contain security information will be intentionally obscured upon reading their value. Examples are passwords where the user will only see "*" characters for values.

Errata

The following is information which may be relevant to viewing or changing a configuration in Intel® PROSet for Windows* Device Manager. It is supplied as a supplement and, in some cases, troubleshooting guide.

Phantom Adapters

When enumerating instances of IANet_PhysicalEthernetAdapter, all installed adapters will be returned whether or not the hardware is actually present. This only occurs when an adapter is installed and then physically removed or replaced by another adapter without first removing its driver. Queries of IANet_PhysicalEthernetAdapter can be modified to filter out these 'phantom' instances.

Amended query:

Select * from IANet_PhysicalEthernetAdapter WHERE StatusInfo = '3'

Permissions

Interaction with the NCS2 provider requires Administrator rights on the operating system. This applies to local and remote access. Windows* Vista may require elevated Administrator rights. These can be obtained by logging in as the Administrator and elevating permissions. Failure to work in an elevated environment may result in failure to obtain and apply client software locks. Intel® PROSet for Windows* Device Manager automatically elevates permissions when it runs in user interface mode.

Diagnostic Results Timeout

The NCS2 provider will automatically terminate within a few minutes of not being used. When this event occurs, all diagnostic results will be lost.

Remote Desktop Limitations

When connecting to a computer remotely with Remote Desktop Protocol and no Administrator locally logged in, use the "/console" option. The other workaround is to make sure a local Administrator account is logged in when initiating remote desktop access. This makes sure the WMI layer can authenticate with the local permissions.

Win32_Product

The Uninstall () method of the Win32_Product class cannot be used to uninstall Intel® PROSet for Windows* Device Manager. Attempting this operation will result in a corrupted installation and, possibly, the need for manual software removal.

Function Return Values

Functions exposed in classes supported by the NCS2 provider will return the value 0 to indicate success. If the function call failed, nothing will be returned. A return value of 0 will only indicate the function was successfully called. It does not validate the final configuration requested by the user. In most cases, a successful function call can be trusted to perform the expected changes. However, it may be necessary to verify those changes before performing further operations.

Passing Client ID Locks in C#

When using client ID locks in C#, the lock number must be cast to type *int* before it is used. This is due to a mismatch between how numbers are treated as data types between WMI, C#, and the NCS2 provider. Attempting to pass the ID without casting it first will result in a failed change request.

C#

```
outParams = netService.InvokeMethod("BeginApply", null, null);
result = (uint)outParams["ReturnValue"];
clientHandle = (uint)outParams["ClientSetHandle"];
g_SValueSet.Add("ClientSetId", (int)clientHandle);
```