Intel® PRO Network Adapters WMI and CDM Providers User Guide

The information below is provided by the supplier of the referenced device without independent verification by Dell and is subject to the restrictions and disclaimers noted below.

Restrictions and Disclaimers

The information contained in this document, including all instructions, cautions, and regulatory approvals and certifications, is provided by the supplier and has not been independently verified or tested by Dell. Dell cannot be responsible for damage caused as a result of either following or failing to follow these instructions.

All statements or claims regarding the properties, capabilities, speeds or qualifications of the part referenced in this document are made by the supplier and not by Dell. Dell specifically disclaims knowledge of the accuracy, completeness or substantiation for any such statements. All questions or comments relating to such statements or claims should be directed to the supplier.

Last revised: 17 November 2008

Initial release: October 2003

Introduction: Intel® PRO Network Adapters WMI and CDM Providers User Guide

Overview

Welcome to the *Intel® PRO Network Adapters WMI and CDM Providers User's Guide*. This document describes the external view of the Intel PRO Network Adapters WMI and CDM Providers. The Windows Management Interface (WMI) Provider is a network configuration block of Network Configuration Services (NCS), which is a means of deploying and managing all Intel end-station networking technologies using industry standard methods. The Intel PRO Common Diagnostic Model (CDM) Provider is an upper interface API that is compliant with the CIM 2.5 and WMI standards. On the lower interface, the CDM Provider implements a client interface into lower layers of the PROSet software stack. Because of this, all PROSet mechanisms for data integrity are maintained.

The WMI and CDM Providers are sets of software components that implements the Intel WMI network classes. These classes are based on the Desktop Management Task Force (DMTF) CIM Schema version 2.5.

This document does not repeat information contained in the Managed Object Format (MOF) files provided with this product (e.g., details of the meanings of individual attributes can be found in the MOF attribute descriptions).

This document describes how a WMI application such as Intel PROSet uses classes to configure a system's network and how a how a WMI application uses classes to test an Intel network interface card. Readers should be familiar with WMI APIs and the WMI SDK (available from http://www.microsoft.com/).

Back to Top

Related Documents

The following documents can be used to better understand WMI technology:

- CIM schema version 2.0, 2.2 published by Desktop Management Task Force (DMTF). Available at http://www.dmtf.org.
- Microsoft Windows Management Interface (and other manageability information). Available at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/wmi/wmi_start_page.asp.
- Web-Based Enterprise Management (WBEM) initiative by DMTF. Available at http://www.dmtf.org/standards/wbem.
- WMI (Microsoft CIM implementation) SDK. Available at http://msdn.microsoft.com/downloads/.
- System Diagnostic Model White Paper by DFTM. Available at http://www.dmtf.org/standards/documents/CIM/DSP0138.pdf.

WARNING: This product contains information that could be used to attack and/or disable a computer system(s) or network(s). A thorough knowledge of Microsoft operating system security features should be a prerequisite to any implementation of this product, and developers and users are strongly encouraged to contact Microsoft regarding any security concerns that they may have prior to using any implementation of this product in a production environment.

Back to Contents Page

WMI: Intel® PRO Network Adapters WMI and CDM Providers User Guide

Overview

Web-Based Enterprise Management (WBEM) is a Desktop Management Task Force (DMTF) initiative intended to provide 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 for representing objects that exist in typical management environments, and the Managed Object Format (MOF) language for defining and storing modeled data.

Windows Management Instrumentation (WMI) is an implementation of the WBEM initiative for Microsoft* Windows* platforms.

WMI consists of three main components:

- Core — These components are part of the Operating System. They are required for a WMI-enabled application to work, and must be installed in order to use the SDK.
- SDK — The SDK contains tools to browse the WMI schema, extend the schema, create providers, register and use the WMI events. It also provides documentation useful for developing applications that will use the WMI. The SDK is installed as part of the Microsoft Platform SDK installation process and is supported on Windows NT4 SP4 or SP5, Windows 2000, Windows Me, Windows XP and Windows Server 2003.
- Tools — The Microsoft WMI Tools provide developers with the tools needed to build a whole new generation of management applications and solutions. It is filled with documents and tools to guide you through the process of accessing management data from WMI.

The WMI architecture consists of the following components:

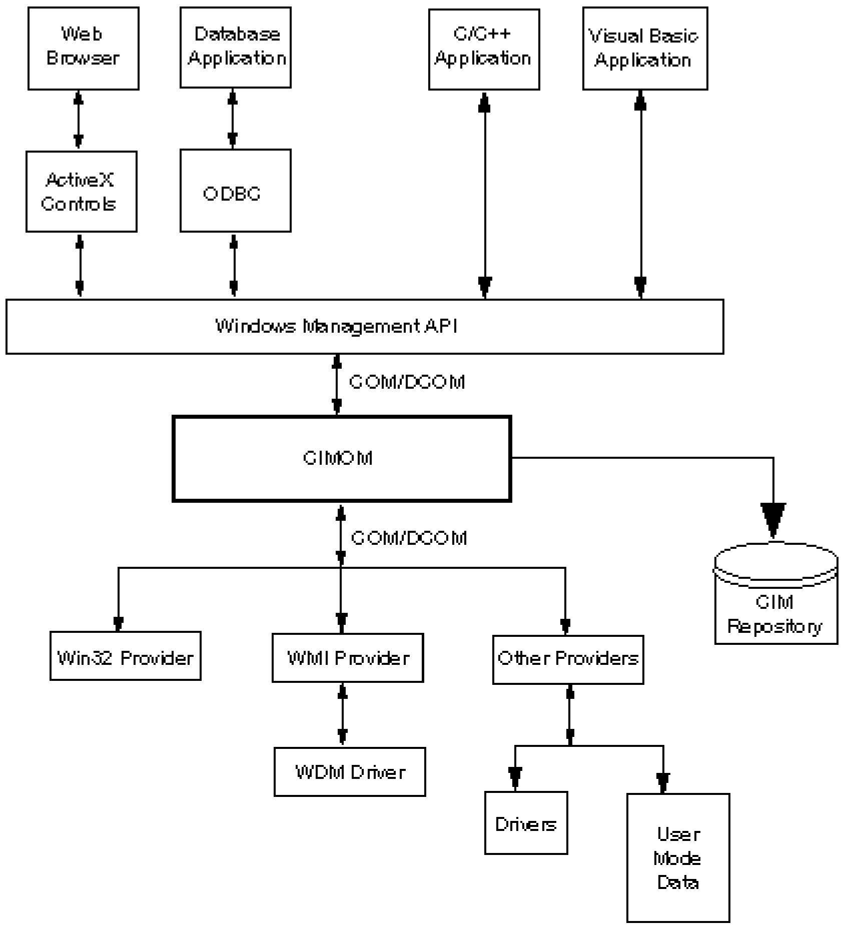
- Management applications
- Managed objects
- Providers
- Management infrastructure (consisting of the Windows Management and Windows Management repository)
- Windows Management API (which uses COM/DCOM to enable providers and management applications to communicate with the Windows Management infrastructure.

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. Providers use the Windows Management API to supply Windows Management with data from managed objects, to handle requests from applications and to generate notification of events.

The management infrastructure consists of Windows Management (for handling the communication between management applications and providers) and the Windows Management repository (for storing data). The Windows Management repository holds static management data. Dynamic data is generated only on request from the providers. Data is placed in the repository using either the MOF language compiler or the Windows Management API.

Applications and providers communicate through Windows Management using the Windows Management API, which supplies services such as event notification and query processing.

The following diagram shows the interrelationship of the WMI architecture components:



Back to Top

Common Information Model (CIM Schema)

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 using object-oriented constructs such as classes. The classes include properties that describe data and methods that describe behavior. The CIM is designed by the DMTF to be operating system and platform independent. The WBEM technology includes an extension of the CIM for the Microsoft Windows operating system platforms. Please refer to the DMTF CIM schema on DMTF web site for more information.

The 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.
- 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, referred to as the extended model.

All classes can be related by inheritance, where a child class includes data and methods from its parent class. 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 obtained using applications that are included in the WMI Tools (see the WMI Tools at http://www.microsoft.com for more information).

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

WBEM supports the concept of schemas to group the classes and instances that are used within a particular management environment. The Platform SDK includes two schemas: the CIM schema and the Microsoft Win32 schema. The CIM schema contains the class definitions for the first two levels of the CIM. These classes represent managed objects that are part of every management environment regardless of platform. The Win32 schema contains class definitions for managed objects that are part of a typical Win32 environment.

For additional information on CIM, visit http://www.dmtf.org.

Back to Contents Page   Back to Top

Back to Contents Page

Main Features: Intel® PRO Network Adapters WMI and CDM Providers User Guide

NCS WMI Provider Features

The main features of the WMI Provider are as follows:

Adapter Features

- Enumerate all physical adapters supported by Intel® PROSet.
- Enumerate an installed adapter's settings.
- Add/remove/update settings for an installed adapter.
- Obtain an adapter's physical device information.
- Obtain an adapter's system slot device information.
- Obtain the IPv4 protocol settings for an adapter.
- Update and change an adapter's boot agent and associated settings.
- Uninstall an adapter.

Team Features

- Enumerate the teams supported by Intel PROSet.
- Create/remove a team of adapters.
- Add/remove/update settings of the team.
- Add/remove member adapters for a team.
- Obtain the IPv4 protocol settings for a team.

VLAN Features

- Enumerate virtual LANs on an adapter or team.
- Create/remove virtual LANs on a physical adapter or a team of adapters.
- Add/remove/update settings of the VLAN.
- Obtain the IPv4 protocol settings for a team.

Event Notification Features

- Allow the client to register.
    - Adapter status events.
    - Adapter configuration events.
    - Session events.
    - Team status events.
    - Team configuration events.
    - VLAN configuration events.

Back to Top

CDM Provider Features

The main features of the CDM Provider are as follows:

- Run tests, stop tests and clear test results without dependency on the type of diagnostic test.
- Use of generic setting class should allow control of tests in ways unanticipated by the CDM software itself.
- CDM Provider is only used for adapters.
- Use of generic result class unbinds specific results messages from the CDM provider code.
- Registry entries controls provider execution.
- Test results written to results log file.

Back to Contents Page Back to Top

Installed Files: Intel® PRO Network Adapters WMI and CDM Providers User Guide

WMI Files

Executables

The WMI Provider executables are as follows:

- **NcsWmiCo.exe** — The core provider. Implements the IANet_NetService and the core event classes.
- **NcsWmiIm.exe** — The instance and method provider. Implements the Ethernet Adapter Schema, the Teaming Schema, the Setting Schema and the VLAN Schema.
- **NcsWmiEv.exe** — The event provider. Implements the adapter, team and VLAN events.

MOF Files

There are separate MOF files for language-neutral and language-specific data. Also, there are separate MOF files for IntelNCS and CIMV2 namespace. See *Locales and Localization* and *Error Reporting* for more details.

The MOF files for IntelNCS namespace are as follows:

- **NcsCmLn.mof** — CIM base classes that the NCS classes depend on.
- **NcsCmEnu.mfl** — US English version of the CIM base classes.
- **NcsCoLn.mof** — Core classes implemented by the core provider.
- **NcsCoEnu.mfl** — US English textual amendments to the core classes.
- **NcsIaLn.mof** — Classes for the IEEE 802.3 adapters, teams and VLANs.
- **NcsIaEnu.mfl** — US English textual amendments to the 802.3 core classes.

The MOF files for CIMV2 namespace are as follows:

- **C2CmLn.mof** — CIM base classes that the NCS classes depend on.
- **C2CmEnu.mfl** — US English version of the CIM base classes.
- **C2CoLn.mof** — Core classes implemented by the core provider.
- **C2CoEnu.mfl** — US English textual amendments to the core classes.
- **C2IaLn.mof** — Classes for the IEEE 802.3 adapters, teams and VLANs.
- **C2IaEnu.mfl** — US English textual amendments to the 802.3 core classes.

Resource Files

The resource files for the WMI Provider are as follows:

- **ENU_8023.dll** — English USA 8023 resource.
- **ENU_NWRC.dll** — English USA WMI resources for core providers.
- **ENU_NWR.dll** — English USA WMI resources for 8023 providers.

The other localized resource files can be loaded on demand. The general pattern for the names of localized resource DLLs is "`_mwr.dll`" which is the language code for the localization (e.g. FRA for standard French).

Back to Top

CDM Provider Files

Executables

The executables for the CDM Provider are as follows:

- **Ncsdiag.exe** is the main executable file for the CDM diagnostics. It conforms to the Microsoft* WMI interface specification and is accessed as an out-of-process COM server.
- Other executables from the Intel® PROSet software stack

MOF Files

The master **.mof** files are not shipped with the product, but are compiled into respective language-dependent and language-neutral components, according to the Microsoft* Windows* Management Instrumentation globalization model. For further information, refer to the Microsoft* WMI SDK (component of Platform SDK) chapter on WMI localization. Pay specific attention to the section *Compiling Localized MOF Files*.

Deleting a **.mof** file (DNcsCdmN.mof) deletes the Intel-derived class definitions, but not the DMTF-defined classes, as this would endanger other existing applications.

Typical usage of this CDM implementation is based on the CIMV2 namespace. The MOF files for an IntelNCS namespace are as follows:

File Name	**Language Type**	**Description**
CdIa.mof	Master	Class definitions for Intel CDM implementation
CdIaLn.mof	Language-neutral	Class definitions for Intel CDM implementation
CdIaEnu.mfl	English language dependent	Language extensions for Intel CDM implementation
CdCm.mof	Master	Core superset CDM class definitions
CdCmLn.mof	Language-neutral	Core superset CDM class definitions
CdCmEnu.mfl	English language-dependent	Language extensions for core superset CDM class definitions
DNcsCdmN.mof	Does not apply	Deletes Intel CDM classes

The MOF Files for a CIMV2 namespace are as follows:

File Name	**Language Type**	**Description**
C2Icd.mof	Master	Class definitions for Intel CDM implementation
C2IcdLn.mof	Language-neutral	Class definitions for Intel CDM implementation
C2IcdEnu.mfl	English language-dependent	Language extensions for Intel CDM implementation
C2Cd.mof	Master	Core superset CDM class definitions
C2CdLn.mof	Language-neutral	Core superset CDM class definitions
C2CdEnu.mfl	English language-dependent	Language extensions for core superset CDM class definitions
DNcsCdm2.mof	Does not apply	Deletes Intel CDM classes

Note: Localization requires addition of the correct language-dependent **.mof** file.

Resource Files

The resource file for the CDM Provider is as follows:

- **ENU_Diag.dll** - English USA WMI Resources for Diagnostic Provider

---

Please read all restrictions and disclaimers.

Back to Contents Page Back to Top

Back to Contents Page

Security: Intel® PRO Network Adapters WMI and CDM Providers User Guide

The WMI and CDM Providers use client impersonation to manage security. Every call into the Providers will be made in the client's own security context, which is passed down to the lower layers. One or all operations may fail if you do not have administrative rights on the target machine.

Please read all restrictions and disclaimers.

Back to Contents Page   Back to Top

Back to Contents Page

Namespace and Context: Intel® PRO Network Adapters WMI and CDM Providers User Guide

The CIM classes reside in a namespace. The standard Microsoft* namespace is called **root/cimv2** and is based on CIM v2.2 or **root/default**. The WMI and CDM Provider classes can be added to this namespace. The Providers are based on CIM v2.5. Because of this and differences used in the keys of the objects, the Providers' classes are located in a separate namespace, **root/IntelNCS**.

WBEM Context

Context objects provide additional information to the Providers that cannot be passed as a parameter to a WMI API method. To register context qualifiers, use the **IWbemContext** to register context qualifiers. The interface pointer for the context object is passed as the last parameter of an **IWbemServices** method.

The following table contains the context qualifiers (named values) used by the Providers. Most qualifiers, like SessionHandle, are only used in conjunction with specific functional areas of the Providers, whereas LocaleID, MachineName and ApplicationName can be set for all **IWbemServices** calls.

If no context is passed to the Providers, they will use the LocaleID passed in the **Initialize** call to the Providers. Any read done with a context will read the current configuration until a write operation is performed. Subsequent reads will show the system as it would be after the write has succeeded. A NULL context can be used for reads.

Context Qualifier	**Variant Type**	**Description**
SessionHandle	VT_BSTR	Identifies the application's copy of IANet network classes. The application cannot make any changes to the classes or their attributes without first establishing a session handle. See the section on the IANet_NetService class to see how to establish and use a session handle. This qualifier is not required if the application is only going to read data from the classes. The session handle allows the NCS software to manage simultaneous multiple accesses to the configuration without one user locking out all others. Each session has a separate cache to store any changes that have been made. If there are multiple users making changes simultaneously then the first user to apply their changes will succeed. All the other users' caches will be invalidated.
LocaleID	VT_BSTR	Microsoft's ID for a locale. This is required if the application requires localized text strings from the Providers. All error messages and warnings will be in English, unless the required LocaleID is used.
ApplicationName	VT_BSTR	The name of the application that made the call. This is required for logging.
MachineName	VT_BSTR	The name of the machine that is connecting to the Providers. This is required for logging.
PreCheck	VT_BOOL	This Boolean value is used to tell the Providers that the client is attempting to verify that an operation is allowed before actually performing the operation. For example, adding an adapter to a team. Values: • TRUE = Provider will not perform the operation, but will return an error code and extended status if the operation is not allowed. • FALSE = Provider will perform the operation. If this qualifier is missing, it has the same effect as if the attribute was FALSE.
WarningErrorCode	VT_I4	Some operations may require warnings to be sent to the user (e.g., adding an adapter to the team may require the team to be reloaded in some circumstances). WMI does not provide a mechanism for this. If this qualifier is present and non-zero, the Provider will return E_FAIL if the operation succeeded, but there was an associated warning. The client should use the extended status to get the text of the warning.

Back to Contents Page Back to Top

Back to Contents Page

Locales and Localization: Intel® PRO Network Adapters WMI and CDM Providers User Guide

There are two aspects to WMI and CDM Provider localization — localized MOF files and localized attribute data.

Localized MOF Files

All the MOF files used by the Providers are localized according to the Microsoft Windows* Management Instrumentation (WMI) globalization model. To accomplish this, each class definition is separated as follows:

- 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.

Supported Languages

Chinese (Taiwan)
Chinese (PRC)
Danish
Dutch (Netherlands)
English (United States)
Finnish
French (France)
German (Germany)
Italian (Italy)
Japanese
Norwegian (Bokmal)
Portuguese (Brazil)
Spanish (Spain - Modern)
Swedish

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 WMI and CDM Provider, a child namespace **ms_409** will exist beneath the **root/Intelncs** namespace for the English locale. Similarly, there exists a child sub-namespace for each supported language beneath the **root/Intelncs** namespace.

Localized MOF Support in cimv2 Namespace

For **root/cimv2** namespaces, the Providers' classes (i.e., IANet_ classes) are derived from base classes added to this namespace by WMI. A sub-namespace with language-specific class definitions for base classes pre-exist beneath the **root/cimv2** namespace. The IA_Net language-specific class definitions will be added to this existing child namespace. Due to this dependency on base classes, MOF localization is done only in the default system locale.

Runtime Support

To retrieve localized data, a WMI application can specify the locale using a 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_USE_AMENDED_QUALIFIERS flag to request localized data along with the basic definition. This is required in all functions that produce displayable values using value maps, display descriptions or other amended qualifiers from the MOF files.

Back to Top

Localized Attribute Data

To get localized attribute data (such as error messages), the Providers need to know the locale of the caller for every call. For this to work correctly, the client must add the locale to the context object that is passed for every call (see *Namespace and Context* on WBEM context). If the Providers need to return a localizable string, then it will attempt to load a resource DLL that is appropriate for the client's locale. If there is no appropriate resource DLL, then the Providers will return strings in US English.

Back to Contents Page   Back to Top

Back to Contents Page

Error Reporting: Intel® PRO Network Adapters WMI and CDM Providers User Guide

Overview

This section about IANet_ExtendedStatus describes how to handle errors generated by WMI and CDM Providers. 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 the Providers generated the error.

To get the error object for synchronous calls, use GetErrorInfo() to get the IErrorInfo object. Use QueryInterface() to get the IWbemClassObject that contains the error information.

To get the error object for 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 Providers create an instance of IANet_ExtendedStatus for errors relating to IANet_ classes. IANet_ExtendedStatus is derived from __ExtendedStatus and contains the following error object qualifiers:

- Description — Description of the error tailored to the current locale.
- File — Code file where the error was generated.
- Line — Line number of the code file with the error.
- ParameterInfo — Class or attribute that was being utilized when the error occurred.
- Operation — Operation being attempted when the error occurred.
- ProviderName — Name of the Provider that caused the error.
- StatusCode — Code returned from the internal call that failed.
- SessionHandle — Session handle used for the operation.
- RuleFailureReasons — 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).

Back to Top

Error Codes

For all error codes, the Providers give 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 the following codes as a basis for a recovery action:

- 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: Providers only support 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: Qualifer 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 NCS Service has been stopped."

- 0x80040801 — "EAL: Internal exception"
- 0x80040802 — "EAL: General failure"
- 0x80040803 — "EAL: Not initialized"
- 0x80040804 — "EAL: Failed to initialize."
- 0x80040805 — "EAL: Session limits exceeded"
- 0x80040806 — "EAL: Out of memory"
- 0x80040807 — "EAL: Rule syntax error"
- 0x80040808 — "EAL: Unexpected end of list"
- 0x80040809 — "EAL: Rule link error"
- 0x8004080A — "EAL: Device Creation Failed"
- 0x8004080B — "EAL: Media service not found"
- 0x8004080C — "EAL: Device service not found"
- 0x8004080D — "EAL: PCI bus module not found"
- 0x8004080E — "EAL: Adapter is a member of a team"
- 0x8004080F — "EAL: Rule Access Point creation error"
- 0x80040810 — "EAL: Registry key error"
- 0x80040811 — "EAL: Registry XML file path error"
- 0x80040812 — "EAL: Unknown event class"
- 0x80040813 — "EAL: Unknown module id"
- 0x80040814 — "EAL: Rule service not found"
- 0x80040815 — "EAL: NULL input pointer"
- 0x80040816 — "EAL: Rule grammar error"
- 0x80040817 — "EAL: Rule failed"
- 0x80040818 — "EAL: Setting is already grouped"

- 0x80040220 — "Sync Layer: Team removal failed."
- 0x80040221 — "Sync Layer: Vlan creation failed."
- 0x80040222 — "Sync Layer: Vlan removal failed."
- 0x80040223 — "Sync Layer: Adapter removal failed."
- 0x80040224 — "Sync Layer: Setting Change/Creation/Removal failed."
- 0x80040225 — "Sync Layer: Parameter Change/Removal failed."
- 0x80040226 — "Sync Layer: NetConfig subsystem locked. "
- 0x80040227 — "Sync Layer: System Update In Progress. Please try again later."
- 0x80040228 — "Sync Layer: Adapter is Locked"
- 0x80040229 — "Sync Layer: Flash read failed."
- 0x8004022A — "Sync Layer:"

- 0x80040210 — "Sync Layer: Invalid event offset"
- 0x80040211 — "Sync Layer: Invalid input"
- 0x80040212 — "Sync Layer: Invalid key"
- 0x80040213 — "Sync Layer: Adapter not team member"
- 0x80040214 — "Sync Layer: Driver not loaded"
- 0x80040215 — "Sync Layer: Client impersonation failed"
- 0x80040216 — "Sync Layer: Caught exception"
- 0x80040217 — "Sync Layer: Session not locked"
- 0x80040218 — "Sync Layer: Hardware access layer is not available"
- 0x80040219 — "Sync Layer: Flash not available"
- 0x8004021A — "Sync Layer: Diagnostics not supported"
- 0x8004021B — "Sync Layer: Diagnostic test not running"
- 0x8004021C — "Sync Layer: Boot Agent update not available"
- 0x8004021D — "Sync Layer: Boot Agent corrupted."
- 0x8004021E — "Sync Layer: Flash write failed."
- 0x8004021F — "Sync Layer: Team creation failed."
- 0x80040201 — "Sync Layer: Initialization failed"
- 0x80040202 — "Sync Layer: Invalid initialization handle"
- 0x80040203 — "Sync Layer: Session handle already exists"
- 0x80040204 — "Sync Layer: Invalid session handle"
- 0x80040205 — "Sync Layer: The maximum number of sessions has been reached."

- 0x80040206 — "Sync Layer: The session lock handle already exists"
- 0x80040207 — "Sync Layer: Invalid session lock handle"
- 0x80040208 — "Sync Layer: Session already locked"
- 0x80040209 — "Sync Layer: Invalid media service module Id"
- 0x8004020A — "Sync Layer: Invalid Advanced Service Module Id"
- 0x8004020B — "Sync Layer: Invalid device service module Id"
- 0x8004020C — "Sync Layer: Invalid component type Id"
- 0x8004020D — "Sync Layer: Invalid bus interface module Id"
- 0x8004020E — "Sync Layer: Invalid sink window handle"
- 0x8004020F — "Sync Layer: Invalid event Id"

- 0x80040401 — "HAM PCI: Invalid memory map address"
- 0x80040402 — "HAM PCI: Configuration driver failed to load"
- 0x80040403 — "HAM PCI: Configuration driver version mismatch"
- 0x80040404 — "HAM PCI: Device slot not found"
- 0x80040405 — "HAM PCI: Diagnostic driver failed to load"
- 0x80040406 — "HAM PCI: Diagnostic driver version mismatch"
- 0x80040407 — "HAM PCI: Diagnostic driver initialization failed"
- 0x80040408 — "HAM PCI: Diagnostics not initialized"
- 0x80040409 — "HAM PCI: Diagnostics already initialized"
- 0x8004040A — "HAM PCI: Diagnostic test already running"
- 0x8004040B — "HAM PCI: Diagnostic test not running"
- 0x8004040C — "HAM PCI: Diagnostic test terminated"
- 0x8004040D — "HAM PCI: Diagnostic Invalid test number"
- 0x8004040E — "HAM PCI: Diagnostic hardware missing"
- 0x8004040F — "HAM PCI: Diagnostic send receive initialization failed"

- 0x80040511 — "Media Service: NDIS IO call failed"
- 0x80040512 — "Media Service: Miniport not loaded"
- 0x8004051B — "Media Service: Invalid device handle"
- 0x8004051C — "Media Service: Invalid adapter handle"
- 0x8004051D — "Media Service: Invalid team handle"
- 0x8004051E — "Media Service: Invalid VLAN handle"
- 0x8004051F — "Media Service: Device missing"
- 0x80040520 — "Media Service: Invalid setting type"
- 0x80040521 — "Media Service: Unknown invalid object"
- 0x80040522 — "Media Service: Invalid Setting Handle"
- 0x80040523 — "Media Service: Invalid Team Mode"
- 0x80040525 — "Media Service: Setting Already Exists"

- 0x80042001 — "RAP: Already initialized"
- 0x80042002 — "RAP: Invalid XML file"
- 0x80042003 — "RAP: XML load error"
- 0x80042004 — "RAP: Not initialized"
- 0x80042005 — "RAP: Rule not extracted before"
- 0x80042006 — "RAP: Conditions count mismatch"
- 0x80042007 — "RAP: Results apply error"
- 0x80042008 — "RAP: Invalid rule"
- 0x80042009 — "RAP: Node not found"
- 0x8004200A — "RAP: Error no single node"
- 0x8004200B — "RAP: No action rule"
- 0x8004200C — "RAP: Zero condition"
- 0x8004200D — "RAP: Zero action"
- 0x8004200E — "RAP: XML Decode error"

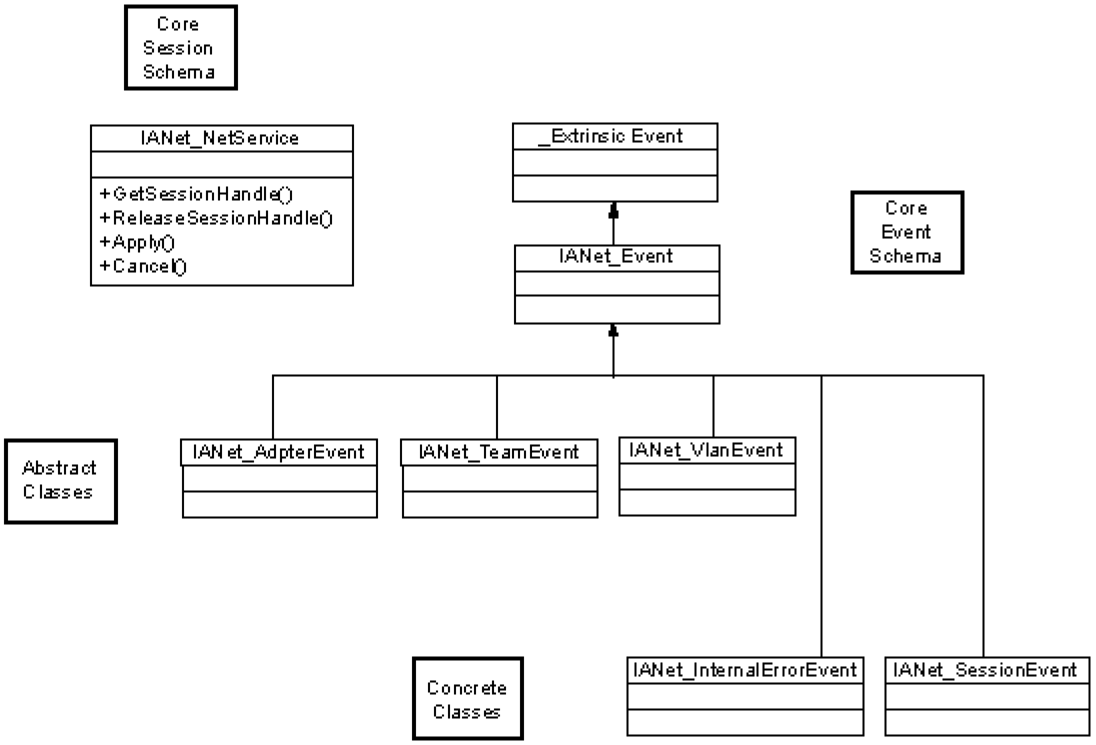
Back to Contents Page   Back to Top

Back to Contents Page

The Core Schema: Intel® PRO Network Adapters WMI and CDM Providers User Guide

Overview

The Core Schema consists of the IANet_NetService class and the core event classes.



Back to Top

IANet_NetService

Purpose
The IANet_NetService class is the root object from the IANet_ schema. This class enables the client to access the sessions that are required to perform sets.

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.

Creating Instances
You cannot create instances of IANet_NetService.

Removing Instances
You cannot delete the instance of IANet_NetService.

Modifying Properties

There are no user-modifiable properties of this class.

Supported Attributes
This class implements two attributes:

- Version — Contains the current version of the core provider.
- InstallDate — Contains the date that the providers were installed.

Methods
The following methods can be used to manage sessions:

- **void GetSessionHandle(([OUT] string SessionHandle, [out] uint32 ActiveSessions)** — Used to set a session handle string, which should be placed in the context object in the SessionHandle qualifier. ActiveSessions returns the number of active sessions for this system. This allows clients to warn that others may be modifying the network settings.
- **void Apply([IN] string sSessionHandle, [OUT] uint32 FollowupAction);** — Applies changes made with a particular session handle. The uint32 argument returned is used by the WMI and CDM Providers to tell the application the server must be rebooted before the changes will take effect. This can be accomplished by calling the **Reboot** method on the class Win32_OperatingSystem.

  Values:
  1 = System reboot required
  0 = No reboot required

- **void ReleaseSessionHandle ([IN] string SessionHandle)** — Releases a session handle after it has been used. Any changes done with this session will be lost. The session handle will no longer be valid after this call and can no longer be used.
- **void Cancel([IN] string SessionHandle);** — Cancels a session. The internal cache will be cleared and any data read following this call will show the current configuration.

Back to Top

Core Events

IANet_SessionEvent

Purpose
This event is used to notify the client about the use of the NCS session API. Clients can use this event to be informed if other clients are creating or using sessions.

Triggers
This event is triggered when a client creates a session, deletes a session, or calls **Apply** for a session.

Event Data
The **EventType** can have one of the following values:

- "New session" indicates that a new session has been created by the client or another client.
- "End session" indicates that a client has finished with a session. The session may have ended by the client or another client.
- "Cache invalidated" indicates that another client has called **Apply** on a session. All other sessions are invalidated and cache associated with their sessions has been deleted.
- "Configuration changed" indicates that the session's configuration has changed.

The **SessionHandle** contains the session handle that triggered the event.

OpenSessions contains the number of open sessions. This data item is NULL for the "Cache invalidated" and "Configuration changed" events.

IANet_InternalErrorEvent

Purpose
This event is used to notify the client that an internal error has occurred in the event Providers. In some cases, this means that the event provider is not able to report further events.

Triggers
This event will occur:

- After the event provider gets an unknown event from an event source
- After the software that provides the events has been shutdown
- After the event provider gets an event but the event source cannot get further data about the event

Event Data
The **EventType** can be one of the following:

- "Could not get event data". An event occurred, but the event source cannot get further data about the event.
- "Event source has shut down". The data source for the event was shutdown. In this case, the event provider will also be shutdown and no more events will be generated until the source is restarted and new notification queries are made.
- "Unexpected message". The event provider received an unexpected event.

Back to Top

Use Cases

A session handle is required to change the configuration. The session handle allows the NCS software to manage simultaneous multiple access to the configuration, thereby preventing a session from locking out all others. Each session has a separate cache to store any changes that have been made. If there are multiple sessions making changes simultaneously then the first to apply its changes will succeed. All other session caches will be invalidated.

Getting a Session Handle

The client must get the object path of the single instance of IANet_NetService before accessing the session handle. Call **IWbemServices::CreateInstanceEnum** and pass the name of the class: IANet_NetService. This is equivalent to calling **IWbemServices::ExecQuery** with the query **SELECT * FROM IANet_NetService**. Before making any changes to the configuration, the client must get a session handle. Use the **GetSesssionHandle** method to start a fresh session.

The client can use **IWbemServices::ExecMethod** to execute a method on a CIM object and will need the object path, from **__PATH** attribute of the instance of IANet_NetService. This method also returns the number of currently active sessions. The client may want to issue a warning if it does not have exclusive access to the Network Configuration Service (NCS).

Using a Session Handle in the IWbemContext Object

After the client obtains a session handle, it must create an IWbemContext object. Store the session handle in the **SessionHandle** qualifier of this object. A pointer to this COM object should be passed to every call into IWbemServices. The session handle is not required when making calls to access the IANet_NetService object as this takes the handle as an explicit argument.

Reading Pending Changes using a Session Handle

When reading the configuration, if you pass the session handle in the context, then the Providers will return the configuration as if the pending updates were applied (e.g., uninstalled adapters will be missing and changed settings will return their new values). However, some objects will not appear until **Apply** has been called (e.g., IANet_IPProtcolEndpoints will not be created until the protocol has been bound to the appropriate miniport).

Finishing with a Session Handle

After changing the configuration, call the **Apply** method to commit the changes. This may return a follow-up action code (e.g., reboot the system before the changes can take effect).

Always call the **ReleaseSessionHandle** after a session is finished, otherwise any changes made will be discarded. Calling the **Cancel** method will also discard any changes made, but the client can continue to use the session handle as if it has just been created.

Registering for the Core Events

Applications should use **IWbemServices::ExecNotificationQuery** or use **IWbemServices:: ExecNotificationQueryAsync** to request event notification. The following queries are examples of event notification queries (this list is not exhaustive as many queries are possible):

- **SELECT * FROM IANet_Event** — Request all events.
- **SELECT * FROM IANet_SessionEvent** — Request all session events.
- **SELECT * FROM IANet_InternalErrorEvent** — Request all internal events.

Back to Contents Page   Back to Top

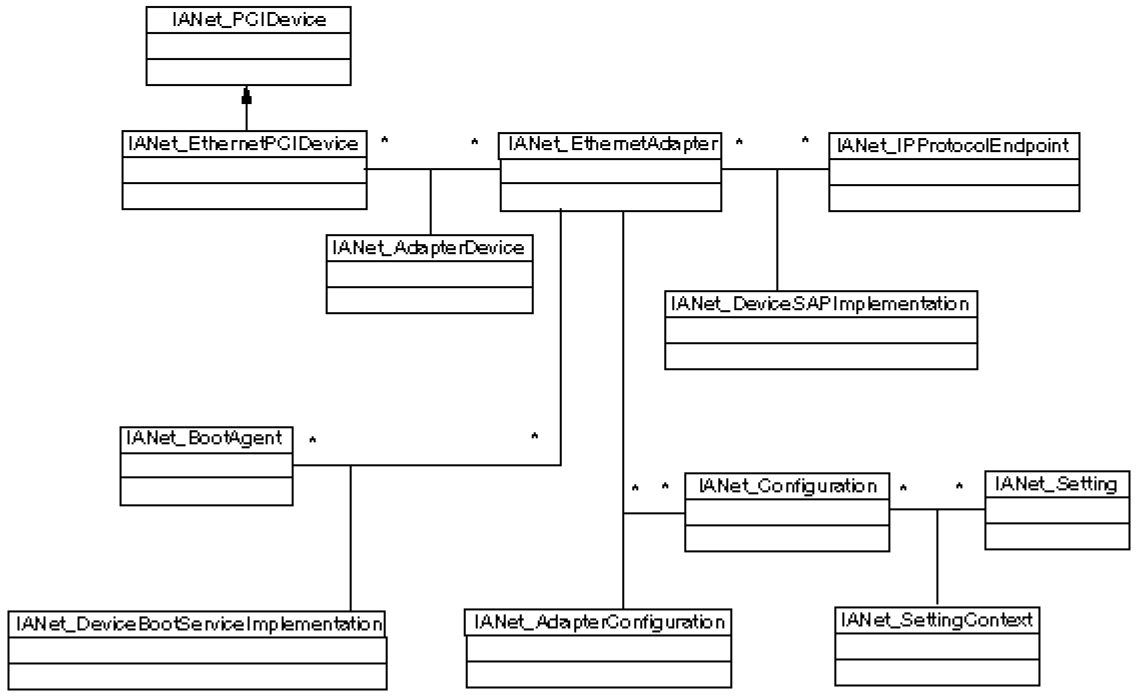
[Back to Contents Page]

The Ethernet Adapter Schema: Intel® PRO Network Adapters WMI and CDM Providers User Guide

---

Overview

The adapter schema is used to model the various configurable Intel® PROSet Ethernet Adapters. This schema is based on the CIM v2.5 schema.



[Back to Top]

IANet_EthernetAdapter

Purpose
IANet_EthernetAdapter defines the capabilities and status of all the installed Intel PRO network adapters as well as any other adapters that can be teamed using Intel intermediate drivers. The class is derived from the CIM_EthernetAdapter superclass defined in CIMv2.5. CIM_EthernetAdapter is derived from CIM_NetworkAdapter, an abstract class defining general networking hardware concepts, such as PermanentAddress, CurrentAddress, Speed of operation, etc.

Instances
Instances of this class will exist for each of the following:

- Supported and installed Intel NICs
- Non-Intel NICs that can participate in an Intel multi-vendor team
- Created Intel adapter team

Creating Instances

You cannot create instances of IANet_EthernetAdapter.

Removing Instances
Deleting an instance of IANet_EthernetAdapter will uninstall physical adapters. Only non-virtual Intel adapters may be uninstalled in this way. A session handle is required for this operation.

Modifying Properties
There are no user-modifiable properties for this class.

Unsupported Attributes
The following attributes are not required for Intel PROSet and are, therefore, not supported:

- AutoSense (this is exposed as a setting)
- ErrorCleared
- OtherIdentifyingInfo
- IdentifyingDescriptions
- InstallDate
- LastErrorCode
- MaxDataSize
- MaxQuiesceTime
- PowerManagementCapabilities (this is exposed as a method)
- PowerManagementSupported (this is exposed as a method)
- PowerOnHours
- ShortFramesReceived
- SymbolErrors
- TotalPowerOnHours

Methods
This class instance supports the following methods:

- **IdentifyAdapter** — Identifies adapter by flashing the light on the adapter for a few seconds. This method will only work for physical adapters.
- **HasVLANs** — Returns number of VLANs on this adapter.
- **IsPowerMgmtSupported** — Indicates whether power management is supported on the adapter.
- **GetPowerUsage** — Detects overall power usage for the adapter.
  0 = Normal Power
  1 = Low power
- **SetPowerUsage** — Reduces overall power usage for the adapter. Power usage setting is not preserved between system restarts or driver reloads. When the system restarts or the driver is reloaded, the adapter automatically returns to normal power consumption.
- **GetPowerUsageOptions** — Detects any optional power usage settings (e.g., power usage for standby, battery operation, etc.).
- **SetPowerUsageOptions** — 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.
- **TestCable** — Performs diagnostic test on a particular adapter. On failure, this method returns possible problems, causes and solutions.
- **AdvancedTestCable** — Performs advanced cable tests on a particular adapter. This test suite is available with 1000 Mbps adapters. The method returns names of tests and their respective results.
  Note: Link failure can occur when **SpeedDuplex** is not set to **Auto Negotiate**. In this instance, **SpeedAndDuplexNotAutomatic** out parameter is TRUE.
- **TestLinkSpeed** — Determines whether adapter is running at full speed. If the adapter is advertising less than 1 Gigabit, then the method states possible reasons (e.g., "Link partner is not capable of running at 1000 Mbps").

Back to Top

IANet_IPProtocolEndpoint

Purpose
This class is used to describe the IP settings for a protocol endpoint in the system. The WMI Provider does not provide information for any other types of networking protocols. The class is derived from the abstract class CIM_IPProtocolEndpoint. The WMI Provider only supplies protocol information when it concerns an entity managed by Intel PROSet.

Instances
An IANet_IPProtocolEndpoint instance will exist for each binding of the IP Protocol stack to an Intel-supported endpoint (i.e., Intel adapters, Intel team-able adapters and VLANs). Some teamed adapters do not have their own IP addresses and so have no IANet_IPProtocolEndpoint directly associated with their adapter instance. The IANet_IPProtocolEndpoint exists only after the operating system has bound the protocol to the adapter or VLAN. Though some adapters may have more than one IP

address, they will be associated with only one IP Protocol endpoint instance. The Provider does not support this advanced use as it is not required or used by Intel PROSet.

Creating Instances
You cannot create instances of IANet_IPProtocolEndpoint. The instance exists only if the operating system has bound the protocol to the endpoint.

Removing Instances
You cannot remove instances of IANet_IPProtocolEndpoint.

Modifying Properties
There are no user-modifiable properties for this class.

Associations
An instance IANet_AdapterProtocolImplementation is used to associate an IANet_EthernetAdapter with an IANet_IPProtocolEndpoint. An instance of IANet_VLANProtocolDependency is used to associate a VLAN with an IANet_IPProtocolEndpoint.

Note: Teams are associated with the endpoint via the adapter that represents the virtual adapter for the team.

Supported Attributes
The following read-only attributes are required by Intel PROSet:

- Address
- AddressType
- DefaultGateway
- DHCPServerAddress
- DHCPAutoAssign
- IPVersionSupport
- SubnetMask

Unsupported Attributes
The following attributes are not required for Intel PROSet and are, therefore, not supported:

- Caption
- Description
- InstallDate
- NameFormat
- OtherTypeInformation
- ProtocolType
- Status

Methods
None.

Back to Top

IANet_BootAgent

Purpose
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). This class is derived from CIM_BootService.

Instances
An IANet_BootAgent instance exists for each adapter that supports boot agent capability, even if the boot agent is not currently installed.

Creating Instances
You cannot create instances of IANet_BootAgent. An instance exists only if the adapter supports boot agent functionality.

Removing Instances
You cannot remove instances of IANet_BootAgent.

Modifying Properties
There are no user-modifiable properties of this class.

Associations
An instance of IANet_DeviceBootServiceImplementation is used to associate an IANet_EthernetAdapter with an IANet_BootAgent, if the adapter supports it.

Supported Attributes
The following read-only attributes are required by Intel PROSet:

- InvalidImageSignature
- Version
- UpdateAvailable
- FlashImageType

Unsupported Attributes
The following attributes are not required by Intel PROSet and are, therefore, not supported:

- Caption
- Description
- InstallDate
- Started
- StartMode
- Status

Methods
The following methods on this class can be used to update the Flash ROM on the NIC:

uint32 ProgramFlash([IN, ValueMap {"0","1"} , Values {"Check Version", "Write Flash"}: Amended] uint32 Action, [IN] uint8 NewFlashData[], [OUT] string strErrorMessage);	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.
uint32 ReadFlash([OUT] uint8 FlashData[]);	This method reads the Flash ROM on the NIC.

Back to Top

IANet_PCIDevice

Purpose
This class is used to describe the properties of a PCI device for a network device in the system. The class is derived from CIM_PCIDevice.

Instances
An instance of this class exists for each PCI card that is a network device in the system. For IA64, only PCI devices that are Intel PROSet supported adapters will have instances.

Creating Instances
You cannot create instances of IANet_PCIDevice.

Removing Instances
You cannot remove instances of IANet_PCIDevice.

Modifying Properties
There are no user-modifiable properties of this class.

Associations
See IANet_EthernetPCIDevice for the class associations.

Methods
There are no supported methods for this class.

Unsupported Attributes
The following attributes are not supported by the WMI Provider:

- AdditionalAvailabitlity
- Capabilities
- CapabilityDescriptions
- Caption
- DeviceSelectTiming
- ErrorCleared
- ErrorDescription
- IdentifyingDescription
- InstallDate
- LastErrorCode
- MaxNumberController
- MaxQuiesceTime
- Name
- OtherIdentifyingInfo
- PowerManagementCapabilities
- PowerManagementSupported
- PowerOnHours
- ProtocolDescription
- ProtocolSupported
- SelfTestEnabled
- TimeOfLastReset
- TotalPowerOnHours

Back to Top

IANet_EthernetPCIDevice

Purpose
This class is used to describe the properties of a PCI device for a Intel PROSet supported Ethernet adapter. This is a subclass of IANet_PCIDevice. The class contains some extra attributes that are only known for Intel PROSet supported PCI devices.

Instances
An instance of this class exists for each PCI card that is a Intel PROSet supported Ethernet adapter.

Creating Instances
You cannot create instances of IANet_EthernetPCIDevice.

Removing Instances
You cannot remove instances of IANet_ EthernetPCIDevice.

Modifying Properties
There are no user-modifiable properties for this class.

Associations
An instance IANet_AdapterDevice is used to associate the IANet_PCIDevice with the IANet_EthernetAdapter. Virtual adapters (i.e., adapters created to represent teams) do not have an associated IANet_PCIDevice.

Unsupported Attributes
The following attributes are not supported by the WMI Provider:

- AdditionalAvailabitlity
- Capabilities
- CapabilityDescriptions
- Caption
- DeviceSelectTiming
- ErrorCleared
- ErrorDescription
- IdentifyingDescription
- InstallDate
- LastErrorCode
- MaxNumberController
- MaxQuiesceTime
- Name
- OtherIdentifyingInfo
- PowerManagementCapabilities

- PowerManagementSupported
- PowerOnHours
- ProtocolDescription
- ProtocolSupported
- SelfTestEnabled
- Status
- StatusInfo
- TimeOfLastReset
- TotalPowerOnHours

Methods
There are no supported methods for this class.

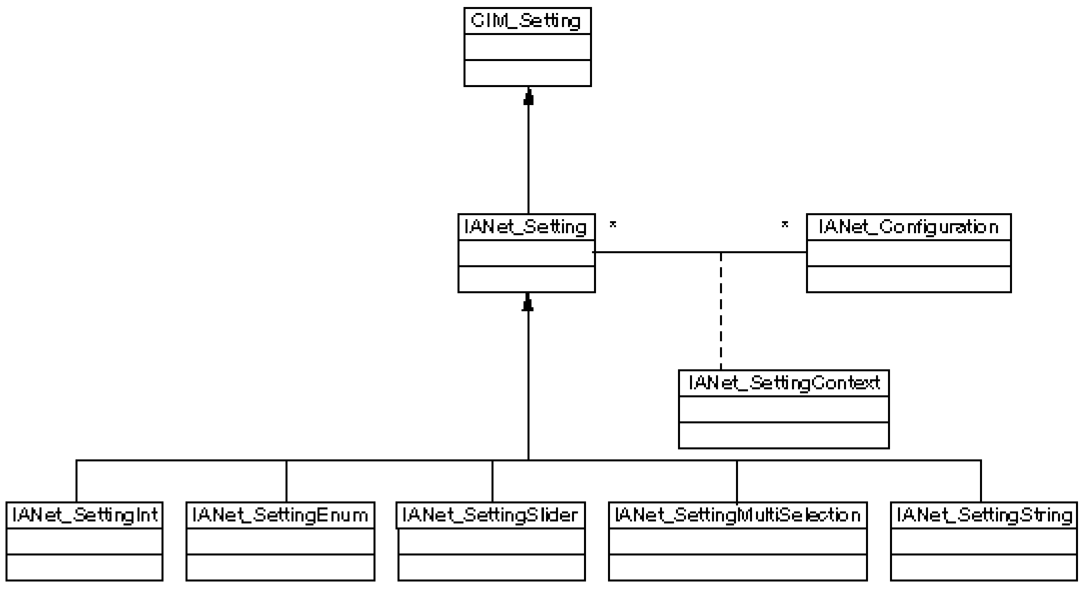
Back to Contents Page   Back to Top

Back to Contents Page

The Setting Schema: Intel® PRO Network Adapters WMI and CDM Providers User Guide

---

Overview



Back to Top

IANet_Configuration

Purpose
This class is used to group a collection of IANet_Setting instances. The class is derived from CIM_Configuration.

Instances
Each adapter, VLAN or Team can have several associated IANet_Configuration instances (each configuration corresponds to a different usage scenario for the adapter.

For this WMI and CDM Providers release, there will be only one instance of IANet_Configuration per adapter, VLAN or team.

Creating Instances
You cannot create instances of IANet_Configuration.

Removing Instances
You cannot remove instances of IANet_Configuration.

Modifying Properties
There are no user-modifiable properties for this class.

Associations
An IANet_AdapterConfiguration instance will exist to associate each adapter (IANet_EthernetAdapter) with its configuration. An IANet_VLANConfiguration instance will exist to associate each VLAN (IANet_VLAN) with its configuration. An IANet_BootAgentConfiguration instance will exist to associate each boot agent (IANet_BootAgent) with its configuration.

Methods
There are no supported methods for this class.

Unsupported Attributes
None.

Back to Top

IANet_Setting

Purpose
This abstract class is used to describe a settable property in a configuration. The class is derived from CIM_Setting.

Instances
A separate instance of this class will exist for each setting on each adapter, VLAN or Team. Settings are not shared between configurations.

There are several sub-classes for IANet_Setting. 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.

Creating Instances
You cannot create instances of IANet_Setting.

Removing Instances
You cannot remove instances of IANet_Setting.

Modifying Properties
This abstract class has no modifiable properties, however, the child classes have modifiable properties (see below).

Associations
Each IANet_Setting instance is associated with an IANet_Configuration instance using an instance of IANet_SettingContext.

Methods
There are no supported methods for this class. To make changes to a setting, modify the required property and call **PutInstance**.

Unsupported Attributes
SettingID is not used.

Back to Top

IANet_SettingInt

Purpose
This class models a setting that takes an integer value. There are several IANet setting classes used to model integers. The differences between these classes is related to the way the GUI displays and modifies the integer and the way the Providers validate. For IANet_SettingInt, 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.

Creating Instances
You cannot create instances of this class.

Removing Instances
You cannot remove instances of this class.

Modifying Properties
The "CurrentValue" attribute is the only modifiable property of this class. You can modify this property by using **IWbemClassObject::Put()** to change the value, then call **IWbemServices::PutInstance()** to update the setting. The Providers will check that:

CurrentValue <= **max**
CurrentValue > = **min**
(**CurrentValue** - **min**) is a multiple of **Step**

Where **max**, **min**, **CurrentValue** and **Step** are all attributes of IANet_SettingInt.

Associations
Each IANet_SettingInt instance is associated with an IANet_Configuration instance using an instance of IANet_SettingContext.

Unsupported Attributes
SettingID is not used.

Methods
There are no supported methods for this class. To make changes to a setting, modify the required property and call **PutInstance**.

Back to Top

IANet_SettingEnum

Purpose
This class models a setting that takes an integer value. There are several IANet setting classes used to model integers. The differences between these classes is related to the way the GUI displays and modifies the integer and the way the Providers validate. For IANet_SettingEnum, it is expected that the GUI will display a list of strings that 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 enum.

Creating Instances
You cannot create instances of this class.

Removing Instances
You cannot remove instances of this class.

Modifying Properties
The **CurrentValue** attribute is the only modifiable property of this class. Modify this property by using **Put()** to change the value, then call **PutInstance()** to update the setting. The Providers will check that **CurrentValue Є PossibleValues[]**

Associations
Each IANet_SettingEnum instance is associated with an IANet_Configuration instance using an instance of IANet_SettingContext.

Unsupported Attributes
SettingID is not used.

Methods
There are no supported methods on this class. To make changes to a setting, modify the required property and call **PutInstance**.

Back to Top

IANet_SettingSlider

Purpose
This class models a setting that takes an integer value. There are several IANet setting classes used to model integers. The differences between these classes is related to the way the GUI displays and modifies the integer and the way the Providers validate. For IANet_SettingSlider, it is expected that the GUI will display a slider that will allow you 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.

Creating Instances
You cannot create instances of this class.

Removing Instances
You cannot remove instances of this class.

Modifying Properties
The **CurrentValue** attribute is the only modifiable property of this class. Modify this property by using **Put()** to change the value, then call **PutInstance()** to update the setting. The Providers will check that **CurrentValue Є PossibleValues[]**

Associations
Each IANet_SettingSlider instance is associated with an IANet_Configuration instance using an instance of IANet_SettingContext.

Unsupported Attributes
SettingID is not used.

Methods
There are no supported methods on this class. To make changes to a setting, modify the required property and call **PutInstance**.

Back to Top

IANet_SettingMultiSelection

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

Instances
An instance of this class exists for each setting that will be displayed as a multi-selection.

Creating Instances
You cannot create instances of this class.

Removing Instances
You cannot remove instances of this class.

Modifying Properties
The **CurrentValue** attribute is the only modifiable property of this class. Modify this property by using **Put()** to change the value, then use **PutInstance()** to update the setting. The Providers will check that **CurrentValue Є PossibleValues[].**

Associations
Each IANet_SettingMultiSelection instance is associated with an IANet_Configuration instance using an instance of IANet_SettingContext.

Unsupported Attributes
SettingID is not used.

Methods
There are no supported methods for this class. To make changes to a setting, modify the required property and call **PutInstance**.

Back to Top

IANet_SettingString

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

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

Creating Instances
You cannot create instances of this class.

Removing Instances
You cannot remove instances of this class.

Modifying Properties
The **CurrentValue** attribute is the only modifiable property of this class. Modify this property by using **Put()** to change the value, then call **PutInstance()** to update the setting.

Associations
Each IANet_SettingMultiSelection instance is associated with an IANet_ElementConfiguration instance using an instance of IANet_SettingString.

Methods
There are no supported methods for this class.

Unsupported Attributes
SettingID is not used.

Methods
There are no supported methods for this class. To make changes to a setting, modify the required property, then call **PutInstance**.

Please read all restrictions and disclaimers.

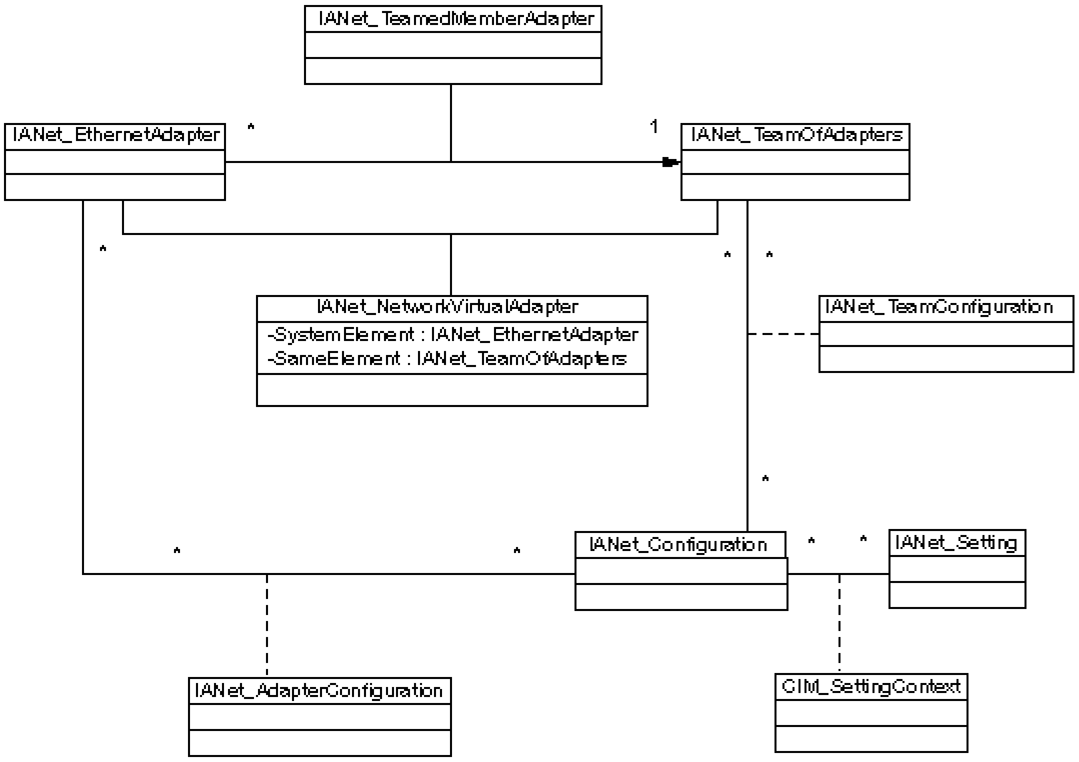
Back to Contents Page   Back to Top

Back to Contents Page

The Team Schema: Intel® PRO Network Adapters WMI and CDM Providers User Guide

Overview

The Team Schema describes how the Ethernet adapters are grouped together into teams.


IANet_TeamedMemberAdapter
IANet_EthernetAdapter
*
1
IANet_TeamOfAdapters
*
*
*
IANet_NetworkVirtualAdapter
-SystemElement : IANet_EthernetAdapter
-SameElement : IANet_TeamOfAdapters
IANet_TeamConfiguration
*
*
*
IANet_Configuration
*
*
IANet_Setting
IANet_AdapterConfiguration
CIM_SettingContext

Back to Top

IANet_TeamOfAdapters

Purpose
This class implements the CIM_RedundancyGroup class. 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 Team.

Creating Instances
To create an empty team, create an instance of IANet_TeamOfAdapters. You must set the correct **TeamingMode** before calling **IWbemServices::PutInstance()** to create the object in the Providers. The Providerswill return a string containing the object path of the new object.

Removing Instances
Correspondingly, to remove a team, delete the instance of IANet_TeamOfAdapters. The Providerswill delete the associations

to the team members and the virtual adapter and settings for the team.

Modifying Properties
Use **Put()** to change the value of the **TeamingMode** property, then call **PutInstance()** to update the team.

Associations
Each adapter in a team is associated with the team's instance of IANet_TeamOfAdapters using an instance of IANet_TeamMemberAdapter. The virtual adapter for the team is associated with this class using an instance of IA_NetNetworkVirtualAdapter.

Methods
This class instance supports the following method:

TestSwitchConfiguration — 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, this test can verify that link partners connected to the adapter support Link Aggregation.

Back to Top

IANet_TeamedMemberAdapter

Purpose
This class is used to associate the adapter with the team, and determines the function of the adapter in the team and establishes that the adapter is currently active in the team. This class implements the CIM class CIM_NetworkAdapterRedundancyComponent.

Instances
An instance of this class exists for each adapter that is a member of a team.

Creating Instances
To add an adapter to a team, create an instance of IANet_TeamedMemberAdapter to associate the adapter with the team.

Removing Instances
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.

Modifying Properties
The **AdapterFunction** property of this class may be modified to describe how the adapter is used within a team.

Associations
This is an association class.

Methods
There are no supported methods on this class.

Back to Top

IANet_NetworkVirtualAdapter

Purpose
This class is used to associate the team's IANet_TeamOfAdapters with the IANet_EthernetAdapter that represents the virtual adapter for the team. The class implements the CIM class CIM_ CIM_NetworkVirtualAdapter.

Instances
An instance of this class exists for each Intel team that has been bound to a virtual adapter.

Creating Instances
You cannot create instances of this class. To create a team, create an instance of IANet_TeamOfAdapters. This class will not exist until after you have called **IANet_NetService .Apply()** within the context of a valid session and the IANet_EthernetAdapter instance has been created.

Removing Instances
You cannot delete instances of this class.

Associations

This is an association class.

Methods
There are no supported methods on this class.

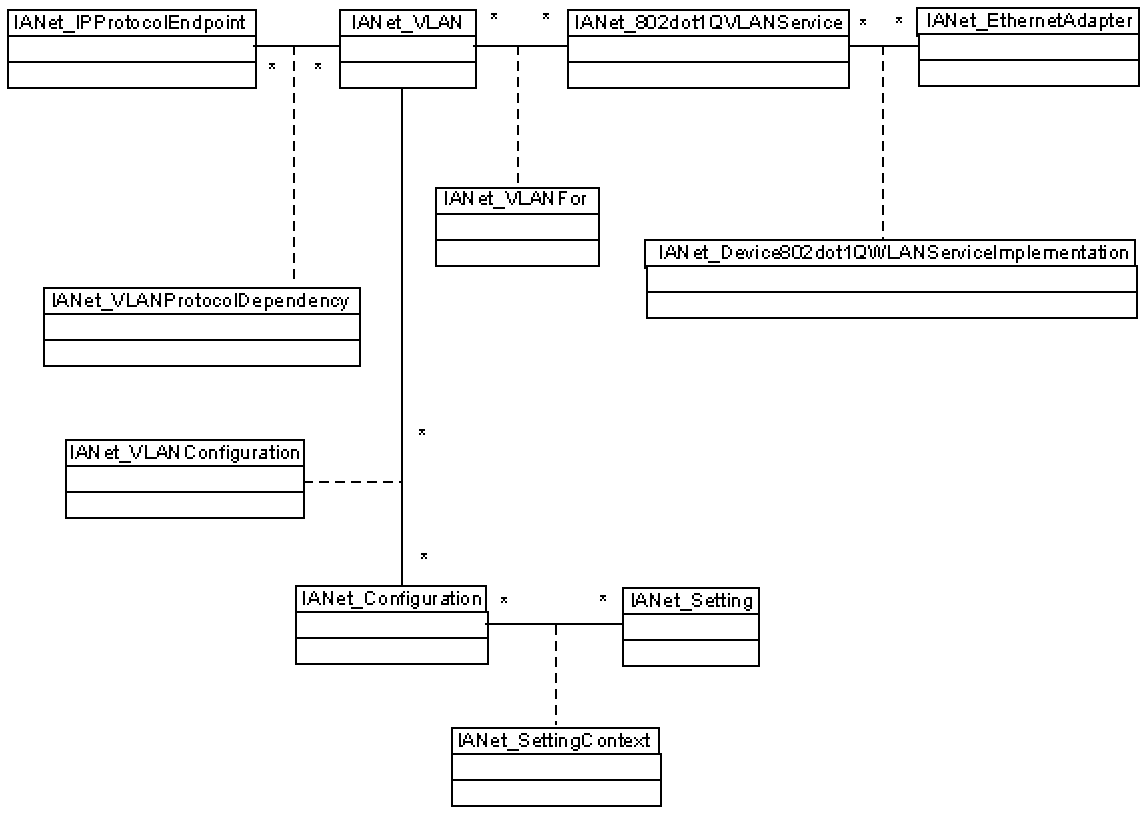
Please read all restrictions and disclaimers.

Back to Contents Page   Back to Top

Back to Contents Page

The VLAN Schema: Intel® PRO Network Adapters WMI and CDM Providers User Guide

Overview



Back to Top

IANet_802dot1QVLANService

Purpose
This class is used to hold the IEEE 802.1Q properties of a network adapter. The 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 can have just one IANet_802dot1QVLANService. Some teams, such as multi-vendor fault tolerant teams, do not support this service.

Anomaly
A team with no VLANs does not have a VLAN service unless you enumerate it within the context of a valid session. For teams, the 802.3QvlanService instance will appear only in the following situations:

- If the team already has VLANs

- If the team does not have VLANs and you use a session handle in the context while enumerating this class.

Creating Instances
You 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.

Removing Instances
You cannot delete instances of this class.

Modifying Properties
There are no modifiable properties of this class.

Associations
Each instance of this class will be associated with one IANet_EthernetAdapter using an instance of IANet_DeviceServiceImplementation.

Each instance of IANet_802dot1QVLANService can support several VLANs; each VLAN will be associated with the instance using IANet_VLANFor association.

Methods
uint16 CreateVLAN([in] uint32 VLANNumber, [in] string Name, [out] IANet_VLAN REF VLANpath); — 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.

Back to Top

IANet_VLAN

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

Instances
An instance of this class will exist for each Intel VLAN.

Creating Instances
To create a VLAN, call **CreateVLAN** on the appropriate instance of IANet_802dot1QVLANService.

Removing Instances
You can remove an instance of this class to remove the corresponding VLAN.

Modifying Properties
You can modify the VLANNumber and Caption attribute.

Associations
Each instance is associated with an instance of IANet_802dot1QVLANService, and therefore to one instance of IANet_EthernetAdapter, using the class IANet_VLANFor.

Each instance can be associated with several IANet_Configuration instances to group a set of settings for the VLAN. For this Provider release, there is only one IANet_Configuration object per VLAN.

Each instance can be associated with one IANet_IPProtocolEndpoint to provide the IP settings for the VLAN using the class IANet_VLANProtocolDependency.

Methods
None

Please read all restrictions and disclaimers.

Back to Contents Page   Back to Top

Back to Contents Page

Getting the Current Configuration: Intel® PRO Network Adapters WMI and CDM Providers User Guide

The client does not need to get a session 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 table, 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.

Getting the Physical Adapters

Task	**WQL Query**	**Result Class**	**Comment**
Enumerate all adapters	SELECT * FROM IANet_EthernetAdapter	IANet_EthernetAdapter	Returns all IANet_EthernetAdapters. This is equivalent to IWbemServices::CreateInstanceEnumAsync.
Determine if adapter is virtual	ASSOCIATORS OF {adapter path} WHERE AssocClass = IANet_NetworkVirtualAdapter	IANet_TeamOfAdapters	If the query results in no classes then the adapter is a real adapter.
Determine if adapter is a phantom adapter	ASSOCIATORS OF {adapter path} WHERE ResultClass = IANet_EthernetPCIDevice	IANet_EthernetPCIDevice	If the adapter is not virtual and this query returns no objects then the adapter is a phantom adapter.

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

Back to Top

Getting the PCI Devices

The main classes are IANet_EthernetPCIDevice, IANet_PCIDevice, and IANet_AdapterDevice (an association class to associate an adapter with its device).

In this case, the association class does not contain any data, i.e., it has no value by itself. IANet_EthernetPCIDevice inherits from IANet_PCIDevice and contains extra attributes that are specific for a PCI device that is an Ethernet adapter.

Task	**WQL Query**	**Result Class**	**Comment**
Enumerate network PCI	SELECT * FROM IANet_PCIDevice	IANet_PCIDevice	The query may return modems and other devices that Intel® PROSet does not

devices			configure. This is equivalent to IWbemServices::CreateInstanceEnumAsync.
Enumerate all PCI devices used by Intel® PROSet configured adapters.	SELECT * FROM IANet_EthernetPCIDevice	IANet_EthernetPCIDevice	The query returns just the PCI devices that Intel® PROSet manages. This is equivalent to IWbemServices::CreateInstanceEnumAsync.
Determine if adapter is installed.	N/A	IANet_EthernetPCIDevice	Check the "Availability" attribute of IANet_EthernetPCIDevice. If equal to 10 - "Not Installed", then the device is not installed. Note: Not much information is known about the device at this stage.
Get the PCI Device associated with an adapter	ASSOCIATORS OF { IANet_EthernetAdapter path} WHERE ResultClass = IANet_EthernetPCIDevice	IANet_EthernetPCIDevice	If this returns no objects, then the adapter is either virtual or a phantom adapter.
Get the adapter associated with a PCI device	ASSOCIATORS OF {Ethernet PCI device path} WHERE ResultClass = IANet_EthernetAdapter	IANet_EthernetAdapter	This query is not optimal for the Provider – it is preferred that the client starts with the adapter.

Back to Top

Getting the Adapter Settings

The setting objects are not associated with the adapter directly. In conformance with the CIM standard, they are associated with a configuration object, which is associated with the adapter.

The classes involved with this part of the schema are IANet_EthernetAdapter, IANet_Configuration, IANet_SettingInt, IANet_SettingString, IANet_SettingEnum, IANet_SettingMultiSelection and IANetSettingSlider.

The association classes IANet_AdapterConfiguration and IANet_SettingContext do not contain any real data - they act as glue between the settings and their parent object.

Task	**WQL Queries**	**Result Class**	**Comments**
Get the Configuration object for the adapter	ASSOCIATORS OF {IANet_EthernetAdapter path} WHERE ResultClass = IANet_Configuration	IANet_Configuration	Returns exactly one object; even if there are no settings there will always be a configuration object. The object path from this object is used in the next query.
Get the settings associated with an adapter	ASSOCIATORS OF {IANet_Configuration path} WHERE AssocClass = IANet_SettingContext	A mixture of IANet_SettingInt, IANet_SettingString, IANet_SettingEnum, IANet_SettingMultiSelection, IANet_SettingSlider	Returns all the setting classes associated with the adapter. The client should use the __CLASS attribute to determine that type of each setting.

Back to Top

Getting the Team Configuration

The main classes in the teaming schema are IANet_EthernetAdapter, IANet_TeamOfAdapters, IANet_NetworkVirtualAdapter and IANet_TeamedMemberAdapter.

The challenge in this schema is that an instance of IANet_EthernetAdapter exists for each physical adapter and for each virtual adapter. The client must be capable of distinguishing between the virtual adapter for a team and the adapters that are members of the team.

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. This is equivalent to IWbemServices::CreateInstanceEnumAsync.
Get the virtual adapter for a team	ASSOCIATORS OF {IANet_TeamOfAdapters path} WHERE AssocClass = IANet_NetworkVirtualAdapter	IANet_EthernetAdapter	Returns only the adapter object for the virtual adapter in the team. This adapter will not exist if the team has been created but Apply has not been called. (see below on updating the configuration).
Enumerate the team's member adapters	ASSOCIATORS OF {IANet_TeamOfAdapters path} WHERE AssocClass = IANet_TeamedMemberAdapter	IANet_EthernetAdapter	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_EthernetAdapter 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.

Back to Top

Getting the Team Settings

The setting objects are not associated with the team directly. In conformance with the CIM standard, they are associated with a configuration object which is associated with the virtual IANet_EthernetAdapter for the team. The same configuration object is also associated with the IANet_TeamOfAdapters object for the team.

The classes involved with this part of the schema are IANet_EthernetAdapter, IANet_TeamOfAdapters, IANet_Configuration, IANet_SettingInt, IANet_SettingString, IANet_SettingEnum, IANet_SettingMultiSelection and IANetSettingSlider.

The association classes IANet_AdapterConfiguration and IANet_SettingContext do not contain any real data - they act as glue between the settings and their parent object. This is exactly the same as the adapter settings case.

Task	**WQL Queries**	**Result Class**	**Comments**
Get the configuration object for the team, starting with the virtual adapter	ASSOCIATORS OF {IANet_EthernetAdapter path} WHERE ResultClass = IANet_Configuration	IANet_Configuration	Returns exactly one object. Even if there are no settings, there will always be a configuration object. The object path from this object is used in the next query.
Get the configuration object for the team, starting with the team of adapters	ASSOCIATORS OF {IANet_TeamOfAdapters path} WHERE ResultClass = IANet_Configuration		
Get the settings associated with an adapter	ASSOCIATORS OF {IANet_Configuration path} WHERE AssocClass = IANet_SettingContext	A mixture of IANet_SettingInt, IANet_SettingString, IANet_SettingEnum, IANet_SettingMultiSelection,	Returns all the setting classes associated with the adapter. The client should use the __CLASS attribute to determine that type of each setting.

		IANet_SettingSlider	

Back to Top

Getting the VLAN Configuration

Each adapter that supports VLANs has an IANet_802dot1QVLANService associated with it, using the association class IANet_Device802do1QVVLANServiceImplementation. If an adapter 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. The VLAN is not directly associated with the adapter - it is associated with the IANet_802dot1QVLANService for the adapter.

The association class IANet_VLANFor is used to associate each VLAN instance with the correct IANet_802dot1QVLANService. This class contains no useful data for the user.

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.

Back to Top

Getting the VLAN Settings

The setting objects are not associated with the VLAN directly. In conformance with the CIM standard, they are associated with a configuration object, which is associated with the IANet_VLAN object for the VLAN.

The classes involved with this part of the schema are IANet_VLAN, IANet_Configuration, IANet_SettingInt, IANet_SettingString, IANet_SettingEnum, IANet_SettingMultiSelection and IANetSettingSlider.

The association classes IANet_VLANConfiguration and IANet_SettingContext do not contain any real data - they act as glue between the settings and their parent object. This is exactly the same as the adapter settings case.

Task	**WQL Queries**	**Result Class**	**Comments**
Get the Configuration object for the VLAN	ASSOCIATORS OF {IANet_VLAN path} WHERE ResultClass = IANet_Configuration	IANet_Configuration	Returns exactly one object; even if there are no settings there will always be a configuration object. The object path from this object is used in the next query.
Get the settings associated with a VLAN	ASSOCIATORS OF {IANet_Configuration path} WHERE AssocClass = IANet_SettingContext	A mixture of IANet_SettingInt, IANet_SettingString, IANet_SettingEnum, IANet_SettingMultiSelection, IANet_SettingSlider	Returns all the setting classes associated with the VLAN. The client should use the __CLASS attribute to determine the type of each setting.

Back to Top

Getting the IP Protocol Information

The Provider will give some limited information about the IP Protocol Endpoints that are associated with adapters, VLANs and teams. No other protocols are supported.

The main class containing the protocol information is IANet_IPProtocolEndpoint. There are two association classes: IANet_VLANProtocolDependency and IANet_AdapterProtocolImplementation. To get the IP endpoint for a team, first get the virtual IANet_EthernetAdapter for the team, i.e., the IP Endpoint is associated with this instance.

Task	WQL Queries	Result Class	Comments
Get the IP Protocol Endpoints associated with an adapter	ASSOCIATORS OF {path of IANet_EthernetAdapter} WHERE ResultClass = IANet_IPProtocolEndpoint	IANet_IPProtocolEndpoint	Though some adapters may have more than one IP address, they will be associated with only one IP Protocol Endpoint instance.
Get the IP Protocol Endpoints associated with a VLAN.	ASSOCIATORS OF {path of IANet_VLAN} WHERE ResultClass = IANet_IPProtocolEndpoint	IANet_IPProtocolEndpoint	There will be one IP Protocol Endpoint associated with the VLAN.

Back to Top

Getting the 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 attributes provide information on the boot ROM image for this adapter: InvalidImageSignature, Version, UpdateAvailable, FlashImageType

Back to Top

Getting the Boot Agent Settings

The setting objects are not associated with the boot agent directly. In conformance with the CIM standard, they are associated with a configuration object which is associated with the boot agent.

The classes involved with this part of the schema are IANet_BootAgent, IANet_Configuration, IANet_SettingInt, IANet_SettingString, IANet_SettingEnum, IANet_SettingMultiSelection and IANetSettingSlider.

The association classes IANet_BootAgentConfiguration and IANet_SettingContext do not contain any real data - they act as glue between the settings and their parent object.

Task	WQL Queries	Result Class	Comments
Get the Configuration object for the boot agent	ASSOCIATORS OF { IANet_BootAgent path} WHERE ResultClass = IANet_Configuration	IANet_Configuration	Returns exactly one object. Even if there are no settings, there will always be a configuration object. The object path from this object is used in the following query.
Get the settings associated with an boot agent	ASSOCIATORS OF {IANet_Configuration path} WHERE AssocClass = IANet_SettingContext	A mixture of IANet_SettingInt, IANet_SettingString, IANet_SettingEnum, IANet_SettingMultiSelection, IANet_SettingSlider	Returns all the setting classes associated with the adapter. The client should use the __CLASS attribute to determine that type of each setting.

Back to Contents Page Back to Top

Back to Contents Page

Updating the Configuration: Intel® PRO Network Adapters WMI and CDM Providers User Guide

---

Overview

In most cases, to update the configuration, the client application will need to get a session handle from the IANet_NetService class and store this handle in an IWbemContext context object. Changes to the configuration will only occur when the **Apply** method on the IANet_NetService is called. There are some exceptions to this requirement:

- Changes to the boot agent class will occur immediately as they are made and do not require a session handle.
- Certain method calls (e.g. identify adapter) cause an operation to be carried out before **Apply** is called.

For some operations you can use the PreCheck qualifier in the context to check to see if an operation is allowed. This is to allow a user interface to disable certain controls or menu items if required.

Back to Top

Changing the Adapter, Team, or VLAN Settings

Changing the adapter, team or VLAN:

- Requires Session Handle.
- PreCheck is available.
- Requires an **Apply** call before operation is executed.

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 (see above).

The easiest way for the client to update a setting is to:

1. Get an instance of the setting object from the WMI.
2. Modify the CurrentValue attribute (using **IWbemClassObject::Put()**).
3. Call **IWbemServices::PutInstance()** to pass the modified instance back to the WMI Provider. **PutInstance** must be called with the flag WBEM_FLAG_UPDATE_ONLY.

The WMI Provider will validate the CurrentValue and return WBEM_E_FAIL if the validation failed. The exact reason for the failure will be returned in the Description attribute of the IANet_ExtendedStatus object.

Setting specific descriptions include:

- 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 setting value is not one of the allowable values.

The last description is returned when the current value for IANet_SettingEnum, IANet_SettingSlider or IANet_SettingMultiSelection is not one of the allowable values.

The only attribute for a setting that the client can change is CurrentValue. The WMI Provider will ignore changes made to any of the other values.

There are no supported methods on the setting class. To make changes to a setting modify the CurrentValue property, call **PutInstance**.

Back to Top

Creating a New (Empty) Team

Creating a new team:

- Requires Session Handle.
- PreCheck is available.
- Requires an **Apply** call before operation is executed.

To create a new team, create an instance of IANet_TeamOfAdapters (i.e., use **IWbemServices::GetObject()** to get a class object for IANet_TeamOfAdapters and then use **IWbemServices::SpawnInstance()** to create an instance of this object).

Then, use **IWbemClassObject::Put** to set the TeamMode attribute in the instance to be the desired team type (e.g., AFT). Finally, call **IWbemServices::PutInstance()** to create the team, passing the flag WBEM_FLAG_CREATE_ONLY.

The object path for the new team is stored in the **IWbemCallResultObject** that is passed back to you when the call has completed. The method **IWbemCallResult::GetResultString** will get the new object path.

If this action fails, the client should check the IANet_ExtendedStatus to get the failure reasons.

The virtual IANet_EthernetAdapter and IANet_IPProtocolEndpoint classes for the team are not available until after the call to **Apply**. The settings for the team can be accessed using the IANet_Configuration object associated with the new IANet_TeamOfAdapters.

Back to Top

Adding an Adapter to a Team

Adding an adapter to a team:

- Requires Session Handle.
- PreCheck is available.
- Requires an **Apply** call before operation is executed.

To add 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 attributes in the object must be set using IWbemClassObject::Put():

- GroupComponent must be set to be the full object path of the IANet_TeamOfAdapters to 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.

You can also set the priority for the adapter in the team. 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.

Back to Top

Removing an Adapter from a Team

Removing an adapter from a team:

- Requires Session Handle.

- PreCheck is available.
- Requires an **Apply** call before operation is executed.

To remove an adapter from a team, 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.

Back to Top

Deleting a Team

Deleting a team:

- Requires Session Handle.
- PreCheck is available.
- Requires an **Apply** call before operation is executed.

To delete a team, delete the IANet_TeamOfAdapters instance using **IWbemServices::DeleteInstance()**. If this action fails, check IANet_ExtendedStatus to get the error code.

Back to Top

Changing the Mode of a Team

Changing the mode of a team:

- Requires Session Handle.
- PreCheck is available.
- Requires an **Apply** call before operation is executed.

To change the mode of a team, get the instance of IANet_TeamOfAdapters for the team (e.g., use **IWbemServices::GetObject** using the object path of the team). Then, use **IWbemClassObject::Put** to change the TeamMode attribute for the team. Finally, call **IWbemClassObject:: PutInstance** to tell the WMI Provider to update the team mode, passing the flag WBEM_FLAG_UPDATE_ONLY. If this action fails, check IANet_ExtendedStatus to get the error code.

Back to Top

Changing an Adapter's Priority within a Team

Changing an adapter's priority within a team:

- Requires Session Handle.
- PreCheck is available.
- Requires an **Apply** call before operation is executed.

To change the priority of an adapter, the client should first get the instance of IANet_TeamedMemberAdapter for the adapter. (e.g. use **IWbemServices::GetObject** using the object path). The client can then use **IWbemClassObject::Put** to change the AdapterFunction attribute for the adapter. Finally the client needs to call **IWbemClassObject:: PutInstance** to tell the WMI Provider to update the adapter's priority. If this action fails, the client should check the IANet_ExtendedStatus for the error code.

Back to Top

Uninstalling an Adapter

Uninstalling an adapter:

- Requires Session Handle.
- PreCheck is available.
- Requires an **Apply** call before operation is executed.

To uninstall an adapter, call **IWbemServices::DeleteInstance**, passing the object path of the adapter to uninstall.

Back to Top

Creating a VLAN

Creating a VLAN:

- Requires Session Handle.
- PreCheck is available.
- Requires an **Apply** call before operation is executed.

To create a VLAN, call the **CreateVLAN** method on the IANet_802dot1QVLANService for the adapter to which the VLAN is to be added. The following arguments must be passed to the method:

- VLANNumber, which is the number of the VLAN. (Range 1- 4094)
- Name, which is a user-definable name to identify the VLAN.

The function will return the object path of the newly created VLAN in the Out parameter VLANpath. If this action fails, check IANet_ExtendedStatus for the error code.

Back to Top

Changing the Attributes of a VLAN

Changing the attributes of a VLAN:

- Requires Session Handle.
- PreCheck is available.
- Requires an **Apply** call before operation is executed.

The client can change the VLANNumber and VLANName attributes for a VLAN. To change the priority of an adapter, first get the instance of IANet_VLAN for the adapter (e.g. use **IWbemServices::GetObject** using the object path).

Then, change VLANNumber or VLANName to the desired values. Finally, call **IWbemClassObject:: PutInstance** to tell the WMI Provider to update the attributes, passing the flag WBEM_FLAG_UPDATE_ONLY. If this action fails, check the IANet_ExtendedStatus for the error code.

Back to Top

Deleting a VLAN

Deleting a VLAN:

- Requires Session Handle.
- PreCheck is available.
- Requires an **Apply** call before operation is executed.

To delete a VLAN, call **IWbemServices::DeleteInstance** passing the object path of the VLAN to delete.

Back to Top

Updating the Boot Agent

Updating the boot agent:

- Does not require Session Handle.
- PreCheck is not available.
- Does not require an **Apply** call before operation is executed.

The client can update the Boot Agent Image by using methods calls. To read/write flash image, first get the instance of

IANet_BootAgent for the adapter (e.g., use **IWbemServices::GetObject** using the object path).

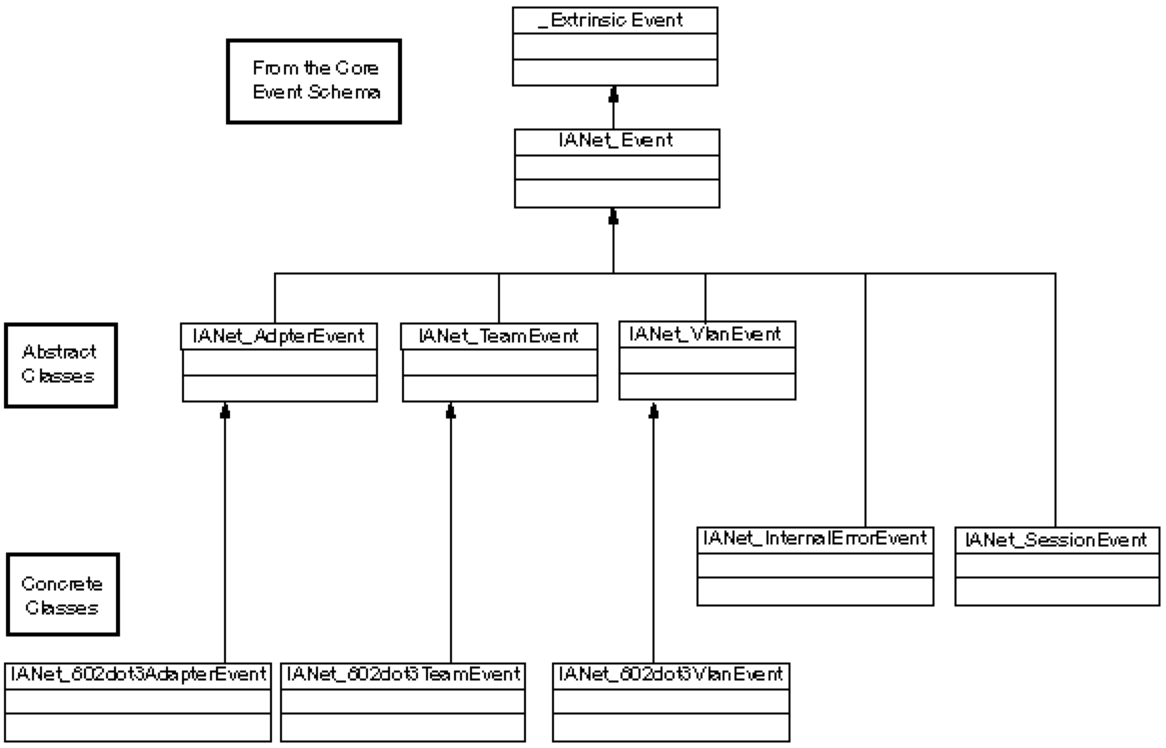
Then, execute ReadFlash() to read the existing flash boot ROM image or ProgramFlash() to update the flash boot ROM image. If this action fails, check the IANet_ExtendedStatus for the error code.

Task	**WMI methods**	**Result**	**Comments**
Update or Insert a boot ROM image for the adapter	uint32 ProgramFlash([IN, ValueMap {"0","1"} , Values {"Check Version", "Write Flash"}: Amended] uint32 Action, [IN] uint8 NewFlashData[], [OUT] string strErrorMessage);	If “Check Version” action is specified, this method will return with a warning message, if boot ROM image being updated as in NewFlashData[] is older than one already present on NIC. If “Write” action is specified, this updates the FLASH ROM on the NIC with NewFlashData[].	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.
Read boot ROM image	uint32 ReadFlash([OUT] uint8 FlashData[]);	FlashData[] contains the Flash ROM image on the NIC.	This method reads the Flash ROM on the NIC which can be saved into a file.

Please read all restrictions and disclaimers.

Back to Contents Page

Event Notifications: Intel® PRO Network Adapters WMI and CDM Providers User Guide



Back to Top

Concrete Event Classes

IANet_802dot3AdapterEvent

Purpose
This event notifies the client about a change in the status or configuration of an adapter.

Triggers
The event occurs after an adapter status changes or after you have changed an adapter setting and called **Apply**.

Event Data
AdapterPath contains the object path of the adapter that caused the event.

IANet_802dot3TeamEvent

Purpose

This event notifies the client about a change in the status or configuration of a team.

Triggers
The event occurs:

- After the status of an adapter changes.
- After you have changed a team setting and called **Apply**.
- After the team configuration has been changed and you call **Apply**.

Event Data
TeamPath contains the object path of the team that caused the event.

IANet_802dot3VlanEvent

Purpose
This event notifies the client about a change in the status or configuration of a VLAN.

Triggers
The event occurs:

- After the status of a VLAN changes.
- After you have changed a VLAN setting and called **Apply**.
- After the VLAN configuration has been changed and you call **Apply**.

Event Data
VlanPath contains the object path of the VLAN that caused the event.

Back to Top

Registering for Events

Applications should use **IWbemServices:: ExecNotificationQuery** or **IWbemServices:: ExecNotificationQueryAsync** to request notifications for events. The following queries are examples of event notification queries. This list is not exhaustive as many queries are possible.

- **SELECT * FROM IANet_Event** — Used to request all events.
- **SELECT * FROM IANet_AdapterEvent** — Used to request all adapter events.
- **SELECT * FROM IANet_TeamEvent** — Used to request all team events.
- **SELECT * FROM IANet_SessionEvent** — Used to request all session events.
- **SELECT * FROM IANet_VlanEvent** — Used to request all VLAN events.
- **SELECT * FROM IANet_InternalErrorEvent** — Used to request all internal events.
- **SELECT * FROM IANet_AdapterEvent WHERE AdapterPath={IANet_EthernetAdapter object path}** — Used to request adapter events for a particular adapter.

Back to Contents Page   Back to Top

Optimized WQL Queries: Intel® PRO Network Adapters WMI and CDM Providers User Guide

---

Overview

The WMI Provider is optimized to allow applications to get settings using queries. The WMI Provider can recognize the following queries and will only return the objects that match. All other queries will result in the WMI Provider getting all the settings on all objects and CIMOM will filter them before they reach the application. With several adapters, teams, and VLANs, this can cause a delay of several seconds when retrieving required data.

Back to Top

Getting the Settings for a Particular Adapter, VLAN, or Team

The following query will get only the settings for a particular adapter, VLAN or team. WQL does not allow any additional clauses in the WHERE clause.

ASSOCIATORS OF {IANet_Configuration path} WHERE AssocClass = IANet_SettingContext

Back to Top

Getting One Setting

The following query can be used to get a single setting for an object without querying to get them all:

SELECT * FROM [SETTING CLASS] WHERE ParentId="[Device ID]" AND ParentType="[type]" AND Caption="[SETTING NAME]"

Notes:

- The class must be the exact setting class, not a base class (e.g. IANet_SettingInt).
- Acceptable ParentTypes are "NIC", "Team" , "VLAN" or "BootAgent".
- ParentId is the GUID that uniquely defines the object with the setting (for an adapter, this is the DeviceId).
- This is not a recommended method of getting the settings associated for an object; the preferred method is use associations. However, WQL does not support the complex query required (i.e., WQL does not support ASSOCIATORS OF {IANet_Configuration path} WHERE AssocClass = IANet_SettingContext AND Caption="[SETTING NAME]")

Diagnostics: Intel® PRO Network Adapters WMI and CDM Providers User Guide

---

Diagnostic Classes

IANet_DiagTest

Purpose
IANet_DiagTest is subclassed from CIM_DiagnosticTest. The class provides a generic vehicle to run and control diagnostic tests for an Intel® PROSet supported Ethernet adapter. The superclass, 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
Key is Name and in this provider it is the concatenation of a numeric index of the test at the GUID of the referenced adapter (e.g. 1@{12345678-9ABC-DEF0-1234-123456789012}). 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 provider will reject the command. The caption property provides the name of the test that this instance will run. Other properties provide other description and run-time information.

Creating Instances
You cannot create instances of IANet_DiagTest.

Deleting Instances
You cannot delete instances of IANet_DiagTest.

Modifying Properties
There are no user-modifiable properties for this class.

Associations

- An instance of IANet_DiagTestForMSE associates an IANet_DiagTest with an IANet_ManagedSystemElement. The IANet_ManagedSystemElement will be an IANet_EthernetAdapter instance.
- An instance of IANet_DiagResultForTest associates an IANet_DiagTest with an IANet_DiagnosticResult instance.
- An instance of IANetDiagSettingForTest associates an IANet_DiagTest with an IANet_DiagSetting.

Unsupported Properties
Install Date, OtherCharacteristicDescription

Methods
This class supports the following methods:

- RunTest — 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.
  - DiagnosticResult — defines an instance of the class CIM_DiagnosticResult, which will always be the class IANet_DiagResult.
- DiscontinueTest — Attempts to stop a diagnostic test in progress as defined by two parameters referencing CIM_ManagedSystemElement and CIM_DiagnosticResult. 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 — Clears test results using parameters:
  - SystemElement
  - 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 DiagnosticResultForTest, 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.

Class Hierarchy

For CimV2. Unused properties and methods not shown.

- CIM_ManagedElement:
  Caption
  Description
- CIM_ManagedSystemElement:
  Install Date
  Name
  Status
- CIM_LogicalElement
- CIM_Service:
  - Key
    - Name (string)
  - Properties
    - Caption (string)
    - CreationClassName (string)
    - Description (string)
    - Started (boolean)
    - StartMode (string)
    - Status (string)
    - SystemCreationClass (string)
    - SystemName (string)
- CIM_DiagnosticTest:
  - Properties
    - Characteristics (uint16 array)
    - IsInUse (Boolean)
    - ResourcesUsed (uint16 array)
  - Methods
    - RunTest
    - ClearResults
    - DiscontinueTest

Executing RunTest, and Other Methods in WbemTest

The RunTest method from the MOF file is as follows:

```
uint32
RunTest([IN] CIM_ManagedSystemElement ref SystemElement,
[IN] CIM_DiagnosticSetting ref Setting,
[OUT] CIM_DiagnosticResult ref Result);
```

The first two parameters are In parameters. You must get the object path of both objects referenced. You must also get the object path of the IANet_DiagTest object, which is exporting the RunTest object.

1. From the main WBEM test dialog box, click **Connect**.
2. Enter the appropriate Server\Namespace. Namespaces IntelNCS and CimV2 are supported.
3. Click the **Enum Instances** button of WBEM test and enter `IANet_DiagTest`.
4. Double-click the instance of IANet_DiagTest. The name will be in the form X@[AdapterGUID}, where X is the test name and AdapterGUID will be the Adapter Name, same as the Name key of the IANet_EthernetAdapter.
5. The following is an example of the object path retrieved:
   `\\MYCOMPUTER\root\Cimv2:IANet_DiagTest.Name="1@{4A0CDABE-F6C3-45D0-B60D-F6E7BAFA2C2C}"`
6. Save the object path.
7. Click the **Enum Instances** button of WBEM test and enter `IANet_EthernetAdapter`.
8. Double-click on the adapter to be tested.
9. The following is an example of the object path retrieved:
   `\\MYCOMPUTER\root\cimv2:IANet_EthernetAdapter.DeviceID="{4A0CDABE-F6C3-45D0-B60D-F6E7BAFA2C2C}"`
10. Save the object path.
11. Click the **Enum Instances** button of WBEM test and enter `IANet_DiagSetting`.
12. Double-click on the setting that represents the adapter/test combination.
13. The following is an example of the object path retrieved:
    `\\MYCOMPUTER\root\cimv2:IANet_DiagSetting.SettingID="1@{4A0CDABE-F6C3-45D0-B60D-F6E7BAFA2C2C}"`
14. Save the object path.
15. From the main WBEM test dialog box, click **Execute Method**.

16. Paste the IANet_DiagTest object path into the dialog box. Click **OK**.
17. Select the test in the drop-down box under method.
18. Click the **Edit In Parameters** button.
19. For RunTest, Setting and SystemElement are the In parameters. Paste the previously saved Setting and Adapter object paths and close.
20. Click the **Execute** button.
21. Enumerate the IANet_DiagResult class, in the same manner as the In parameters.
22. Examine the selected result object as needed.

IANet_DiagSetting

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

Creating Instances
You cannot create instances of this class.

Deleting Instances
You cannot delete instances of this class.

Modifying Properties
UpdateInstanceAsync is implemented and can be used to set test parameters to "Halt On Error", "Report Soft Errors", "Report Status Messages", "Quick Mode", "Test Warning Level" and "Percent Of Test Coverage".

Associations
An instance of IANetDiagSettingForTest associates an IANet_DiagTest with an IANet_DiagSetting.

Unsupported Properties
The following properties are not supported by NCS:

- Caption
- Description

Methods
None

Class Hierarchy

For CimV2. Unused properties and methods not shown.

- CIM_ManagedElement
- CIM_Setting:
  - Properties
  - SettingID
  - Methods (none are supported)
  - VerifyOKToApplyToMSE
  - ApplyToMSE
  - VerifyOKToApplyToCollection
  - ApplyToCollection
  - VerifyOKToApplyIncrementalChangeToMSE
  - ApplyIncrementalChangesToMSE
  - ApplyIncrementalChangeToCollection
- CIM_DiagnosticSetting:
  - Key
    - SettingID (string)
  - Properties
    - TestWarningLevel (uint16)
    - ReportSoftErrors (Boolean)
    - ReportStatusMessages (Boolean)
    - HaltOnError (Boolean)
    - QuickMode (Boolean)
    - PercentOfTestCoverage (uint8)

IANet_DiagResult

Purpose

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
Instances of IANet_DiagResult correspond to results of a particular test run on a specific adapter. The format for the key is the same as IANet_DiagTest and IANet_DiagSetting. The instance can store any arbitrary test results as any data, which does not fit the defined properties, can be placed into the TestResults Array property. Any time a new test is run on an adapter, the new instance overwrites the existing instance of test results corresponding to that adapter and test combination.

Creating Instances
You cannot create instances of this class.

Deleting Instances
You cannot delete instances of this class.

Modifying Properties
You cannot modify instances of this class.

Associations
An instance of IANet_DiagResultForTest associates an IANet_DiagTest with an IANet_DiagnosticResult instance.

Unsupported Properties
The following properties are not supported by NCS:

- EstimatedTimeOfPerforming
- HaltOnError
- OtherStateDescription
- ReportSoftErrors
- TestWarningLevel

Methods
None

Class Hierarchy

For CimV2. Unused properties and methods not shown.

- CIM_DiagnosticResult:
  - Keys
    - DiagnosticCreationClassName (string)
    - DiagnosticName (string)
    - ExecutionID (string)
    - DiagSystemCreationClassName (string)
    - DiagSystemName (string)
  - Properties
    - TimeStamp (string)
    - IsPackage (Boolean)
    - TestStartTime (datetime)
    - TestCompletionTime (datetime)
    - TestState (uint16)
    - TestResults (string)
    - PercentComplete (uint8)
- IANet_DiagResult

Back to Top

Registry Entries

The following entries are entered into the registry during installation under **HKLM\Software\Intel\NETWORK_SERVICES\NCS\NcsDiag**. These keys and values control the execution of the diagnostic tests and are defined below.

The following table contains the values in the key, their type, and a brief explanation of their usage:

Value	**Type**	**Default**	**Usage**
Check Time	REG_DWORD	2 seconds	Duration of time period between checks for completion of sender or

			responder test
Enable	REG_DWORD	0	Enable (1) or Disable (0) usage of a results log file.
FileAppend	REG_DWORD	1	Append (1) log results file to existing log results file. If 0, delete existing file.
LogFileName	REG_SZ	NcsDiag.log	The name of the log results file.
MaxFileSize	REG_DWORD	0x10000	Maximum size of the log results file.
MaxPktsRcvd	REG_DWORD	200	In Quick Mode (from IANet_DiagSetting), send/receive test will terminate when the number of packets received is greater than this value.
TimeoutSndRsp	REG_DWORD	100	Test is exited when duration of test (in seconds) has exceeded this value.

Back to Top

Logging

Results Log

The Results log primarily shows information, which could also be obtained from IANet_DiagResult objects. The difference is that information obtained from a CIM browser will only show the immediate last results of particular tests on a particular adapter. Subsequent tests overwrite earlier test results. The Results log is more convenient for setting up and viewing multiple runs of such a test.

Enabling the Results Log

To enable the results log:

1. In the registry key **HKLM\Software\Intel\NETWORK_SERVICES\NCS\NCSDiag**, set the value Enable to 1.
2. Set the value LogFileName to the preferred log filename or leave at default **NcsDiag.log**.
3. The log file will be found in the directory indicated by the value InstalledDir.

Back to Top

Associative Classes

Associative Class	Reference Property/Value	Reference Property/Value
IANet_DiagTestForMSE	Antecedent = IANet_DiagTest	Dependent = IANet_EthernetAdapter
IANet_DiagResultForTest	DiagnosticResult = IANet_DiagResult	DiagnosticTest = IANet_DiagTest
IANet_DiagSettingForTest	Element = IANet_DiagTest	Setting = IANet_DiagSetting
IANet_DiagResultForTest	DiagnosticResult = IANetDiagResult	DiagnosticTest = IANet_DiagTest

Back to Top

Tests

The implemented tests can run on a single machine or on two machines. Detailed descriptions of the tests are beyond the scope of this document because the CDM Provider is intended to be a generic test vehicle, which is not dependent on the particulars of the tests. However, there is some dependency built into the code and that is explained in this section.

Single Adapter Tests

The following tests are run on a single adapter and need no interaction with any other adapter:

- EEPROM
- Control Registers

- MAC Loopback
- PHY Loopback
- Link

All error messages from these tests result from HRESULT error codes returned from calls to lower stack layers. The error codes are stored internally as error codes and are not translated into error messages until the IANet_DiagResult objects are de-referenced by enumeration or object calls are received from a management application.

Tests Requiring Two Adapters

The sender and responder tests are codependent tests in which one adapter (the sender) sends packets to another adapter (the responder), which sends packets back to the sender, thus completing the loop. These are the same tests, which can be run from Intel ® PROSet. However, Intel PROSet does not use CDM and does not allow you to run two tests on the same machine at the same time. CDM will allow different tests to be run concurrently on the same machine.

Sender/Responder Tests

Sender/Responder requires two Intel adapters, one to be the sender and one to be the responder. This test is the only one in which the test is run based on a second thread that continues running until the test completes according to completion criteria or is stopped by the primary thread. The completion criteria are timeouts based either on the length of the testing time or number of packets received. Both of these values are obtained from the registry. The test may only be ended based on the number of packets received if Quick mode has been enabled. Quick Mode is a property of the IANet_DiagSetting class and hence may be set on a per adapter basis. CDM responders will respond to PROSet responders and vice versa.

Two types of error values are returned from the sender/responder tests. First, an Error Code (HRESULT) may be returned from the lower layers. Secondly, while the test is running and, unless the test is terminated prematurely by returned error code, the test thread will return intermediate and then final test statistics, which will include the following:

- Link Status
- Using Auto-Negotiation
- Collisions
- Packets Received
- Packets Received Total
- Packets Sent
- Transmit Oks
- Receive Oks
- Transmit Errors
- Receive Errors
- Collisions
- Diagnostic Phase

Back to Contents Page   Back to Top

Executing Methods in IANet_DiagTest: Intel® PRO Network Adapters WMI and CDM Providers User Guide

Executing RunTest, and Other Methods in WbemTest

The RunTest method from the MOF file is as follows:

```
uint32
RunTest([IN] CIM_ManagedSystemElement ref SystemElement,
[IN] CIM_DiagnosticSetting ref Setting,
[OUT] CIM_DiagnosticResult ref Result);
```

The first two parameters are In parameters. You must get the object path of both objects referenced. You must also get the object path of the IANet_DiagTest object, which is exporting the RunTest object.

1. From the main WBEM test dialog box, click **Connect**.
2. Enter the appropriate Server\Namespace. Namespaces IntelNCS and CimV2 are supported.
3. Click the **Enum Instances** button of WBEM test and enter `IANet_DiagTest`.
4. Double-click the instance of IANet_DiagTest. The name will be in the form X@[AdapterGUID}, where X is the test name and AdapterGUID will be the Adapter Name, same as the Name key of the IANet_EthernetAdapter.
5. The following is an example of the object path retrieved:
   `\\MYCOMPUTER\root\Cimv2:IANet_DiagTest.Name="1@{4A0CDABE-F6C3-45D0-B60D-F6E7BAFA2C2C}"`
6. Save the object path.
7. Click the **Enum Instances** button of WBEM test and enter `IANet_EthernetAdapter`.
8. Double-click on the adapter to be tested.
9. The following is an example of the object path retrieved:
   `\\MYCOMPUTER\root\cimv2:IANet_EthernetAdapter.DeviceID="{4A0CDABE-F6C3-45D0-B60D-F6E7BAFA2C2C}"`
10. Save the object path.
11. Click the **Enum Instances** button of WBEM test and enter `IANet_DiagSetting`.
12. Double-click on the setting that represents the adapter/test combination.
13. The following is an example of the object path retrieved:
    `\\MYCOMPUTER\root\cimv2:IANet_DiagSetting.SettingID="1@{4A0CDABE-F6C3-45D0-B60D-F6E7BAFA2C2C}"`
14. Save the object path.
15. From the main WBEM test dialog box, click **Execute Method**.
16. Paste the IANet_DiagTest object path into the dialog box. Click **OK**.
17. Select the test in the drop-down box under method.
18. Click the **Edit In Parameters** button.
19. For RunTest, Setting and SystemElement are the In parameters. Paste the previously saved Setting and Adapter object paths and close.
20. Click the **Execute** button.
21. Enumerate the IANet_DiagResult class, in the same manner as the In parameters.
22. Examine the selected result object as needed.

Please read all restrictions and disclaimers.

Summary of CIM Classes: Intel® PRO Network Adapters WMI and CDM Providers User Guide

Class	Can Create?	Can Delete?	Implemented Methods	Settable Properties	Unsupported Properties	Instance Count	Related Association Classes
IANet_802dot1QVLANService	N	N	CreateVLAN	GRVPEnabled, JoinTime	Description, Install Date, Started, StartMode, Status	One instance for each team or adapter that supports VLANs	IANet_Device802dot1QVLANServiceImplementation, IANet_VLANFor
IANet_AdapterConfiguration	N	N	None	None	None	One instance for each adapter	This class associates IANet_EthernetAdapter with IANet_Configuration.
IANet_AdapterDevice	N	N	None	None	None	One instance for each non-phantom adapter	This class associates IANet_EthernetAdapter with IANet_EthernetPCIDevice.
IANet_AdapterProtocolImplementation	N	N	None	None	None	One instance for each IP protocol endpoint which is bound to an adapter	This class associates IANet_EthernetAdapter with IANet_IPProtocolEndpoint.
IANet_BootAgent	N	N	ProgramFlash ReadFlash	None	Caption, Description, InstallDate, Started, StartMode, Status	One instance for each adapter that supports the boot agent capability	IANet_DeviceBootServiceImplementation, IANet_BootAgentConfiguration.
IANet_BootAgentConfiguration	N	N	None	None	None	One instance for each boot agent.	This class associates IANet_BootAgent with IANet_Configuration.
IANet_Configuration	N	N	None	None	None	One Instance for each adapter, VLAN and team	IANet_AdapterConfiguration, IANet_VLANConfiguration, IANet_SettingContext.
IANet_Device802dot1QVLANServiceImplementation	N	N	None	None	None	One instance for each adapter or team which supports VLANs	This class associates IANet_EthernetAdapter with IANet_802dot1QVLANService.
IANet_DiagTest	N	N	RunTest, DiscontinueTest, ClearResults	None	InstallDate, OtherCharacteristicsDescription	One for each adapter/test combination	IANet_DiagTestForMSE, IANet_DiagResultForTest, IANet_DiagSettingForTest
IANet_DiagSetting	N	N	None	HaltOnError, ReportSoftErrors, ReportStatusMessages, QuickMode, PercentOfTestCoverage, TestWarningLevel	Caption, Description	One for each adapter/test combination	IANet_DiagSettingForTest

IANet_DiagResult	N	N	None	None	EstimatedTimeOfPerforming, HaltOnError, OtherStateDescription, ReportSoftErrors, TestWarningLevel	One for each adapter/test combination	IANet_DiagResultForTest, IANet_DiagResultForMSEM
IANet_EthernetAdapter	N	Y	IdentifyAdapter HasVLANs IsPowerMgmtSupported GetPowerUsage SetPowerUsage GetPowerUsageOptions SetPowerUsageOptions TestCable AdvancedTestCable TestLinkSpeed	None	AutoSense – (this is exposed as a setting), ErrorCleared, OtherIdentifyingInfo, IdentifyingDescriptions, InstallDate, LastErrorCode, MaxDataSize, MaxQuiesceTime, PowerManagementCapabilities– (this is exposed as a method), PowerManagementSupported– (this is exposed as a method), PowerOnHours , ShortFramesReceived, SymbolErrors, TotalPowerOnHours	One for each Intel® PROSet supported installed adapter, phantom adapter and team	IANet_AdapterProtocolImplementation, IANet_AdapterDevice, IANet_AdapterConfiguration, IANet_TeamedMemberAdapter, IANet_NetworkVirtualAdapter, IANet_Device802dot1QVLANServiceImplementation, IANet_DeviceBootServiceImplementation
IANet_EthernetPCIDevice	N	N	None	None	AdditionalAvailability, Capabilities, CapabilityDescriptions, Caption, Description, DeviceSelectTiming, ErrorCleared, ErrorDescription, IdentifyingDescription, InstallDate, LastErrorCode, MaxNumberController, MaxQuiesceTime, Name, OtherIdentifyingInfo, PowerManagementCapabilities, PowerManagementSupported, PowerOnHours, ProtocolDescription, ProtocolSupported, SelfTestEnabled, Status, StatusInfo, TimeOfLastReset, TotalPowerOnHours	One instance for each PCI card that is an Intel PROSet supported Ethernet adapter	IANet_AdapterDevice
IANet_IPProtocolEndpoint	N	N	None	None	Caption, Description, InstallDate, NameFormat, OtherTypeInformation, Status	One instance for each binding of the IP Protocol stack to an Intel supported endpoint	IANet_AdapterProtocolImplementation, IANet_VLANProtocolDependency
IANet_NetService	N	N	GetSessionHandle, Apply, ReleaseSessionHandle, Cancel	None	Caption, Description, Install Date, Started, Start Mode, Status	Exactly one.	None
IANet_NetworkVirtualAdapter	N	N	None	None	None	One instance for each team.	This class associates IANet_TeamOfAdapters with an IANet_EthernetAdapter.
IANet_PCIDevice	N	N	None	None	AdditionalAvailability, Capabilities, CapabilityDescriptions, Caption, DeviceSelectTiming, ErrorCleared, ErrorDescription, IdentifyingDescription, InstallDate, LastErrorCode, MaxNumberController, MaxQuiesceTime, Name, OtherIdentifyingInfo, PowerManagementCapabilities, PowerManagementSupported, PowerOnHours, ProtocolDescription, ProtocolSupported, SelfTestEnabled, TimeOfLastReset, TotalPowerOnHours	One instance for each PCI card that is a network device in the system	

IANet_Setting	N	N	None	None	SettingID	This is an abstract class.	IANet_SettingContext
IANet_SettingContext	N	N	None	None	None	One instance for each setting	This class associates IANet_Setting with an IANet_Configuration.
IANet_SettingInt	N	N	None	CurrentValue	SettingID	One instance for each integer setting	IANet_SettingContext
IANet_SettingMultiSelection	N	N	None	CurrentValue	SettingID	One instance for each multi-selection setting	IANet_SettingContext
IANet_SettingSlider	N	N	None	CurrentValue	SettingID	One instance for each slider setting	IANet_SettingContext
IANet_SettingString	N	N	None	CurrentValue	SettingID	One instance for each string setting	IANet_SettingContext
IANet_TeamedMemberAdapter	Y	Y	None	AdapterFunction	PrimaryAdapter, ScopeOfBalancing	One instance for every adapter that is in a team	This class associates IANet_TeamOfAdapters with an IANet_EthernetAdapter.
IANet_TeamOfAdapters	Y	Y	TestSwitchConfiguration	TeamingMode	Install Date, Status	One instance for each team	IANet_NetworkVirtualAdapter, IANet_TeamedMemberAdapter
IANet_VLAN	N	Y	None	VLANNumber, Caption	Description, Install Date, StartMode, Status	One instance for each VLAN	IANet_VLANFor
IANet_VLANConfiguration	N	N	None	None	None	One instance for each VLAN	This class associates IANet_VLAN with IANet_Configuration
IANet_VLANFor	N	N	None	None	None	One instance for each VLAN	This class associates IANet_VLAN with IANet_802dot1QVLANService.
IANet_VLANProtocolDependency	N	N	None	None	None	One instance for each VLAN.	This class associates IANet_VLAN with IANet_IPProtocolEndpoint.

Back to Contents Page   Back to Top

Intel Software License Agreement (Final, License): Intel® PRO Network Adapters WMI and CDM Providers User Guide

you retain no copies of the Software.

LIMITED MEDIA WARRANTY. If the Software has been delivered by Intel on physical media, Intel warrants the media to be free from material physical defects for a period of ninety days after delivery by Intel. If such a defect is found, return the media to Intel for replacement or alternate delivery of the Software as Intel may select.

EXCLUSION OF OTHER WARRANTIES. EXCEPT AS PROVIDED ABOVE, THE SOFTWARE IS PROVIDED "AS IS" WITHOUT ANY EXPRESS OR IMPLIED WARRANTY OF ANY KIND INCLUDING WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR A PARTICULAR PURPOSE. Intel does not warrant or assume responsibility for the accuracy or completeness of any information, text, graphics, links or other items contained within the Software.

LIMITATION OF LIABILITY. IN NO EVENT SHALL INTEL OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, LOST PROFITS, BUSINESS INTERRUPTION, OR LOST INFORMATION) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF INTEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME JURISDICTIONS PROHIBIT EXCLUSION OR LIMITATION OF LIABILITY FOR IMPLIED WARRANTIES OR CONSEQUENTIAL OR INCIDENTAL DAMAGES, SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU. YOU MAY ALSO HAVE OTHER LEGAL RIGHTS THAT VARY FROM JURISDICTION TO JURISDICTION.

TERMINATION OF THIS AGREEMENT. Intel may terminate this Agreement at any time if you violate its terms. Upon termination, you will immediately destroy the Software or return all copies of the Software to Intel.

APPLICABLE LAWS. Claims arising under this Agreement shall be governed by the laws of California, excluding its principles of conflict of laws and the United Nations Convention on Contracts for the Sale of Goods. You may not export the Software in violation of applicable export laws and regulations. Intel is not obligated under any other agreements unless they are in writing and signed by an authorized representative of Intel.

GOVERNMENT RESTRICTED RIGHTS. The Software is provided with "RESTRICTED RIGHTS." Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor. Use of the Software by the Government constitutes acknowledgment of Intel's proprietary rights therein. Contractor or Manufacturer is Intel.

Please read all restrictions and disclaimers.

Back to Contents Page   Back to Top

Support: Intel® PRO Network Adapters WMI and CDM Providers User Guide

Web and Internet Sites

http://www.dell.com

Customer Support Technicians

If the troubleshooting procedures in this document do not resolve the problem, please contact Dell Computer Corporation for technical assistance (refer to the "Getting Help" section in your system documentation).

Before you call...

You need to be at your computer with your software running and the product documentation at hand.

The technician may ask for the following:

- Your address and telephone number
- The name and model number of the product you are calling about
- The serial number and service tag of the product
- The names and version numbers of the software you are using to operate the product
- The name and version number of the operating system you are using
- The computer type (manufacturer and model number)
- Expansion boards or add-in cards in your computer
- The amount of memory in your computer

Please read all restrictions and disclaimers.

Back to Contents Page