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

Legal Notices and Disclaimers

Table of Contents

1 Introduction

1.1 Scope

Network Configuration Services version 2 (NCS2) is an easy to use solution for deploying and managing all Intel end-station networking technologies using industry standard methods. This document describes the external view of the Intel® PRO Network Connections WMI[1] Provider (referred to throughout this document as "NCS2 WMI Provider"). The NCS2 WMI Provider is a network configuration block of NCS2.

The NCS2 WMI Provider is a set of software components that implements the WMI network classes. These classes are based on the Distributed Management Task Force (DMTF) CIM Schema version 2.6.

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 for Windows* Device Manager uses classes to configure a system's network. Readers should be familiar with WMI APIs and the WMI SDK (available from http://www.microsoft.com/).

1.2 Related Documents

- CIM schema version 2.0, 2.2 published by Distributed Management Task Force (DMTF), http://www.dmtf.org.
- Microsoft* Windows Management Instrumentation (and other manageability information) http://www.microsoft.com/hwdev/WMI/.
- Web-based Enterprise Management (WBEM) initiative by DMTF http://www.dmtf.org/wbem/index.html.
- WMI (Microsoft CIM implementation) SDK http://msdn.microsoft.com/code/sample.asp?url=/msdn-files/027/001/566/msdncompositedoc.xml

---

[1] WMI stands for Windows Management Instrumentation

2 WMI

Web-based Enterprise Management (WBEM) is a Distributed 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)mof_8opx.htm 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 two main components: the Core and the SDK.

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 WMI events. It also provides documentation useful in developing applications that will use WMI. The SDK is installed as part of the Microsoft Platform SDK installation process.

The SDK is supported on Microsoft Windows NT4* SP4 or SP5, Windows 2000, Windows Me, Windows XP and Microsoft Windows Server* 2003.

The WMI architecture consists of the following components:

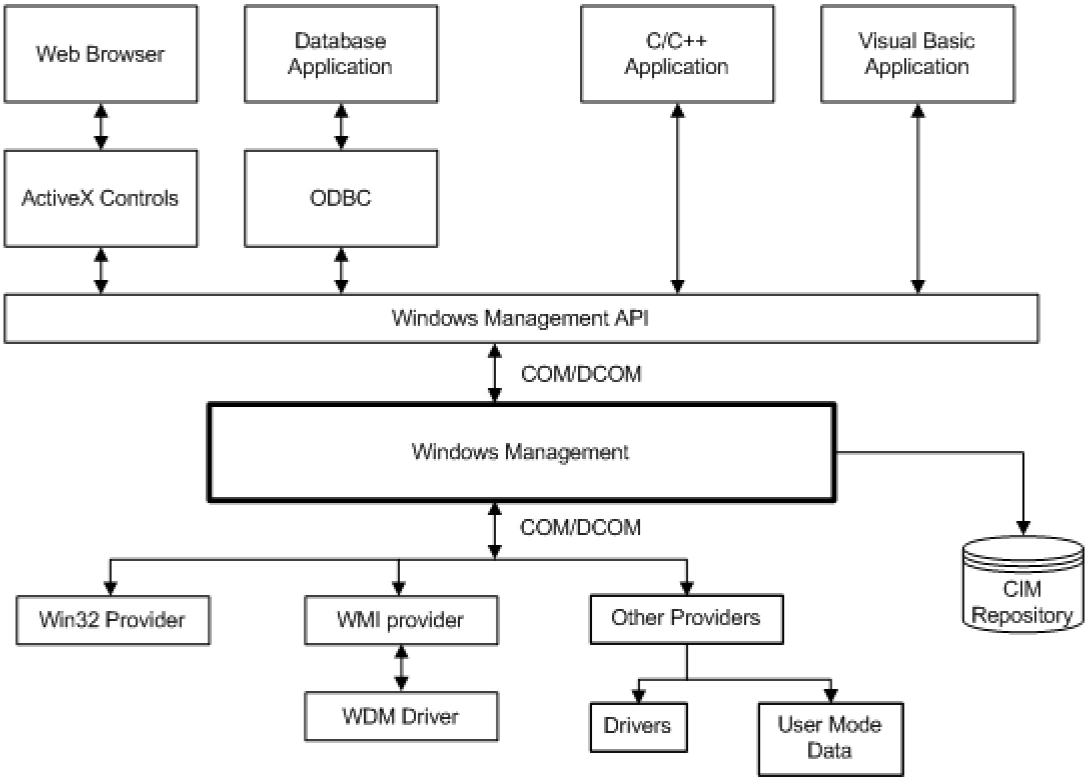
- Management applications
- Managed objects
- Providers
- Management infrastructure (consisting of the Windows Management and Windows Management repository)
- Windows Management API (which uses COM/DCOM to enable providers and management applications to communicate with the Windows Management infrastructure.

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 such services as event notification and query processing.

The following diagram shows the interrelationship of these components:



2.1 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, however the Microsoft implementation predominates the specification. 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 - an extension of the core model.
- Classes representing managed objects that are technology-specific additions to the common model. These classes typically apply to specific platforms such as UNIX or the Microsoft Win32 environment.

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.

3 Main Features

The main features exposed by the NCS2 WMI Provider are divided into the following categories:

3.1 Adapter

- 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.
- Uninstall an adapter.
- Update and change an adapters Boot Agent and associated settings.

3.2 Team

- 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.

3.3 VLAN

- 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 VLAN.

3.4 Diagnostics

- Enumerate all supported diagnostic tests/settings/result for all physical Intel adapters.
- Run/Stop diagnostic test on a physical Intel adapter.

4 Installed Files

4.1 Executables

There are six separate dynamic linking libraries and one executable for the Provider:

Filename	Description
Ncs2Prov.exe	The instance and method provider. Implements the Ethernet Adapter Schema, the Teaming Schema, the Setting Schema, the VLAN Schema and the Diagnostic Schema.
Ncs2Core.dll	Implements the Ethernet Adapter Schema.
Ncs2Diag.dll	Implements the Diagnostics Schema.
Ncs2Boot.dll	Implements the Boot Agent Schema.
Ncs2Team.dll	Implements the Team Schema.
Ncs2VLAN.dll	Implements the VLAN Schema.
Ncs2InstUtility.dll	Implements the common utility functions.

4.2 MOF Files

There are separate MOF files for language neutral and language specific data. For more information on localization, refer to section 7.

4.3 MOF Files for IntelNCS2 Namespace

Filename	Description
ICmLn.mof	CIM base classes on which the NCS2 classes depend.
ICmEnu.mfl	US English version of the CIM base classes.
ICoreLn.mof	Classes for the IEEE 802.3 adapters.
ICoreEnu.mfl	US English textual amendments to the adapter classes.
IBootLn.mof	Classes for the IEEE 802.3 boot service.
IBootEnu.mfl	US English textual amendments to the 802.3 boot service classes.
IDiagLn.mof	Classes for the CDM (Common Diagnostic Model).
IDiagEnu.mfl	US English textual amendments to the CDM classes.
ITeamLn.mof	Classes for the IEEE 802.3 teams.
ITeamEnu.mfl	US English textual amendments to the team classes.
IVLANLn.mof	Classes for the IEEE 802.3 VLANs.
IVLANEnu.mfl	US English textual amendments to the VLAN classes.

5 Security

The NCS2 WMI Provider uses client impersonation to manage the security. Every call into the Provider will be made in the client’s own security context. This context is passed down to the lower layers. An operation may fail if the user does not have suitable administrative rights on the target machine.

6 Namespace and Context

6.1 Namespace

The CIM classes reside in a namespace. The standard Microsoft namespace is called “root/cimv2” and is based on CIM v2.2.

The NCS2 WMI Provider is based on CIM v2.6. Because of this, and because of differences used in the keys of the objects, the NCS2 WMI Provider classes reside in a separate namespace called “root/IntelNCS2”. Intel PROSet for Windows Device Manager uses the “root/IntelNCS2” namespace.

6.2 WBEM Context

Context objects are used to provide additional information to the NCS2 WMI Provider that cannot be passed as a parameter to a WMI API method. 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 NCS2 WMI Provider. ClientSetId is only used in conjunction with specific functional areas of the Provider, whereas MachineName can be set for all IWbemServices calls.

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
ClientSetId	VT_BSTR	Identifies the application's copy of IANet network classes. The application cannot make any changes to the classes or their properties without first establishing a client handle. See the section on the IANet_NetService class to see how to establish and use a client handle. This qualifier is not required if the application is only going to read data from the classes. The client handle allows the NCS2 software to manage single access to the configuration.
MachineName	VT_BSTR	The name of the machine that is connecting to the Provider. This is required for logging.

7 Locales and Localization

7.1 Localized MOF files

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

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

7.2 Class Storage

The language-specific class definitions are stored in a child sub-namespace beneath the namespace that contains a language-neutral basic class definition. For example, for the NCS2 WMI Provider, a child namespace ms_409 will exist beneath the root/intelncs2 namespace for the English locale. Similarly, there exists a child sub-namespace for each supported language beneath the root/intelncs2 namespace.

7.3 Runtime Support

To retrieve localized data, a WMI application can specify the locale using strLocale parameter in SWbemLocator.ConnectServer and IWbemLocator::ConnectServer calls. If the locale is not specified, the default locale for that system will be used. (e.g. MS_409 for US English). This locale is used to select the correct namespace when adding in the English strings.

In addition, IWbemServices::GetObject, SWbemServices.GetObject, IWbemServices:: ExecQuery, and SWbemServices.ExecQuery must specify the WBEM_FLAG_USE_AMENDED_QUALIFIERS flag to request localized data stored in the localized namespace, along with the basic definition. This is required in all functions that produce displayable values using value maps or display descriptions or other amended qualifiers from the MOF files.

8 Error Reporting

8.1 IANet_ExtendedStatus

This section details how to handle errors generated by NCS2 WMI Provider.

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

8.2 Getting the Error Object

8.2.1 Synchronous Calls

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

8.2.2 Asynchronous Calls

The IWbemClassObject is passed back as the last item in the last SetStatus() call.

After you get the error object instance, you can check the __Class property to determine the origin of the error. WMI creates an instance of __ExtendedStatus, and the NCS2 WMI Provider creates an instance of IANet_ExtendedStatus for errors relating to IANet_ classes and NCS2 WMI Provider.

IANet_ExtendedStatus is derived from __ExtendedStatus and contains the following attributes:

8.3 Error Object Qualifiers

Context Qualifier	Description
Description	Description of the error tailored to the current locale.
File	Code file where the error was generated.
Line	Line number in 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.
ClientSetHandle	Client Set 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).

8.4 Error Codes

For all error codes, the NCS2 WMI Providers gives a description customized to the locale. Below is a list of possible error codes that the Provider may return. Error codes are in the form of HRESULT with severity set to one (1) and facility set to ITF. An application may use these codes as a basis for a recovery action.

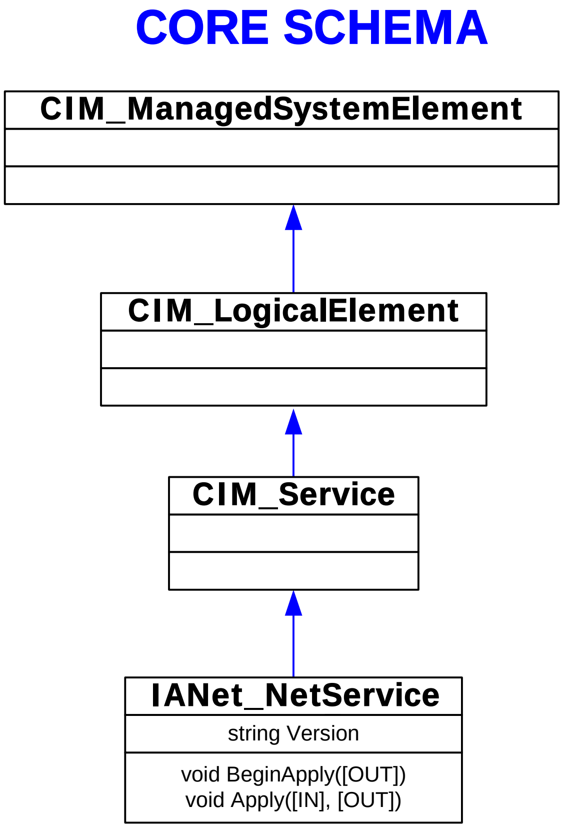
0x80040901	"WMI: Put property failed"
0x80040902	"WMI: No class object"
0x80040903	"WMI: Failed to create class"
0x80040904	"WMI: Failed to spawn instance of class"

0x80040905	"WMI: Failed to create safe array"
0x80040906	"WMI: Failed to put safe array"
0x80040907	"WMI: Failed to return object to WMI"
0x80040908	"WMI: Get property failed"
0x80040909	"WMI: Unexpected type while getting property"
0x8004090A	"WMI: Class not implemented by this provider"
0x8004090B	"WMI: Unable to parse WQL statement"
0x8004090C	"WMI: Provider only supports WQL"
0x8004090D	"WMI: Parameter in context has the wrong type"
0x8004090E	"WMI: Error formatting debug log"
0x8004090F	"WMI: bad object path"
0x80040910	"WMI: Failed to update setting"
0x80040911	"WMI:[Null parameter passed to method"
0x80040912	"Setting value too small"
0x80040913	"Setting value too big"
0x80040914	"Setting not in step"
0x80040915	"String setting is too long"
0x80040916	"Setting is not one of the allowed values"
0x80040917	"WMI: Qualifier not found"
0x80040918	"WMI: Qualifier set not found"
0x80040919	"WMI: Safe array access failed"
0x8004091A	"WMI: Unhandled exception"
0x8004091B	"WMI: Operation is not supported for this class"
0x8004091C	"WMI: Unexpected event class"
0x8004091D	"WMI: Bad event data"
0x8004091E	"WMI: Operation succeeded with warnings"
0x8004081F	"WMI: The NCS2 Service has been stopped"

9 The Core Schema

The Core Schema consists of the IANet_NetService class.

9.1 Core Schema Diagram



9.2 IANet_NetService

9.2.1 Purpose

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

9.2.2 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.

9.2.3 Creating Instances

The user is not able to create instances of IANet_NetService.

9.2.4 Removing Instances

The user is not able to delete the instance of IANet_NetService.

9.2.5 Local Properties

This class implements the following local attribute:

Property	**Description**
Version	Contains the current version of the Core Provider.

9.2.6 Modifiable Properties

There are no user modifiable properties of this class.

9.2.7 Unsupported Properties

The following properties are not required for Intel PROSet and are, therefore, not supported:

Caption, Description, Install Date, Started, Start Mode, Status

Methods

The following methods are implemented in IANet_NetService:

void BeginApply(([OUT] uint32 ClientSetHandle)	Used to set a Client session handle , which should be placed in the context object in the ClientSetId qualifier.
void Apply([IN] uint32 ClientSetHandle, [OUT] uint32 FollowupAction);	Applies changes made with a particular session handle and releases a session handle after it has been used. The uint32 argument returned is used by the Provider to tell the application the server must be rebooted before the changes will take effect. (This can be accomplished by calling the Reboot method on the class Win32_OperatingSystem). Values: 1 – system reboot required 0 – no reboot required

9.2.8 Use Cases

A session handle is required to change the configuration. The session handle allows the NCS2 software to manage single access to the configuration, thereby preventing multiple changes to the configuration.

9.2.8.1 Getting a Client Handle

The client must get the object path of the single instance of IANet_NetService before accessing the client 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 client handle. Use the BeginApply method to start a fresh client change configuration. 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.

9.2.8.2 Using a Client Handle in the IWbemContext Object

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

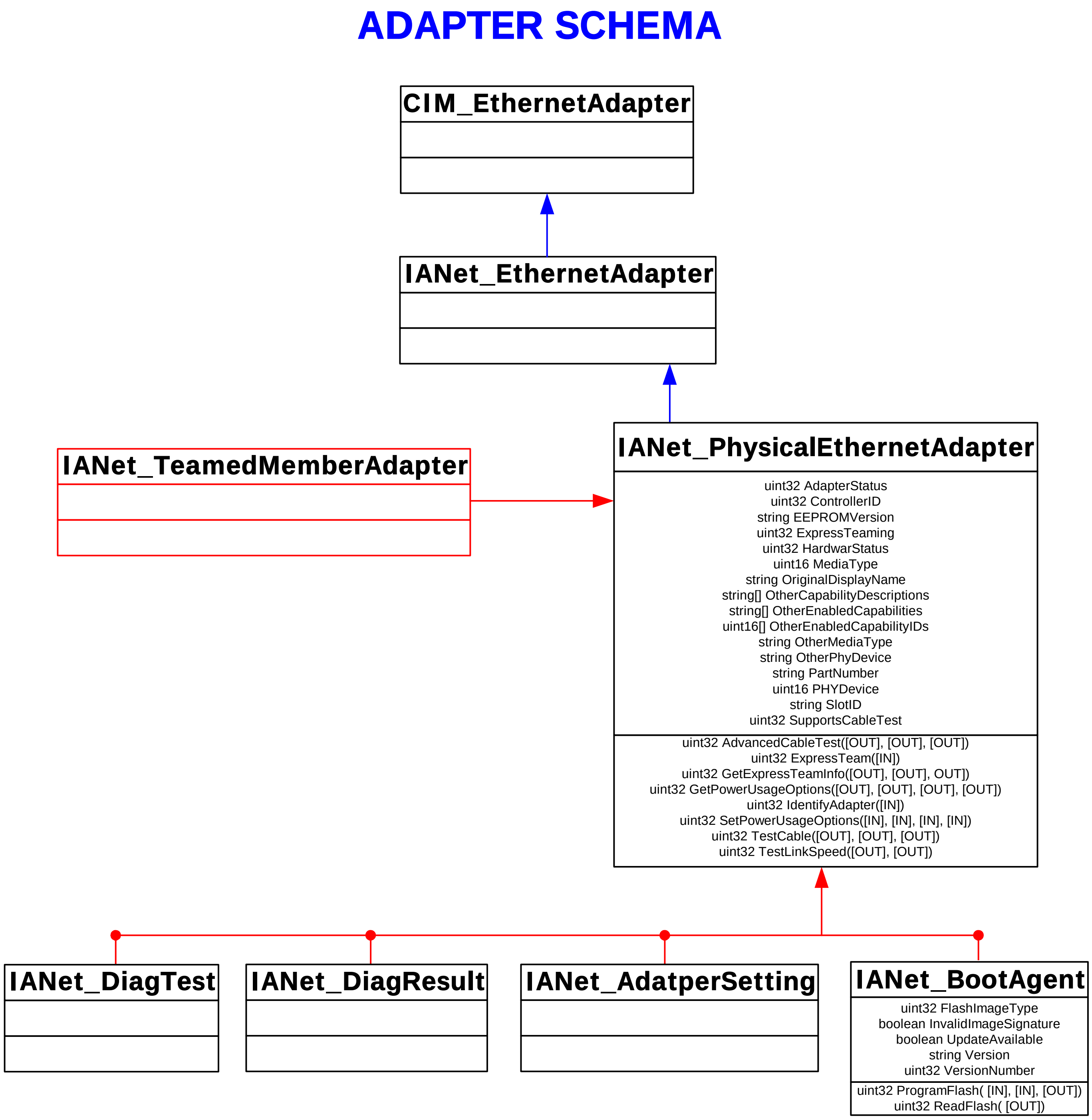
9.2.8.3 Finishing with a Client Handle

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).

10 Ethernet Adapter Schema

The adapter schema is used to model the various configurable Intel adapters. This schema is based on the CIM v2.6 schema.

10.1 Adapter Schema Diagram


ADAPTER SCHEMA
CIM_EthernetAdapter
IANet_EthernetAdapter
IANet_TeamedMemberAdapter
IANet_PhysicalEthernetAdapter
uint32 AdapterStatus
uint32 ControllerID
string EEPROMVersion
uint32 ExpressTeaming
uint32 HardwarStatus
uint16 MediaType
string OriginalDisplayName
string[] OtherCapabilityDescriptions
string[] OtherEnabledCapabilities
uint16[] OtherEnabledCapabilityIDs
string OtherMediaType
string OtherPhyDevice
string PartNumber
uint16 PHYDevice
string SlotID
uint32 SupportsCableTest
uint32 AdvancedCableTest([OUT], [OUT], [OUT])
uint32 ExpressTeam([IN])
uint32 GetExpressTeamInfo([OUT], [OUT], OUT])
uint32 GetPowerUsageOptions([OUT], [OUT], [OUT], [OUT])
uint32 IdentifyAdapter([IN])
uint32 SetPowerUsageOptions([IN], [IN], [IN], [IN])
uint32 TestCable([OUT], [OUT], [OUT])
uint32 TestLinkSpeed([OUT], [OUT])
IANet_DiagTest
IANet_DiagResult
IANet_AdatperSetting
IANet_BootAgent
uint32 FlashImageType
boolean InvalidImageSignature
boolean UpdateAvailable
string Version
uint32 VersionNumber
uint32 ProgramFlash([IN], [IN], [OUT])
uint32 ReadFlash([OUT])

10.2 IANet_PhysicalEthernetAdapter

10.2.1 Purpose

IANet_PhysicalEthernetAdapter defines the capabilities and status of all the installed Intel adapters. The class is derived from the an abstract class IANet_EthernetAdapter. IANet_EthernetAdapter is derived from 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.

10.2.2 Instances

Instances of this class will exist for supported and installed Intel adapters.

10.2.3 Creating Instances

The user cannot create instances of IANet_PhysicalEthernetAdapter.

10.2.4 Removing Instances

Deleting an instance of IANet_PhysicalEthernetAdapter will uninstall physical adapters. A client handle is required for this operation.

10.2.5 Modifying Properties

There are no user-modifiable properties for this class.

10.2.6 Local Properties

This class implements the following local properties:

Property	**Description**
AdapterStatus	Specifies the current status of the adapter.
ControllerID	The unique identifier for this adapter
EEPROMVersion	Contains device EEPROM version.
ExpressTeaming	Indicates if Express Teaming is enabled.
HardwareStatus	Specifies current status of device.
MediaType	The media which interfaces to this phy.
OriginalDisplayName	Original name value will show if Express Teaming is enabled.
OtherCapabilityDescriptions	An array of descriptions of OtherEnabledCapabilities of the adapter.
OtherEnabledCapabilities	Used to state Other Capabilities of the adapter.
OtherEnabledCapabilityIDs	An unmapped value list of IDs of Other Capabilities.
OtherMediaType	Shows media type if MediaType value is ‘Other’.
OtherPhyDevice	Shows name of device if PHYDevice value is ‘Other’.
PartNumber	The PBA manufacturing part number.
PHYDevice	PHY used on this adapter.
SlotID	Value that correlates the physical attributes to the logical access method.
SupportsCableTest	Value to indicate support for advanced cable test.

10.2.7 Unsupported Properties

The following properties are not required for Intel PROSet and are, therefore, not supported:

AlignmentErrors, AutoSense, CarrierSenseErrors, DeferredTransmissions, DriverComments, DriverDescription, DriverFileSize, DriverFileVersion, DriverLegalCopyright, DriverPath, DriverProductVersion, EnabledCapabilities ErrorCleared, ErrorDescription, ExcessiveCollisions, FCSErrors, FlowControlPacketsReceived, FlowControlPacketsTransmitted, FrameTooLongs, FullDuplex, GeneralReceiveErrors, GeneralTransmitErrors, IdentifyingDescriptions, InstallDate, InternalMACReceiveErrors, InternalMACTransmitErrors, LastErrorCode, LateCollisions, MaxDataSize, MaxQuiesceTime, MultipleCollisionFrames, NoBufferReceiveErrors, NoBufferXmitErrors, OctetsReceived, OctetsTransmitted, OtherIdentifyingInfo, PacketTaggingStatus, PowerManagementCapabilities (this is exposed as a method), PowerManagementSupported (this is exposed as a method), PowerOnHours, ShortFramesReceived, SingleCollisionFrames, SymbolErrors, SQETestErrors, TCOFramesReceived, TCOFramesTransmitted, TotalHostErrors, TotalPacketsReceived, TotalPacketsTransmitted, TotalPowerOnHours, TotalWireErrors, TroubleShootingCauses, TroubleShootingProblems, TroubleShootingSeverityLevels, TroubleShootingSolutions

10.2.8 Methods

This class implements the following methods:

uint32 AdvancedCableTest([OUT] boolean, [OUT] array[string], [OUT] array[string]);	Performs a set of advanced cable tests on supported adapters.
uint32 ExpressTeam([IN Boolean]);	Creates/Removes the express team.
uint32 GetExpressTeamInfo([OUT] boolean, [OUT] uint16, [OUT] uint32);	Gets the express team information.
uint32 GetPowerUsageOptions([OUT] uint32, [OUT] uint32, [OUT] uint32, [OUT] uint32);	Detects any optional power usage settings (e.g., power usage for standby, battery operation, etc.).
uint32 IdentifyAdapter([IN uint16]);	Identifies adapter by flashing the light on the adapter for a few seconds. This method will only work for physical adapters.
uint32 SetPowerUsageOptions ([IN] uint32, [IN] uint32, [IN] uint32, [IN] uint32);	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.
uint32 TestCable ([OUT] array[string], [OUT] array[string], [OUT] array[string]);	Analyzes the network cable connected to the adapter and reports aspects of the cable such as length, quality and signal quality.
unit32 TestLinkSpeed ([OUT] uint32, [OUT] string);	Determines whether the adapter is running at full speed.

10.2.9 Unsupported Methods

The following methods are not required for Intel PROSet and are, therefore, not supported:

EnableDevice, OnlineDevice, QuiesceDevice, Reset, RestoreProperties, SaveProperties, SetPowerState.

10.2.10 Associations

- IANet_DiagTestForMSE is used to associate an IANet_DiagTest with an IANet_PhysicalEthernetAdapter.
- IANet_DiagResultForMSE is used to associate an IANet_DiagResult with an IANet_PhysicalEthernetAdapter.
- IANet_DeviceBootServiceImplementation is used to associate an IANet_BootAgent with an IANet_PhysicalEthernetAdapter.
- IANet_AdapterToSettingAssoc is used to associate an IANet_AdapterSetting with an IANet_PhysicalEthernetAdapter.
- IANet_TeamedMemberAdapter is used to associate a IANet_TeamOfAdapters with an IANet_PhysicalEthernetAdapter.

10.3 IANet_BootAgent

10.3.1 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.

10.3.2 Instances

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

10.3.3 Creating Instances

The user cannot create instances of IANet_BootAgent. An instance exists only if the adapter supports boot agent functionality.

10.3.4 Removing Instances

The user cannot remove instances of IANet_BootAgent.

10.3.5 Modifying Properties

There are no user-modifiable properties of this class.

10.3.6 Associations

- IANet_DeviceBootServiceImplementation is used to associate an IANet_PhysicalEthernetAdapter with an IANet_BootAgent, if the adapter supports it.
- IANet_BootAgentToBootAgentSettingAssoc is used to associate an IANet_BootAgentSetting with an IANet_BootAgent.

10.3.7 Local Properties

The following read only properties are required by Intel PROSet:

Property	**Description**
FlashImageType	The boot agent flash image type.
InvalidImageSignature	Boolean value denoting corrupted flash image.
UpdateAvailable	Indicates if install or upgrade to boot agent software is available.
Version	String value of boot agent version.
VersionNumber	Unsigned integer value of boot agent version.

10.3.8 Unsupported Properties

The following properties are not required by Intel PROSet and are, therefore, not supported:

Caption, Description, InstallDate, Started, StartMode, Status.

10.3.9 Methods

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

uint32 ProgramFlash([IN] uint32, [IN] array[uint8], [OUT] uint32,);	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] array[uint8]);	This method reads the Flash ROM on the NIC.

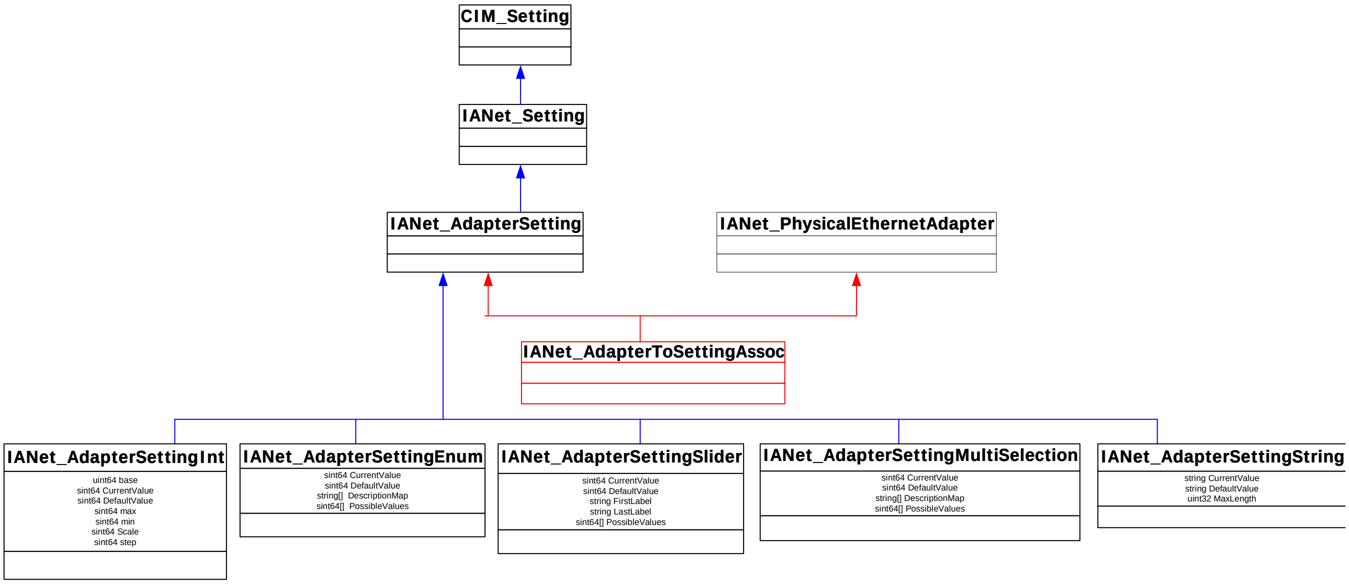
10.3.10 Unsupported Methods

StartService, StopService

11 Adapter Setting Schema

11.1 Adapter Setting Schema Diagram

ADAPTER SETTING SCHEMA



11.2 IANet_AdapterToSettingAssoc

11.2.1 Purpose

This class is used to group a collection of IANet_AdapterSetting instances.

11.2.2 Instances

Each adapter can have several associated IANet_AdapterToSettingAssoc instances.

11.2.3 Creating instances

The user cannot create instances of IANet_ AdapterToSettingAssoc.

11.2.4 Removing instances

The user cannot remove instances of IANet_ AdapterToSettingAssoc.

11.2.5 Modifying properties

There are no user-modifiable properties for this class.

11.2.6 Associations

An IANet_AdapterToSettingAssoc instance will exist to associate each IANet_PhysicalEthernetAdapter with its IANet_AdapterSetting.

11.2.7 Methods

There are no supported methods for this class.

11.2.8 Unsupported Properties

None

11.3 IANet_AdapterSetting

11.3.1 Purpose

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

11.3.2 Instances

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

There are several sub-classes for IANet_AdapterSetting. The sub-classes correspond to the different types and ranges of values that settings can take. Each sub-class corresponds to a different style of GUI that may be used to display or change the settings.

11.3.3 Creating instances

The user cannot create instances of IANet_AdapterSetting.

11.3.4 Removing instances

The user cannot remove instances of IANet_AdapterSetting.

11.3.5 Modifying properties

This abstract class has no modifiable properties, however, the child classes have modifiable properties (see sub-classes listed in this section).

11.3.6 Associations

Each IANet_AdapterSetting instance is associated with an IANet_PhysicalEthernetAdapter instance using an instance of IANet_ AdapterToSettingAssoc.

11.3.7 Methods

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

11.3.8 Unsupported Properties

SettingID and RequiresSession are not used.

11.4 IANet_AdapterSettingInt

11.4.1 Purpose

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

11.4.2 Instances

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

11.4.3 Creating Instances

The user cannot create instances of this class.

11.4.4 Removing Instances

The user cannot remove instances of this class.

11.4.5 Modifying Properties

The “CurrentValue” attribute is the only modifiable property of this class. The user can modify this property by using IWbemClassObject::Put() to change the value, then call “IWbemServices::PutInstance()” to update the setting. The Provider will check that:

CurrentValue <= max
CurrentValue >= min
(CurrentValue – min) is a multiple of step

Where max, min, CurrentValue and step are all properties of IANet_SettingInt.

11.4.6 Local Properties

This class implements the following local properties:

Property	Description
base	Root from which the integer value may take values (example; decimal = base 10).
CurrentValue	The actual value of the integer setting.
DefaultValue	The initial value of the integer setting.
max	The maximum value that the setting can have.
min	The minimum value that the setting can have.
Scale	Unit to measure value of setting.
step	The granularity of the integer value.

11.4.7 Associations

Each IANet_AdapterSettingInt instance is associated with an IANet_PhysicalEthernetAdapter instance using an instance of IANet_AdapterToSettingAssoc.

11.4.8 Unsupported Properties

SettingID and RequiresSession are not used.

11.4.9 Methods

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

11.5 IANet_AdapterSettingEnum

11.5.1 Purpose

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

11.5.2 Instances

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

11.5.3 Creating Instances

The user cannot create instances of this class.

11.5.4 Removing Instances

The user cannot remove instances of this class.

11.5.5 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 Provider will check that:

$$\text{CurrentValue} \in \text{PossibleValues[]}$$

11.5.6 Associations

Each IANet_AdapterSettingEnum instance is associated with an IANet_PhysicalEthernetAdapter instance using an instance of IANet_AdapterToSettingAssoc.

11.5.7 Local Properties

This class implements the following local properties:

Property	**Description**
CurrentValue	The actual value of the setting.
DefaultValue	The initial value of the setting.
DescriptionMap	An array of descriptions mapped to the PossibleValues property.
PossibleValues	The values that correspond to the DescriptionMap.

11.5.8 Unsupported Properties

SettingID and RequiresSession are not used.

11.5.9 Methods

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

11.6 IANet_AdapterSettingSlider

11.6.1 Purpose

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

11.6.2 Instances

An instance of this class exists for each setting that will be displayed as a slider.

11.6.3 Creating Instances

The user cannot create instances of this class.

11.6.4 Removing Instances

The user cannot remove instances of this class.

11.6.5 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 Provider will check that:

$$\text{CurrentValue} \in \text{PossibleValues[]}$$

11.6.6 Local Properties

This class implements the following local properties:

Property	Description
CurrentValue	The actual value of the setting.
DefaultValue	The initial value of the setting.
FirstLabel	The label that should be displayed to the left of the slider.
LastLabel	The label that should be displayed to the right of the slider.
PossibleValues	The range of values which should be displayed with the first value on the left and last value on the right side of the slider.

11.6.7 Associations

Each IANet_AdapterSettingSlider instance is associated with an IANet_PhysicalEthernetAdapter instance using an instance of IANet_AdapterToSettingAssoc.

11.6.8 Unsupported Properties

SettingID and RequiresSession are not used.

11.6.9 Methods

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

11.7 IANet_AdapterSettingMultiSelection

11.7.1 Purpose

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

11.7.2 Instances

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

11.7.3 Creating Instances

The user cannot create instances of this class.

11.7.4 Removing Instances

The user cannot remove instances of this class.

11.7.5 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 Provider will check that:

$$\text{CurrentValue} \in \text{PossibleValues[]}$$

11.7.6 Local Properties

This class implements the following properties:

Property	Description
CurrentValue	The actual value of the setting.
DefaultValue	The initial value of the setting.
DescriptionMap	An array of descriptions mapped to the PossibleValues property.
PossibleValues	The values that correspond to the DescriptionMap.

11.7.7 Associations

Each IANet_AdapterSettingMultiSelection instance is associated with an IANet_PhysicalEthernetAdapter instance using an instance of IANet_AdapterToSettingAssoc.

11.7.8 Unsupported Properties

SettingID and RequiresSession are not used.

11.7.9 Methods

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

11.8 IANet_AdapterSettingString

11.8.1 Purpose

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

11.8.2 Instances

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

11.8.3 Creating Instances

The user cannot create instances of this class.

11.8.4 Removing Instances

The user cannot remove instances of this class.

11.8.5 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.

11.8.6 Local Properties

This class implements the following local properties:

Property	Description
CurrentValue	The actual value of the setting.
DefaultValue	The initial value of the setting.
MaxLength	The maximum string length allowed.

11.8.7 Associations

Each IANet_AdapterSettingString instance is associated with an IANet_PhysicalEthernetAdapter instance using an instance of IANet_AdapterToSettingAssoc.

11.8.8 Unsupported Properties

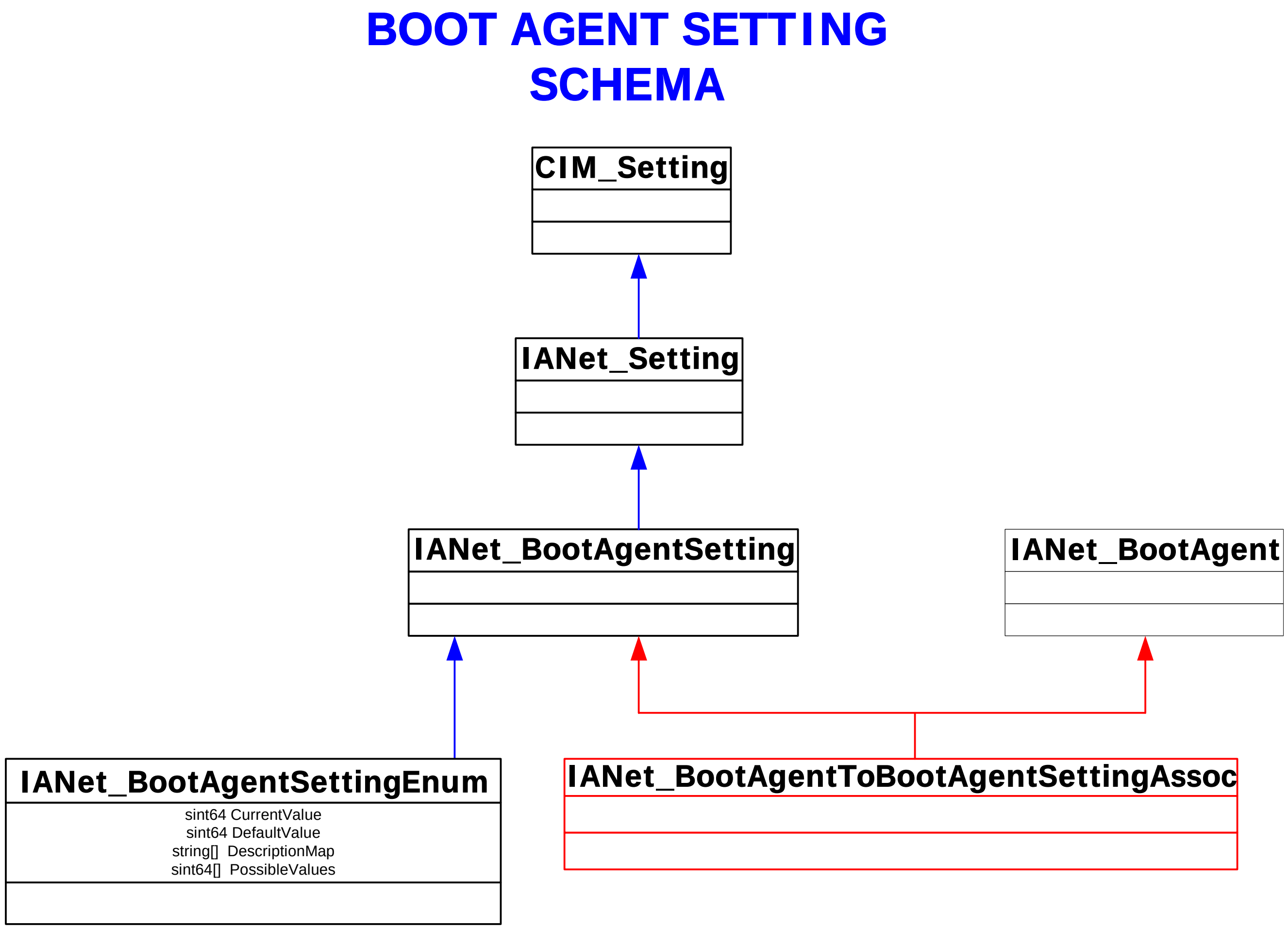
SettingID and RequiresSession are not used.

11.8.9 Methods

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

12 Boot Agent Setting Schema

12.1 Boot Agent Setting Schema Diagram



12.2 IANet_BootAgentToBootAgentSettingAssoc

12.2.1 Purpose

This class is used to group a collection of IANet_BootAgentSetting instances.

12.2.2 Instances

Each BootAgent can have several associated IANet_BootAgentToBootAgentSettingAssoc instances.

12.2.3 Creating Instances

The user cannot create instances of IANet_BootAgentToBootAgentSettingAssoc.

12.2.4 Removing Instances

The user cannot remove instances of IANet_ BootAgentToBootAgentSettingAssoc.

12.2.5 Modifying Properties

There are no user-modifiable properties for this class.

12.2.6 Associations

An IANet_BootAgentToBootAgentSettingAssoc instance will exist to associate each Boot Agent (IANet_BootAgent) with its setting.

12.2.7 Methods

There are no supported methods for this class.

12.2.8 Unsupported Properties

None

12.3 IANet_BootAgentSetting

12.3.1 Purpose

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

12.3.2 Instances

Instances of this class will exist for each setting on each Boot Agent.

There are several sub-classes for IANet_BootAgentSetting. 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.

12.3.3 Creating Instances

The user cannot create instances of IANet_BootAgentSetting.

12.3.4 Removing Instances

The user cannot remove instances of IANet_BootAgentSetting.

12.3.5 Modifying Properties

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

12.3.6 Associations

Each IANet_BootAgentSetting instance is associated with an IANet_BootAgent instance using an instance of IANet_BootAgentToBootAgentSettingAssoc.

12.3.7 Methods

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

12.3.8 Unsupported Properties

SettingID and RequiresSession are not used.

12.4 IANet_BootAgentSettingEnum

12.4.1 Purpose

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

12.4.2 Instances

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

12.4.3 Creating Instances

The user cannot create instances of this class.

12.4.4 Removing Instances

The user cannot remove instances of this class.

12.4.5 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 Provider will check that:

$$\text{CurrentValue} \in \text{PossibleValues[]}$$

12.4.6 Local Properties

This class implements the following local properties:

Property	Description
CurrentValue	The actual value of the setting.
DefaultValue	The initial value of the setting.
DescriptionMap	An array of descriptions mapped to the PossibleValues property.
PossibleValues	The values that correspond to the DescriptionMap.

12.4.7 Associations

Each IANet_BootAgentSettingEnum instance is associated with an IANet_BootAgent instance using an instance of IANet_BootAgentToBootAgentSettingAssoc.

12.4.8 Unsupported Properties

SettingID and RequiresSession are not used.

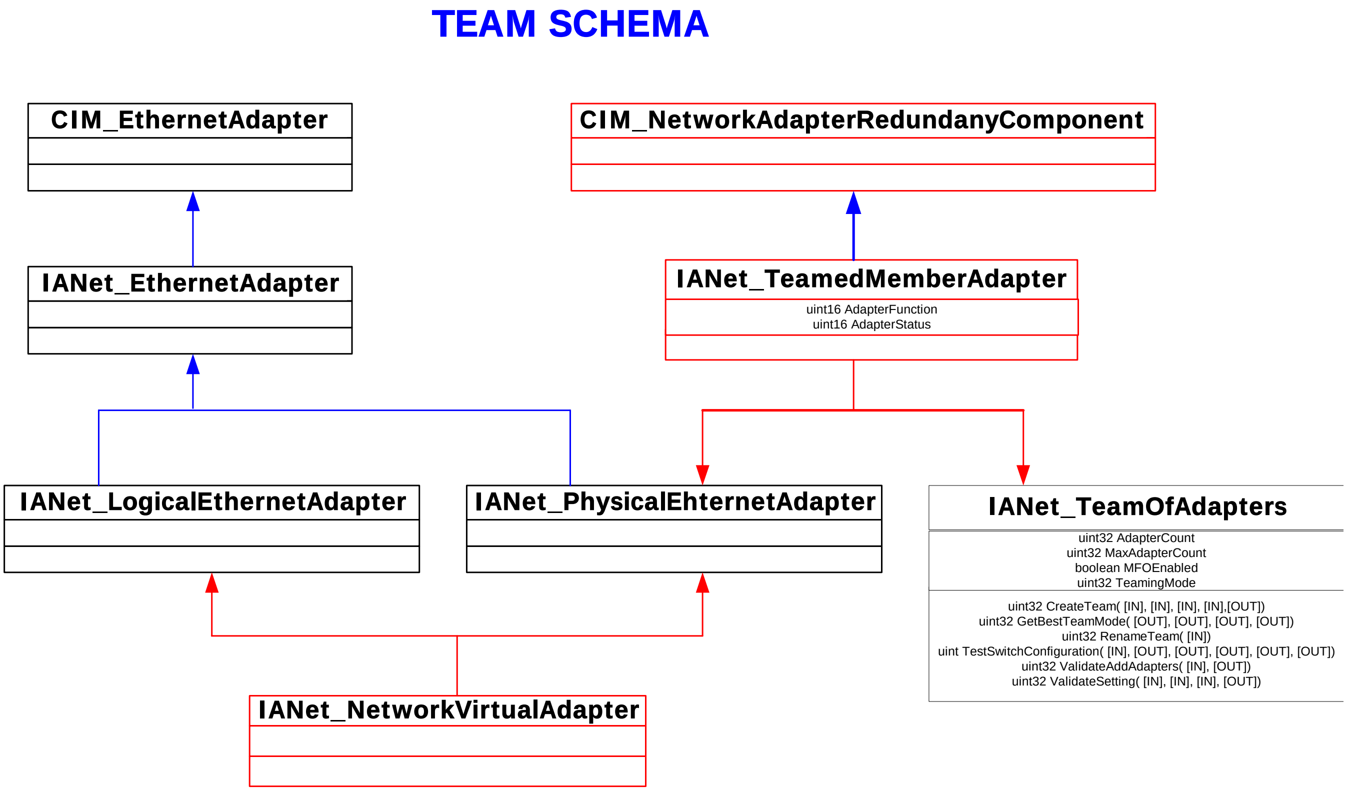
12.4.9 Methods

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

13 Team Schema

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

13.1 Team Schema Diagram


TEAM SCHEMA
CIM_EthernetAdapter
CIM_NetworkAdapterRedundanyComponent
IANet_EthernetAdapter
IANet_TeamedMemberAdapter
uint16 AdapterFunction
uint16 AdapterStatus
IANet_LogicalEthernetAdapter
IANet_PhysicalEhternetAdapter
IANet_TeamOfAdapters
uint32 AdapterCount
uint32 MaxAdapterCount
boolean MFOEnabled
uint32 TeamingMode
uint32 CreateTeam([IN], [IN], [IN], [IN],[OUT])
uint32 GetBestTeamMode([OUT], [OUT], [OUT], [OUT])
uint32 RenameTeam([IN])
uint TestSwitchConfiguration([IN], [OUT], [OUT], [OUT], [OUT], [OUT])
uint32 ValidateAddAdapters([IN], [OUT])
uint32 ValidateSetting([IN], [IN], [IN], [OUT])
IANet_NetworkVirtualAdapter

13.2 IANet_TeamOfAdapters

13.2.1 Purpose

This class implements the CIM_ExtraCapacityGroup 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.

13.2.2 Instances

There is an instance of this class for each Intel adapter team.

13.2.3 Creating Instances

To create an empty team, the user will create an instance of IANet_TeamOfAdapters. The user must set the correct “TeamingMode” before calling IWbemServices::PutInstance() to create the object in the Provider. The Provider will return a string containing the object path of the new object.

13.2.4 Removing Instances

Correspondingly, to remove a team the user should delete the instance of IANet_TeamOfAdapters. The Provider will delete the associations to the team members, and will also delete the virtual adapter and settings for the team.

13.2.5 Modifying Properties

Use Put() to change the value of the “TeamingMode” property, then call PutInstance() to update the team.

13.2.6 Local Properties

This class implements the following local properties:

Property	Description
AdapterCount	The number of adapters currently in the team.
MaxAdapterCount	The maximum number of adapters allowed in created team.
MFOEnabled	Represents the MFO enabled/disabled in the current team.
TeamingMode	The type of the current team.

13.2.7 Associations

Each adapter in a team is associated with the team’s instance of IANet_TeamOfAdapters using an instance of IANet_TeamedMemberAdapter.

The virtual adapter (IANet_LogicalEthernetAdapter) for the team is associated with this class using an instance of IANet_NetworkVirtualAdapter.

13.2.8 Methods

This class instance supports following methods:

Method	Description
uint32 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, then this test can verify that link partners connected to the adapter support Link Aggregation.

```
uint32 TestSwitchConfiguration(
   [IN, ValueMap {"0","1","2"}:Amended,
         Values {"Start", "Cancel",
                     "Results"}: Amended
    ] uint32 Cmd,
    [OUT, ValueMap
                    {"0","1","2"}:Amended,
              Values {"OK", "Error",
                         "Progress"}: Amended
   ] uint32 Status,
   [OUT] uint16 CauseMessageId[],
   [OUT] string strCause[],
   [OUT] uint16 SolutionMessageId[]
   [OUT]string strSolution[]
);
```

```
uint32 GetBestTeamMode(
   [OUT,
         ValueMap {"Passed", "Failed",
         "In Progress",
         "Unknown"}:Amended ,
         Values {"0", "1", "2", "3"}:
                    Amended]
       uint32 Status,
   [OUT,
         Units ("Percent"):Amended,
         MinValue (0), MaxValue (100)]
       uint8 PercentOfCoverage,
   [OUT, ValueMap {"0", "1", "2", "4",
                          "5"}:Amended ,
            Values {"AFT", "ALB", "SLA",
                      "IEEE 802.3ad",
                      "SFT"}:Amended]
       uint32 TeamingMode,
   [OUT]  uint16 ErrorMessageId
);
```

GetBestTeamMode selects the most appropriate teaming mode to use for teaming.

```
uint32 RenameTeam([IN] string TeamName);
```

RenameTeam changes the name of an existing Intel adapter Team in the system.

```
uint32 CreateTeam(
   [IN] IANet_EthernetAdapter REF Adapters[],
   [IN, ValueMap {"0", "1", "2", "4", "5"}:
                      Amended,
       Values {"AFT", "ALB", "SLA",
                 "IEEE 802.3ad",
                 "SFT"}:Amended]
        uint32 TeamingMode,
   [IN] string TeamName,
   [IN]  boolean MFOEnable,
   [OUT] IANet_TeamOfAdapters REF
                                    TeamPath
);
```

CreateTeam adds a new Intel adapter Team to the system.

```
uint32 ValidateAddAdapters(
   [IN] IANet_PhysicalEthernetAdapter REF
                                      Adapters[],
   [OUT] uint16 ValResult
);
```

Validates the adapters which will be added to a team.

```
uint32 ValidateSetting(
   [IN] IANet_PhysicalEthernetAdapter REF
                                      Adapter,
   [IN] string SettingName,
   [IN] sint64 Value,
   [OUT] uint16 ValResult
);
```

Validates the member adapter setting.

13.2.9 Unsupported Properties

InstallDate and Status are not used.

13.3 IANet_TeamedMemberAdapter

13.3.1 Purpose

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

13.3.2 Instances

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

13.3.3 Creating Instances

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

13.3.4 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.

13.3.5 Modifying Properties

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

13.3.6 Local Properties

This class implements the following local properties:

Property	Description
AdapterFunction	Describes how the adapter is used in a team.
AdapterStatus	Describes the adapters status within the team.

13.3.7 Associations

This is an association class.

13.3.8 Methods

There are no supported methods on this class.

13.4 IANet_NetworkVirtualAdapter

13.4.1 Purpose

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

13.4.2 Instances

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

13.4.3 Creating Instances

The user cannot create instances of this class. To create a team the user creates an instance of IANet_TeamOfAdapters. This class will not exist until after the user has called IANet_NetService.Apply() within the context of a valid client handle and the IANet_EthernetAdapter instance has been created.

13.4.4 Removing Instances

The user cannot delete instances of this class.

13.4.5 Associations

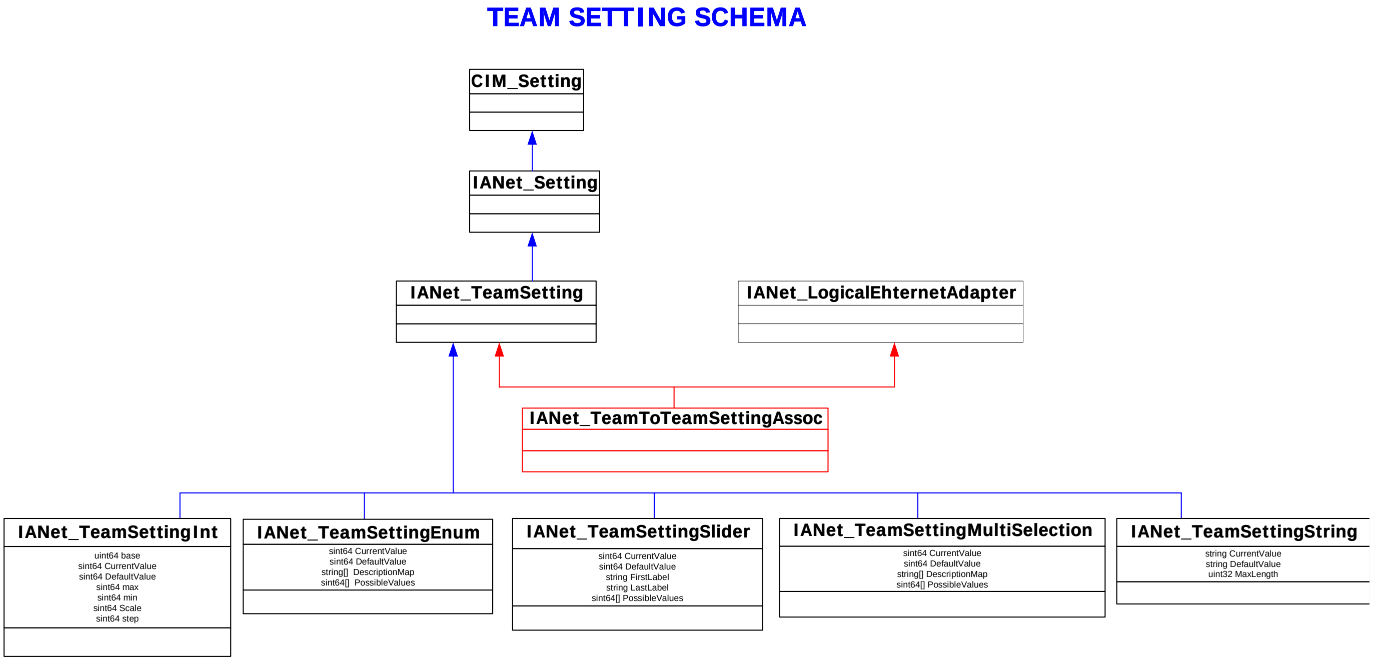
This is an association class.

13.4.6 Methods

There are no supported methods on this class.

14 Team Setting Schema

14.1 Team Setting Schema Diagram



14.2 IANet_TeamToTeamSettingAssoc

14.2.1 Purpose

This class is used to group a collection of IANet_TeamSetting instances.

14.2.2 Instances

Each Team can have several associated IANet_TeamToTeamSettingAssoc instances.

14.2.3 Creating Instances

The user cannot create instances of IANet_TeamToTeamSettingAssoc.

14.2.4 Removing Instances

The user cannot remove instances of IANet_TeamToTeamSettingAssoc.

14.2.5 Modifying Properties

There are no user-modifiable properties for this class.

14.2.6 Associations

An IANet_TeamToTeamSettingAssoc instance will exist to associate each Team (IANet_LogicalEthernetAdapter) with its setting (IANet_TeamSetting).

14.2.7 Methods

There are no supported methods for this class.

14.2.8 Unsupported Properties

None.

14.3 IANet_TeamSetting

14.3.1 Purpose

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

14.3.2 Instances

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

There are several sub-classes for IANet_TeamSetting. 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.

14.3.3 Creating Instances

The user cannot create instances of IANet_TeamSetting.

14.3.4 Removing Instances

The user cannot remove instances of IANet_TeamSetting.

14.3.5 Modifying Properties

This abstract class has no modifiable properties, however, the child classes have modifiable properties (see sub-classes listed in this section).

14.3.6 Associations

Each IANet_TeamSetting instance is associated with an IANet_LogicalEthernetAdapter instance using an instance of IANet_TeamToTeamSettingAssoc.

14.3.7 Methods

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

14.3.8 Unsupported Properties

SettingID and RequiresSession are not used.

14.4 IANet_TeamSettingInt

14.4.1 Purpose

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

14.4.2 Instances

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

14.4.3 Creating Instances

The user cannot create instances of this class.

14.4.4 Removing Instances

The user cannot remove instances of this class.

14.4.5 Modifying Properties

The “CurrentValue” attribute is the only modifiable property of this class. The user can modify this property by using IWbemClassObject::Put() to change the value, then call “IWbemServices::PutInstance()” to update the setting. The Provider will check that:

CurrentValue <= max
CurrentValue >= min
(CurrentValue – min) is a multiple of step

Where max, min, CurrentValue and step are all properties of IANet_TeamSettingInt.

14.4.6 Local Properties

This class implements the following local properties:

Property	Description
base	Root from which the integer value may take values. (example; decimal = base 10)
CurrentValue	The actual value of the integer setting
DefaultValue	The initial value of the integer setting
max	The maximum value that the setting can have
min	The minimum value that the setting can have
Scale	Unit to measure value of setting
step	The granularity of the integer value

14.4.7 Associations

Each IANet_TeamSettingInt instance is associated with an IANet_LogicalEthernetAdapter instance using an instance of IANet_TeamToTeamSettingAssoc.

14.4.8 Unsupported Properties

SettingID and RequiresSession are not used.

14.4.9 Methods

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

14.5 IANet_TeamSettingEnum

14.5.1 Purpose

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

14.5.2 Instances

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

14.5.3 Creating Instances

The user cannot create instances of this class.

14.5.4 Removing Instances

The user cannot remove instances of this class.

14.5.5 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 Provider will check that:

$$\text{CurrentValue} \in \text{PossibleValues[]}$$

14.5.6 Local Properties

This class implements the following local properties:

Property	Description
CurrentValue	The actual value of the setting
DefaultValue	The initial value of the setting
DescriptionMap	An array of descriptions mapped to the PossibleValues property
PossibleValues	The values that correspond to the DescriptionMap

14.5.7 Associations

Each IANet_TeamSettingEnum instance is associated with an IANet_LogicalEthernetAdapter instance using an instance of IANet_TeamToTeamSettingAssoc.

14.5.8 Unsupported Properties

SettingID and RequiresSession are not used.

14.5.9 Methods

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

14.6 IANet_TeamSettingSlider

14.6.1 Purpose

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

14.6.2 Instances

An instance of this class exists for each setting that will be displayed as a slider.

14.6.3 Creating Instances

The user cannot create instances of this class.

14.6.4 Removing Instances

The cannot remove instances of this class.

14.6.5 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 Provider will check that:

$$\text{CurrentValue} \in \text{PossibleValues[]}$$

14.6.6 Local Properties

This class implements the following local properties:

Property	Description
CurrentValue	The actual value of the setting
DefaultValue	The initial value of the setting
FirstLabel	The label that should be displayed to the left of the slider
LastLabel	The label that should be displayed to the right of the slider
PossibleValues	Range of values which should be displayed with the first value on the left and last value on the right side of the slider

14.6.7 Associations

Each IANet_TeamSettingSlider instance is associated with an IANet_LogicalEthernetAdapter instance using an instance of IANet_TeamToTeamSettingAssoc.

14.6.8 Unsupported Properties

SettingID and RequiresSession are not used.

14.6.9 Methods

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

14.7 IANet_TeamSettingMultiSelection

14.7.1 Purpose

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

14.7.2 Instances

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

14.7.3 Creating Instances

The user cannot create instances of this class.

14.7.4 Removing Instances

The user cannot remove instances of this class.

14.7.5 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 Provider will check that:

$$\text{CurrentValue} \in \text{PossibleValues[]}$$

14.7.6 Local Properties

This class implements the following local properties:

Property	Description
CurrentValue	The actual value of the setting
DefaultValue	The initial value of the setting
DescriptionMap	An array of descriptions mapped to the PossibleValues property
PossibleValues	The values that correspond to the DescriptionMap

14.7.7 Associations

Each IANet_TeamSettingMultiSelection instance is associated with an IANet_LogicalEthernetAdapter instance using an instance of IANet_TeamToTeamSettingAssoc.

14.7.8 Unsupported Properties

SettingID and RequiresSession are not used.

14.7.9 Methods

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

14.8 IANet_TeamSettingString

14.8.1 Purpose

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

14.8.2 Instances

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

14.8.3 Creating Instances

The user cannot create instances of this class.

14.8.4 Removing Instances

The user cannot remove instances of this class.

14.8.5 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.

14.8.6 Local Properties

This class implements the following local properties:

Property	Description
CurrentValue	The actual value of the setting
DefaultValue	The initial value of the setting
MaxLength	The maximum string length allowed

14.8.7 Associations

Each IANet_TeamSettingString instance is associated with an IANet_LogicalEthernetAdapter instance using an instance of IANet_TeamToTeamSettingAssoc.

14.8.8 Unsupported Properties

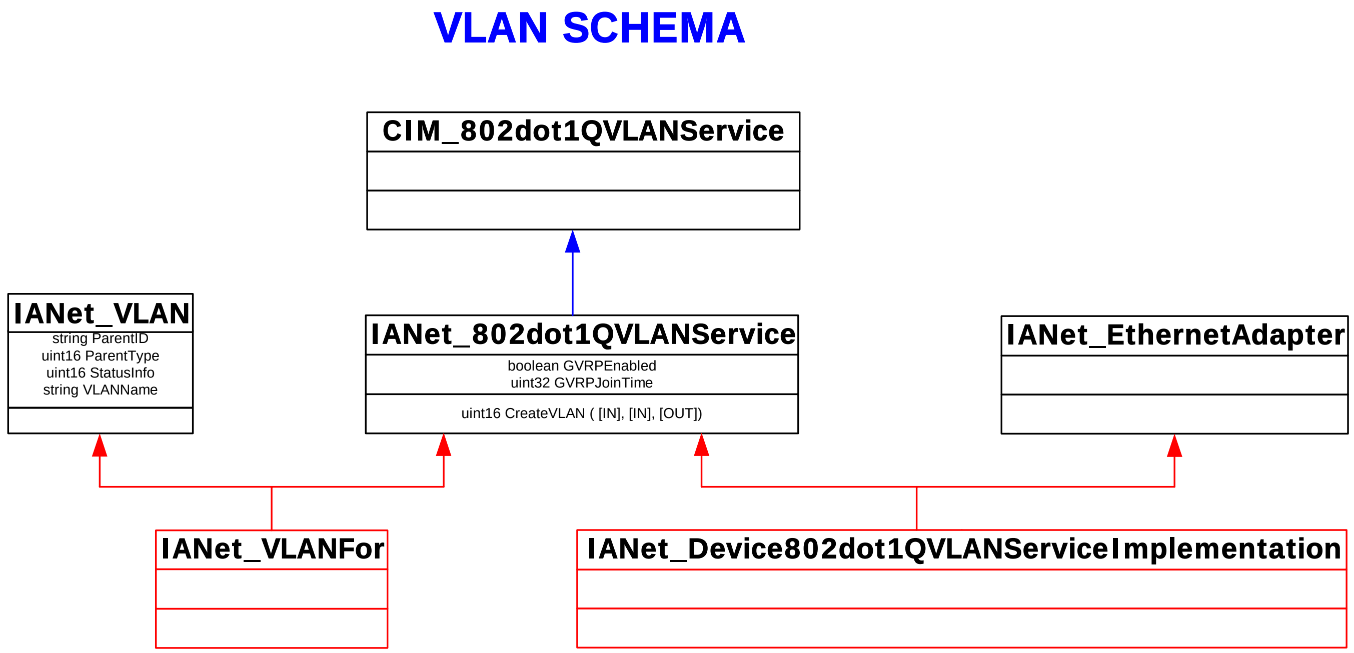
SettingID and RequiresSession are not used.

14.8.9 Methods

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

15 VLAN Schema

15.1 VLAN Schema Diagram



15.2 IANet_802dot1QVLANService

15.2.1 Purpose

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

15.2.2 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.

15.2.3 Creating Instances

The user cannot create instances of this class If the adapter does not have an instance associated with it, then the adapter does not support this service.

15.2.4 Removing Instances

The user cannot delete instances of this class.

15.2.5 Modifying Properties

There are no modifiable properties of this class.

15.2.6 Associations

Each instance of this class will be associated with one IANet_EthernetAdapter using an instance of IANet_Device802dot1QVLANServiceImplementation.

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

15.2.7 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.

15.3 IANet_VLAN

15.3.1 Purpose

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

15.3.2 Instances

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

15.3.3 Creating instances

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

15.3.4 Removing Instances

The user can remove an instance of this class to remove the corresponding VLAN.

15.3.5 Modifying properties

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

15.3.6 Local properties

This class implements the following local properties:

Property	**Description**
ParentID	The VLAN parent device ID
ParentType	The VLAN parent device type
StatusInfo	Status information of logical device (enabled, disabled, other, unknown)
VLANName	Name of the VLAN set by the user

15.3.7 Associations

Each instance is associated with an instance of IANet_VLANSetting using an instance of the class IANet_VLANToVLANSettingAssoc.

15.3.8 Unsupported Properties

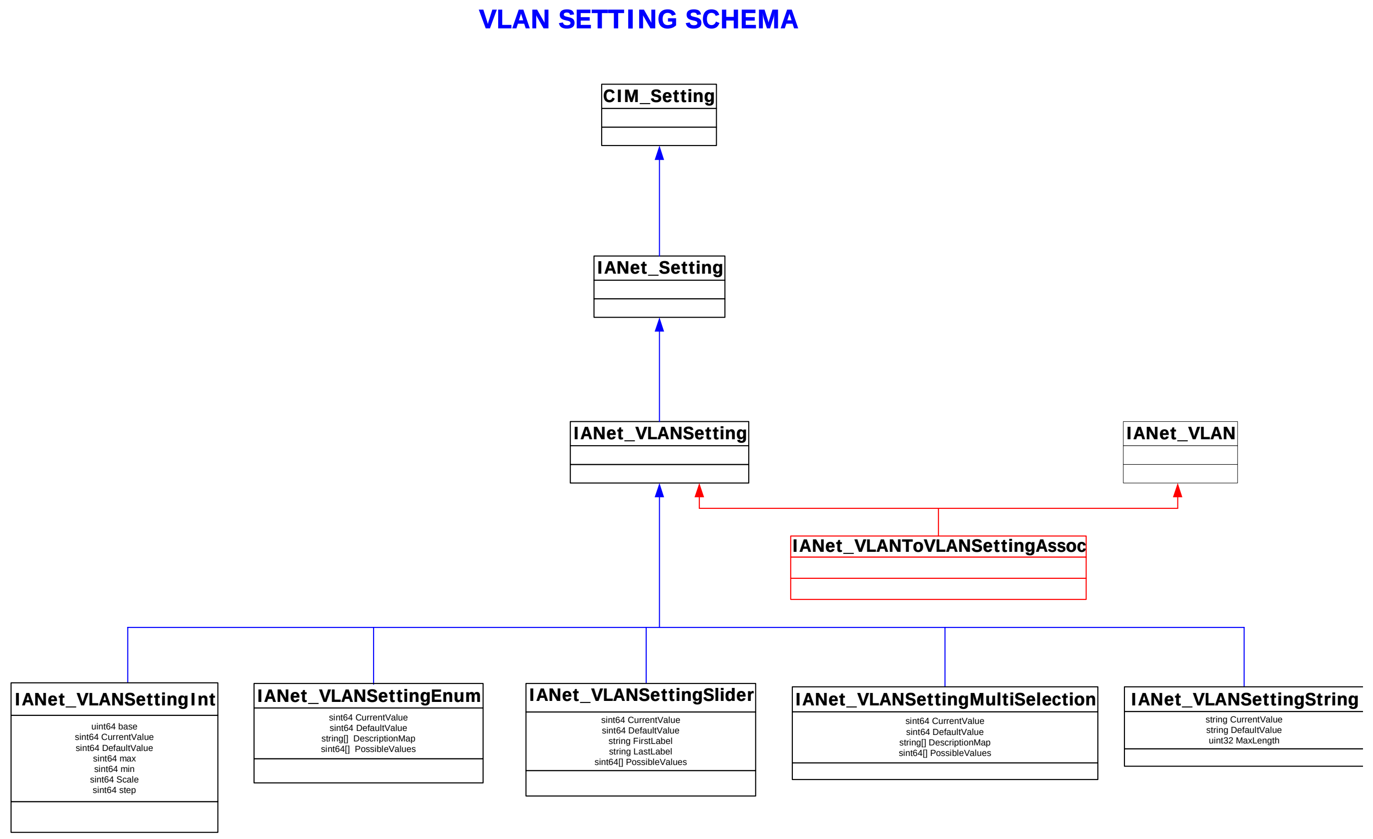
Description, Install Date, StartMode, and Status are not used.

15.3.9 Methods

None

16 VLAN Setting Schema

16.1 VLAN Setting Schema Diagram



16.2 IANet_VLANToVLANSettingAssoc

16.2.1 Purpose

This class is used to group a collection of IANet_VLANSetting instances.

16.2.2 Instances

Each VLAN can have several associated IANet_VLANToVLANSettingAssoc instances.

16.2.3 Creating Instances

The user cannot create instances of IANet_VLANToVLANSettingAssoc.

16.2.4 Removing Instances

The user cannot remove instances of IANet_VLANToVLANSettingAssoc.

16.2.5 Modifying Properties

There are no user-modifiable properties for this class.

16.2.6 Associations

An IANet_VLANToVLANSettingAssoc instance will exist to associate each IANet_VLAN (IANet_LogicalEthernetAdapter) with its setting.

16.2.7 Methods

There are no supported methods for this class.

16.2.8 Unsupported Properties

None.

16.3 IANet_VLANSetting

16.3.1 Purpose

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

16.3.2 Instances

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

There are several sub-classes for IANet_VLANSetting. The sub-classes correspond to the different types and range 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.

16.3.3 Creating Instances

The user cannot create instances of IANet_VLANSetting.

16.3.4 Removing Instances

The user cannot remove instances of IANet_VLANSetting.

16.3.5 Modifying Properties

This abstract class has no modifiable properties, however, the child classes do have modifiable properties (see sub-classes listed in this section).

16.3.6 Associations

Each IANet_VLANSetting instance is associated with an IANet_VLAN instance using an instance of IANet_VLANToVLANSettingAssoc.

16.3.7 Methods

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

16.3.8 Unsupported Properties

SettingID and RequiresSession are not used.

16.4 IANet_VLANSettingInt

16.4.1 Purpose

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

16.4.2 Instances

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

16.4.3 Creating Instances

The user cannot create instances of this class.

16.4.4 Removing Instances

The user cannot remove instances of this class.

16.4.5 Modifying Properties

The “CurrentValue” attribute is the only modifiable property of this class. The user can modify this property by using IWbemClassObject::Put() to change the value, then call “IWbemServices::PutInstance()” to update the setting. The Provider will check that:

CurrentValue <= max
CurrentValue >= min
(CurrentValue – min) is a multiple of step

Where max, min, CurrentValue and step are all properties of IANet_SettingInt.

16.4.6 Local Properties

This class implements the following local properties:

Property	Description
base	Root from which the integer value may take values. (example; decimal = base 10)
CurrentValue	The actual value of the integer setting
DefaultValue	The initial value of the integer setting
max	The maximum value that the setting can have
min	The minimum value that the setting can have
Scale	Unit to measure value of setting
step	The granularity of the integer value

16.4.7 Associations

Each IANet_VLANSettingInt instance is associated with an IANet_VLAN instance using an instance of IANet_VLANToVLANSettingAssoc.

16.4.8 Unsupported Properties

SettingID and RequiresSession are not used.

16.4.9 Methods

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

16.5 IANet_VLANSettingEnum

16.5.1 Purpose

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

16.5.2 Instances

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

16.5.3 Creating Instances

The user cannot create instances of this class.

16.5.4 Removing Instances

The user cannot remove instances of this class.

16.5.5 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 Provider will check that:

$$\text{CurrentValue} \in \text{PossibleValues[]}$$

16.5.6 Local Properties

This class implements the following local properties:

Property	Description
CurrentValue	The actual value of the setting
DefaultValue	The initial value of the setting
DescriptionMap	An array of descriptions mapped to the PossibleValues property
PossibleValues	The values that correspond to the DescriptionMap

16.5.7 Associations

Each IANet_VLANSettingEnum instance is associated with an IANet_VLAN instance using an instance of IANet_VLANToVLANSettingAssoc.

16.5.8 Unsupported Properties

SettingID and RequiresSession are not used.

16.5.9 Methods

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

16.6 IANet_VLANSettingSlider

16.6.1 Purpose

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

16.6.2 Instances

An instance of this class exists for each setting that will be displayed as a slider.

16.6.3 Creating Instances

The user cannot create instances of this class.

16.6.4 Removing Instances

The user cannot remove instances of this class.

16.6.5 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 Provider will check that:

$$\text{CurrentValue} \in \text{PossibleValues[]}$$

16.6.6 Local Properties

This class implements the following local properties:

Property	Description
CurrentValue	The actual value of the setting
DefaultValue	The initial value of the setting
FirstLabel	The label that should be displayed to the left of the slider
LastLabel	The label that should be displayed to the right of the slider
PossibleValues	Range of values which should be displayed with the first value on the left and last value on the right side of the slider

16.6.7 Associations

Each IANet_VLANSettingSlider instance is associated with an IANet_VLAN instance using an instance of IANet_VLANToVLANSettingAssoc.

16.6.8 Unsupported Properties

SettingID and RequiresSession are not used.

16.6.9 Methods

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

16.7 IANet_VLANSettingMultiSelection

16.7.1 Purpose

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

16.7.2 Instances

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

16.7.3 Creating Instances

The user cannot create instances of this class.

16.7.4 Removing Instances

The user cannot remove instances of this class.

16.7.5 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 Provider will check that:

$$\text{CurrentValue} \in \text{PossibleValues[]}$$

16.7.6 Local Properties

This class implements the following properties:

Property	Description
CurrentValue	The actual value of the setting
DefaultValue	The initial value of the setting
DescriptionMap	An array of descriptions mapped to the PossibleValues property
PossibleValues	The values that correspond to the DescriptionMap

16.7.7 Associations

Each IANet_VLANSettingMultiSelection instance is associated with an IANet_VLAN instance using an instance of IANet_VLANToVLANSettingAssoc.

16.7.8 Unsupported Properties

SettingID and RequiresSession are not used.

16.7.9 Methods

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

16.8 IANet_VLANSettingString

16.8.1 Purpose

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

16.8.2 Instances

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

16.8.3 Creating Instances

The user cannot create instances of this class.

16.8.4 Removing Instances

The user cannot remove instances of this class.

16.8.5 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.

16.8.6 Local Properties

This class implements the following local properties:

Property	Description
CurrentValue	The actual value of the setting
DefaultValue	The initial value of the setting
MaxLength	The maximum string length allowed

16.8.7 Associations

Each IANet_VLANSettingString instance is associated with an IANet_VLAN instance using an instance of IANet_VLANToVLANSettingAssoc.

16.8.8 Unsupported Properties

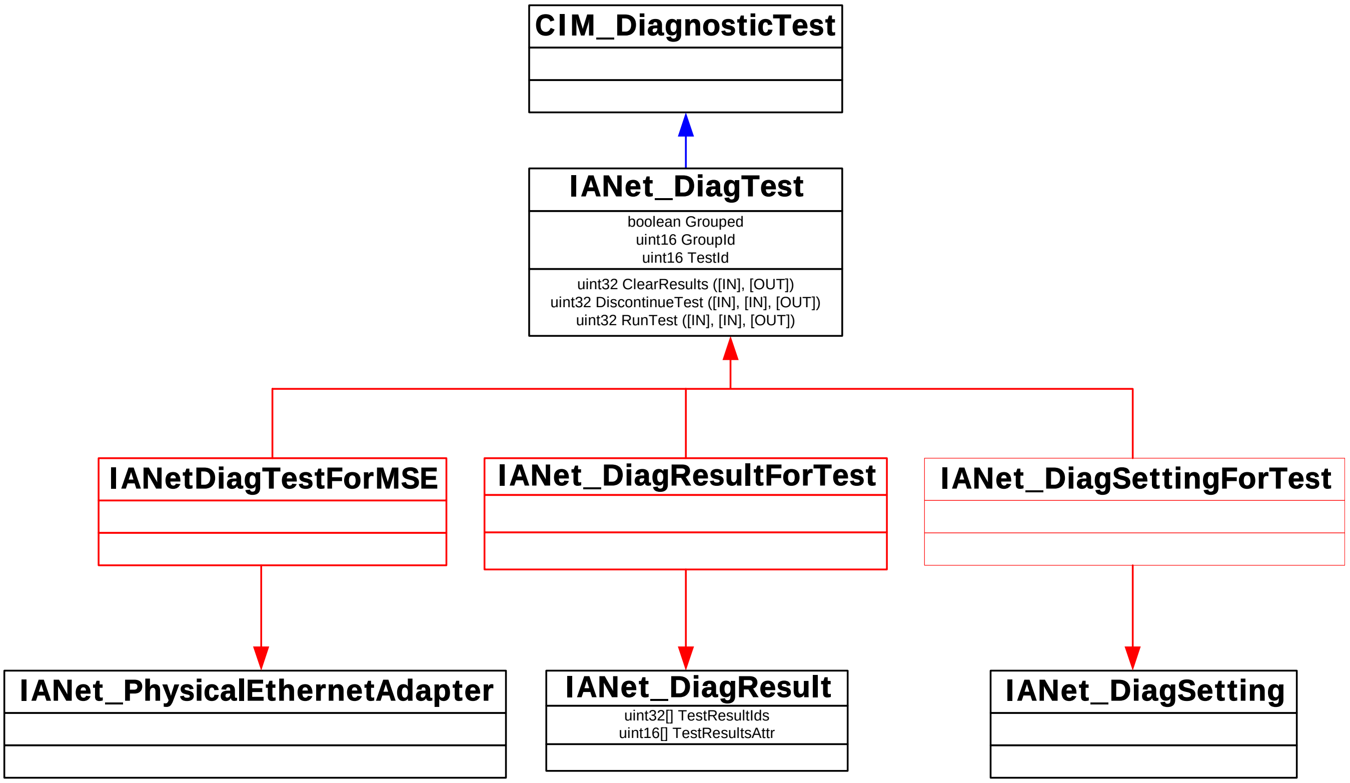
SettingID and RequiresSession are not used.

16.8.9 Methods

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

17 Diagnostic Classes

17.1 Diagnostic Test Schema



Diagnostic Test Schema

17.2 IANet_DiagTest

17.2.1 Purpose

IANet_DiagTest is subclassed from CIM_DiagnosticTest. The class provides a generic vehicle to run and control Diagnostic tests for an Intel PROSet for Microsoft Device Manager 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.

17.2.2 Instances

Key is Name and in this provider it is the concatenation of a numeric index of the test @ 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. Other properties provide other description and run time information.

17.2.3 Creating Instances

The user cannot create instances of IANet_DiagTest.

17.2.4 Deleting Instances

The user cannot delete instances of IANet_DiagTest.

17.2.5 Modifying Properties

There are no user-modifiable properties for this class.

17.2.6 Local Properties

This class implements the following local properties:

Property	Description
Grouped	Indicates whether the test are grouped under a specific category
GroupID	The identification number of the group of which this test belongs
TestID	The specific test identification number

17.2.7 Associations

- An instance of IANet_DiagTestForMSE associates an IANet_DiagTest with an IANet_PhysicalEthernetAdapter.
- An instance of IANet_DiagResultForTest associates an IANet_DiagTest with an IANet_DiagResult instance.
- An instance of IANetDiagSettingForTest associates an IANet_DiagTest with an IANet_DiagSetting.

17.2.8 Unsupported Properties

Caption, Description, InstallDate, OtherCharacteristicDescription,

17.2.9 Methods

This class supports the following methods:

uint32 RunTest([IN] CIM_ManagedSystemElement ref SystemElement, [IN] CIM_DiagnosticSetting ref Setting, [OUT] CIM_DiagnosticResult ref Result);	Runs a test as defined by three parameters referencing: SystemElement - defines the adapter, which we are to run the test on by referring to an instance of SystemElement, which will always be the subclass IANet_EthernetAdapter. Setting - defines the test to be run, and the manner in which it is run by referring to an instance of CIM_DiagnosticSetting, which will always be the subclass IANet_DiagSetting. Result - defines an instance of the class CIM_DiagnosticResult, which will always be the class IANet_DiagResult.
uint32 DiscontinueTest([IN] CIM_ManagedSystemElement ref SystemElement, [IN] CIM_DiagnosticResult ref Result, [OUT] Boolean TestingStopped);	Attempts to stop a diagnostic test in progress as defined by two parameters referencing SystemElement and Result. These parameters function the same as RunTest. A third parameter TestingStopped returns a BOOLEAN value, which indicates if the command was successful in stopping the test.

uint32 ClearResults([IN] CIM_ManagedSystemElement ref SystemElement, [OUT] String ResultsNotCleared[]);	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.

17.3 IANet_DiagSetting

17.3.1 Purpose

Instances of IANet_DiagSetting provide specific run time diagnostic test directives. Directives used are in 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.

17.3.2 Creating Instances

The user cannot create instances of this class.

17.3.3 Deleting Instances

The user cannot delete instances of this class.

17.3.4 Modifying properties

UpdateInstanceAsync is implemented and can be used to set test parameters to HaltOnError, ReportSoftErrors, ReportStatusMessages, QuickMode, TestWarningLevel, and PercentOfTestCoverage.

17.3.5 Associations

An instance of IANetDiagSettingForTest associates an IANet_DiagTest with an IANet_DiagSetting.

17.3.6 Unsupported properties

The following properties are not supported by NCS2:

Caption, Description

17.3.7 Methods

None

17.4 IANet_DiagResult

17.4.1 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.

17.4.2 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 is able to 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.

17.4.3 Creating Instances

The user cannot create instances of this class

17.4.4 Deleting Instances

The user cannot delete instances of this class

17.4.5 Modifying Properties

The user cannot modify instances of this class

17.4.6 Local Properties

This class implements the following local properties:

Property	**Description**
TestResultIds	Indicates the result string ID’s
TestResultsAttr	The type of the result string

17.4.7 Associations

- An instance of IANet_DiagResultForTest associates an IANet_DiagTest with an IANet_DiagResult instance.
- An instance of IANet_DiagResultForMSE associates an IANet_PhysicalEthernetAdapter with an IANet_DiagResult instance.

17.4.8 Unsupported Properties

The following properties are not supported by NCS2:

EstimatedTimeOfPerforming OtherStateDescription, HaltOnError, ReportSoftErrors, and TestWarningLevel

17.4.9 Methods

None

18 Getting the Current Configuration

The client does not need to get a client handle to read the current configuration. Clients can use a NULL context, however, any error messages will be returned in the default language for the managed machine.

In the following 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.

18.1 Getting the Physical Adapters

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

Task	WQL Query	Result Class	Comment
Enumerate all adapters	SELECT * FROM IANet_EthernetAdapter	IANet_EthernetAdapter	Returns all IANet_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.

18.2 Getting the Team Configuration

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

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_LogicalEthernetAdapter	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_PhysicalEthernetAdapter	Returns the adapters which are in the team, but does not describe what role the adapter plays.

Task	WQL Queries	Result Class	Comments
Determine an adapter's role in a team	REFERENCES OF {IANet_PhysicalEthernetAdapter path} WHERE ResultClass = IANet_TeamedMemberAdapter	IANet_TeamedMemberAdapter	The class contains information about how the member adapter relates to the team and its current status within the team.

18.3 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.

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

19 Updating the configuration

In most cases, to update the configuration, the client application will need to get a client handle from the IANet_NetService class and store this handle in an IWbemContext context object. Changes to the configuration will only occur when the “Apply” method on the IANet_NetService is called.

19.1 Changing the adapter, team or VLAN settings

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

The easiest way for the client to update a setting, is to: (1) get an instance of the setting object from WMI, (2) modify the CurrentValue attribute (using IWbemClassObject::Put()), and (3) call IWbemServices::PutInstance() to pass the modified instance back to the Provider. PutInstance must be called with the flag WBEM_FLAG_UPDATE_ONLY.

The 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 whenever 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 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, then call PutInstance.

19.2 Creating a new team

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 the user 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.

19.3 Adding an adapter to a team

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 properties 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.

The following properties may optionally be set:

- can be used to 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.

19.4 Removing an adapter from a team

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.

19.5 Deleting a team

To delete a team, delete the IANet_TeamOfAdapters instance using IWbemServices::DeleteInstance()

If this action fails, check IANet_ExtendedStatus to get the error code.

19.6 Changing the mode of a team

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 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.

19.7 Changing an adapter’s priority within a team

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 Provider to update adapter’s priority.

If this action fails the client should check the IANet_ExtendedStatus for the error code.

19.8 Uninstalling an adapter

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

19.9 Creating a VLAN

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 the number of the VLAN. (Range 1- 4094)
- Name 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.

19.10 Changing the Properties of a VLAN

The client can change the VLANNumber and VLANName properties 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 Provider to update the properties, passing the flag WBEM_FLAG_UPDATE_ONLY.

If this action fails, check the IANet_ExtendedStatus for the error code.

19.11 Deleting a VLAN

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

19.12 Updating the Boot Agent

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] uint32 FlashRetCode);	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.

19.13 Executing methods in IANet_DiagTest

Here is the RunTest method, from the MOF file:

```
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.

From the main WBEM test dialog box, click “Connect”.

Enter the appropriate Server\Namespace. Namespaces IntelNCS2 and CimV2 are supported.

Click the “Enum Instances” button of WBEM test and enter “IANet_DiagTest”

Double click the desired 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.

The following is an example of the object path retrieved:

\\MYCOMPUTER\root\Cimv2:IANet_DiagTest.Name="1@{4A0CDABE-F6C3-45D0-B60D-F6E7BAFA2C2C}"

Save the object path.

Click the “Enum Instances” button of WBEM test and enter “IANet_EthernetAdapter”

Double click on the desired adapter, to be tested.

Following is an example of the object path retrieved.

\\MYCOMPUTER\root\cimv2:IANet_EthernetAdapter.DeviceID="{4A0CDABE-F6C3-45D0-B60D-F6E7BAFA2C2C}"

Save the object path.

Click the “Enum Instances” button of WBEM test and enter “IANet_DiagSetting”

Double click on the setting which represents the desired adapter/test combination.

Following is an example of the object path retrieved:

\\MYCOMPUTER\root\cimv2:IANet_DiagSetting.SettingID="1@{4A0CDABE-F6C3-45D0-B60D-F6E7BAFA2C2C}"

Save the object path.

From the main WBEM test dialog box, click “Execute Method”

Paste the IANet_DiagTest object path into the dialog box. Click OK

Select the desired test in the drop down box under method.

Click the “Edit In Parameters” button.

For RunTest, Setting and SystemElement are the in parameters, paste the previously saved Setting and Adapter object paths. Close.

Click the execute button.

Enumerate the IANet_DiagResult class, in the same manner as the In parameters were.

Examine the selected result object as needed.

20 Summary of CIM classes

20.1.1 IANet_802dot1QVLANService

Can the user create? No

Can the user delete? No

Implemented Methods: CreateVLAN

Settable Properties: None

Unsupported Properties: Description, Install Date, Started, StartMode, Status

Instance Count: One instance for each team or adapter that supports VLANs

Related Association Classes: IANet_Device802dot1QVLANServiceImplementation, IANet_VLANFor

20.1.2 IANet_BootAgent

Can the user create? No

Can the user delete? No

Implemented Methods: ProgramFlash, ReadFlash

Settable Properties: None

Unsupported Properties: Caption, Description, InstallDate, Started, StartMode, Status

Instance Count: One instance for each adapter that supports the boot agent capability.

Related Association Classes: IANet_DeviceBootServiceImplementation, IANet_BootAgentToBootAgentSettingAssoc

20.1.3 IANet_Device802dot1QVLANServiceImplementation

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: None

Unsupported Properties: None

Instance Count: One instance for each adapter or team which supports VLANs

Related Association Classes: This class associates IANet_EthernetAdapter with IANet_802dot1QVLANService.

20.1.4 IANet_PhysicalEthernetAdapter

Can the user create? No

Can the user delete? Yes

Implemented Methods: AdvancedCableTest, ExpressTeam, GetExpressTeamInfo, GetPowerUsageOptions, IdentifyAdapter, SetPowerUsageOptions, TestCable, TestLinkSpeed

Settable Properties: None

Unsupported Properties: AlignmentErrors, AutoSense, CarrierSenseErrors, DeferredTransmissions, DriverComments, DriverDescription, DriverFileSize, DriverFileVersion, DriverLegalCopyright, DriverPath, DriverProductVersion, EnabledCapabilities ErrorCleared, ErrorDescription, ExcessiveCollisions, FCSErrors, FlowControlPacketsReceived, FlowControlPacketsTransmitted, FrameTooLongs, FullDuplex, GeneralReceiveErrors, GeneralTransmitErrors, IdentifyingDescriptions, InstallDate, InternalMACReceiveErrors, InternalMACTransmitErrors, LastErrorCode, LateCollisions, MaxDataSize, MaxQuiesceTime, MultipleCollisionFrames, NoBufferReceiveErrors, NoBufferXmitErrors, OctetsReceived, OctetsTransmitted, OtherIdentifyingInfo, PacketTaggingStatus, PowerManagementCapabilities (this is exposed as a method), PowerManagementSupported (this is exposed as a method), PowerOnHours, ShortFramesReceived, SingleCollisionFrames, SymbolErrors, SQETestErrors, TCOFramesReceived, TCOFramesTransmitted, TotalHostErrors, TotalPacketsReceived, TotalPacketsTransmitted, TotalPowerOnHours, TotalWireErrors, TroubleShootingCauses, TroubleShootingProblems, TroubleShootingSeverityLevels, TroubleShootingSolutions

Instance Count: One for each Intel PROSet supported installed adapter.

Related Association Classes: IANet_Device802dot1QVLANServiceImplementation, IANet_DeviceBootServiceImplementation, IANet_DiagTestForMSE, IANet_DiagResultForMSE, IANet_AdapterToSettingAssoc, IANet_TeamedMemberAdapter

20.1.5 IANet_NetService

Can the user create? No

Can the user delete? No

Implemented Methods: BeginApply,Apply

Settable Properties: None

Unsupported Properties: Caption, Description, Install Date, Started, Start Mode, Status

Instance Count: Exactly one.

Related Association Classes: None

20.1.6 IANet_EthernetAdapter

Can the user create? No

Can the user delete? Yes

Implemented Methods: None

Settable Properties: None

Unsupported Properties: AlignmentErrors, AutoSense, CarrierSenseErrors, DeferredTransmissions, DriverComments, DriverDescription, DriverFileSize, DriverFileVersion, DriverLegalCopyright, DriverPath, DriverProductVersion, EnabledCapabilities ErrorCleared, ErrorDescription, ExcessiveCollisions, FCSErrors, FlowControlPacketsReceived, FlowControlPacketsTransmitted, FrameTooLongs, FullDuplex, GeneralReceiveErrors, GeneralTransmitErrors, OtherIdentifyingInfo, IdentifyingDescriptions, InstallDate, InternalMACReceiveErrors, InternalMACTransmitErrors, LastErrorCode, LateCollisions, MaxDataSize, MaxQuiesceTime, MultipleCollisionFrames, NoBufferReceiveErrors, NoBufferXmitErrors, OctetsReceived, OctetsTransmitted, OtherIdentifyingInfo, PacketTaggingStatus, PowerManagementCapabilities (this is exposed as a method), PowerManagementSupported (this is exposed as a method), PowerOnHours, ShortFramesReceived, SingleCollisionFrames, SymbolErrors, SQETestErrors, SymbolErrors, TCOFramesReceived, TCOFramesTransmitted, TotalHostErrors, TotalPacketsReceived, TotalPacketsTransmitted, TotalPowerOnHours, TotalWireErrors, TroubleShootingCauses, TroubleShootingProblems, TroubleShootingSeverityLevels, TroubleShootingSolutions

Instance Count: This is an abstract class.

Related Association Classes: IANet_Device802dot1QVLANServiceImplementation.

20.1.7 IANet_LogicalEthernetAdapter

Can the user create? No

Can the user delete? Yes

Implemented Methods: None

Settable Properties: None

Unsupported Properties: AlignmentErrors, AutoSense, CarrierSenseErrors, DeferredTransmissions, DriverComments, DriverDescription, DriverFileSize, DriverFileVersion, DriverLegalCopyright, DriverPath, DriverProductVersion, EnabledCapabilities ErrorCleared, ErrorDescription, ExcessiveCollisions, FCSErrors, FlowControlPacketsReceived, FlowControlPacketsTransmitted, FrameTooLongs, FullDuplex, GeneralReceiveErrors, GeneralTransmitErrors, OtherIdentifyingInfo, IdentifyingDescriptions, InstallDate, InternalMACReceiveErrors, InternalMACTransmitErrors, LastErrorCode, LateCollisions, MaxDataSize, MaxQuiesceTime, MultipleCollisionFrames, NoBufferReceiveErrors, NoBufferXmitErrors, OctetsReceived, OctetsTransmitted, OtherIdentifyingInfo, PacketTaggingStatus, PowerManagementCapabilities (this is exposed as a method), PowerManagementSupported (this is exposed as a method), PowerOnHours, ShortFramesReceived, SingleCollisionFrames, SymbolErrors, SQETestErrors, SymbolErrors, TCOFramesReceived, TCOFramesTransmitted, TotalHostErrors, TotalPacketsReceived, TotalPacketsTransmitted, TotalPowerOnHours, TotalWireErrors, TroubleShootingCauses, TroubleShootingProblems, TroubleShootingSeverityLevels, TroubleShootingSolutions

Instance Count: One for each team.

Related Association Classes: IANet_NetworkVirtualAdapter., IANet_TeamToTeamSettingAssoc.

20.1.8 IANet_NetworkVirtualAdapter

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: None

Unsupported Properties: None

Instance Count: One instance for each team.

Related Association Classes: This class associates IANet_TeamOfAdapters with an IANet_LogicalEthernetAdapter.

20.1.9 IANet_Setting

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: None

Unsupported Properties: SettingID, RequiresSession

Instance Count: This is an abstract class.

Related Association Classes: None

20.1.10 IANet_AdapterSetting

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: None

Unsupported Properties: SettingID, RequiresSession

Instance Count: This is an abstract class.

Related Association Classes: IANet_AdapterToSettingAssoc

20.1.11 IANet_AdapterSettingInt

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: CurrentValue

Unsupported Properties: SettingID, RequiresSession

Instance Count: One instance for each integer setting

Related Association Classes: IANet_AdapterToSettingAssoc

20.1.12 IANet_AdapterSettingMultiSelection

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: CurrentValue

Unsupported Properties: SettngID

Instance Count: One instance for each multi-selection setting

Related Association Classes: IANet_AdapterToSettingAssoc

20.1.13 IANet_AdapterSettingEnum

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: CurrentValue

Unsupported Properties: SettingID, RequiresSession

Instance Count: One instance for each enum setting

Related Association Classes: IANet_AdapterToSettingAssoc

20.1.14 IANet_AdapterSettingSlider

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: CurrentValue

Unsupported Properties: SettingID, RequiresSession

Instance Count: One instance for each slider setting

Related Association Classes: IANet_AdapterToSettingAssoc

20.1.15 IANet_AdapterSettingString

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: CurrentValue

Unsupported Properties: SettingID, RequiresSession

Instance Count: One instance for each string setting

Related Association Classes: IANet_AdapterToSettingAssoc

20.1.16 IANet_AdapterToSettingAssoc

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: None

Unsupported Properties: None

Instance Count: One instance for each Aadapter setting

Related Association Classes: This class associates IANet_AdapterSetting with IANet_PhysicalEthernetAdapter.

20.1.17 IANet_TeamSetting

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: None

Unsupported Properties: SettingID, RequiresSession

Instance Count: This is an abstract class.

Related Association Classes: None

20.1.18 IANet_TeamSettingInt

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: CurrentValue

Unsupported Properties: SettingID, RequiresSession

Instance Count: One instance for each integer setting

Related Association Classes: IANet_TeamToTeamSettingAssoc

20.1.19 IANet_TeamSettingEnum

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: CurrentValue

Unsupported Properties: SettingID, RequiresSession

Instance Count: One instance for each enum setting

Related Association Classes: IANet_TeamToTeamSettingAssoc

20.1.20 IANet_TeamSettingMultiSelection

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: CurrentValue

Unsupported Properties: SettngID

Instance Count: One instance for each multi-selection setting

Related Association Classes: IANet_ TeamToTeamSettingAssoc

20.1.21 IANet_TeamSettingSlider

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: CurrentValue

Unsupported Properties: SettingID, RequiresSession

Instance Count: One instance for each slider setting

Related Association Classes: IANet_ TeamToTeamSettingAssoc

20.1.22 IANet_TeamSettingString

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: CurrentValue

Unsupported Properties: SettingID, RequiresSession

Instance Count: One instance for each string setting

Related Association Classes: IANet_TeamToTeamSettingAssoc

20.1.23 IANet_ TeamToTeamSettingAssoc

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: None

Unsupported Properties: None

Instance Count: One instance for each Team setting

Related Association Classes: This class associates IANet_TeamSetting with IANet_LogicalEthernetAdapter.

20.1.24 IANet_VLANSetting

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: None

Unsupported Properties: SettingID, RequiresSession

Instance Count: This is an abstract class.

Related Association Classes: None

20.1.25 IANet_VLANSettingInt

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: CurrentValue

Unsupported Properties: SettingID, RequiresSession

Instance Count: One instance for each integer setting

Related Association Classes: IANet_VLANToVLANSettingAssoc

20.1.26 IANet_VLANSettingEnum

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: CurrentValue

Unsupported Properties: SettingID, RequiresSession

Instance Count: One instance for each enum setting

Related Association Classes: IANet_VLANToVLANSettingAssoc

20.1.27 IANet_VLANSettingMultiSelection

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: CurrentValue

Unsupported Properties: SettngID

Instance Count: One instance for each multi-selection setting

Related Association Classes: IANet_ VLANToVLANSettingAssoc

20.1.28 IANet_VLANSettingSlider

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: CurrentValue

Unsupported Properties: SettingID, RequiresSession

Instance Count: One instance for each slider setting

Related Association Classes: IANet_ VLANToVLANSettingAssoc

20.1.29 IANet_VLANSettingString

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: CurrentValue

Unsupported Properties: SettingID, RequiresSession

Instance Count: One instance for each string setting

Related Association Classes: IANet_VLANToVLANSettingAssoc

20.1.30 IANet_ VLANToVLANSettingAssoc

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: None

Unsupported Properties: None

Instance Count: One instance for each VLAN setting

Related Association Classes: This class associates IANet_VLANSetting with IANet_VLAN.

20.1.31 IANet_BootAgentSetting

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: None

Unsupported Properties: SettingID, RequiresSession

Instance Count: This is an abstract class.

Related Association Classes: None

20.1.32 IANet_BootAgentSettingEnum

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: CurrentValue

Unsupported Properties: SettingID, RequiresSession

Instance Count: One instance for each enum setting

Related Association Classes: IANet_BootAgentToBootAgentSettingAssoc

20.1.33 IANet_BootAgentToBootAgentSettingAssoc

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: None

Unsupported Properties: None

Instance Count: One instance for each Boot agent setting

Related Association Classes: This class associates IANet_BootAgentSetting with IANet_BootAgent.

20.1.34 IANet_TeamedMemberAdapter

Can the user create? Yes

Can the user delete? Yes

Implemented Methods: None

Settable Properties: AdapterFunction

Unsupported Properties: PrimaryAdapter, ScopeOfBalancing

Instance Count: One instance for every adapter which is in a team

Related Association Classes: This class associates IANet_TeamOfAdapters with an IANet_PhysicalEthernetAdapter.

20.1.35 IANet_TeamOfAdapters

Can the user create? Yes

Can the user delete? Yes

Implemented Methods: TestSwitchConfiguration, GetBestTeamMode, RenameTeam, CreateTeam, ValidateAddAdapters, ValidateSetting

Settable Properties: TeamingMode

Unsupported Properties: Install Date, Status

Instance Count: One instance for each team

Related Association Classes: IANet_NetworkVirtualAdapter, IANet_TeamedMemberAdapter

20.1.36 IANet_VLAN

Can the user create? No

Can the user delete? Yes

Implemented Methods: None

Settable Properties: VLANNumber, Caption

Unsupported Properties: Description, Install Date, StartMode, Status

Instance Count: One instance for each VLAN

Related Association Classes: IANet_VLANFor, IANet_VLANSetting

20.1.37 IANet_VLANFor

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: None

Unsupported Properties: None

Instance Count: One instance for each VLAN

Related Association Classes: This class associates IANet_VLAN with IANet_802dot1QVLANService.

20.1.38 IANet_DiagTest

Can the user create? No

Can the user delete? No

Implemented Methods: RunTest, DiscontinueTest, ClearResults

Settable Properties: None

Unsupported Properties: Caption, Description, InstallDate, OtherCharacteristicDescription

Instance Count: One for each Adapter/test combination

Related Association Classes: IANet_DiagTestForMSE, IANet_DiagResultForTest, IANet_DiagSettingForTest

20.1.39 IANet_DiagSetting

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: HaltOnError, ReportSoftErrors, ReportStatusMessages, QuickMode, PercentOfTestCoverage, TestWarningLevel,

Unsupported Properties: Caption, Description

Instance Count: One for each Adapter/test combination

Related Association Classes: IANet_DiagSettingForTest

20.1.40 IANet_DiagResult

Can the user create? No

Can the user delete? No

Implemented Methods: None

Settable Properties: None

Unsupported Properties: EstimatedTimeOfPerforming, HaltOnError, OtherStateDescription, ReportSoftErrors, TestWarningLevel

Instance Count: One for each Adapter/test combination

Related Association Classes: IANet_DiagResultForTest, IANet_DiagResultForMSE