Document Number: DSP1002

Date: 2009-02-24

Version: 1.0.0

Diagnostics Profile

Document Type: Specification

Document Status: DMTF Standard

Document Language: E

CONTENTS

Figures

Tables

Foreword

The *Diagnostics Profile* (DSP1002) was prepared by the Diagnostics Working Group.

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability.

Introduction

A *profile* is a collection of Common Information Model (CIM) elements and behavior rules that represent a specific area of management. The purpose of a profile is to ensure interoperability in the use of web-based enterprise management (WBEM) services for a specific subset of the Distributed Management Task Force (DMTF) CIM schema for a specific management area—in this case, diagnostics.

Diagnostics is a critical component of systems management. Diagnostic services are used in problem containment to maintain availability, achieve fault isolation for system recovery, establish system integrity during boot, increase system reliability, and perform routine proactive system verification. The goal of the Common Diagnostic Model (CDM) is to define industry-standard building blocks, based on and consistent with the DMTF CIM, that enables seamless integration of vendor-supplied diagnostic services into system and SAN management frameworks.

The CDM is an architecture and methodology for exposing system diagnostic instrumentation through the CIM standard interfaces.

The ability to transparently run diagnostic tests and exercisers while the user operating system is functional (no reboot required) may significantly contribute to the reduction of Total Cost of Ownership (TCO) and will also lower warranty costs by reducing the return of defect-free parts for service. This functionality is referred to as *OS-Present Diagnostics* (also known as On-line Diagnostics and Concurrent Diagnostics).

A primary objective of the CDM is to standardize the interfaces that diagnostic developers create for their OS-Present Diagnostics in the operating environment, making the diagnostics accessible to all applications that query CIM for diagnostic data or register with CIM to execute diagnostic methods and receive results.

Standardization of these interfaces means that clients, providers, and tests gain a certain degree of portability and, in many cases, need only be written once to satisfy multiple environments and platforms. OEMs can differentiate their diagnostic offerings by how effectively their applications use the information and capabilities available through CIM to maintain and service their systems.

Reduced cost through standardization is accompanied by the initial investment of coding to a new interface. The CDM Forum intends to ease this burden by developing tools to generate most of the interface code necessary to communicate with CIM.

Diagnostics Profile

1 Scope

The information in this specification should be sufficient for a provider or consumer of this data to identify unambiguously the classes, properties, methods, and values that shall be instantiated and manipulated to represent and manage the diagnostic service components of systems and subsystems that are modeled using the DMTF CIM core and extended model definitions.

The target audience for this specification is implementers who are writing CIM-based providers or consumers of management interfaces that represent the component described in this document.

2 Normative References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

2.1 Approved References

DMTF DSP0200, *CIM Operations over HTTP 1.2.0*,
http://www.dmtf.org/standards/published_documents/DSP200.pdf

DMTF DSP0004, *CIM Infrastructure Specification 2.3.0*,
http://www.dmtf.org/standards/published_documents/DSP0004V2.3_final.pdf

DMTF DSP1001, *Management Profile Specification Usage Guide 1.0.0*,
http://www.dmtf.org/standards/published_documents/DSP1001.pdf

DMTF DSP1033, *Profile Registration Profile 1.0.0*,
http://www.dmtf.org/standards/published_documents/DSP1033_1.0.0.pdf

2.2 Other References

ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards*,
http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype

Unified Modeling Language (UML) from the Open Management Group (OMG),
http://www.omg.org/docs/formal/07-11-04.pdf

3 Terms and Definitions

For the purposes of this document, the following terms and definitions apply. The terms and definitions given in DSP1033 and DSP1001 also apply.

3.1
can
used for statements of possibility and capability, whether material, physical, or causal

3.2
cannot
used for statements of possibility and capability, whether material, physical, or causal

3.3
conditional
indicates requirements to be followed strictly in order to conform to the document when the specified conditions are met

3.4
mandatory
indicates requirements to be followed strictly in order to conform to the document and from which no deviation is permitted

3.5
may
indicates a course of action permissible within the limits of the document

3.6
need not
indicates a course of action permissible within the limits of the document

3.7
optional
indicates a course of action permissible within the limits of the document

3.8
referencing profile
indicates a profile that owns the definition of this class and can include a reference to this profile in its “Related Profiles” table

3.9
shall
indicates requirements to be followed strictly in order to conform to the document and from which no deviation is permitted

3.10
shall not
indicates requirements to be followed strictly in order to conform to the document and from which no deviation is permitted

3.11
should
indicates that among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required

3.12
should not
indicates that a certain possibility or course of action is deprecated but not prohibited

3.13
unspecified
indicates that this profile does not define any constraints for the referenced CIM element or operation

4 Symbols and Abbreviated Terms

The following abbreviations are used in this document.

4.1
CDM
Common Diagnostic Model

4.2
CIM
Common Information Model

4.3
CRU
Customer Replaceable Unit

4.4
FRU
Field Replaceable Unit

4.5
ME
Managed Element

4.6
MOF
Managed Object Format

4.7
PD
Problem Determination

4.8
PFA
Predictive Failure Analysis

4.9
SAN
Storage Area Network

4.10
WBEM
Web-Based Enterprise Management

5 Synopsis

Profile Name: Diagnostics

Version: 1.0.0

Organization: DMTF

CIM schema version: 2.9.0

Central Class: CIM_DiagnosticTest

Scoping Class: CIM_ComputerSystem

The *Diagnostics Profile* extends the management capability of referencing profiles by adding the capability to run diagnostic services in a managed system. This profile includes a specification of the Diagnostic Test Service, its configuration, its associated capabilities, its logging mechanisms, and its profile registration information.

Table 1 identifies profiles on which this profile has a dependency.

CIM_DiagnosticTest shall be the Central Class of this profile. The instance of CIM_DiagnosticTest shall be the Central Instance of this profile. CIM_ComputerSystem shall be the Scoping Class of this profile. The instance of CIM_ComputerSystem with which the Central Instance is associated through an instance of CIM_HostedService shall be the Scoping Instance of this profile.

Table 1 – Related Profiles

Profile Name	Organization	Version	Relationship	Behavior
Profile Registration	DMTF	1.0.0	Mandatory	

6 Description

This profile describes the CIM schema extensions that compose the Common Diagnostic Model (CDM) and provides guidelines for the development of diagnostic clients and providers that will promote seamless integration of option diagnostics into Problem Determination and Systems Management applications. Using this profile as a guide, WBEM clients can discover diagnostic services that have been installed on the system and invoke these services to run on their respective devices. The client can monitor the progress of the service, obtain and modify the status of the service, and query for results.

The architecture of the CDM is described in the *CIM Diagnostic Model White Paper*. This profile is a normative presentation of the model described in the white paper, and it suggests implementation techniques that will result in the highest degree of interoperability. It is targeted at developers of diagnostic applications (WBEM clients) and hardware instrumentation (for the WBEM server) to help them understand the CDM.

Figure 1 presents the class schema for the *Diagnostics Profile*. For simplicity, the prefix CIM_ has been removed from the names of the classes.

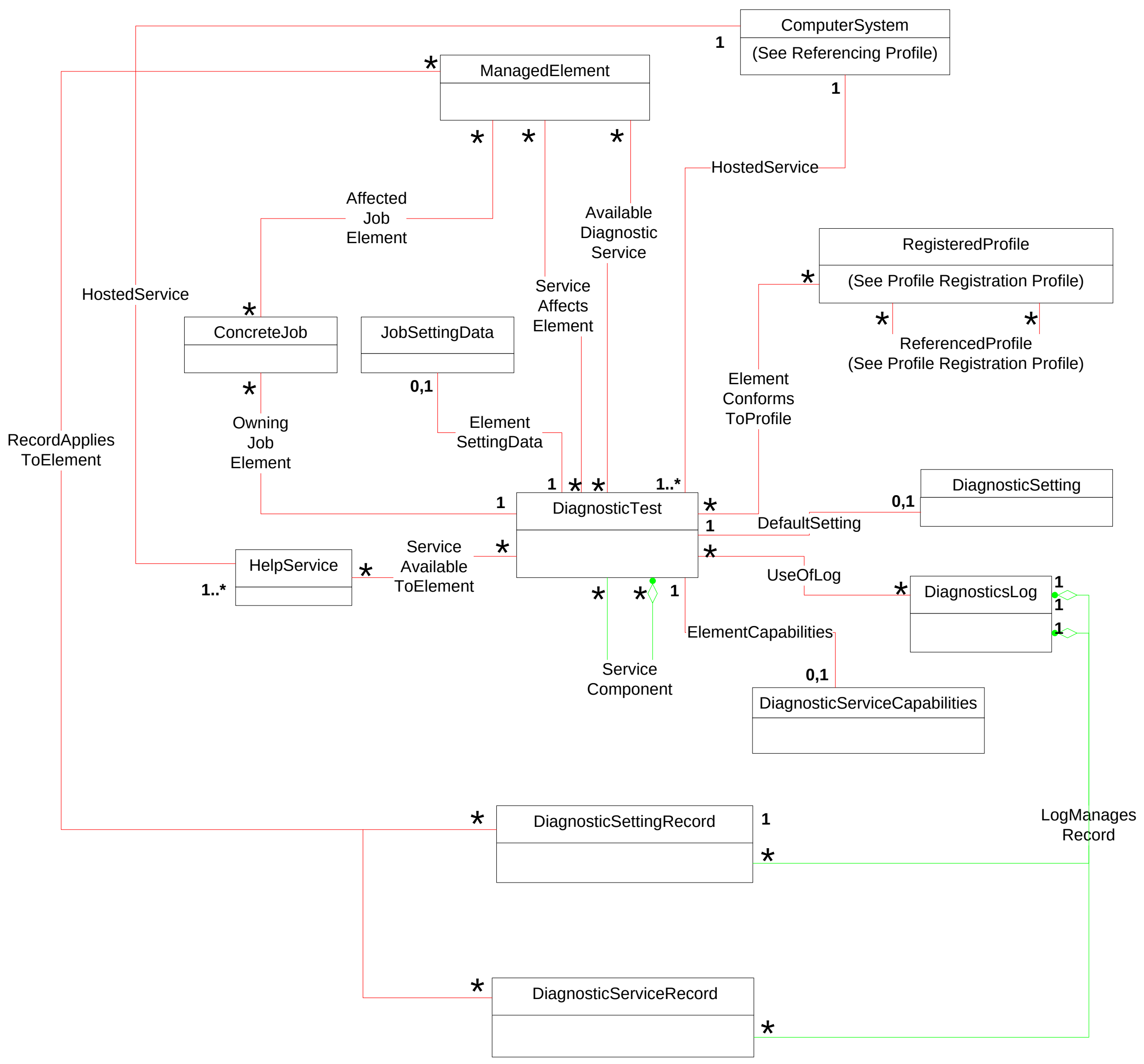

Figure 1 – Diagnostics Profile: Class Diagram

7 Implementation

This section details the requirements related to the arrangement of instances and their properties for implementations of this profile.

The *Diagnostics Profile* consists of definitions for classes related to the CIM_DiagnosticService class, such as CIM_DiagnosticTest, CIM_DiagnosticSetting, and CIM_DiagnosticServiceCapabilities. It also defines the CIM_DiagnosticsLog class and its related classes, CIM_DiagnosticRecord, CIM_DiagnosticServiceRecord, and CIM_DiagnosticSettingRecord. Requirements for propagating and formulating certain properties of these classes and their parents are discussed in this section. Required methods are listed in section 8, and properties are listed in section 10.

7.1 CIM_DiagnosticTest

CIM_DiagnosticTest is the only defined subclass of CIM_DiagnosticService. CIM_DiagnosticTest inherits the RunDiagnostic( ) method, which is used to execute a diagnostic test on a managed element.

Each diagnostic test shall be represented by an instance of either CIM_DiagnosticTest or a subclass. Note that a test that actually packages multiple subtests shall also be represented by such an instance and shall set the IsPackage characteristic for that instance (see section 7.1.3.5).

Depending on the implementation, a provider may use

- an instance of CIM_DiagnosticTest for each test
- an instance of a single subclass (for example, ST_DiskDiagnosticTest) for each test
- a different subclass and its instance (for example, ST_DiskDiagnosticSelfTest, ST_DiskDiagnosticRWVTest) for each test

The same provider may use a combination of the preceding approaches.

7.1.1 CIM_DiagnosticTest.Name

The Name property uniquely identifies the service and provides an indication of the functionality that is managed. The value of the Name property shall be unique and should indicate the nature of the service (for example, EjectTest).

7.1.2 CIM_DiagnosticTest.ElementName

The ElementName property shall be used to provide a user-friendly name for the service. This name shall be used by clients to identify the service to the user.

7.1.3 CIM_DiagnosticTest.Characteristics

This section defines the values of the Characteristics property.

7.1.3.1 2 (Is Exclusive)

Use this value to indicate that only one instance of the diagnostic test may be running at one time, even if more than one target device exists.

7.1.3.2 3 (Is Interactive)

Use this value to indicate that the test requires some interaction with the client at the system under test (for example, when media is required in a device for the test to run).

7.1.3.3 4 (Is Destructive)

Use this value to indicate that the test has the potential for destroying data, permanently altering the state, or reconfiguring the device.

7.1.3.4 5 (Is Risky)

Use this value to indicate that data loss, state change, or reconfiguration may occur if the test is interrupted. For example, a test saves some device data or configuration, changes the device state, performs some operation, and then restores the saved data. If this process is interrupted, the device may be left in an altered state.

7.1.3.5 6 (Is Package)

Use this value to indicate that the test is actually a set of lower-level diagnostics that are packaged together by the test. This packaging is implemented by the test, not aggregated by CIM. Information and results associated with the individual tests in the package may be requested by using the Subtests value in the CIM_DiagnosticSetting.LogOptions array.

If the lower-level diagnostics are themselves CIM_DiagnosticTest instances, the packaging test shall be associated to those lower-level diagnostics through an instance of the CIM_ServiceComponent association. See section 7.8.

7.1.3.6 7 (Reserved)

This value originally contained “Supports PercentOfTestCoverage”, which was deprecated and added to the CIM_DiagnosticServiceCapabilities class.

7.1.3.7 8 (Is Synchronous)

Use this value to indicate that this diagnostic service will complete before the RunDiagnostic( ) method returns to the caller. A job is still created that the client may access for accounting purposes, but the ability to track the progress and status of the job are lost. Additionally, in certain environments, the client may be "blocked" from further action until the service completes. Development of synchronous diagnostic services is not recommended.

7.1.3.8 9 (Media Required)

Use this value to indicate that media shall be inserted into the device to perform the service.

7.1.3.9 10 (Additional Hardware Required)

Use this value to indicate that some additional hardware (for example, a wrap plug) shall be installed to perform the service.

7.1.4 Looping Tests

Looping tests or groups of tests is useful for detecting intermittent faults. The client, provider, or test may control looping, and the method chosen depends on many factors, a few of which follow:

- A client may want to loop a test that does not support looping.
- A provider may choose to support looping even though its tests do not.
- A stress test may, by its nature, want to repeat a certain operation a large number of times.

Looping in the provider and test is under control of the LoopControl( ) and LoopControlParameter( ) properties of the CIM_DiagnosticSetting class. These properties are used to specify the number of iterations in the loop, either directly or through a termination condition. If more than one control is set, the first one that reaches its condition terminates the loop.

Looping in the client is entirely under the control of the client and would generally not affect the CIM_DiagnosticSetting object.

Note: A remote client may incur network delays and CIM Server delays during every iteration of its loop, and this is not an effective way to stress a device.

It is recommended that all diagnostic tests support looping. Exceptions exist where looping a test leads to an undesirable condition (for example, a risky test, certain user interactions, or excessive mechanical wear).

7.1.5 Test Effectiveness

Although the focus of this profile is use of the CIM schema, the CDM includes the notion of test effectiveness. A perfectly implemented CDM provider wrapped around an ineffective test is not very useful.

Diagnostic tests should provide support for all properties in the CIM_DiagnosticSetting class.

The QuickMode property of the CIM_DiagnosticSetting class shall be supported for “long-running” tests (that is, tests with running times in excess of what would be considered compatible with a quick system “health check” of a few minutes). QuickMode need not be supported for interactive, risky, or destructive tests, because these tests would not be useful as a health check.

Note: QuickMode is distinct from PercentOfTestCoverage in that it is a Boolean property that may be set by a client without any particular knowledge of the test. Use of PercentOfTestCoverage requires that the client be aware of the effects and expected outcome of this “throttling” setting control. Diagnostic tests should support the ability to surface logs that may be useful in the problem-determination process.

7.2 CIM_AvailableDiagnosticService

An instance of CIM_AvailableDiagnosticService shall associate a managed element with a diagnostic service that is available for that element. This instance is the means by which clients discover the diagnostic services that are installed for a particular managed element.

7.2.1 CIM_AvailableDiagnosticService.EstimatedDurationOfService

All tests shall attempt to accurately set the EstimatedDurationOfService property. As stated in the MOF file for this class, this property is an estimation of magnitude, not absolute time, and is to be used as a guide for the client.

The CIM_DiagnosticSetting.LoopControl property allows a client to indicate how long a test should run. Tests should use their default values for the LoopControl properties when determining a value for EstimatedDurationOfService.

Interactive tests have an additional complication because their test execution depends on the responses from the user. However, this type of test is not much different than a test whose execution depends on information from a device and the response time of the hardware, or even on how much CPU time or other system resources are allocated to the test. Interactive tests should assume a user response time. If a test cannot reasonably determine an EstimatedDurationOfService value (for example, a completely interactive test that does not know anything about what it will do until a user tells it what tests to run), it can set the value to 0 (Unknown).

7.2.2 CIM_AvailableDiagnosticService.EstimatedDurationQualifier

The EstimatedDurationQualifier property allows for more accurate quantification of the value specified for the EstimatedDurationOfService property. This property shall be implemented only if further quantification is possible.

7.3 CIM_DiagnosticServiceCapabilities

CIM_DiagnosticServiceCapabilities is the means by which a diagnostic service may publish its support for various options—in particular, settings. If a setting is supported, the client may assign it, usually in satisfaction of a user request. The client gains access to an instance of CIM_DiagnosticServiceCapabilities through an instance of CIM_ElementCapabilities.

7.4 CIM_DiagnosticSetting

This class defines specific diagnostic service parameters and execution instructions. To provide more detailed settings for a type of test (that is, additional properties), subclassing is appropriate.

The default settings for a diagnostic service are obtained by using the CIM_DefaultSetting association to an instance of (a subclass of) CIM_DiagnosticSetting. If a service does not publish defaults in this manner, the client should either avoid settings altogether or use only those settings supported by an instance of CIM_DiagnosticServiceCapabilities.

Note that the CIM_DiagnosticSetting subclass may have extensions. If the client is aware of the extensions, these may be modified as well. If the client is unaware, the default values should be used.

If a client chooses to accept the default settings (published or not), the CIM_DiagnosticSetting object may be excluded from the method parameter list (entered as NULL).

7.5 CIM_ConcreteJob

This section defines the properties of the CIM_ConcreteJob class. All executing diagnostics will be represented by instances of CIM_ConcreteJob so that a client can track the progress and control the execution of the executing diagnostic.

7.5.1 CIM_ConcreteJob.TimeBeforeRemoval

To properly implement the functionality implied by this property, the job completion time shall be determined. The algorithm is

> If JobState=Completed OR Terminated OR Killed OR Exception OR ShuttingDown, then Completion Time=StartTime+ElapsedTime.

The job may be deleted at Completion Time+TimeBeforeRemoval.

7.5.2 CIM_ConcreteJob.PercentComplete

This property indicates the percentage of the job that has completed at the time that this value is requested.

Implementation of this property is mandatory in order to provide progress indication to clients.

The value of this property shall be kept current to be useful. Service providers should update this property within one second of becoming aware of a progress change.

The PercentComplete property shall always report the actual percent complete of how much testing was done. It shall be set to 100 percent only when the test is complete. It shall not be set to 100 percent if the test stops for any other reason (for example, the test stopped or was killed by user, the test exited due to a critical failure, or the test found an error and HaltOnError is TRUE) because the actual percent complete is not 100 percent.

7.6 CIM_DiagnosticsLog

All diagnostic result messages may be represented by instances of CIM_DiagnosticRecord subclasses. Moreover, those records may be aggregated to an instance of CIM_DiagnosticsLog. A diagnostic service may also implement other additional logging mechanisms. Any other implemented logging mechanism shall be indicated in the LogStorage property of the published capabilities.

7.6.1 Logging Results

The ways to record the results of running a diagnostic service are specified by the LogOptions and LogStorage properties of the CIM_DiagnosticsSetting class. Use LogOptions to specify *what* to log and

LogStorage to specify *where* to log it. The MOF file describes these properties in some detail, but it is useful to emphasize the mandatory mechanism here.

Diagnostic Records aggregated to the Diagnostic Log are highly recommended for several reasons:

- The heterogeneous nature of the log entries more easily fits into a self-describing record paradigm.
- Keyed records are easier to manage and retrieve.

7.7 CIM_DiagnosticRecord

CIM_DiagnosticRecord has two subclasses: CIM_DiagnosticServiceRecord and CIM_DiagnosticSettingRecord.

CIM_DiagnosticServiceRecord is structured to hold the information that is generated while a particular service is running.

CIM_DiagnosticSettingRecord is structured to hold the attributes of the setting object that was used as an input parameter to the RunDiagnostic( ) method.

7.7.1 CIM_DiagnosticRecord.ExpirationDate

After a diagnostic service produces results, the result objects need to persist for a minimum amount of time to allow diagnostic CIM clients to capture what the application needs. When the data has been captured, the containing objects need to be deleted in a timely fashion.

CIM_DiagnosticSetting.ResultPersistence shall be used by the client to specify to the diagnostic service provider how long the results generated by that service shall persist. A value shall be chosen that allows the minimum time needed by the client to record the data. When the timeout value has been reached, the provider shall expire the data objects that contain the results.

The value of CIM_DiagnosticRecord.ExpirationDate shall be calculated by the provider to account for the persistence setting value, time zone, and other applicable factors. When this expiration value has been reached, the record is eligible for immediate deletion by the provider. It is the provider’s responsibility to manage the logs to prevent accumulation of expired records.

A ResultPersistence value of 0 (zero) indicates that the result does not need to persist; the ExpirationDate is set to the current date and time. A ResultPersistence value of 0xFFFFFFFF indicates that the result shall persist until it is explicitly deleted by a client DeleteInstance or ClearLog call; the ExpirationDate is set to NULL, indicating no expiration date.

7.8 CIM_ServiceComponent

CIM_ServiceComponent is the means by which clients discover any individual tests that are also subtests within a packaging test. This association does not imply any order, number, or method of subtest execution, nor that all subtests executed within a packaging test shall be individual tests, nor even that all the subtests would be executed for any specific execution of the packaging test.

The packaging test shall ensure that the values in CIM_DiagnosticTest.Characteristics of the packaging test are consistent with the values in CIM_DiagnosticTest.Characteristics of the subtests unless the packaging test can execute the subtest such that it does not have those characteristics. For example, if a subtest sets the values Is Destructive or Is Interactive, the packaging test values in CIM_DiagnosticTest.Characteristics should reflect those same characteristics, unless the packaging test can execute the subtest so that it is not destructive or interactive.

8 Methods

This section details the requirements for supporting intrinsic operations and extrinsic methods for the CIM elements defined by this Profile.

8.1 CIM_DiagnosticService.RunDiagnostic( )

The RunDiagnostic( ) method is invoked to commence execution of a diagnostic service on a specific managed element. The input parameters specify this managed element and the settings that are to be applied to the diagnostic service and the resultant job. The method returns a reference to the CIM_ConcreteJob instance that is created.

Before invoking this method, clients examine the appropriate capabilities and create valid CIM_DiagnosticSetting and CIM_JobSettingData instances to apply as input parameters. The RunDiagnostic( ) method shall capture the attributes of CIM_DiagnosticSetting in an instance of CIM_DiagnosticSettingRecord. This information is useful for post-mortem analysis of diagnostic results.

A job may be instantiated to monitor the diagnostic service as it runs and to provide useful accounting and status information when the diagnostic service has completed.

RunDiagnostic( ) return values are specified in Table 2 and parameters are specified in Table 3. No standard messages are defined.

Table 2 – RunDiagnostic( ) Method: Return Code Values

Value	Description
0	Job completed with no error
1	Not supported
2	Unknown
3	Timeout
4	Failed
5	Invalid parameter
0x1000	Method parameters checked – job started
0x1001..0x7FFF	Method reserved
0x8000..0xFFFF	Vendor Specific

Table 3 – RunDiagnostic( ) Method: Parameters

Qualifiers	Name	Type	Description/Values
IN	ManagedElement	CIM_ManagedElement	A reference that specifies the element upon which to run the diagnostic service This parameter is Mandatory.
IN	DiagSetting	CIM_DiagnosticSetting	A reference that specifies the settings to be applied to the diagnostic service. If NULL, the diagnostic service's defaults are used.
IN	JobSetting	CIM_JobSettingData	A reference that specifies the settings to be applied to the resulting job. If NULL, the job's defaults are used.
OUT	Job	CIM_ConcreteJob	Returns a reference to the resulting job

8.2 CIM_ConcreteJob.RequestStateChange( )

All CIM_DiagnosticService.RunDiagnostic( ) calls will return a reference to a CIM_ConcreteJob instance, which represents the diagnostic execution. The CIM_ConcreteJob.RequestStateChange( ) method is invoked to control the diagnostic program execution. The input parameters specify the execution control to be performed (Suspend, Kill, Terminate) and a timeout period that specifies the maximum amount of time that the client expects the transition to the new state to take.

Before invoking this method, clients examine the appropriate capabilities to verify whether the execution control is supported. The RequestStateChange( ) method shall change the JobState value if the transition is successfully performed.

RequestStateChange( ) return values are specified in Table 4 and parameters are specified in Table 5. No standard messages are defined.

Table 4 – RequestStateChange( ) Method: Return Code Values

Value	Description
0	Completed with No Error
1	Not Supported
2	Unknown/Unspecified Error
3	Can NOT complete within Timeout Period
4	Failed
5	Invalid parameter
6	In Use
7..4095	DMTF reserved
4096	Method parameters checked – transition started
4097	Invalid state transition
4098	Use of timeout parameter not supported
4099	Busy
4100 - 32767	Method reserved
32768-65535	Vendor specific

Table 5 – RequestStateChange( ) Method: Parameters

Qualifiers	Name	Type	Description/Values
IN	RequestedState	uint16	The requested state of a job, which may be one of the following values: Start (2), Suspend (3), Terminate (4), Kill (5), or Service (6)
IN	TimeoutPeriod	datetime	A timeout period that specifies the maximum amount of time that the client expects the transition to the new state to take. The interval format shall be used to specify the TimeoutPeriod.

8.3 CIM_Log.ClearLog( )

The ClearLog( ) method is invoked to delete all records (instances of CIM_DiagnosticRecord subclasses) that are associated with the log instance through the CIM_LogManagesRecord association. This method has no parameters, and no standard messages are defined.

ClearLog return values are specified in Table 6.

Table 6 – ClearLog( ) Method: Return Code Values

Value	Description
0	Completed with no error
1	Not supported
2	Unspecified Error
3	Timeout
4	Failed
5	Invalid parameter
"6..0x0FFF"	DMTF reserved
0x1000..0x7FFF	Method_Reserved
0x8000..0xFFFF	Vendor_Reserved

8.4 CIM_HelpService.GetHelp( )

The GetHelp( ) method is invoked to obtain documentation about a diagnostic service. The input parameters provide the name, format, and delivery type of a document.

The CIM_HelpService class has some attributes that publish the available documents, supported delivery types, and formats. See Table 8 for additional information. Before invoking this method, clients check these attributes in order to request an available document, format, and delivery type.

GetHelp( ) return values are specified in Table 7 and parameters are specified in Table 8. No standard messages are defined.

Table 7 – GetHelp( ) Method: Return Code Values

Value	Description
0	Document returned with no error
1	Not supported
2	Unspecified Error
3	Timeout
4	Failed
5	Invalid parameter
6..0x0FFF	DMTF reserved
0x1000	Busy
0x1001	Requested Document not found
0x1002..0x7FFF	Method Reserved
0x8000..0xFFFF	Vendor Reserved

Table 8 – GetHelp( ) Method: Parameters

Qualifiers	Name	Type	Description/Values
IN	RequestedDocument	string	The document that should be made available to the client. The available documents are published in the DocumentsAvailable attribute.
IN	Format	uint16	The format that the document should have. The supported formats are published in the DocumentFormat attribute.
IN	RequestedDelivery	uint16	The way in which the document should be made available (fully specified path, launch a program, file contents, and so on)
OUT	DocumentInfo	string	This parameter returns information about the document. The format and content will depend on the RequestedDelivery parameter.

8.5 Profile Conventions for Operations

Support for operations for each Profile class (including associations) is specified in the following subclauses. Each subclause includes either the statement “All operations in the default list in section 8.5 are supported as described by DSP0200 version 1.2” or a table listing all of the operations that are not supported by this Profile or where the Profile requires behavior other than that described by DSP0200 version 1.2.

The default list of operations is as follows:

- GetInstance
- Associators
- AssociatorNames
- References
- ReferenceNames
- EnumerateInstances
- EnumerateInstanceNames

A compliant implementation shall support all of the operations in the default list for each class, unless the “Requirement” column states something other than *Mandatory*.

8.6 CIM_DiagnosticTest

Table 9 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

Table 9 – Operations: CIM_DiagnosticTest

Operation	Requirement	Messages
References	Unspecified	None
ReferenceNames	Unspecified	None

8.7 CIM_AvailableDiagnosticService

Table 10 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

Table 10 – Operations: CIM_AvailableDiagnosticService

Operation	Requirement	Messages
Associators	Unspecified	None
AssociatorNames	Unspecified	None
References	Unspecified	None
ReferenceNames	Unspecified	None

8.8 CIM_ServiceAffectsElement

Table 11 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

Table 11 – Operations: CIM_ServiceAffectsElement

Operation	Requirement	Messages
Associators	Unspecified	None
AssociatorNames	Unspecified	None
References	Unspecified	None
ReferenceNames	Unspecified	None

8.9 CIM_HelpService

Table 12 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

Table 12 – Operations: CIM_HelpService

Operation	Requirement	Messages
References	Unspecified	None
ReferenceNames	Unspecified	None

8.10 CIM_ServiceAvailableToElement

Table 13 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

Table 13 – Operations: CIM_ServiceAvailableToElement

Operation	Requirement	Messages
Associators	Unspecified	None
AssociatorNames	Unspecified	None
References	Unspecified	None
ReferenceNames	Unspecified	None

8.11 CIM_DiagnosticSetting

Table 14 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

CreateInstance, DeleteInstance and ModifyInstance shall be supported if the provider supports the DiagnosticServiceCapabilities class and the DiagnosticServiceCapabilities class indicates that settings other than default are supported. DeleteInstance and ModifyInstance operations shall return CIM_ERR_ACCESS_DENIED if the DiagnosticSetting instance is associated to the DiagnosticTest instance via the DefaultSetting association.

Table 14 – Operations: CIM_DiagnosticSetting

Operation	Requirement	Messages
CreateInstance	Conditional	None
ModifyInstance	Conditional	None
DeleteInstance	Conditional	None
References	Unspecified	None
ReferenceNames	Unspecified	None

8.12 CIM_DiagnosticServiceCapabilities

Table 15 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

Table 15 – Operations: CIM_DiagnosticServiceCapabilities

Operation	Requirement	Messages
References	Unspecified	None
ReferenceNames	Unspecified	None

8.13 CIM_ElementCapabilities

Table 16 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

Table 16 – Operations: CIM_ElementCapabilities

Operation	Requirement	Messages
Associators	Unspecified	None
AssociatorNames	Unspecified	None
References	Unspecified	None
ReferenceNames	Unspecified	None

8.14 CIM_ConcreteJob

Table 17 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

Table 17 – Operations: CIM_ConcreteJob

Operation	Requirement	Messages
ModifyInstance	Optional	None
References	Unspecified	None
ReferenceNames	Unspecified	None

8.15 CIM_OwningJobElement

Table 18 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

Table 18 – Operations: CIM_OwningJobElement

Operation	Requirement	Messages
Associators	Unspecified	None
AssociatorNames	Unspecified	None
References	Unspecified	None
ReferenceNames	Unspecified	None

8.16 CIM_AffectedJobElement

Table 19 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

Table 19 – Operations: CIM_AffectedJobElement

Operation	Requirement	Messages
Associators	Unspecified	None
AssociatorNames	Unspecified	None
References	Unspecified	None
ReferenceNames	Unspecified	None

8.17 CIM_JobSettingData

Table 20 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

CreateInstance, DeleteInstance and ModifyInstance shall be supported if the provider supports job settings other than default. DeleteInstance and ModifyInstance operations shall return CIM_ERR_ACCESS_DENIED if the JobSettingData instance is associated to the DiagnosticTest instance via the ElementSettingData association where ElementSettingData.IsDefault property is True.

Table 20 – Operations: CIM_JobSettingData

Operation	Requirement	Messages
CreateInstance	Conditional	None
ModifyInstance	Conditional	None
DeleteInstance	Conditional	None
References	Unspecified	None
ReferenceNames	Unspecified	None

8.18 CIM_DefaultSetting

Table 21 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

Table 21 – Operations: CIM_DefaultSetting

Operation	Requirement	Messages
Associators	Unspecified	None
AssociatorNames	Unspecified	None
References	Unspecified	None
ReferenceNames	Unspecified	None

8.19 CIM_DiagnosticsLog

Table 22 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

Table 22 – Operations: CIM_DiagnosticsLog

Operation	Requirement	Messages
References	Unspecified	None
ReferenceNames	Unspecified	None

8.20 CIM_UseOfLog

Table 23 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

Table 23 – Operations: CIM_UseOfLog

Operation	Requirement	Messages
Associators	Unspecified	None
AssociatorNames	Unspecified	None
References	Unspecified	None
ReferenceNames	Unspecified	None

8.21 CIM_DiagnosticServiceRecord

Table 24 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

Table 24 – Operations: CIM_DiagnosticServiceRecord

Operation	Requirement	Messages
CreateInstance	Optional	None
DeleteInstance	Mandatory	None
References	Unspecified	None
ReferenceNames	Unspecified	None

8.22 CIM_DiagnosticSettingRecord

Table 25 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

Table 25 – Operations: CIM_DiagnosticSettingRecord

Operation	Requirement	Messages
CreateInstance	Optional	None
DeleteInstance	Mandatory	None
References	Unspecified	None
ReferenceNames	Unspecified	None

8.23 CIM_LogManagesRecord

Table 26 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

Table 26 – Operations: CIM_LogManagesRecord

Operation	Requirement	Messages
Associators	Unspecified	None
AssociatorNames	Unspecified	None
References	Unspecified	None
ReferenceNames	Unspecified	None

8.24 CIM_RecordAppliesToElement

Table 27 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

Table 27 – Operations: CIM_RecordAppliesToElement

Operation	Requirement	Messages
Associators	Unspecified	None
AssociatorNames	Unspecified	None
References	Unspecified	None
ReferenceNames	Unspecified	None

8.25 CIM_ServiceComponent

Table 28 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

Table 28 – Operations: CIM_ServiceComponent

Operation	Requirement	Messages
Associators	Unspecified	None
AssociatorNames	Unspecified	None
References	Unspecified	None
ReferenceNames	Unspecified	None

8.26 CIM_LogRecord

Table 29 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

Table 29 – Operations: CIM_LogRecord

Operation	Requirement	Messages
CreateInstance	Optional	None
DeleteInstance	Mandatory	None
References	Unspecified	None
ReferenceNames	Unspecified	None

8.27 CIM_ElementSettingData

Table 30 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

Table 30 – Operations: CIM_ElementSettingData

Operation	Requirement	Messages
Associators	Unspecified	None
AssociatorNames	Unspecified	None
References	Unspecified	None
ReferenceNames	Unspecified	None

8.28 CIM_CorrespondingSettingsRecord

Table 31 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

Table 31 – Operations: CIM_CorrespondingSettingsData

Operation	Requirement	Messages
Associators	Unspecified	None
AssociatorNames	Unspecified	None
References	Unspecified	None
ReferenceNames	Unspecified	None

8.29 CIM_HostedService

Table 32 lists operations that either have special requirements beyond those from DSP0200 version 1.2 or shall not be supported.

Table 32 – Operations: CIM_HostedService

Operation	Requirement	Messages
Associators	Unspecified	None
AssociatorNames	Unspecified	None
References	Unspecified	None
ReferenceNames	Unspecified	None

9 Use Cases

This section contains object diagrams and use cases for the *Diagnostics Profile*.

9.1 Profile Conformance

Conformance of a central class instance and its associated instances to a particular profile may be identified by examining instances of the CIM_ElementConformsToProfile association class according to the Central Class Methodology. In some environments, an alternative method that relies on the Scoping Class Methodology through the scoping class instance may be desirable.

With CIM_ComputerSystem as the Scoping Class of this profile, the object diagram in Figure 2 shows how instances of CIM_RegisteredProfile may be used to identify the version of the *Diagnostics Profile* to which an instance of CIM_DiagnosticTest and its associated instances conform. In this example (using BaseServer as the system configuration), one instance of CIM_RegisteredProfile identifies the "*Base Server Profile v1.0"* and the other instance identifies the "*Diagnostics Profile v1.0*."

To support the Scoping Class Methodology for advertising profile implementation conformance, a CIM_DiagnosticTest instance is associated to an instance of the Scoping Class, CIM_ComputerSystem, through an instance of CIM_HostedService. This instance of CIM_ComputerSystem is advertised as being in implementation conformance with the *Base Server Profile v1.0* as indicated by the CIM_ElementConformsToProfile association to the "server" CIM_RegisteredProfile instance. The CIM_ReferencedProfile relationship between "server" and "diagnostic" places the CIM_DiagnosticTest instance within the scope of "diagnostic." Thus, the CIM_DiagnosticTest instance is conformant with the *Diagnostics Profile v1.0*.

To support the Central Class Methodology for advertising profile implementation conformance, a CIM_ElementConformsToProfile association is established between the CIM_DiagnosticTest central class instance and the instance of CIM_RegisteredProfile that represents the *Diagnostics Profile*.

For these methodologies to be successful, profiles for systems that can support diagnostics need to reference the *Diagnostics Profile*. In this example, the *Base Server Profile* would need to include the *Diagnostics Profile* in its "Related Profiles" table.

The CIM_ prefix has been omitted from the class names in Figure 2 for simplicity and readability.

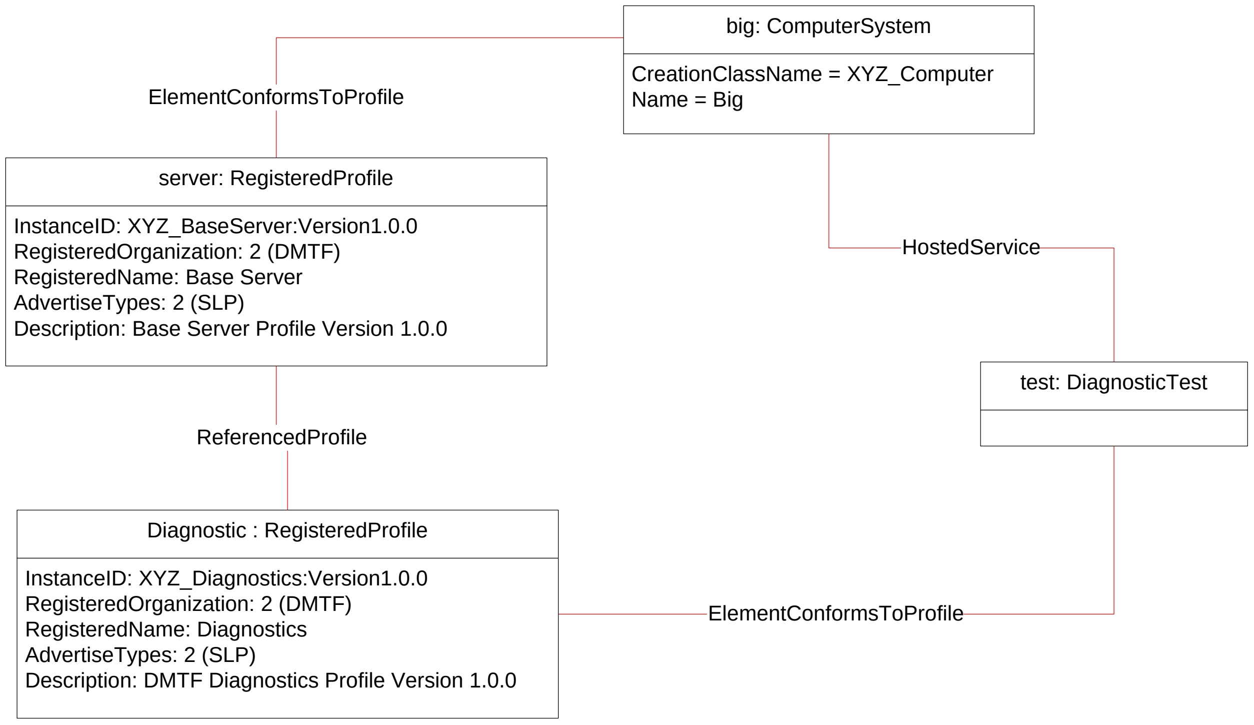
big: ComputerSystem
CreationClassName = XYZ_Computer
Name = Big
ElementConformsToProfile
server: RegisteredProfile
InstanceID: XYZ_BaseServer:Version1.0.0
RegisteredOrganization: 2 (DMTF)
RegisteredName: Base Server
AdvertiseTypes: 2 (SLP)
Description: Base Server Profile Version 1.0.0
HostedService
test: DiagnosticTest
ReferencedProfile
Diagnostic : RegisteredProfile
InstanceID: XYZ_Diagnostics:Version1.0.0
RegisteredOrganization: 2 (DMTF)
RegisteredName: Diagnostics
AdvertiseTypes: 2 (SLP)
Description: DMTF Diagnostics Profile Version 1.0.0
ElementConformsToProfile

Figure 2 – Registered Profile

9.2 Use Case Summary

Table 33 summarizes the use cases that are described in this section. The use cases are categorized and named, and references are provided to the body text that describes the use case.

Note: Although use case names follow the convention for naming classes, properties, and methods in the schema, this naming was done for readability only and does not imply any functionality attached to the name.

The CIM_ prefix has been omitted from the class names in the use cases for readability.

Table 33 – Diagnostics Profile Use Cases

Category	Name	Description
Discover Available Diagnostics See section 9.4.	GetAllDiagnostics	Find all diagnostics available on a system. See section 9.4.
	GetAllDiagnosticMEPairs	Find all diagnostic/managed elements pairs available on a system. See section 9.4.2.
	GetDiagnosticsForME	Find all the diagnostics available on a system, for a managed element. See section 9.4.3.
	GetMEsForDiagnostic	Find all the managed elements that support a particular diagnostic. See section 9.4.4.
	GetCapabilitiesOfDiagnostic	Find the capabilities of a particular diagnostic. See section 9.4.5.
	GetCharacteristicsOfDiagnostic	Find the characteristics of a particular diagnostic. See section 9.4.6.
	GetDiagnosticsWithCharacteristicsForME	Find all the diagnostics available on a system, for a managed element, with certain characteristics. See section 9.4.7.
	GetDiagnosticsWithCapabilitiesForME	Find all the diagnostics available on a system, for a managed element, with certain capabilities. See section 9.4.8.
	GetPackageSubtests	Find the subtests for a diagnostic test with the value of the DiagnosticTest.Characteristics property set to Is Package. See section 9.4.9.
Configure Diagnostic See section 9.5.	GetDefaultDiagnosticSettings	Find the default diagnostic settings for a diagnostic. See section 9.5.1.
	CreateDiagnosticSettings	Create a unique setting for a diagnostic. See section 9.5.2.
	GetDefaultJobSettings	Find the default job settings for a diagnostic. See section 9.5.3.
	CreateJobSettings	Create a unique setting for a diagnostic job. See section 9.5.4.
Execute and Control Diagnostic See section 9.6.	RunDiagnostic	Run a diagnostic with default and unique settings. See section 9.6.1.
	SuspendDiagnostic	Suspend a running diagnostic. See section 9.6.2.
	ResumeDiagnostic	Resume a running diagnostic. See section 9.6.3.
	AbortDiagnostic	Abort a running diagnostic. See section 9.6.4.
	KillDiagnostic	Abort a running diagnostic immediately, with no attempt to perform a clean shutdown. See section 9.6.5.

Category	Name	Description
Discover Diagnostic Executions See section 9.7.	GetAffectedMEs	Find all the managed elements affected by a running diagnostic. See section 9.7.1.
	GetAllDiagnosticExecutionsForME	Find all the diagnostic executions on a system for a managed element. See section 9.7.2.
	GetSpecificDiagnosticExecutions	Find all the executions of a specific diagnostic. See section 9.7.3.
	GetSpecificDiagnosticExecutionsForME	Find all the executions of a specific diagnostic for a particular managed element. See section 9.7.4.
Discover Diagnostic Results (in-progress and final) See section 9.8.	GetLogRecordsForDiagnostic	Find all the diagnostic log records for a particular diagnostic. See section 9.8.1.
	GetLogRecordsForME	Find all the diagnostic log records for a particular managed element. See section 9.8.2.
	GetLogRecordsForMEAndDiagnostic	Find all the diagnostic log records for a particular diagnostic run on a particular managed element. See section 9.8.3.
	GetDiagnosticExecutionResults	Find all diagnostic log records for a particular execution (job). See section 9.8.4.
	GetDiagnosticExecutionSettings	Find the settings used in a diagnostic execution. See section 9.8.5.
	GetDiagnosticProgress	Get the progress of a running diagnostic. See section 9.8.6.
	GetDiagnosticExecutionFinalResults	Find the diagnostic log record with final results for a particular execution (job). See section 9.8.7.

9.3 Diagnostic Services Object Diagram

Figure 3 is an object diagram for diagnostic services for a fictitious device called “Widget.” Only classes, properties, and methods that are of particular interest for the diagnostic model are shown. Refer to this diagram for the use cases in this section.

The CIM_ prefix has been omitted from the class names in the diagram for readability.

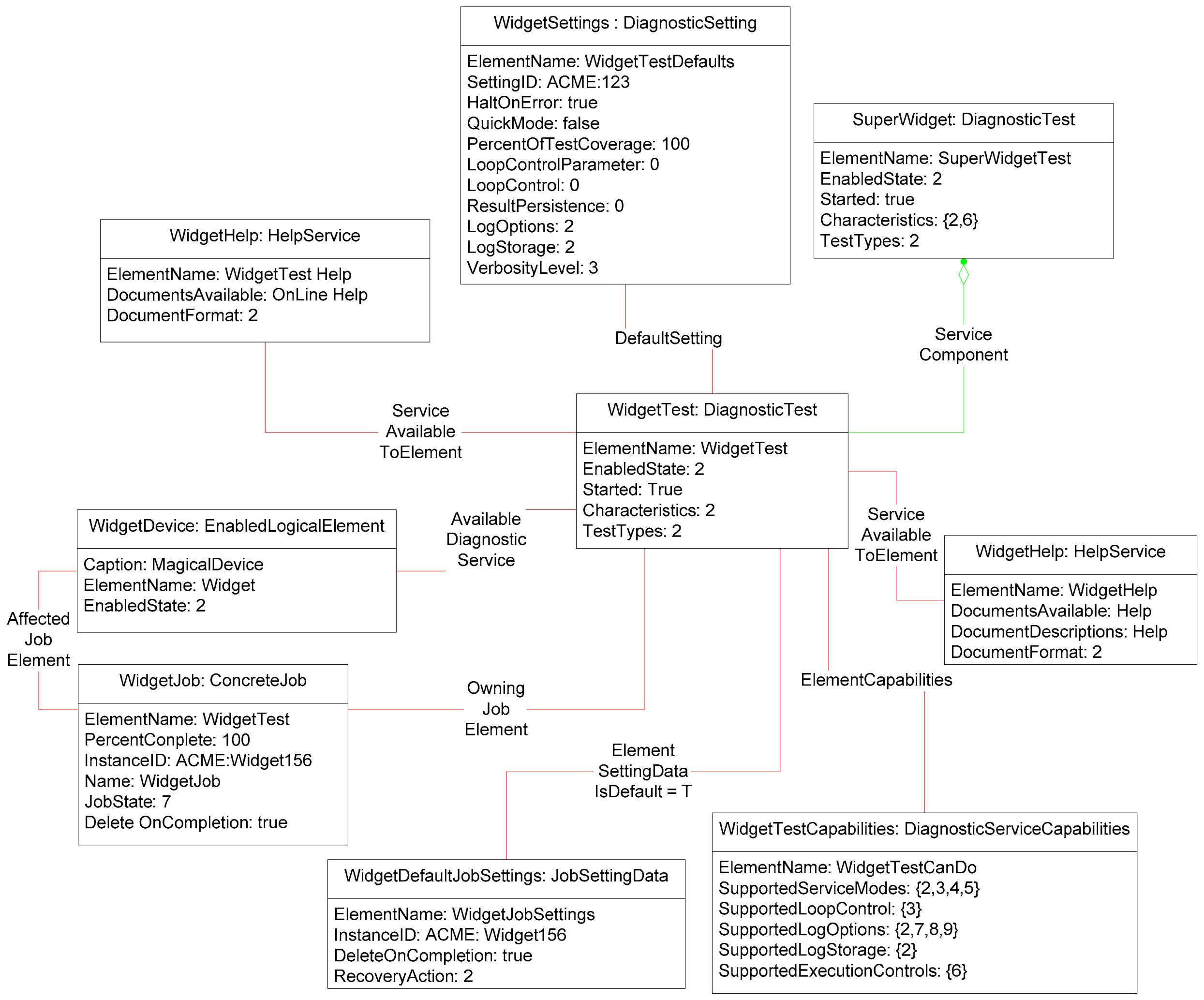
WidgetSettings : DiagnosticSetting
ElementName: WidgetTestDefaults
SettingID: ACME:123
HaltOnError: true
QuickMode: false
PercentOfTestCoverage: 100
LoopControlParameter: 0
LoopControl: 0
ResultPersistence: 0
LogOptions: 2
LogStorage: 2
VerbosityLevel: 3
SuperWidget: DiagnosticTest
ElementName: SuperWidgetTest
EnabledState: 2
Started: true
Characteristics: {2,6}
TestTypes: 2
WidgetHelp: HelpService
ElementName: WidgetTest Help
DocumentsAvailable: OnLine Help
DocumentFormat: 2
DefaultSetting
Service Component
Service Available ToElement
WidgetTest: DiagnosticTest
ElementName: WidgetTest
EnabledState: 2
Started: True
Characteristics: 2
TestTypes: 2
WidgetDevice: EnabledLogicalElement
Caption: MagicalDevice
ElementName: Widget
EnabledState: 2
Available Diagnostic Service
Service Available ToElement
WidgetHelp: HelpService
ElementName: WidgetHelp
DocumentsAvailable: Help
DocumentDescriptions: Help
DocumentFormat: 2
Affected Job Element
WidgetJob: ConcreteJob
ElementName: WidgetTest
PercentConplete: 100
InstanceID: ACME:Widget156
Name: WidgetJob
JobState: 7
Delete OnCompletion: true
Owning Job Element
ElementCapabilities
Element SettingData IsDefault = T
WidgetTestCapabilities: DiagnosticServiceCapabilities
ElementName: WidgetTestCanDo
SupportedServiceModes: {2,3,4,5}
SupportedLoopControl: {3}
SupportedLogOptions: {2,7,8,9}
SupportedLogStorage: {2}
SupportedExecutionControls: {6}
WidgetDefaultJobSettings: JobSettingData
ElementName: WidgetJobSettings
InstanceID: ACME: Widget156
DeleteOnCompletion: true
RecoveryAction: 2

Figure 3 – Diagnostic Services Object Diagram

9.4 Discover Available Diagnostics

The use cases in this section describe how the client can find available diagnostics. The CIM_ prefix has been omitted from the class names in the use cases for readability.

9.4.1 GetAllDiagnostics

The client can find all of the diagnostics that are available on a system as follows:

1) The client calls the EnumerateInstances (or EnumerateInstanceNames) operation using the DiagnosticTest class.
2) The operation returns DiagnosticTest instances that represent a diagnostic that is available on the system.

9.4.2 GetAllDiagnosticMEPairs

The client can find all of the diagnostics/managed element pairs that are available on a system as follows. Each pair comprises a diagnostic and a ManagedElement (device) that is supported by the diagnostic.

1) The client calls the EnumerateInstances (or EnumerateInstanceNames) operation using the AvailableDiagnosticService class.
2) The operation returns AvailableDiagnosticService instances that have a reference to the DiagnosticTest instance and another reference to the ManagedElement instance.

9.4.3 GetDiagnosticsForME

The client can find all of the diagnostics on a system that can be launched against a specific device (managed element) as follows. Assume that the client starts at a known ManagedElement instance, which represents the device to be tested.

1) From the ManagedElement instance, the client calls the Associators operation using AvailableDiagnosticService as the association class.
2) The operation returns DiagnosticTest instances that represent a diagnostic that can be launched against the ManagedElement.

9.4.4 GetMEsForDiagnostic

The client can find all managed elements (devices) that are supported by a specific diagnostic as follows. Assume that the client starts at a known DiagnosticTest instance.

1) From the DiagnosticTest instance, the client calls the Associators operation using AvailableDiagnosticService as the association class.
2) The operation returns ManagedElement instances that represent a device that is supported by the DiagnosticTest.

9.4.5 GetCapabilitiesOfDiagnostic

A diagnostic service publishes its support for various options—in particular, settings—through a DiagnosticServiceCapabilities instance. If a setting is supported, the client can assign it, usually to satisfy a user request. The client should be able to find the capabilities of a diagnostic as follows. Assume that the client starts at a known DiagnosticTest instance.

1) From the DiagnosticTest instance, the client calls the Associators operation using ElementCapabilities as the association class and DiagnosticServiceCapabilities as the result class.
2) The operation should return only one DiagnosticServiceCapabilities instance, which represents the diagnostic capabilities.

Note: Because the implementation of DiagnosticServiceCapabilities is optional, it may not be available. In this case, no assumptions should be made regarding the diagnostic capabilities.

9.4.6 GetCharacteristicsOfDiagnostic

The client can discover all of the characteristics (is destructive, is interactive, is synchronous, and so on) of a diagnostic. From the DiagnosticTest instance, the client reads just the Characteristics and OtherCharacteristicsDescriptions attributes, which contain the diagnostic characteristics.

9.4.7 GetDiagnosticswithCharacteristicsForME

The client can find all of the diagnostics that can be launched against a specific device (managed element) and have specific characteristics as follows. Assume that the client starts at a known ManagedElement instance, which represents the device to be tested.

1) The client discovers all of the diagnostics that are available for the specific ManagedElement. The GetDiagnosticsForME use case (section 9.4.3) describes the necessary steps.
2) For each DiagnosticTest instance, the client checks the diagnostic characteristics. The GetCharacteristicsOfDiagnostic use case (section 9.4.6) describes the necessary steps.
3) If the characteristics of the DiagnosticTest instance match the desired characteristics, the DiagnosticTest instance is the one desired.

9.4.8 GetDiagnosticswithCapabilitiesForME

The client can find all of the diagnostics that can be launched against a specific device (managed element) and have specific capabilities as follows. Assume that the client starts at a known ManagedElement instance, which represents the device to be tested.

1) The client discovers all of the diagnostics that are available for the specific ManagedElement. The GetDiagnosticsForME use case (section 9.4.3) describes the necessary steps.
2) For each DiagnosticTest instance, the client checks the diagnostic capabilities. The GetCapabilitiesOfDiagnostic use case (section 9.4.5) describes the necessary steps.
3) If the capabilities of the DiagnosticTest instance match the desired capabilities, the DiagnosticTest instance is the one desired.

9.4.9 GetPackageSubtests

The client can find the subtests for a diagnostic test with the IsPackage value set in the DiagnosticTest.Characteristics property, using the following procedure. Assume that the client starts at a known DiagnosticTest instance.

1) The client checks the DiagnosticTest.Characteristics property for the IsPackage value.
2) If the IsPackage value is present, the client calls the Associators operation using ServiceComponent as the association class and DiagnosticTest as the result class.
3) The operation returns the DiagnosticTest instances that are subtests of the known DiagnosticTest.

9.5 Configure Diagnostic

The use cases in this section describe how the client can find and create settings for diagnostics. The CIM_ prefix has been omitted from the class names in the use cases for readability.

9.5.1 GetDefaultDiagnosticSettings

The client can obtain the default settings for a diagnostic service as follows. Assume that the client starts at a known DiagnosticTest instance.

1) From the DiagnosticTest instance, the client calls the Associators operation using DefaultSetting as the association class and DiagnosticSetting as the result class.

2) The operation should return only one DiagnosticSetting instance, which represents the default settings of the diagnostic.

Note: Because the implementation of DiagnosticSetting is optional, it may not be available. In this case, no assumptions should be made regarding the default settings of diagnostic.

9.5.2 CreateDiagnosticSettings

The client may modify the diagnostic settings as follows if it wants to use settings different than the default settings. Note that the diagnostic default settings are represented by a DiagnosticSetting subclass that may have extensions. If the client is aware of the extensions, they may be modified as well. If the client is unaware, the default values should be used. Assume that the client starts at a known DiagnosticTest instance.

1) The client discovers the diagnostic capabilities of the DiagnosticTest instance. The GetCapabilitiesOfDiagnostic use case (section 9.4.5) describes the necessary steps. If no capability information is available, the client shall use the default settings (empty string or NULL) because it cannot assume any diagnostic capability.

2) The client discovers the diagnostic default settings of the DiagnosticTest instance. The GetDefaultDiagnosticSettings use case (section 9.5.1) describes the necessary steps. If no instance is returned, the client shall use the default settings (empty string or NULL) because it cannot assume support for any diagnostic setting..

3) The client modifies the created DiagnosticSetting instance as necessary. However, the client should consider the diagnostic capabilities during the changes and shall modify the SettingID attribute.

4) Finally, the client calls CreateInstance operation passing the instance modified on step 3.

9.5.3 GetDefaultJobSettings

The client can obtain the default job settings for a diagnostic service as follows. Assume that the client starts at a known DiagnosticTest instance.

1) From the DiagnosticTest instance, the client calls the Associators operation using ElementSettingData as the association class and JobSettingData as the result class. The operation returns JobSettingData instances.

2) For each JobSettingData instance, the client calls the References operation using ElementSettingData as the result class. The operation returns ElementSettingData instances.

3) For each ElementSettingData instance, the client determines whether the value of the ElementSettingData.ManagedElement property matches the reference to the instance of DiagnosticTest and the value of the ElementSettingData.IsDefault property is 1 (“Is Default”). If so, the JobSettingData instance represents the default job settings. The name of this JobSettingData instance may also be retrieved from ElementSettingData.SettingData property.

Note: Because the implementation of JobSettingData is optional, it may not be available. In this case, no assumptions should be made regarding the default job settings of diagnostic.

9.5.4 CreateJobSettings

The client can modify the diagnostic job settings as follows if it wants to use settings different than the default job settings. Note that the diagnostic default job settings are represented by a JobSettingData subclass that may have extensions. If the client is aware of the extensions, they may be modified as well. If the client is unaware, the default values should be used. Assume that the client starts at a known DiagnosticTest instance.

1) The client discovers the diagnostic capabilities of the DiagnosticTest instance. The GetCapabilitiesOfDiagnostic use case (section 9.4.5) describes the necessary steps. If no capability information is available, the client shall use the default settings (empty string or NULL) because it cannot assume any diagnostic capability.
2) The client discovers the diagnostic default settings of the DiagnosticTest instance. The GetDefaultJobSettings use case (section 9.5.3) describes the necessary steps. If no instance is returned, the client shall use the default settings (empty string or NULL) because it cannot assume support for any job setting.
3) The client modifies the created JobSettingData instance as necessary. However, the client should consider the diagnostic capabilities during the changes and shall modify the InstanceID attribute.
4) Finally, the client calls CreateInstance operation passing the instance modified on step 3.

9.6 Execute and Control Diagnostic

The RunDiagnostic( ) method is invoked to start the diagnostic service. Input parameters are the ManagedElement being tested and the settings (optional). A reference to a ConcreteJob instance is returned.

An instance of ConcreteJob is created by the diagnostic provider to allow monitoring and control of the running service. By invoking the RequestStateChange method, the client may start, stop, suspend, and resume the job. By inspecting the value of PercentComplete, the client may determine the job's progress.

The ManagedElement being tested and the DiagnosticTest instance that launched the test are related to the job instance through the OwningJobElement and the AffectedJobElement associations. The client may find jobs associated with services or managed elements of interest by using these associations.

Figure 4 is an object diagram that shows the state of instances when a DiagnosticTest RunDiagnostic( ) method has been called three times. Two of the times were to run a test on the same device, ManagedElement2.

Note: Not all diagnostic tests are capable of running on the same device simultaneously. If this had been the case in this example, the DiagnosticTest would have returned an error on the second RunDiagnostic( ) method call to run a test on ManagedElement2.

Note: Diagnostic tests that do not return a reference to a ConcreteJob instance are assumed to have completed the execution of the test upon return from the RunDiagnostic() method and thus there is no need for a ConcreteJob reference to provide execution status or control execution. In this case, only the RunDiagnostic use case is valid.

The CIM_ prefix has been omitted from the class names in the diagram and the use cases for readability.

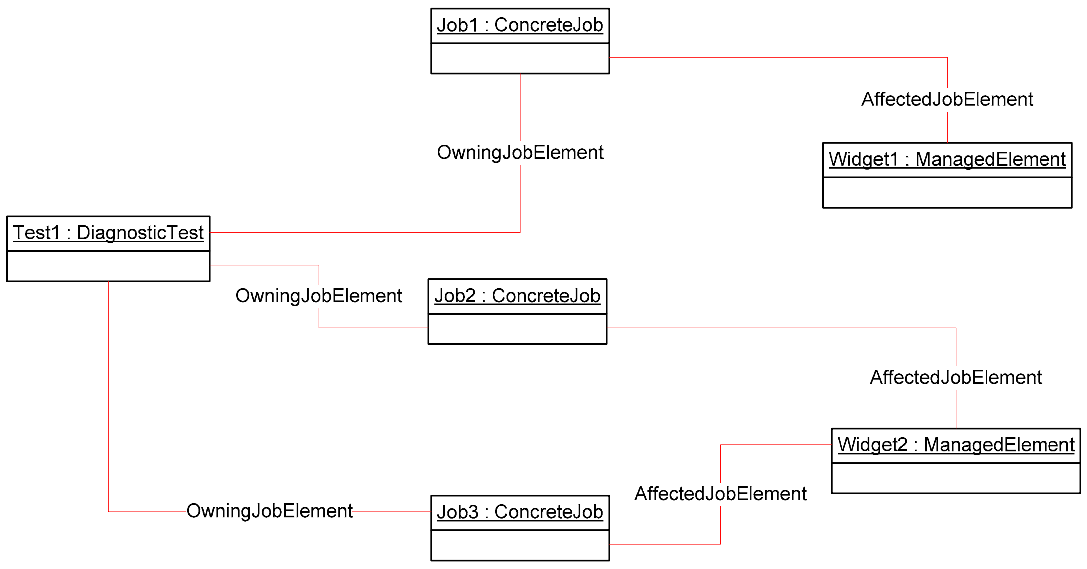

Figure 4 – Job Example

9.6.1 RunDiagnostic

The client can run a diagnostic with default and unique settings as follows. (See section 9.4 for use cases related to finding diagnostics that can be initiated. See section 9.5 for use cases related to creating and modifying diagnostic settings to configure diagnostic execution.)

1) The client calls the RunDiagnostic() method, passing in references of DiagnosticSetting and JobSetting to use to execute the test as well as the reference to the ManagedElement to test. If the client passes in NULL or an empty string for these classes, the default values are used.

2) The diagnostic service creates a Job instance to represent that test running on that managed element and returns a reference to it in the return call from RunDiagnostic(). In addition, the diagnostic service creates the OwningJobElement association between the Job and the Service and the AffectedJobElement association between the Job and the ManagedElement.

3) If the diagnostic service does not create and return a Job instance, the test is assumed to have completed execution upon return from the RunDiagnostic() method.

9.6.2 SuspendDiagnostic

The client can suspend the execution of the test by using the RequestStateChange() method call on the Job instance that is returned from the RunDiagnostic() method, as shown in the following procedure. Assume that the client starts at a known DiagnosticTest instance.

1) The client follows the ElementCapabilities association from the DiagnosticTest to the DiagnosticServiceCapabilities for the service.

2) The client checks the DiagnosticServiceCapabilities.SupportedExecutionControls() property for the value of “Suspend Job”. If the value exists, the Job supports suspending.

3) The client finds the appropriate Job instances. The GetSpecificDiagnosticExecutions use case (section 9.7.3) describes the necessary steps.

4) The client calls the RequestStateChange() method, passing in a RequestedState value of “Suspend”.

5) When the transition completes successfully, the ConcreteJob that represents the test will set the value of the JobState property to “Suspended” and set the value of TimeOfLastStateChange to the current time.

9.6.3 ResumeDiagnostic

The client can resume the execution of a test by using the RequestStateChange() method call on the Job instance that is returned from the RunDiagnostic() method, as shown in the following procedure. Assume that the client starts at a known DiagnosticTest instance.

1) The client follows the ElementCapabilities association from the DiagnosticTest to the DiagnosticServiceCapabilities for the service.
2) The client checks the DiagnosticServiceCapabilities.SupportedExecutionControls() property for the value of "Resume Job". If the value exists, the Job supports resuming.
3) The client finds the appropriate Job instances. The GetSpecificDiagnosticExecutions use case (section 9.7.3) describes the necessary steps.
4) The client calls the RequestStateChange() method, passing in a RequestedState value of "Start".
5) When the transition completes successfully, the ConcreteJob that represents the test will set the value of the JobState property to "Running" and set the value of TimeOfLastStateChange to the current time.

Note: The JobState property may transition to "Starting" before the final transition to "Running".

9.6.4 AbortDiagnostic

The client can cleanly abort the execution of a test by using the RequestStateChange() method call on the Job instance that is returned from the RunDiagnostic() method, as shown in the following procedure. Assume that the client starts at a known DiagnosticTest instance.

1) The client follows the ElementCapabilities association from the DiagnosticTest to the DiagnosticServiceCapabilities for the service.
2) The client checks the DiagnosticServiceCapabilities.SupportedExecutionControls() property for the value of "Terminate Job". If the value exists, the Job supports termination.
3) The client finds the appropriate Job instances. The GetSpecificDiagnosticExecutions use case (section 9.7.3) describes the necessary steps.
4) The client calls the RequestStateChange() method, passing in a RequestedState value of "Terminate".
5) When the transition completes successfully, the ConcreteJob that represents the test will set the value of the EnabledState property to "Terminated" and set the value of TimeOfLastStateChange to the current time.

Note: The JobState property may transition to "Shutting Down" before the final transition to "Terminated".

9.6.5 KillDiagnostic

The client can immediately abort the execution of a test, with no attempt to perform a clean shutdown, by using the RequestStateChange() method call on the Job instance that is returned from the RunDiagnostic() method, as shown in the following procedure. Assume that the client starts at a known DiagnosticTest instance.

1) The client follows the ElementCapabilities association from the DiagnosticTest to the DiagnosticServiceCapabilities for the service.
2) The client checks the DiagnosticServiceCapabilities.SupportedExecutionControls() property for the value of "Kill Job". If the value exists, the Job supports kill.
3) The client finds the appropriate Job instances. The GetSpecificDiagnosticExecutions use case (section 9.7.3) describes the necessary steps.

4) The client calls the RequestStateChange( ) method, passing in a RequestedState value of “Kill”.

5) When the transition completes successfully, the ConcreteJob that represents the test will set the value of the EnabledState property to “Killed” and set the value of TimeOfLastStateChange to the current time.

9.7 Discover Diagnostic Executions

In the following use cases, the term *execution* refers to an instance of the ConcreteJob class created to control a diagnostic service that was started on a managed element. The job may be in any of the states represented by the JobState property value, not necessarily active and running. The use cases in this section only apply to implementations that return a reference to ConcreteJob upon execution of the RunDiagnostic() method.

The CIM_ prefix has been omitted from the class names in the use cases for readability.

9.7.1 GetAffectedMEs

The client can find all of the managed elements that are affected by a diagnostic execution as follows. Assume that the client starts at a known DiagnosticTest instance.

1) From the DiagnosticTest instance, the client calls the Associators operation using OwningJobElement as the association class and ConcreteJob as the result class. The operation returns the ConcreteJob instances launched by the DiagnosticTest.

2) For each ConcreteJob instance, the client calls the Associators operation using AffectedJobElement as the association class and ManagedElement as the result class. The operation returns the ManagedElement instances that this DiagnosticTest affects.

Note: This use case depends on the optional AffectedJobElement association. If that association does not exist, this use case is invalid.

9.7.2 GetAllDiagnosticExecutionsForME

The client can find all of the diagnostic executions on a system for a managed element as follows. Assume that the client starts at a known ManagedElement instance.

1) From the ManagedElement instance, the client calls the Associators operation using AffectedJobElement as the association class. The operation returns the ConcreteJob instances launched against this ManagedElement.

2) For each ConcreteJob instance, the client calls the AssociatorNames operation using OwningJobElement as the association class and DiagnosticTest as the result class. The operation returns the instance paths to the DiagnosticTest instances that launched the ConcreteJob against this ManagedElement.

3) Each ConcreteJob instance that is associated with a DiagnosticTest represents an execution of a diagnostic service on that ManagedElement.

Note: This use case depends on the optional AffectedJobElement association. If that association does not exist, this use case is invalid.

9.7.3 GetSpecificDiagnosticExecutions

The client can find all of the executions of a specific diagnostic as follows. Assume that the client starts at a known DiagnosticTest instance.

1) From the DiagnosticTest instance, the client calls the Associators operation using OwningJobElement as the association class. The operation returns the ConcreteJob instances launched by the DiagnosticTest.

2) Each ConcreteJob instance represents an execution of that diagnostic service.

9.7.4 GetSpecificDiagnosticExecutionsForME

The client can find all of the executions of a specific diagnostic for a particular managed element using either of the following methods:

- starting at the known ManagedElement instance
- starting at the known DiagnosticTest instance

9.7.4.1 Starting at the Managed Element

Note: This use case depends on the optional AffectedJobElement association. If that association does not exist, this use case is invalid.

Assume that the client starts at the known ManagedElement instance and knows the particular DiagnosticTest instance.

1) From the ManagedElement instance, the client calls the Associators operation using AffectedJobElement as the association class and ConcreteJob as the result class. The operation returns the ConcreteJob instances that are running against this ManagedElement.
2) For each ConcreteJob instance, the client calls the AssociatorNames operation using OwningJobElement as the association class and DiagnosticTest as the result class. The operation returns the instance paths to the DiagnosticTest instances that launched the ConcreteJob instances against this ManagedElement.
3) For each DiagnosticTest instance path returned, the client determines if it is the instance path of the known DiagnosticTest instance. If the instance path matches, the ConcreteJob instance represents an execution of that diagnostic service on that ManagedElement.

9.7.4.2 Starting at the DiagnosticTest

Note: This use case depends on the optional AffectedJobElement association. If that association does not exist, this use case is invalid.

Assume that the client starts at the known DiagnosticTest instance and knows the particular ManagedElement instance.

1) From the DiagnosticTest instance, the client calls the Associators operation using OwningJobElement as the association class and ConcreteJob as the result class. The operation returns the ConcreteJob instances launched by the DiagnosticTest.
2) For each ConcreteJob instance, the client calls the AssociatorNames operation using AffectedJobElement as the association class and ManagedElement as the result class. The operation returns the instance paths to the ManagedElement instances against which this DiagnosticTest launched the ConcreteJob instances.
3) For each ManagedElement instance path returned, the client determines if it is the instance path of the known ManagedElement instance. If the instance path matches, the ConcreteJob instance represents an execution of that diagnostic service on that ManagedElement.

9.8 Discover Diagnostic Results (In Progress and Final)

In the following use cases, the term *execution* refers to an instance of the ConcreteJob class created to control a diagnostic service that was started on a managed element. The job may be in any of the states represented by the JobState property value, not necessarily active and running. Some of the use cases in this section only apply to implementations that return a reference to ConcreteJob upon execution of the RunDiagnostic() method.

Figure 5 is an object diagram that represents the results logging process for a diagnostic service on a fictitious device called “Widget”. Only classes, properties, and methods that are of particular interest for the diagnostic model are shown.

Figure 5 shows the logging implementation, using the DiagnosticsLog class. DiagnosticsLog is a special subclass of RecordLog that supports a standard mechanism for organizing and retrieving the records that diagnostics services generate. Use of this common logging mechanism can substantially increase interoperability and simplify client design.

The diagnostic provider will store the results of running the diagnostic in the manner selected through the LogStorage setting. The most common mechanism is for the provider to create instances of DiagnosticRecord to record the results and status of running diagnostic services. DiagnosticRecord has two subclasses: DiagnosticServiceRecord for recording test results, and DiagnosticSettingRecord for preserving the test settings. The providers for these classes can implement ExecQuery to simplify the retrieval of records. The use cases below provide alternatives that use ExecQuery as well as do not.

The records are aggregated to a log by the LogManagesRecord association.

The CIM_ prefix has been omitted from the class names in the diagram and use cases for readability.

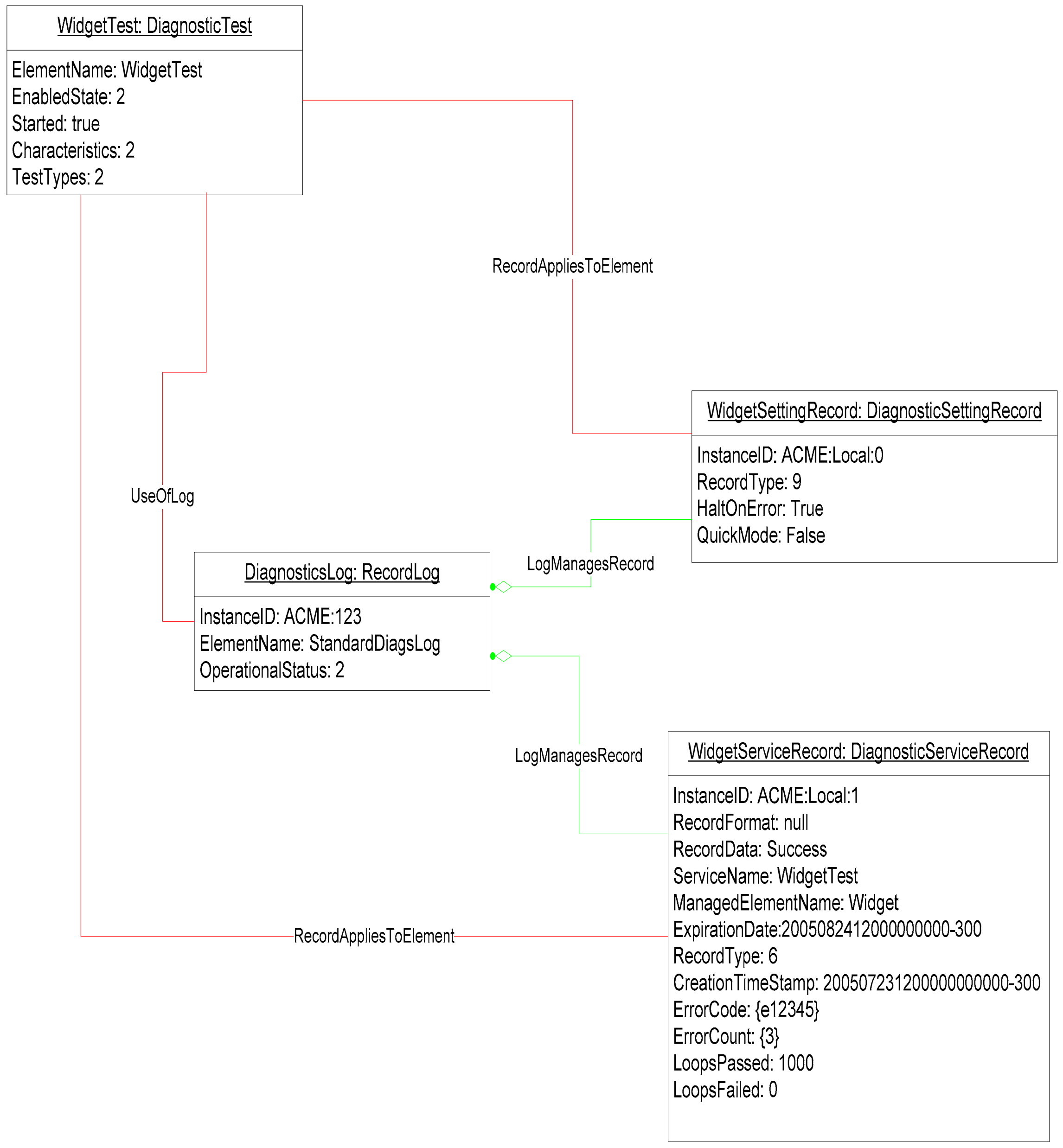

Figure 5 – Diagnostic Logging Object Diagram

9.8.1 GetLogRecordsForDiagnostic

The client can find all of the diagnostic log records for a particular diagnostic as follows. Assume that the client starts at the known DiagnosticTest instance and that the DiagnosticRecord.ServiceName property is implemented according to this Profile.

1) The client calls the ExecQuery operation as follows:

```
SELECT * FROM CIM_DiagnosticRecord
WHERE ServiceName = '<DiagnosticTest.Name>'
```

2) The operation returns the DiagnosticRecord instances created for the specific DiagnosticTest, independently if they are related to different managed elements or executions.

An alternate method without using ExecQuery can be done by the following:

Assume that the client starts at the known DiagnosticTest instance.

1) From the DiagnosticTest instance, the client calls the Associators operation using UseOfLog as the association class and DiagnosticsLog as the result class. The operation returns the DiagnosticsLog instances that contain records for the DiagnosticTest.
2) For each DiagnosticsLog instance, the client calls the Associators operation using LogManagesRecord as the association class and DiagnosticRecord as the result class. The operation returns the DiagnosticRecord instances in the Log.
3) For each returned instance, the client compares DiagnosticRecord.ServiceName with DiagnosticTest.Name to determine if the instance is one created for the specific DiagnosticTest.

9.8.2 GetLogRecordsForME

The client can find all of the diagnostic log records for a particular managed element as follows. Assume that the client starts at the known ManagedElement instance and that the DiagnosticRecord.ManagedElementName property is implemented according to this Profile.

1) The client calls the ExecQuery operation as follows:

```
SELECT * FROM CIM_DiagnosticRecord
WHERE ManagedElementName = '<ManagedElement.ElementName>'
```

2) The operation returns the DiagnosticRecord instances created for the specific ManagedElement, independently if they are related to different diagnostics or executions.

An alternate method without using ExecQuery can be done by the following:

Assume that the client starts at the known ManagedElement instance.

1) From the ManagedElement instance, the client calls the Associators operation using ServiceAvailableToElement as the association class and DiagnosticTest as the result class. The operation returns the DiagnosticTest instances for the ManagedElement.
2) For each DiagnosticTest instance, the client calls the Associators operation using UseOfLog as the association class and DiagnosticsLog as the result class. The operation returns the DiagnosticsLog instances that contain records for the DiagnosticTest.
3) For each DiagnosticsLog instance, the client calls the Associators operation using LogManagesRecord as the association class and DiagnosticRecord as the result class. The operation returns the DiagnosticRecord instances in the Log.
4) For each returned instance, the client compares DiagnosticRecord.ManagedElementName with ManagedElement.ElementName to determine if the instance is one created for the specific ManagedElement.

9.8.3 GetLogRecordsForMEAndDiagnostic

The client can find all of the diagnostic log records for a particular diagnostic run on a particular managed element as follows.

Assume that the client starts at the known DiagnosticTest and ManagedElement instances and that the DiagnosticRecord.ServiceName and DiagnosticRecord.ManagedElementName properties are implemented according to this Profile.

1) The client calls the ExecQuery operation as follows:

```
SELECT * FROM CIM_DiagnosticRecord
WHERE ManagedElementName = '<ManagedElement.ElementName>' and ServiceName =
'<DiagnosticTest.Name>'
```

2) The operation returns the DiagnosticRecord instances created for the specific ManagedElement and DiagnosticTest, independently if they were created in different executions.

An alternate method without using ExecQuery can be done by the following:

Assume that the client starts at the known DiagnosticTest instance.

1) From the DiagnosticTest instance, the client calls the Associators operation using UseOfLog as the association class and DiagnosticsLog as the result class. The operation returns the DiagnosticsLog instances that contain records for the DiagnosticTest.

2) For each DiagnosticsLog instance, the client calls the Associators operation using LogManagesRecord as the association class and DiagnosticRecord as the result class. The operation returns the DiagnosticRecord instances in the Log.

3) For each returned instance, the client compares DiagnosticRecord.ServiceName with DiagnosticTest.Name and DiagnosticRecord.ManagedElementName with ManagedElement.ElementNameto determine if the instance is one created for the specific DiagnosticTest and ManagedElement.

9.8.4 GetDiagnosticExecutionResults

The client can find all diagnostic log records for a particular execution (job) as follows.

Assume that the client starts at the known ConcreteJob instance and that the DiagnosticRecord.InstanceID property follows the format defined in this Profile (CIM_DiagnosticRecord.InstanceID *should* be <ConcreteteJob.InstanceID>:<n>). This use case is also applicable after the job completes and is removed if the client knows the original ConcreteJob.InstanceID.

1) The client calls the ExecQuery operation as follows:

```
SELECT * FROM CIM_DiagnosticRecord
WHERE InstanceID LIKE '<ConcreteJob.InstanceID>%'
```

2) The operation returns the DiagnosticRecord instances created for the specific ConcreteJob.

An alternate method without using ExecQuery can be done as follows:

Assume that the client starts at the known DiagnosticTest instance.

1) From the DiagnosticTest instance, the client calls the Associators operation using UseOfLog as the association class and DiagnosticsLog as the result class. The operation returns the DiagnosticsLog instances that contain records for the DiagnosticTest.

2) For each DiagnosticsLog instance, the client calls the Associators operation using LogManagesRecord as the association class and DiagnosticRecord as the result class. The operation returns the DiagnosticRecord instances in the Log.

3) For each returned instance, the client compares portion of DiagnosticRecord.InstanceID that contains the ConcreteJob.InstanceID with ConcreteJob.InstanceID to determine if the instance is one created for the specific execution of the DiagnosticTest.

An alternate method without using ExecQuery can be done as follows:

Note: This alternative use case depends upon the implementation of DiagnosticSettingRecord and CorrespondingSettingsRecord.

Assume that the client starts at the known DiagnosticTest instance.

1) From the DiagnosticTest instance, the client calls the Associators operation using UseOfLog as the association class and DiagnosticsLog as the result class. The operation returns the DiagnosticsLog instances that contain records for the DiagnosticTest.

2) For each DiagnosticsLog instance, the client calls the Associators operation using LogManagesRecord as the association class and DiagnosticSettingRecord as the result class. The operation returns the DiagnosticSettingRecord instances in the Log.

3) For each returned instance, the client compares portion of DiagnosticSettingRecord.InstanceID that contains the ConcreteJob.InstanceID with ConcreteJob.InstanceID to determine if the instance is the one created for the specific execution of the DiagnosticTest.

4) From the DiagnosticSettingRecord instance, the client calls the Associators operation using CorrespondingSettingsRecord as the association class and DiagnosticServiceRecord as the result class. The operation returns the DiagnosticServiceRecord instances created for the specific execution of the DiagnosticTest

Note: All these alternatives only apply to implementations that return a reference to ConcreteJob from the RunDiagnostic() method.

9.8.5 GetDiagnosticExecutionSettings

The client can find the settings used to execute a diagnostic as follows.

Assume that the client starts at the known ConcreteJob instance and that the DiagnosticRecord.InstanceID property follows the format defined in this Profile (CIM_DiagnosticRecord.InstanceID *should* be <ConcreteteJob.InstanceID>:<n>). This use case is also applicable after the job completes and is removed if the client knows the original ConcreteJob.InstanceID.

1) The client calls the ExecQuery operation as follows:

```
SELECT * FROM CIM_DiagnosticSettingRecord
WHERE InstanceID LIKE '<ConcreteJob.InstanceID>%'
```

2) The operation returns the DiagnosticSettingRecord instance created for the specific ConcreteJob.

3) The client reads the DiagnosticSettingRecord properties, such as HaltOnError or QuickMode, which are a copy of the properties from the DiagnosticSetting instance that passed as a parameter in the RunDiagnostic() method.

An alternate method without using ExecQuery can be done as follows:

Assume that the client starts at the known DiagnosticTest instance.

1) From the DiagnosticTest instance, the client calls the Associators operation using UseOfLog as the association class and DiagnosticsLog as the result class. The operation returns the DiagnosticsLog instances that contain records for the DiagnosticTest.

2) For each DiagnosticsLog instance, the client calls the Associators operation using LogManagesRecord as the association class and DiagnosticSettingRecord as the result class. The operation returns the DiagnosticSettingRecord instances in the Log.

3) For each returned instance, the client compares portion of DiagnosticSettingRecord.InstanceID that contains the ConcreteJob.InstanceID with ConcreteJob.InstanceID to determine if the instance is the one created for the specific execution of the DiagnosticTest.

Note: This use case only applies to implementations that return a reference to ConcreteJob from the RunDiagnostic() method.

9.8.6 GetDiagnosticProgress

The client can get the progress of a running diagnostic as follows.

The client may poll the ConcreteJob.PercentComplete property to determine test progress or register for an indication that this property has changed. The value of this property shall be kept current to be useful.

Service providers should update this property within one second of becoming aware of a progress change.

1) The client may use any of the Discover Diagnostic Execution use cases (section 9.7) to find the desired ConcreteJob instances.

2) The client reads the ConcreteJob.PercentComplete property to determine test progress.

Assuming CIM_InstModification indications are supported, the client may register to receive indications when the particular ConcreteJob.PercentComplete property changes value.

1) The client can use any of the Discover Diagnostic Execution use cases (section 9.7) to find the desired ConcreteJob instances.

2) The client can register to receive a CIM_InstModification indication by creating an indication subscription using the following CIM_IndicationFilter.Query:

```
SELECT * FROM CIM_InstModification
WHERE "SourceInstance.ISA("CIM_ConcreteJob") and SourceInstance.InstanceID =
<ConcreteJob.InstanceID> and PreviousInstance.PercentComplete <>
SourceInstance.PercentComplete
```

3) The indication received will notify the client that the PercentComplete property for the specific ConcreteJob has changed. The client can use the SourceInstance property in the indication to see the actual PercentComplete value to determine test progress.

Note: This use case only applies to implementations that return a reference to ConcreteJob from the RunDiagnostic() method.

9.8.7 GetDiagnosticExecutionFinalResults

The client can find the final results log record for a particular execution (job) as follows.

Assume that the client starts at the known ConcreteJob instance and that the DiagnosticRecord.InstanceID property follows the format defined in this Profile (CIM_DiagnosticRecord.InstanceID *should* be <ConcreteteJob.InstanceID>:<n>). This use case is also applicable after the job completes and is removed if the client knows the original ConcreteJob.InstanceID.

1) Client determines that the Job has completed by examining ConcreteJob.JobState. Value should be Completed, Terminated or Killed.

2) The client uses GetDiagnosticExecutionResults to get the DiagnosticServiceRecord instances for the particular execution of the DiagnosticTest.

3) For each returned DiagnosticServiceRecord, the client reads the DiagnosticServiceRecord.RecordType to find the instance with RecordType = "Results" to find the final results record.

4) Client reads properties of interest to determine the final result, such as LoopsPassed, LoopsFailed, ErrorCode[], ErrorCount[] and RecordData.

Note: This use case only applies to implementations that return a reference to ConcreteJob from the RunDiagnostic() method.

10 CIM Elements

Table 34 shows the instances of CIM Elements for this Profile. Instances of the CIM Elements shall be implemented as described in Table 34. Section 8 ("Methods") may impose additional requirements on these elements.

1327 **Table 34 – CIM Elements: Diagnostics Profile**

Element Name	Requirement	Description
Classes		
CIM_AffectedJobElement	Optional	Association to link a job to a managed element See section 10.1.
CIM_AvailableDiagnosticService	Mandatory	Association to link diagnostic services which can be launched against managed elements See section 10.2.
CIM_ConcreteJob	Mandatory	Used by the client to monitor and control the execution of a diagnostic service See section 10.3.
CIM_DiagnosticsLog	Optional	Although several legitimate mechanisms for logging results exist (see CIM_DiagnosticSetting.LogStorage), aggregation of diagnostic records to a diagnostic log is expected. See section 10.4.
CIM_DiagnosticServiceCapabilities	Conditional	This class is mandatory if CIM_DiagnosticSetting is implemented and non-default settings are supported. See sections 7.2 and 10.5.
CIM_DiagnosticServiceRecord	Conditional	Records that contain the results of running a diagnostic service. If CIM_DiagnosticsLog is implemented, this class is mandatory. See section 10.6.
CIM_DiagnosticSetting	Optional	See section 10.7.
CIM_DiagnosticSettingRecord	Conditional	Records that contain the settings that were used by the diagnostic service. If CIM_DiagnosticSetting and CM_DiagnosticsLog are implemented, this class is Mandatory. See section 10.8.
CIM_DiagnosticTest	Mandatory	See section 10.9.
CIM_ElementCapabilities	Optional	Association to link a Capabilities object to a diagnostic service. If Capabilities is implemented, this association is Mandatory. See section 10.10.
CIM_DefaultSetting (DiagnosticSetting)	Optional	If CIM_DiagnosticSetting is implemented, this association is Mandatory. See section 10.11.
CIM_DefaultSetting (JobSettingData)	Optional	See section 10.12
CIM_ElementSettingData	Optional	If CIM_JobSettingData is implemented, this association is Mandatory. See section 10.21.
CIM_HelpService	Optional	See section 10.13.

Element Name	Requirement	Description
CIM_JobSettingData	Optional	See section 10.14.
CIM_LogManagesRecord	Optional	If CIM_DiagnosticsLog is implemented, this class is Mandatory. See section 10.15.
CIM_OwningJobElement	Mandatory	Association to link a job to a diagnostic service See section 10.16.
CIM_RecordAppliesToElement	Optional	Association to link records to the managed element to which they apply See section 10.17.
CIM_ServiceAffectsElement	Optional	See section 10.18.
CIM_ServiceAvailableToElement	Optional	See section 10.19.
CIM_ServiceComponent	Optional	Association to link a packaging test to its subtests. If a CIM_DiagnosticTest.Characteristic property contains the IsPackage value and the subtests are also instances of CIM_DiagnosticTest, this association shall be used to associate those subtests to the packaging CIM_DiagnosticTest. See section 10.20.
CIM_UseOfLog	Optional	See section 10.22.
CIM_CorrespondingSettingsRecord	Optional	If CIM_DiagnosticsLog and CIM_DiagnosticSetting are implemented, this class is Mandatory. See section 10.23.
CIM_HostedService (DiagnosticTest)	Mandatory	The association between CIM_DiagnosticTest and CIM_ComputerSystem instance is Mandatory. See section 10.24.
CIM_HostedService (HelpService)	Optional	If the HelpService class is implemented for the associated system, this class is mandatory. See section 10.25.
CIM_RegisteredProfile	Mandatory	See section 10.26.
Indications		
None defined in this Profile		

10.1 CIM_AffectedJobElement

CIM_AffectedJobElement is used to associate a job with its affected managed elements (devices). Table 35 provides information about the properties of CIM_AffectedJobElement.

Table 35 – Class: CIM_AffectedJobElement

Properties	Requirement	Notes
AffectedElement	Mandatory	**Key** This property shall be a reference to an instance of CIM_ManagedElement.
AffectingElement	Mandatory	**Key** This property shall be a reference to an instance of CIM_ConcreteJob.

10.2 CIM_AvailableDiagnosticService

CIM_AvailableDiagnosticService is used to discover the diagnostic services that are installed for a particular managed element. Table 36 provides information about the properties of CIM_AvailableDiagnosticService.

Table 36 – Class: CIM_AvailableDiagnosticService

Properties	Requirement	Notes
ServiceProvided	Mandatory	**Key** This property shall be a reference to an instance of CIM_DiagnosticService.
UserOfService	Mandatory	**Key** This property shall be a reference to an instance of CIM_ManagedElement.

10.3 CIM_ConcreteJob

Each successful RunDiagnostic( ) call will return a CIM_ConcreteJob instance. Each CIM_ConcreteJob instance represents a diagnostic execution. Table 37 provides information about the properties of CIM_ConcreteJob.

Table 37 – Class: CIM_ConcreteJob

Properties	Requirement	Notes
InstanceID	Mandatory	**Key** InstanceID should be constructed using the following preferred algorithm: <OrgID>:<LocalID> (See the MOF file for more detail.) (pattern "^.*[:].*$")
Name	Mandatory	The property will be formatted as a free-form string of variable length. (pattern ".*")
JobState	Mandatory	None
TimeBeforeRemoval	Mandatory	
StartTime	Mandatory	None

Properties	Requirement	Notes
ElapsedTime	Mandatory	This property should be updated periodically so as to be useful as a “heartbeat.”
PercentComplete	Mandatory	
DeleteOnCompletion	Optional	The default value for this property is TRUE.
ErrorCode	Optional	0 if the Job completed without error.
ErrorDescription	Conditional	If ErrorCode has a value other than 0, , ErrorDescription is Mandatory.
RequestStateChange()	Mandatory	See section 8.2.

10.4 CIM_DiagnosticsLog

CIM_DiagnosticsLog represents a log that aggregates all of the results (records) that the execution of a diagnostic generates. Table 38 provides information about the properties of CIM_DiagnosticsLog.

Table 38 – Class: CIM_DiagnosticsLog

Properties	Requirement	Notes
InstanceID	Mandatory	**Key** InstanceID should be constructed using the following preferred algorithm: <OrgID>:<LocalID> (See the MOF file for more detail.) (pattern "^.*[:].*$")
ClearLog()	Mandatory	See section 8.3.

10.5 CIM_DiagnosticServiceCapabilities

CIM_DiagnosticServiceCapabilities publishes the diagnostic service’s capabilities, such as settings and execution controls that are supported. Table 39 provides information about the properties of CIM_DiagnosticServiceCapabilities.

Table 39 – Class: CIM_DiagnosticServiceCapabilities

Properties	Requirement	Notes
InstanceID	Mandatory	**Key** InstanceID shall be unique and should be constructed using the following preferred algorithm: <OrgID>:<LocalID> (See the MOF file for more detail.) (pattern "^.*[:].*$")

Properties	Requirement	Notes
ElementName	Optional	This property shall contain the value of the Service's ElementName property. The property will be formatted as a free-form string of variable length. (pattern ".*")
SupportedServiceModes	Optional	If service modes are supported, they shall be published using this property.
OtherSupportedServiceModesDescriptions	Conditional	If SupportedServiceModes has the value 1 (Other), this property is Mandatory.
SupportedLoopControl	Optional	If looping is supported, its controls shall be published using this property.
OtherSupportedLoopControlDescriptions	Conditional	If SupportedLoopControl has the value 1 (Other), this property is Mandatory.
SupportedLogOptions	Optional	If any log options are supported, they shall be published using this property.
OtherSupportedLogOptionsDescriptions	Conditional	If SupportedLogOptions has the value 1(Other), this property is Mandatory.
SupportedLogStorage	Optional	If any log storage options are supported, they shall be published using this property.
OtherSupportedLogStorageDescriptions	Conditional	If SupportedLogStorage has the value 1 (Other), this property is Mandatory.
SupportedExecutionControls	Optional	If any execution controls are supported, they shall be published using this property.
OtherSupportedExecutionControls Descriptions	Conditional	If SupportedExecutionControls has the value 1 (Other), this property is Mandatory.

10.6 CIM_DiagnosticServiceRecord

CIM_DiagnosticServiceRecord is used to report diagnostic service messages such as results, errors, warnings, and status when CIM_DiagnosticsLog is implemented. Table 40 provides information about the properties of CIM_DiagnosticServiceRecord.

Table 40 – Class: CIM_DiagnosticServiceRecord

Properties	Requirement	Notes
InstanceID	Mandatory	**Key** InstanceID should be constructed using the following preferred algorithm: <ConcreteJob.InstanceID>:<n> Where < ConcreteJob.InstanceID> is <OrgID>:<LocalID> as described in ConcreteJob and <n> is an increment value that provides uniqueness. <n> should be set to \"0\" for the first record created by the job, and incremented for each subsequent record. (pattern "^.*[:].*[:][0123456789]*$")
CreationTimeStamp	Mandatory	None
RecordData	Mandatory	None

Properties	Requirement	Notes
RecordFormat	Mandatory	None
LoopsPassed	Conditional	This property shall contain the number of loops passed when RecordType is a "Results", 3 (Subtests), 7 (Device Errors) or 8 (Service Errors) at the time when the record was logged. This property may be NULL if RecordType is different than the aforementioned record types.
LoopsFailed	Conditional	This property shall contain the number of loops failed when RecordType is a "Results", 3 (Subtests), 7 (Device Errors) or 8 (Service Errors) at the time when the record was logged. This property may be NULL if RecordType is different than the aforementioned record types.
ErrorCode	Conditional	This property is Mandatory and shall contain only the error code number when RecordType is 7 (Device Errors) or 8 (Service Errors). If the RecordType value is 2 (Results), this property shall be an array that contains the error codes of all errors generated by the diagnostic service or subtest execution at the time when the record was logged. This property may be NULL if RecordType is different than the aforementioned record types. The property will be formatted as a free-form string of variable length. (pattern ".*")
ErrorCount	Conditional	This property is Mandatory and shall contain only "1" in the first position of that array when RecordType is 7 (Device Errors) or 8 (Service Errors). This property is Mandatory when the RecordType value is 2 (Results), this property shall be an array where each position shall contain the number of times that an error (which can be identified by the same position of ErrorCode array) happened. This property may be NULL if RecordType is different than the aforementioned record types.
ServiceName	Mandatory	This property shall be constructed as follows: <OrgID>:<TestName>. It should be set to the value of the associated DiagnosticTest.Name. (pattern "^.*[:].*$")

Properties	Requirement	Notes
ManagedElementName	Mandatory	This property shall be formatted as a free-form string of variable length. It should be set to the value of the associated ManagedElement.ElementName. (pattern ".*")
RecordType	Mandatory	A RecordType value of 2 (Results) shall be used to report the final results. A RecordType value of 6 (Status) shall be used to report interim results.
OtherRecordTypeDescription	Conditional	If the RecordType value is 1 (Other), this property is Mandatory.
ExpirationDate	Mandatory	

1356

10.7 CIM_DiagnosticSetting

1357 Diagnostic services use CIM_DiagnosticSetting to publish default settings, and clients use this class to
1358 change defaults and run a diagnostic service using specific settings. Table 41 provides information about
1359 the properties of CIM_DiagnosticSetting.

1360 **Table 41 – Class: CIM_DiagnosticSetting**

Properties	Requirement	Notes
SettingID	Mandatory	**Key**
ElementName	Optional	This property shall be formatted as a free-form string of variable length. (pattern ".*")
HaltOnError	Optional	When this property is TRUE, the service should halt after finding the first error.
QuickMode	Optional	When this property is TRUE, the service should attempt to run in an accelerated fashion either by reducing the coverage or number of tests performed.
PercentOfTestCoverage	Optional	This property requests the service to reduce test coverage to the specified percentage.
LoopControl	Optional	This property, which is used in conjunction with LoopControlParameter property, sets one or more loop control mechanisms that limit the number of times that a test should be repeated.
LoopControlParameter	Conditional	If LoopControl has a value other than null, LoopControlParameter shall be filled in for corresponding LoopControl settings. If LoopControl matches "Count", "Timer", or "ErrorCount", LoopControlParameter represents a uint32 numeric value. (pattern "^b[01]* | ^d[0123456789]* | ^x[0123456789ABCDEFabcdef]* | ^[0123456789]*")
OtherLoopControlDescriptions	Conditional	If LoopControl has the value 1 (Other), this property is Mandatory.

Properties	Requirement	Notes
ResultPersistence	Mandatory	This property specifies how many seconds the records should persist after service execution finishes. 0 (zero) indicates "no persistence" and 0xFFFFFFFF indicates "persist forever".
LogOptions	Optional	This property specifies the types of data that should be logged by the diagnostic service.
OtherLogOptionsDescriptions	Conditional	If LogOptions has the value 1 (Other), this property is Mandatory.
LogStorage	Optional	This property specifies the logging mechanism to store the diagnostic results.
OtherLogStorageDescriptions	Conditional	If LogStorage has the value 1 (Other), this property is Mandatory.
VerbosityLevel	Optional	This property specifies the desired volume or detail logged by a diagnostic service.
Locales	Optional	None

1361 ## 10.8 CIM_DiagnosticSettingRecord

1362 CIM_DiagnosticSettingRecord stores the settings used in a specific diagnostic service execution when
1363 CIM_DiagnosticsLog is implemented. Table 42 provides information about the properties of
1364 CIM_DiagnosticSettingRecord.

1365 **Table 42 – Class: CIM_DiagnosticSettingRecord**

Properties	Requirement	Notes
InstanceID	Mandatory	**Key** InstanceID should be constructed using the following preferred algorithm: <ConcreteJob.InstanceID>:<n> < ConcreteJob.InstanceID> is <OrgID>:<LocalID> as described in CIM_ConcreteJob, and <n> is an increment value that provides uniqueness. <n> should be set to \"0\" for the first record created by the job, and incremented for each subsequent record. (pattern "^.*[:].*[:][0123456789]*$")
CreationTimeStamp	Mandatory	None
ServiceName	Mandatory	This property shall be constructed as follows: <OrgID>:<TestName>. It should be set to the value of the associated DiagnosticTest.Name. (pattern "^.*[:].*$")
ManagedElementName	Mandatory	This property will be formatted as a free-form string of variable length. It should be set to the value of the associated ManagedElement.ElementName. (pattern ".*")

Properties	Requirement	Notes
ExpirationDate	Mandatory	
HaltOnError	Optional	When this property is TRUE, the service should halt after finding the first error.
QuickMode	Optional	When this property is TRUE, the service should attempt to run in an accelerated fashion either by reducing the coverage or number of tests performed.
PercentOfTestCoverage	Optional	This property requests the service to reduce test coverage to the specified percentage.
LoopControl	Optional	This property, which is used in conjunction with the LoopControlParameter property, sets one or more loop control mechanisms that limit the number of times that a test should be repeated.
LoopControlParameter	Conditional	If LoopControl has a value other than null, LoopControlParameter shall be filled in for corresponding LoopControl settings. If LoopControl matches "Count", "Timer", or "ErrorCount", LoopControlParameter represents a uint32 numeric value. (pattern "^b[01]* | ^d[0123456789]* | ^x[0123456789ABCDEFabcdef]* | ^[0123456789]*")
OtherLoopControlDescriptions	Conditional	If LoopControl has the value 1 (Other), this property is Mandatory.
ResultPersistence	Mandatory	This property specifies how many seconds the records should persist after service execution finishes. 0 (zero) indicates “no persistence” and 0xFFFFFFFF indicates “persist forever”.
LogOptions	Optional	This property specifies the types of data that should be logged by the diagnostic service.
OtherLogOptionsDescriptions	Conditional	If LogOptions has the value 1 (Other), this property is Mandatory.
VerbosityLevel	Optional	This property specifies the desired volume or detail logged by a diagnostic service.

10.9 CIM_DiagnosticTest

CIM_DiagnosticTest is a class that represents a diagnostic service developed to exercise and observe the behavior of a device that is implicated in some level of system malfunction. It contains properties useful in test configuration and the RunDiagnostic( ) method, a standard mechanism for invoking the test.

Table 43 provides information about the properties of CIM_DiagnosticTest.

Table 43 – Class: CIM_DiagnosticTest

Properties	Requirement	Notes
SystemCreationClassName	Mandatory	**Key**
SystemName	Mandatory	**Key**
CreationClassName	Mandatory	**Key**

Properties	Requirement	Notes
Name	Mandatory	**Key** The Name property shall be constructed as follows: <OrgID>:<TestName>. (pattern "^.*[:].*$")
ElementName	Mandatory	The property will be formatted as a free-form string of variable length. (pattern ".*")
Characteristics	Mandatory	
OtherCharacteristicsDescriptions	Conditional	If Characteristics has the value 1 (Other), this property is Mandatory.
RunDiagnostic( )	Mandatory	See section 8.1.

10.10 CIM_ElementCapabilities

CIM_ElementCapabilities associates a diagnostic service with its capabilities. Table 44 provides information about the properties of CIM_ElementCapabilities.

Table 44 – Class: CIM_ElementCapabilities

Properties	Requirement	Notes
ManagedElement	Mandatory	**Key** This property shall be a reference to an instance of CIM_DiagnosticService.
Capabilities	Mandatory	**Key** This property shall be a reference to an instance of CIM_DiagnosticServiceCapabilities. Cardinality 0..1

10.11 CIM_DefaultSetting (DiagnosticSetting)

CIM_DefaultSetting associates the diagnostic service with the settings for the service itself and the resulting job. Table 45 provides information about the properties of CIM_DefaultSetting.

Table 45 – Class: CIM_DefaultSetting (DiagnosticSetting)

Properties	Requirement	Notes
Element	Mandatory	**Key** This property shall be a reference to an instance of CIM_DiagnosticService. Cardinality 1
Setting	Mandatory	**Key** This property shall be a reference to an instance of CIM_DiagnosticSetting. Cardinality 0..1

10.12 CIM_DefaultSetting (JobSettingData)

CIM_DefaultSetting associates the diagnostic service with the settings for the service itself and the resulting job. Table 46 provides information about the properties of CIM_DefaultSetting.

Table 46 – Class: CIM_DefaultSetting (JobSettingData)

Properties	Requirement	Notes
Element	Mandatory	**Key** This property shall be a reference to an instance of CIM_DiagnosticService. Cardinality 1
Setting	Mandatory	**Key** This property shall be a reference to an instance of CIM_JobSettingData. Cardinality 0..1

10.13 CIM_HelpService

CIM_HelpService is the preferred way for a service to publish online help information. Table 47 provides information about the properties of CIM_HelpService.

Table 47 – Class: CIM_HelpService

Properties	Requirement	Notes
SystemCreationClassName	Mandatory	**Key**
SystemName	Mandatory	**Key**
CreationClassName	Mandatory	**Key**
Name	Mandatory	**Key** This property will be formatted as a free-form string of variable length. (pattern ".*")
ElementName	Mandatory	This property will be formatted as a free-form string of variable length. (pattern ".*")
DeliveryOptions	Mandatory	None
OtherDeliveryOptionDescription	Conditional	If DeliveryOptions has the value 1 (Other), this property is Mandatory.
DocumentsAvailable	Mandatory	This property will be formatted as a free-form string of variable length. (pattern ".*")
DocumentDescriptions	Mandatory	None
DocumentFormat	Mandatory	None
OtherDocumentFormatDescription	Conditional	If DocumentFormat has the value 1 (Other), this property is Mandatory.
GetHelp( )	Mandatory	See section 8.4.

10.14 CIM_JobSettingData

Diagnostic services use CIM_JobSettingData to publish default job settings (for the jobs that they launch), and clients use this class to change the default job settings when invoking the RunDiagnostic( ) method. Table 48 provides information about the properties of CIM_JobSettingData.

Table 48 – Class: CIM_JobSettingData

Properties	Requirement	Notes
InstanceID	Mandatory	**Key**
ElementName	Mandatory	This property shall be formatted as a free-form string of variable length. (pattern ".*")
DeleteOnCompletion	Mandatory	

10.15 CIM_LogManagesRecord

CIM_LogManagesRecord associates a DiagnosticsLog with its records (service records, setting records, or completion records). Table 49 provides information about the properties of CIM_LogManagesRecord.

Table 49 – Class: CIM_LogManagesRecord

Properties	Requirement	Notes
Log	Mandatory	**Key** This property shall be a reference to an instance of CIM_Log.
Record	Mandatory	**Key** This property shall be a reference to an instance of CIM_DiagnosticRecord.

10.16 CIM_OwningJobElement

CIM_OwningJobElement associates a diagnostic service with its jobs (jobs that are launched by this diagnostic). Table 50 provides information about the properties of CIM_OwningJobElement.

Table 50 – Class: CIM_OwningJobElement

Properties	Requirement	Notes
OwningElement	Mandatory	**Key** This property shall be a reference to an instance of CIM_DiagnosticService. Cardinality 1
OwnedElement	Mandatory	**Key** This property shall be a reference to an instance of CIM_ConcreteJob.

10.17 CIM_RecordAppliesToElement

CIM_RecordAppliesToElement associates a record with the managed elements (diagnostic service and device) that have a relationship with this record. Table 51 provides information about the properties of CIM_RecordAppliesToElement.

Table 51 – Class: CIM_RecordAppliesToElement

Properties	Requirement	Notes
Antecedent	Mandatory	**Key** This property shall be a reference to an instance of CIM_DiagnosticRecord.
Dependent	Mandatory	**Key** This property shall be a reference to an instance of CIM_ManagedElement.

10.18 CIM_ServiceAffectsElement

CIM_ServiceAffectsElement is used to associate to the diagnostic service any managed elements that are affected by the running of the service. Table 52 provides information about the properties of CIM_ServiceAffectsElement.

Table 52 – Class: CIM_ServiceAffectsElement

Properties	Requirement	Notes
AffectedElement	Mandatory	**Key** This property shall be a reference to an instance of CIM_ManagedElement.
AffectingElement	Mandatory	**Key** This property shall be a reference to an instance of CIM_DiagnosticService.

10.19 CIM_ServiceAvailableToElement

CIM_ServiceAvailableToElement associates the diagnostic service with its help service information. Table 53 provides information about the properties of CIM_ServiceAvailableToElement.

Table 53 – Class: CIM_ServiceAvailableToElement

Properties	Requirement	Notes
ServiceProvided	Mandatory	**Key** This property shall be a reference to an instance of CIM_HelpService. Cardinality 1
UserOfService	Mandatory	**Key** This property shall be a reference to an instance of CIM_DiagnosticService. Cardinality 1

10.20 CIM_ServiceComponent

CIM_ServiceComponent associates a test that is also part of another test. Table 54 provides information about the properties of CIM_ServiceComponent.

Table 54 – Class: CIM_ServiceComponent

Properties	Requirement	Notes
GroupComponent	Mandatory	**Key** This property shall be a reference to an instance of CIM_DiagnosticService.
PartComponent	Mandatory	**Key** This property shall be a reference to an instance of CIM_DiagnosticService.

10.21 CIM_ElementSettingData

CIM_ElementSettingData associates the diagnostic service with the settings for the resulting job. Table 55 provides information about the properties of CIM_ElementSettingData.

Table 55 – Class: CIM_ElementSettingData

Properties	Requirement	Notes
ManagedElement	Mandatory	**Key** This property shall be a reference to an instance of CIM_DiagnosticService. Cardinality 1
SettingData	Mandatory	**Key** This property shall be a reference to an instance of CIM_JobSettingData Cardinality 0..1
IsDefault	Mandatory	None

10.22 CIM_UseOfLog

CIM_UseOfLog associates a log with a managed element (a device or diagnostic service) whose information is stored in the log. Table 56 provides information about the properties of CIM_UseOfLog.

Table 56 – Class: CIM_UseOfLog

Properties	Requirement	Notes
Antecedent	Mandatory	**Key** This property shall be a reference to an instance of CIM_Log.
Dependent	Mandatory	**Key** This property shall be a reference to an instance of CIM_DiagnosticService.

10.23 CIM_CorrespondingSettingsRecord

CIM_RecordAppliesToElement associates a setting record with the service records related to this setting record. Table 57 provides information about the properties of CIM_CorrespondingSettingsRecord.

Table 57 – Class: CIM_CorrespondingSettingsRecord

Properties	Requirement	Notes
DataRecord	Mandatory	**Key** This property shall be a reference to an instance of CIM_DiagnosticRecord.
SettingsRecord	Mandatory	**Key** This property shall be a reference to an instance of CIM_DiagnosticSettingRecord.

10.24 CIM_HostedService (DiagnosticTest)

CIM_HostedService is used to associate an instance of CIM_DiagnosticTest with an instance of CIM_ComputerSystem to which the CIM_DiagnosticTest is scoped. Table 58 provides information about the properties of CIM_HostedService (DiagnosticTest).

Table 58 – Class: CIM_HostedService (DiagnosticTest)

Properties	Requirement	Notes
Antecedent	Mandatory	**Key** This property shall be a reference to an instance of CIM_ComputerSystem. Cardinality 1
Dependent	Mandatory	**Key** This property shall be a reference to an instance of CIM_DiagnosticTest. Cardinality 1..*

10.25 CIM_HostedService (HelpService)

CIM_HostedService is used to associate an instance of CIM_HelpService with an instance of CIM_ComputerSystem to which the CIM_HelpService is scoped. Table 59 provides information about the properties of CIM_HostedService.

Table 59 – Class: CIM_HostedService (HelpService)

Properties	Requirement	Notes
Antecedent	Mandatory	**Key** This property shall be a reference to an instance of CIM_ComputerSystem. Cardinality 1
Dependent	Mandatory	**Key** This property shall be a reference to an instance of CIM_HelpService. Cardinality 1..*

10.26 CIM_RegisteredProfile

CIM_RegisteredProfile identifies the *Diagnostics Profile* in order for a client to determine whether an instance of CIM_DiagnosticService is conformant with this profile. The CIM_RegisteredProfile class is defined by the *Profile Registration Profile*. With the exception of the mandatory values specified in Table 60, the behavior of the CIM_RegisteredProfile instance is in accordance with the *Profile Registration Profile*.

Table 60 – Class: CIM_RegisteredProfile

Properties	Requirement	Notes
RegisteredName	Mandatory	This property shall have a value of "Diagnostics".
RegisteredVersion	Mandatory	This property shall have a value of "1.0.0".
RegisteredOrganization	Mandatory	This property shall have a value of 2 (DMTF).

ANNEX A
(informative)

Change Log

Version	Date	Description
1.0.0a	07/12/07	Preliminary
1.0.0	02/13/09	Standard

ANNEX B
(informative)

Acknowledgements

The authors wish to acknowledge the following people.

Editors:

- Andre Asselin – IBM Corporation
- Mateus Baur – Hewlett-Packard Company
- Barbara Craig – Hewlett-Packard Company
- Carl Chan – WBEM Solutions, Inc.

Contributors:

- Jim Davis – WBEM Solutions, Inc.
- Members of the DMTF Diagnostics SIG (sub-Working Group of CIM Core)

Bibliography

DMTF DSP1004, Base Server Profile *1.0.0*,
http://www.dmtf.org/standards/published_documents/DSP1004.pdf

DMTF DSP2000, *CIM Diagnostic Model White Paper 1.0.0*,
http://www.dmtf.org/standards/published_documents/DSP2000.pdf