Document Number: DSP1002

Date: 2013-06-13

Version: 2.1.0a

Diagnostics Profile

Information for Work-in-Progress version:

IMPORTANT: This document is not a standard. It does not necessarily reflect the views of the DMTF or all of its members. Because this document is a Work in Progress, it may still change, perhaps profoundly. This document is available for public review and comment until the stated expiration date.

It expires on: 2013-09-27

Provide any comments through the DMTF Feedback Portal:
http://www.dmtf.org/standards/feedback

Document Type: Specification

Document Status: Work in Progress

Document Language: en-US

CONTENTS

Figures

Tables

Foreword

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

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability. For information about the DMTF, see http://www.dmtf.org.

Acknowledgments

The DMTF acknowledges the following individuals for their contributions to this document:

- Rodney Brown – IBM Corporation
- Carl Chan – WBEM Solutions, Inc.
- Peter Lamanna – EMC Corporation
- Mike Walker – Storage Networking Industry Association

Introduction

A *profile* is a collection of Common Information Model (CIM) elements and behavior rules that represents 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 systems management frameworks, for example 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, implementations, 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.

225 # Diagnostics Profile

226 ## 1 Scope

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

231 The target audience for this specification is implementers who are developing implementations or
232 consumers of management interfaces that represent the functionality described in this document.

233 ## 2 Normative references

234 The following referenced documents are indispensable for the application of this document. For dated or
235 versioned references, only the edition cited (including any corrigenda or DMTF update versions) applies.
236 For references without a date or version, the latest published edition of the referenced document
237 (including any corrigenda or DMTF update versions) applies.

238 DMTF DSP0004, *CIM Infrastructure Specification 2.6,*
239 http://www.dmtf.org/standards/published_documents/DSP0004_2.6.pdf

240 DMTF DSP0200, *CIM Operations over HTTP 1.3*,
241 http://www.dmtf.org/standards/published_documents/DSP0200_1.3.pdf

242 DMTF DSP0215, *SM Managed Element Addressing Specification (SM ME Addressing) 1.0*,
243 http://www.dmtf.org/sites/default/files/standards/documents/DSP0215_1.0.pdf

244 DMTF DSP1001, *Management Profile Specification Usage Guide 1.0*,
245 http://www.dmtf.org/standards/published_documents/DSP1001_1.0.pdf

246 DMTF DSP1004, *Base Server Profile 1.0,*
247 http://www.dmtf.org/standards/published_documents/DSP1004_1.0.pdf

248 DMTF DSP1033, *Profile Registration Profile 1.0*,
249 http://www.dmtf.org/standards/published_documents/DSP1033_1.0.pdf

250 DMTF DSP1054, *Indications Profile 1.2*,
251 http://dmtf.org/sites/default/files/standards/documents/DSP1054_1.2.pdf

252 DMTF DSP1119, *Diagnostic Job Control Profile 1.0.0b*,
253 http://dmtf.org/sites/default/files/standards/documents/DSP1119_1.0.pdf

254 DMTF DSP8055, *Diagnostics Message Registry 1.0.0a*,
255 http://dmtf.org/sites/default/files/standards/documents/DSP8055_1.0a.xml

256 IETF RFC5234, *ABNF: Augmented BNF for Syntax Specifications, January 2008*,
257 http://tools.ietf.org/html/rfc5234

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

260

3 Terms and definitions

In this document, some terms have a specific meaning beyond the normal English meaning. Those terms are defined in this clause.

The terms "shall" ("required"), "shall not," "should" ("recommended"), "should not" ("not recommended"), "may," "need not" ("not required"), "can" and "cannot" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Annex H. The terms in parenthesis are alternatives for the preceding term, for use in exceptional cases when the preceding term cannot be used for linguistic reasons. Note that ISO/IEC Directives, Part 2, Annex H specifies additional alternatives. Occurrences of such additional alternatives shall be interpreted in their normal English meaning.

The terms "clause," "subclause," "paragraph," and "annex" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 5.

The terms "normative" and "informative" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 3. In this document, clauses, subclauses, or annexes labeled "(informative)" do not contain normative content. Notes and examples are always informative elements.

The terms defined in DSP0004, DSP0200, DSP1001, and DSP1033 apply to this document. For the purposes of this document, the following terms and definitions also apply.

3.1
conditional

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

3.2
mandatory

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

3.3
optional
indicates a course of action permissible within the limits of the document

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
CIMOM

CIM Object Manager

4.4
CRU
Customer Replaceable Unit

4.5
FRU
Field Replaceable Unit

4.6
ME
Managed Element

4.7
MOF
Managed Object Format

4.8
PD
Problem Determination

4.9
PFA
Predictive Failure Analysis

4.10
SAN
Storage Area Network

4.11
WBEM
Web-Based Enterprise Management

5 Synopsis

Profile Name: Diagnostics Profile

Version: 2.1.0

Organization: DMTF

CIM schema version: 2.36

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.

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

339 **Table 1 – Related profiles**

Profile Name	Organization	Version	Relationship	Behavior
Profile Registration	DMTF	1.0	Mandatory	
Diagnostic Job Control	DMTF	1.0.0b	Mandatory	
Indications	DMTF	1.2.2	Mandatory	

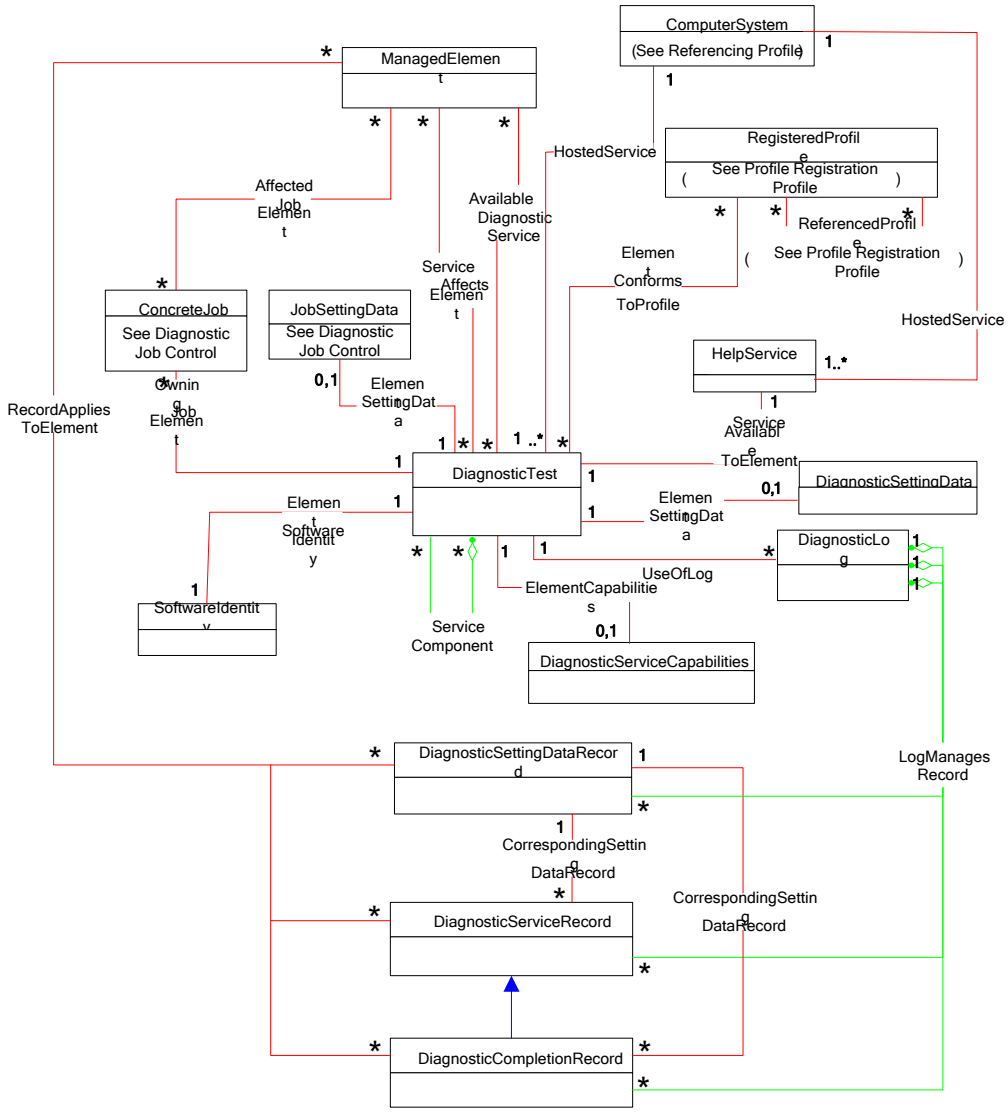
340

341 6 Description

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

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

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



355

356 **Figure 1 – Diagnostics Profile: Class diagram**

357

7 Implementation

This clause 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_DiagnosticSettingData, and CIM_DiagnosticServiceCapabilities. It also defines the CIM_DiagnosticLog class and its related classes, CIM_DiagnosticRecord, CIM_DiagnosticServiceRecord, and CIM_DiagnosticSettingDataRecord. Requirements for propagating and formulating certain properties of these classes and their parents are discussed in this clause. Required methods are listed in clause 8, and properties are listed in clause 10.

7.1 CIM_DiagnosticTest

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

Each diagnostic test shall be represented by an instance of either the 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 7.1.3.5).

An implementation 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 implementation 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 clause defines the values of the Characteristics property.

7.1.3.1 Is Exclusive (value=2)

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.

If the test can run on multiple target devices, but only one instance per device, use CIM_AvailableDiagnosticService.IsExclusiveForMSE.

7.1.3.2 Is Interactive (value=3)

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).

For a description of how a client application interacts with a diagnostic test, see the *Diagnostics Job Control Profile* (DSP1119).

7.1.3.3 Is Destructive (value=4)

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 Is Risky (value=5)

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 Is Package (value=6)

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_DiagnosticSettingData.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 7.8.

7.1.3.6 Reserved (value=7)

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

7.1.3.7 Is Synchronous (value=8)

Use this value to indicate that this diagnostic service will be completed before the RunDiagnosticService( ) 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 is completed. Development of synchronous diagnostic services is not recommended.

7.1.3.8 Media Required (value=9)

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

7.1.3.9 Additional Hardware Required (value=10)

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

7.1.4 CIM_DiagnosticTest.TestTypes

The TestTypes is an optional array property that provides a high-level description of the nature of the test If supplied, the possible values are 1 (Other), 2 (Functional), 3 (Stress), 4 (Health Check), 5 (Access Test), or 6 (Media Verify).

7.1.5 OtherTestTypesDescriptions

The “Other” TestType is provided for vendor-specific service modes. If this property is specified, the OtherTestTypesDescriptions shall have at least one value.

7.1.6 Looping tests

Looping tests or groups of tests are useful for detecting intermittent faults. The client, implementation, 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.
- An implementation may choose to support looping even though its tests do not.
- A stress test may, by its nature, want to repeat a certain operation multiple times.

Looping in the implementation and test is under control of the LoopControl() and LoopControlParameter() properties of the CIM_DiagnosticSettingData 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_DiagnosticSettingData object.

NOTE A remote client may incur network delays and CIMOM delays during each 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).

7.1.7 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 implementation coupled with an ineffective test is not very useful.

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

The QuickMode property of the CIM_DiagnosticSettings 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.

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_DiagnosticSettingData.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. For example, if EstimatedDurationOfService has the value 2 (Seconds) and EstimatedDurationQualifier has a value of 20, the service has an estimated duration of 20 seconds. This property should be implemented if further quantification is possible. In contrast, if EstimatedDurationOfService has the value 0 (Unknown), EstimatedDurationQualifier may be NULL.

7.3 CIM_DiagnosticServiceCapabilities

A diagnostic service publishes its support for various options by using CIM_DiagnosticServiceCapabilities. A client uses CIM_ElementCapabilities to find the diagnostic service capabilities. CIM_DiagnosticServiceCapabilities and CIM_DiagnosticSettingData are closely related and have similar properties. The settings used to control the execution of a diagnostic test cannot specify unsupported capabilities.

7.3.1 CIM_DiagnosticServiceCapabilities.SupportedServiceModes

This property identifies the service modes supported by the DiagnosticTest. Multiple entries may be provided in the SupportedServiceModes. If service modes are supported, they shall be published by using this property. That is, a test may support none, one, or many of the service modes. The service modes that may be specified are 1 (Other), 2 (PercentOfTestCoverage), 3 (QuickMode), 4 (HaltOnError), 5 (ResultPersistence),6 (NonDestructive), 7 (No Service Modes).

7.3.1.1 Other

The “Other” service mode is provided for vendor-specific service modes. If this mode is specified, the OtherServiceModesDescriptions shall have at least one value.

NOTE If “Other” is specified, the implementing vendor should also extend the CIM_DiagnosticSettingData class to include any specification of support needed.

7.3.1.2 PercentOfTestCoverage

If this service mode is supported, the client may request the test to reduce its coverage to the specified percentage set in the PercentOfTestCoverage property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method. The effect of the percentage specified is determined by the provider implementation.

7.3.1.3 QuickMode

If this service mode is supported, the client may request the test attempt to run in an accelerated manner either by reducing the coverage or reducing the of number of tests performed (as determined by the provider implementation). The client requests this mode by specifying QuickMode=TRUE in the QuickMode property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method.

7.3.1.4 HaltOnError

If this service mode is supported, the client may request the test to halt after finding the first error. The client requests this mode by specifying HaltOnError=TRUE in the HaltOnError property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method.

Depending on the test, it may make sense to have this mode set to FALSE to allow all errors to be captured.

7.3.1.5 ResultPersistence

If this service mode is supported, the client may request how many seconds the records should persist after test execution finishes. The client requests this mode by specifying the number of seconds in the ResultPersistence property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method. Supplying 0 (zero) indicates “no persistence” and supplying 0xFFFFFFFF indicates “persist forever”.

If an implementation claims support for ResultPersistence, it shall support any value supplied in the DiagnosticSettingData.ResultPersistence, except the 0xFFFFFFFF (“persist forever”) value. The 0xFFFFFFFF may or may not be supported. If an implementation cannot support all other values for ResultPersistence, it shall not include ResultPersistence in its list of SupportedServiceModes.

7.3.1.6 NonDestructive

If this service mode is supported, the client may request the test only run nondestructive tests (as determined by the provider implementation). The client requests this mode by specifying NonDesctructive=TRUE in the NonDesctructive property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method.

What constitutes a destructive test may vary depending on the device and how it is configured. For example, if you are performing a random write test on a new disk drive that you have not put into service, the test would not be destructive to any data or configuration. However, a random write test would be destructive to a disk drive that is configured and in use in an array.

When a client specifies this mode, the client does not have to identify what subtests should not be run. The provider will determine which subtests are destructive and not execute them.

7.3.2 CIM_DiagnosticServiceCapabilities.OtherSupportedServiceModesDescriptions

The OtherSupportedServiceModesDescriptions provide additional information for SupportedServiceModes when the corresponding value is set to 1 ("Other"). This is intended for vendor-specific extensions to the profile.

7.3.3 CIM_DiagnosticServiceCapabilities.SupportedLoopControl

This property identifies the loop controls supported by the DiagnosticTest. If looping is supported (see 7.1.6), its controls shall be published by using this property. Multiple entries may be provided in the SupportedLoopControl. That is, a test may support none, one, or many of the loop controls. If multiple loop controls are specified, all the specified controls will be applied and the first limit that is reached causes the test to terminate. The loop controls that may be specified are 1 (Other), 2 (Continuous), 3 (Count), 4 (Timer), 5 (ErrorCount), 0x8000 (No Loop Control).

7.3.3.1 Other

The “Other” loop control is provided for vendor-specific loop controls. If this mode is specified, the OtherLoopControlDescriptions should have at least one value.

NOTE If “Other” is specified, the implementing vendor should also extend the CIM_DiagnosticSettingData class to include any loop control specification support needed.

7.3.3.2 Continuous

If this loop control is supported, the client may request that the test will execute continuously. The client requests this mode by specifying 2 (Continuous) in the LoopControl property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method.

NOTE If a loop control of 2 (Continuous) is specified, the corresponding LoopControlParameter property of the DiagnosticSettings parameter is ignored.

NOTE If a LoopControl of 0x8000 (No Loop Control) is specified, no other entry should be in the array property. No Loop Control means the client cannot specify loop controls in the DiagnosticSettingData.

7.3.3.3 Count

If this loop control is supported, the client may request that the test will execute a specified number of times with a single invocation of a test method. The client requests this mode by specifying 3 (Count) in the LoopControl property and the number of loops desired in the LoopControlParameter property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method. The corresponding LoopControlParameter property specifies the count in string format.

7.3.3.4 Timer

If this loop control is supported, the client may request that the test will execute for a specified number of seconds and then terminate. The client requests this mode by specifying 4 (Timer) in the LoopControl property and the number of seconds in the LoopControlParameter property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method. The corresponding LoopControlParameter property specifies the timer (in seconds) in string format.

7.3.3.5 ErrorCount

If this loop control is supported, the client may request that the test will execute until the number of errors that have occurred exceeds a specified ErrorCount. The client requests this mode by specifying 5 (ErrorCount) in the LoopControl property and the error count in the LoopControlParameter property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method. The corresponding LoopControlParameter property specifies the error count in string format.

NOTE The ErrorCount only refers to device errors. It does not include processing errors or warnings.

7.3.3.6 No Loop Control

If this loop control is specified, the test shall have no loops. The client may not specify this mode; it is a capability of the test. The implementation will ignore any loop control settings.

7.3.4 CIM_DiagnosticServiceCapabilities.OtherSupportedLoopControlDescriptions

The OtherSupportedLoopControlDescriptions provide additional information for SupportedLoopControl when the corresponding value is set to 1 ("Other"). This property is intended for vendor-specific extensions to the profile.

7.3.5 CIM_DiagnosticServiceCapabilities.SupportedLogOptions

This property identifies the log options supported by the DiagnosticTest. If any log options are supported, they shall be published by using this property. Multiple entries may be provided in the SupportedLogOptions. That is, a test may support none, one, or many of the log options. The options that may be specified are 1 (Other), 2 (Subtests), 3 (Results), 4 (Actions), 5 (Warnings), 6 (Status), 7 (Device Errors), 8 (Service Errors), 9 (Setting Data), 10 (Statistics), 11 (Hardware Configuration), 12 (Software Configuration), 13 (References), 14 (Debug), and 0x8000 (No Log Options).

7.3.5.1 Other

The “Other” log option is provided for vendor-specific log options. If this option is specified, the OtherLogOptionsDescriptions should have at least one value.

NOTE If “Other” is specified, the implementing vendor should also extend the CIM_DiagnosticSettingData class to include any log option specification support needed.

7.3.5.2 Subtests

If this log option is supported, the client may request the test produce a summary report upon completion of each subtest and each loop iteration. The client requests this option by specifying a value of 3 in the LogOptions property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method. The summary reports should state whether the individual subtest or iteration passed or failed and list relevant error codes and respective error counts.

7.3.5.3 Results

If this log option is supported, the client may request that the test log the results. The client requests this option by specifying a value of 2 in the LogOptions property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method. This option is the most common value for reporting the test results.

NOTE This RecordType may also be specified for interim results from subtests.

7.3.5.4 Actions

If this log option is supported, the client may request that the test log corrective action and instructional messages to guide service personnel. The client requests this option by specifying a value of 4 in the LogOptions property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method. For example, the test might present a prioritized list of actions to perform to isolate a failure or correct a problem. When ordering steps or prioritizing actions, a number should precede the text. For example, 1) Do this first, 2) Do this next, etc.

7.3.5.5 Warnings

If this log option is supported, the client may request that the test log warning messages. The client requests this option by specifying a value of 5 in the LogOptions property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method. For example, log records for alerts that identify warnings (such as DIAG4, see 7.9.4) would be logged.

7.3.5.6 Status

If this log option is supported, the client may request that the test log status messages. The client requests this options by specifying a value of 6 in the LogOptions property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method. For example, the test might log status messages about state information for the driver, device, or system.

7.3.5.7 Device Errors

If this log option is supported, the client may request that the test log errors related to the managed element being tested. The client requests this option by specifying a value of 7 in the LogOptions property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method.

7.3.5.8 Service Errors

If this log option is supported, the client may request that the test log errors related to the test itself rather than the element being tested. The client requests this option by specifying a value of 8 in the LogOptions property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method.

Support for this option means that the test logs errors related to the test itself rather than the element being tested, such as log records associated with DIAG26 (see 7.9.20).

7.3.5.9 Setting Data

If this log option is supported, the client may request that the test log the property values of the DiagnosticSettingData object that is used to configure the test. The client requests this option by specifying a value of 9 in the LogOptions property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method.

7.3.5.10 Statistics

If this log option is supported, the client may request that the test log statistical messages. The client requests this option by specifying a value of 10 in the LogOptions property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method.

Support for this option means the client may request that the test log statistical messages, such as packets sent per second.

7.3.5.11 Hardware Configuration

If this log option is supported, the client may request that the test log messages that contain information about the hardware configuration as viewed by the test. The client requests this option by specifying a value of 11 in the LogOptions property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method. This information might include vendor, version, FRU identification, and location information. The format and contents of this property are element dependent.

7.3.5.12 Software Configuration

If this log option is supported, the client may request that the test log messages that contain information about the software environment as viewed by the test. The client requests this option by specifying a value of 12 in the LogOptions property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method. This information might include the name and version of all the critical software elements controlling the device under test. Each configuration message should have the following common format: element name; element type; manufacturer name; version.

7.3.5.13 References

If this log option is supported, the client may request that the test log the keys of an CIM object of interest. The client requests this option by specifying a value of 13 in the LogOptions property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the

RunDiagnosticService( ) method. Typically this information should include the keys of the object under test. However, it might also include the keys of the DiagnosticsLog.

7.3.5.14 Debug

If this log option is supported, the client may request that the test log debug messages. The client requests this option by specifying a value of 14 in the LogOptions property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method. The debug messages would be vendor-specific messages to aid in debugging the test logic.

7.3.5.15 No Log Messages

If this log option is supported, the client may request that the test not log any messages. The client requests this option by specifying a value of 0x8000 in the LogOptions property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method.

7.3.6 CIM_DiagnosticServiceCapabilities.OtherSupportedLogOptionsDescriptions

The OtherSupportedLogOptionsDescriptions provide additional information for SupportedLogOptions when the corresponding value is set to 1 ("Other"). This option is intended for vendor-specific extensions to the profile.

7.3.7 CIM_DiagnosticServiceCapabilities.SupportedLogStorage

This property identifies the log storage options supported by the DiagnosticTest. Multiple entries may be provided in the SupportedLogStorage. That is, a test may support none, one, or many of the log storage options. The options that may be specified are 1 (Other), 2 (DiagnosticLog), and 0x8000 (No Log Storage).

7.3.7.1 Other

The “Other” log storage property is provided for vendor-specific log storage. If this property is specified, the OtherLogStorageDescriptions should have at least one value.

NOTE If “Other” is specified, the implementing vendor should also extend the CIM_DiagnosticSettingData class to include any log storage specification support needed.

7.3.7.2 DiagnosticLog

If this log storage is supported, the client may request that the test use a DiagnosticLog class for aggregating diagnostic records. The client requests this option by specifying a value of 2 in the LogStorage property of the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService( ) method.

7.3.7.3 No Log Storage

If this log storage is specified, the client may not request any form of log storage. If anything is specified in the DiagnosticSettings parameter, it will be ignored and a DIAG43 alert message will be issued (see 7.9.27).

7.3.8 CIM_DiagnosticServiceCapabilities.OtherSupportedLogStorageDescriptions

The OtherSupportedLogStorageDescriptions provide additional information for SupportedLogStorage when the corresponding value is set to 1 ("Other"). This option is intended for vendor-specific extensions to the profile.

DEPRECATED

7.3.9 CIM_DiagnosticServiceCapabilities.SupportedExecutionControls

NOTE CIM_DiagnosticServiceCapabilities.SupportedExecutionControls and CIM_DiagnosticServcieCapabilities.OtherSupportedExecutionControlsDescriptions are being deprecated in favor of CIM_DiagnosticServiceJobCapabilities.RequestedStatesSupported.

This option identifies the execution control options supported by the DiagnosticTest. Multiple entries may be provided in the SupportedExecutionControls. That is, a test may support none, one, or many of the execution controls. The options that may be specified are 1 (Other), 3 (Kill Job), 4 (Suspend Job), 5 (Terminate Job), 0x8000 (No Execution Controls).

7.3.10 CIM_DiagnosticServiceCapabilities.OtherSupportedExecutionControlsDescriptions

The OtherSupportedExecutionControlsDescriptions provide additional information for SupportedExecutionControls when the corresponding value is set to 1 ("Other"). This option is intended for vendor-specific extensions to the profile.

DEPRECATED

7.4 CIM_DiagnosticSettingData

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. This class can be used in two different ways: 1) by the test to optionally publish its default settings; or 2) by the client to optionally override the test default settings.

NOTE A CIM_DiagnosticSettingData object shall not contain any values that are unsupported by the diagnostic service's CIM_DiagnosticServiceCapabilities object. For example, if CIM_DiagnosticServiceCapabilities.SupportedLoopControl includes the value 5 (No Loop Control), CIM_DiagnosticSettingData.LoopControl cannot include the value 3 (Count). Unsupported values shall be ignored by the implementation.

7.4.1 CIM_DiagnosticSettingData.HaltOnError

When the default DiagnosticSettingData version of this property is TRUE, the test should halt after finding the first error. If the implementation includes a DiagnosticServiceCapabilities instance for the test, HaltOnError shall only be set to TRUE when DiagnosticServiceCapabilities.SupportedServiceModes includes HaltOnError. If HaltOnError is not included in the DiagnosticServiceCapabilities.SupportedServiceModes, the default DiagnosticSettingData.HaltOnError shall be set to FALSE.

If a client sets HaltOnError to TRUE in the DiagnosticsSettings parameter for a RunDiagnosticService method when the DiagnosticServiceCapabilities.SupportedServiceModes does not include HaltOnError, the HaltOnError specification will be ignored. The unsupported setting parameter will result in a DIAG43 alert indication (see subclause 7.9.27) identifying that HaltOnError is not supported, if the client has subscribed to the indication.

If HaltOnError is in effect, at the first device error, an alert message indicating the test was terminated based on HaltOnError will be sent to any client subscribed to the indication. (See 7.9.8.)

7.4.2 CIM_DiagnosticSettingData.QuickMode

When the default DiagnosticSettingData version of this property is TRUE, the test should attempt to run in an accelerated manner either by reducing the coverage or by reducing the number of tests performed. If

the implementation includes a DiagnosticServiceCapabilities instance for the test, QuickMode should only be set to true when DiagnosticServiceCapabilities.SupportedServiceModes includes QuickMode.

If a client sets QuickMode to TRUE in the DiagnosticsSettings parameter for a RunDiagnosticService method when the DiagnosticServiceCapabilities.SupportedServiceModes does not include QuickMode, the QuickMode specification will be ignored. This conflict will result is a DIAG43 alert indication (see subclause 7.9.27) identifying that QuickMode is not supported, if the client has subscribed to the indication.

7.4.3 CIM_DiagnosticSettingData.PercentOfTestCoverage

This property requests the test to reduce test coverage to the specified percentage. If the implementation includes a DiagnosticServiceCapabilities instance for the test, PercentOfTestCoverage should only be set to true when DiagnosticServiceCapabilities.SupportedServiceModes includes PercentOfTestCoverage.

If a client sets PercentOfTestCoverage to anything other than NULL in the DiagnosticsSettings parameter for a RunDiagnosticService method when the DiagnosticServiceCapabilities.SupportedServiceModes does not include PercentOfTestCoverage, the PercentOfTestCoverage specification will be ignored. This conflict will result is a DIAG43 alert indication (see subclause 7.9.27) identifying that PercentOfTestCoverage is not supported, if the client has subscribed to the indication.

7.4.4 CIM_DiagnosticSettingData.NonDestructive

When the default DiagnosticSettingData version of this property is TRUE, the test should not run any tests that would be destructive to the device or data on the device. If the implementation includes a DiagnosticServiceCapabilities instance for the test, NonDestructive should only be set to TRUE when DiagnosticServiceCapabilities.SupportedServiceModes includes NonDestructive.

If a client sets NonDestructive to TRUE in the DiagnosticsSettings parameter for a RunDiagnosticService method when the DiagnosticServiceCapabilities.SupportedServiceModes does not include NonDestructive, the test will be terminated without executing.

7.4.5 CIM_DiagnosticSettingData.LoopControl and LoopControlParameter

The LoopControl property is used in combination with the LoopControlParameter to set one or more loop control mechanisms that limit the number of times that a test should be repeated.

With these properties, it is possible to loop a test (if supported) under control of a counter, timer, and other loop terminating facilities. If the implementation includes a DiagnosticServiceCapabilities instance for the test, LoopControl should only be set to a value contained in the DiagnosticServiceCapabilities.SupportedLoopControl property (see subclause 7.3.3).

NOTE The No Loop Control option cannot be specified in the DiagnosticSettingData property. It is a capability of the implementation, but cannot be requested.

7.4.6 CIM_DiagnosticSettingData.ResultPersistence

This property specifies how many seconds the log records should persist after service execution finishes. If the implementation includes a DiagnosticServiceCapabilities instance for the test, ResultPersistence should be set when DiagnosticServiceCapabilities.SupportedServiceModes includes ResultPersistence. If ResultPersistence is not specified in the DiagnosticServiceCapabilities.SupportedServiceModes, but is specified in the default DiagnosticSettingData, the results will be retained for the default ResultPersistence value.

ResultPersistence is specified in seconds. If it is set to zero (0), the provider need not persist the diagnostic result. The diagnostic information is only available while the diagnostic is executing or at its conclusion.

If ResultPersistence is set to 0xFFFFFFFF, the provider shall persist results forever. In this case, the client bears the responsibility for deleting them. An implementation might not support the 0xFFFFFFFF value even though it claims support for ResultPersistence in its DiagnosticServiceCapabilities.SupportedServiceModes. If the client request to persist results forever is rejected, the client may specify any other value and the implementation shall support that time period.

NOTE Results (e.g., DiagnosticLog information) are independent of the job that creates the results. The life of the job is controlled by the TimeBeforeRemoval property of the ConcreteJob. The life of the DiagnosticLog information is controlled by ResultPersistence. One may be deleted before the other.

7.4.7 CIM_DiagnosticSettingData.LogOptions

This property specifies the types of data that should be logged by the diagnostic test.

This property supports specification of the nature of data being logged by the test through the addition of the LogOptions enumeration. If the implementation includes a DiagnosticServiceCapabilities instance for the test, LogOptions should only be set to a value contained in the DiagnosticServiceCapabilities.SupportedLogOptions property (see subclause 7.3.5).

7.4.8 CIM_DiagnosticSettingData.LogStorage

This property specifies the logging mechanism to store the diagnostic results. If the implementation includes a DiagnosticServiceCapabilities instance for the test, LogStorage should only be set to a value contained in the DiagnosticServiceCapabilities.SupportedLogStorage property (see subclause 7.3.7).

NOTE The No Log Storage option cannot be specified in DiagnosticSettingData.

7.4.9 CIM_DiagnosticSettingData.VerbosityLevel

This property specifies the desired volume or detail logged by a diagnostic test. The possible values include Minimum, Standard, and Full. The exact meaning of each of these are vendor specific. The definitions in this subclause are guidelines.

The VerbosityLevel property is an array property that is correlated with the LogOptions property. That is, VerbosityLevel can be set for each log option specified in the LogOptions setting property.

In the default CIM_DiagnosticSettingData, the provider would identify the default VerbosityLevel for each of the log options that it supports.

In the DiagnosticSettings parameter (an embedded instance of CIM_DiagnosticSettingData) of the RunDiagnosticService method, the client would specify the verbosity level desired for each of the log options.

7.4.9.1 Minimum

This value would be specified if the least amount of information is desired.

7.4.9.2 Standard

This level is the standard level of messaging provided by the test. This value would be specified to get the default level of logging.

7.4.9.3 Full

This value would be specified when all information, regardless of size, is desired.

7.4.10 Default setting

The default settings for a diagnostic service are obtained by using the CIM_ElementSettingData association to an instance of (a subclass of) CIM_DiagnosticSettingData where the IsDefault property has the value of TRUE.

7.4.11 Client override

A client can choose to accept the default settings (published or not) or override the default settings by creating a CIM_DiagnosticSettingData object based upon the settings that an implementation indicates are supported in its CIM_DiagnosticServiceCapabilities object.

If a client chooses to accept the default settings (published or not), the DiagnosticSettings argument to the RunDiagnosticService( ) method of DiagnosticTest should be set to NULL or an empty string.

If a client chooses to override the default settings, the DiagnosticSettings argument to the RunDiagnosticService( ) method of DiagnosticTest is set to an encoded form of the CIM_DiagnosticSettingData object.

Note that the CIM_DiagnosticSettingData 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.

7.5 CIM_DiagnosticLog

All diagnostic result messages shall be represented by instances of CIM_DiagnosticRecord subclasses. Moreover, those records shall be aggregated to an instance of CIM_DiagnosticLog. Each invocation of the RunDiagnosticService method of DiagnosticTest shall instantiate a new CIM_DiagnosticLog object. 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.

7.5.1 Logging results

The methods to record the results of running a diagnostic service are specified by the LogOptions and LogStorage properties of the CIM_DiagnosticSettingData 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 is mandatory 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.

7.6 CIM_DiagnosticRecord

CIM_DiagnosticRecord has two subclasses: CIM_DiagnosticServiceRecord and CIM_DiagnosticSettingDataRecord. CIM_DiagnosticServiceRecord has a single subclass: CIM_DiagnosticCompletionRecord.

CIM_DiagnosticServiceRecord is structured to hold the information that is generated while a particular service is running. One or more CIM_DiagnosticServiceRecord objects may be created during a single execution of a test.

CIM_DiagnosticSettingDataRecord is structured to hold the attributes of the setting object that was used as the DiagnosticSettings parameter to the RunDiagnosticService( ) method. The record that gets written to the log is the “effective” DiagnosticSettingData that includes default and overridden values. At most, a single CIM_DiagnosticSettingDataRecord may be created during a single execution of a test.

CIM_DiagnosticCompletionRecord is structured to hold the information that is generated as a result of running the particular service. A single CIM_DiagnosticCompletionDataRecord shall be created during a single execution of a test.

7.6.1 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 manner.

CIM_DiagnosticSettingData.ResultPersistence shall be used by the client to specify to the diagnostic service implementation 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 implementation shall delete the data objects that contain the results.

The value of CIM_DiagnosticRecord.ExpirationDate shall be calculated by the implementation 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 implementation. It is the implementation'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.

7.6.2 CIM_DiagnosticRecord.InstanceID

To simplify the retrieval of test data for a specific test execution, the value of InstanceID for CIM_ConcreteJob is closely related to the InstanceID for the subclasses of CIM_DiagnosticRecord.

CIM_DiagnosticRecord.InstanceID should be constructed by 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 test during this job, and incremented for each subsequent record created by the test during this job. Each new test execution can reset the <n> to 0.

7.6.3 CIM_DiagnosticRecord.RecordType

The RecordType property of DiagnosticRecords correlates with the CIM_DiagnosticSettingData.LogOptions property. Each DiagnosticRecord in the log identifies the RecordType for the LogOptions value specified in the DiagnosticSettingData. If a Log Option is not included in the DiagnosticSettingData, the DiagnosticRecords that would have contained that RecordType shall not be logged.

The RecordType also identifies which DiagnosticProperties are populated.

7.7 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 of 4 (Is Destructive) or 3 (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.

7.8 Diagnostics Profile Indications support

The Diagnostics Profile constrains certain elements in its support for the DMTF Indications Profile. This subclause identifies those constraints.

7.8.1 CIM_IndicationFilter (StaticIndicationFilter)

The Diagnostics Profile constrains some of the properties of the StaticIndicationFilter version of the CIM_IndicationFilter class and makes the class mandatory. The class is mandatory because some of the alert indication filters are mandatory and the Diagnostics Profile requires that static versions of mandatory indication filters be populated.

7.8.1.1 CIM_IndicationFilter.Name

The Diagnostics Profile constrains names of the profile defined alert indication filters as prescribed by DSP1054. The names for the indication filters are identified in the entries for the indications in Table 28. The Name shall be formatted as defined by the following ABNF rule:

"DMTF:Diagnostics:" MessageID

The MessageID shall have the same value of the MessageID in the Query for the filter.

7.8.1.2 CIM_IndicationFilter.Query

The Diagnostics Profile constrains the Query properties of the profile defined alert indication filters as prescribed by DSP1054. The Query properties for the indication filters are identified in the entries for the indications in Table 28.

7.8.1.3 CIM_IndicationFilter.QueryLanguage

The Diagnostics Profile constrains the QueryLanguage properties of the profile defined alert indication filters as prescribed by DSP1054. The QueryLanguage properties for the indication filters are identified in the entries for the indications in Table 28.

7.8.2 CIM_FilterCollection (ProfileSpecificFilterCollection)

The Diagnostics Profile constrains the CollectionName property of the ProfileSpecificFilterCollection version of the CIM_FilterCollection class.

7.8.2.1 CIM_FilterCollection.CollectionName

The Diagnostics Profile constrains CollectionName of the profile defined ProfileSpecificFilterCollection filter collection as prescribed by DSP1054. The CollectionName for the filter collection shall be formatted as defined by the following ABNF rule:

"DMTF:Diagnostics:ProfileSpecifiedAlertIndicationFilterCollection"

7.8.3 CIM_MemberOfCollection (IndicationFilterInFilterCollection)

The Diagnostics Profile constrains the properties of the IndicationFilterInFilterCollection version of the CIM_MemberOfCollection class.

7.8.3.1 CIM_MemberOfCollection.Collection

The Diagnostics Profile constrains the Collection property to be the reference to the ProfileSpecificFilterCollection filter collection.

7.8.3.2 CIM_MemberOfCollection.Member

The Diagnostics Profile constrains the Member property to be a reference to one of the profile defined alert indication filters.

7.8.4 CIM_OwningCollectionElement (IndicationServiceOfFilterCollection)

The Diagnostics Profile constrains the OwnedElement property of the IndicationServiceOfFilterCollection version of the CIM_OwningCollectionElement class.

7.8.4.1 CIM_OwningCollectionElement.OwnedElement

The Diagnostics Profile constrains OwnedElement property to be the reference to the ProfileSpecificFilterCollection filter collection.

7.9 Diagnostics alert indications and standard messages

7.9.1 DIAG0 – The test passed.

The test executed with no errors or warnings.

This alert would be sent if the inputs were accepted, defaulted, or reset and the test executed to completion (as defined by the inputs) and the test reported no errors or warnings.

This indication is the last alert indication that would be sent in a successful test execution. Any indications that precede it would be informational messages.

The variables in this message are:

- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

  – The Object Path of the element
  – The ElementName of the element
  – A unique user friendly name not in the model (such as, asset name)

  The Element Moniker can be any of these, but whichever one is used shall be used consistently for all managed elements of the same type within the scoping profile (such as, all disk drives in a system).
- Log Object Path – Identifies the Object Path of the CIM_DiagnosticLog instance.

With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType should be set to "Successful Completion".

With this alert, the PerceivedSeverity shall have the value 2 (Information).

7.9.2 DIAG1 – The reason for the test failure is unknown

The test failed to execute for unknown reasons.

This alert would be sent if the test failed to execute for any one of a number of reasons. A client should refer to other alert indications that may have been sent to determine what (if anything) can be done.

The variables in this message are:

- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

This could be one of the following:

- The Object Path of the element
- The ElementName of the element
- A unique user friendly name not in the model (such as, asset name)

The Element Moniker can be any of these, but whichever one is used shall be used consistently for all managed elements of the same type within the scoping profile (such as, all disk drives in a system).

- Log Object Path – Identifies the Object Path of the CIM_DiagnosticLog instance.
- AlertType – Identifies the AlertType for this alert indication (such as, Processing Error or Device Alert).

With this alert the AlertType shall have the value 4 (Processing Error) or 5 (Device Alert). If the test failed before the actual test was started, the AlertType shall have the value 4 (Processing Error). If the test had started and then failed, the AlertType should have the value 5 (Device Alert).

With this alert, the PerceivedSeverity shall have the value 0 (Unknown), 4 (Minor), 5 (Major), 6 (Critical), or 7 (Fatal/NonRecoverable). If the severity is unknown, 0 (unknown) should be specified. If this is a processing error (see above), the PerceivedSeverity should be coded as 4 (Minor) or 5 (Major). If this is a device alert (see above), this may be 4 (Minor), 5 (Major), 6 (Critical) or 7 (Fatal/NonRecoverable). It should only be 7 (Fatal/NonRecoverable) if recovery cannot be done.

7.9.3 DIAG3 – The device test failed

The test ran, but the element under test reported device alert errors.

This alert would only be sent if the element under test reported errors. This would be the last Alert Indication and device error alerts should have preceded this alert.

It would NOT be sent if

- there were errors in processing the RunDiagnosticsService parameters, or
- the element under test only issued warnings.

The variables in this message are:

- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

  - The Object Path of the element
  - The ElementName of the element
  - A unique user friendly name not in the model (such as, asset name)

  The Element Moniker can be any of these, but whichever one is used shall be used consistently for all managed elements of the same type within the scoping profile (such as, all disk drives in a system).
- Log Object Path – Identifies the Object Path of the CIM_DiagnosticLog instance.

With this alert, the AlertType shall have the value 5 (Device Alert).

With this alert, the PerceivedSeverity shall have the value 4 (Minor), 5 (Major), 6 (Critical), or 7 (Fatal/NonRecoverable).

7.9.4 DIAG4 – The test completed with warnings

The test ran, but the element under test reported warnings.

This alert would be sent if the test ran to completion with no errors, but reported warnings. This would be the last alert indication sent for the test run. Informational and warning messages may have preceded this message.

The variables in this message are:

- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

  – The Object Path of the element
  – The ElementName of the element
  – A unique user friendly name not in the model (such as, asset name)

  The Element Moniker can be any of these, but whichever one is used shall be used consistently for all managed elements of the same type within the scoping profile (such as, all disk drives in a system).
- Log Object Path – Identifies the Object Path of the CIM_DiagnosticLog instance.

With this alert, the AlertType shall have the value 4 (Processing Error) or 5 (Device Alert). Processing Error means input was given but ignored or input was reset by the test job. Device Alert means the device reported the warning.

With this alert, the PerceivedSeverity shall have the value 3 (Warning).

7.9.5 DIAG5 – The requested test is not supported

The test as requested in the RunDiagnosticService extrinsic method is not supported on the element specified.

This alert would be sent if the instance of DiagnosticTest (and the test that it represents) is not supported on the ManagedElement specified by the RunDiagnosticService extrinsic method. The test may well be supported on other elements, but not on the element specified in the request.

The variables in this message are:

- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

  – The Object Path of the element
  – The ElementName of the element
  – A unique user friendly name not in the model (such as, asset name)

  The Element Moniker can be any of these, but whichever one is used shall be used consistently for all managed elements of the same type within the scoping profile (such as, all disk drives in a system).
- Log Object Path – Identifies the Object Path of the CIM_DiagnosticLog instance.

With this alert, the AlertType shall have the value 5 (Device Alert). The device does not support the test.

With this alert, the PerceivedSeverity shall have the value 3 (Warning). There is no device problem, but the test cannot be run on the specified element.

7.9.6 DIAG6 – The required test setup steps have not been performed

The test did not run because the proper set up steps were not done to support the test.

This alert would be sent if in processing the request, the test detected that certain conditions are not present to execute the test. For example, a setup file is missing or the element in question is disabled or the device is not connected. This alert will be followed by a “test did not start” or “test aborted” test completion status alert (see 7.9.28 and 7.9.29).

The variables in this message are:

- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

  – The Object Path of the element
  – The ElementName of the element
  – A unique user friendly name not in the model (such as, asset name)

  The Element Moniker can be any of these, but whichever one is used shall be used consistently for all managed elements of the same type within the scoping profile (such as, all disk drives in a system).

With this alert, the AlertType shall have the value 4 (Processing Error). That is, the test was not run because the proper set up has not been done.

With this alert, the PerceivedSeverity shall have the value 3 (Warning). That is, there is no real error with the element under test, just a setup error. The client needs to ensure that the proper setup is done before running the test again.

7.9.7 DIAG7 – The test ran but HaltOnError is not supported

The test ran and found one or more errors, but the test did not halt on the first error because HaltOnError is not supported by the test on the specified element.

This alert would be sent if the DiagnosticSettings parameter of the RunDiagnosticService method included HaltOnError=TRUE, but the device did not support HaltOnError.

The variables in this message are:

- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

  – The Object Path of the element
  – The ElementName of the element
  – A unique user friendly name not in the model (such as, asset name)

1133 The Element Moniker can be any of these, but whichever one is used shall be used consistently
1134 for all managed elements of the same type within the scoping profile (such as, all disk drives in
1135 a system).

1136 With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType should be set to
1137 Parameter Ignored.

1138 With this alert, the PerceivedSeverity shall have the value 3 (Warning). The Alert warns the client that the
1139 test was not run with HaltOnError in effect.

1140 7.9.8 DIAG8 – The test halted due to an error

1141 The test ran until it found a Device Error and was terminated because the DiagnosticSettings parameter
1142 of the RunDiagnosticService method called for HaltOnError.

1143 This alert would be sent if the client set HaltOnError to TRUE and the test encountered a Device Error.
1144 The test does not run to completion, but it is terminated. The resulting JobState for the ConcreteJob is 8
1145 (Terminated), just as though the client had issued a RequestedStateChange requesting termination.

1146 To determine the error that caused the test to be halted, see prior (error) alert indications or see the
1147 DiagnosticLog records.

1148 The variables in this message are:

1149 • Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name
1150 property of the DiagnosticTest instance.

1151 • Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive)
1152 that was specified.

1153 This could be one of the following:

1154 – The Object Path of the element
1155 – The ElementName of the element
1156 – A unique user friendly name not in the model (such as, asset name)

1157 The Element Moniker can be any of these, but whichever one is used shall be used consistently
1158 for all managed elements of the same type within the scoping profile (such as, all disk drives in
1159 a system).

1160 With this alert, the AlertType shall have the value 5 (Device Alert).

1161 With this alert, the PerceivedSeverity shall have the value 3 (Warning).

1162 7.9.9 DIAG10 – QuickMode is not supported

1163 The test ran but QuickMode is not supported.

1164 This alert would be sent if the client requested QuickMode=TRUE in the DiagnosticSettings parameter of
1165 the RunDiagnosticService method, but QuickMode is not supported for the test or the element under test.
1166 The QuickMode parameter applies to the test invoked by RunDiagnosticService (and may or may not
1167 apply to lower level tests).

1168 The variables in this message are:

1169 • Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name
1170 property of the DiagnosticTest instance.

1171 • Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive)
1172 that was specified.

1173 This could be one of the following:

1174 – The Object Path of the element
1175 – The ElementName of the element
1176 – A unique user friendly name not in the model (such as, asset name)

1177 The Element Moniker can be any of these, but whichever one is used shall be used consistently
1178 for all managed elements of the same type within the scoping profile (such as, all disk drives in
1179 a system).

1180 With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType should be set to
1181 Parameter Not Supported.

1182 With this alert, the PerceivedSeverity shall have the value 3 (Warning). The alert warns the client that the
1183 test was not run in QuickMode.

1184 7.9.10 DIAG11 – Requested LoopControl type not supported

1185 The test may or may not have run, but a LoopControl specified in the DiagnosticSettings parameter of the
1186 RunDiagnosticService method was not supported. Another test completion status alert indicates whether
1187 or not the test was run.

1188 This alert would be sent if the request asked for a LoopControl that was not supported by the test or the
1189 element under test.

1190 NOTE If multiple LoopControl types were not supported, multiple alert messages will be sent.

1191 The variables in this message are:

1192 • Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name
1193 property of the DiagnosticTest instance.

1194 • Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive)
1195 that was specified.

1196 This could be one of the following:

1197 – The Object Path of the element
1198 – The ElementName of the element
1199 – A unique user friendly name not in the model (such as, asset name)

1200 The Element Moniker can be any of these, but whichever one is used shall be used consistently
1201 for all managed elements of the same type within the scoping profile (such as, all disk drives in
1202 a system).

1203 • LoopControl – Identifies the LoopControl in the DiagnosticSettings parameter of the
1204 RunDiagnosticService method that was not supported.

1205 With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType should be set to Loop
1206 Control Type Not Supported.

1207 With this alert, the PerceivedSeverity shall have the value 3 (Warning).

1208 7.9.11 DIAG13 – Logging could not be started

1209 The test ran, but the logging requested could not be started.

1210 This alert would be sent if a client requested some type of logging, but logging could not be started. If
1211 multiple logs could not be started, the client may receive multiple versions of this message. This alert
1212 would be sent as soon as the problem is discovered (before or as the test is running). Clients would have
1213 the opportunity to kill or terminate the test.

1214 The variables in this message are:

- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

  – The Object Path of the element
  – The ElementName of the element
  – A unique user friendly name not in the model (such as, asset name)

  The Element Moniker can be any of these, but whichever one is used shall be used consistently for all managed elements of the same type within the scoping profile (such as, all disk drives in a system).
- Log Storage – Identifies the type of log storage the client requested.

With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType would be Log Not Started.

With this alert, the PerceivedSeverity shall have the value 3 (Warning).

7.9.12 DIAG14 – Logging errors occurred

The test ran, but logging errors (such as, errors writing the log) occurred.

This alert would be sent if the test ran and logging errors occurred in one of the logs specified by the DiagnosticSetting parameter of the RunDiagnosticService method request. If more than one log storage experiences errors, the multiple alerts will be sent. This message would be sent when the first error writing to the log is encountered.

The variables in this message are:

- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

  – The Object Path of the element
  – The ElementName of the element
  – A unique user friendly name not in the model (such as, asset name)

  The Element Moniker can be any of these, but whichever one is used shall be used consistently for all managed elements of the same type within the scoping profile (such as, all disk drives in a system).
- Log Storage – Identifies the log storage that experienced the errors.

With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType should be set to Log Errors Occurred.

With this alert, the PerceivedSeverity shall have the value 3 (Warning).

7.9.13 DIAG15 – LogStorage type not supported

The test ran, but a LogStorage request was not supported.

1254 This alert would be sent if the client requested one or more log storage types, but one of them is not
1255 supported by the implementation.

1256 NOTE If multiple log storage types are not supported, multiple DIAG15 alerts would be sent. DIAG15 alerts do not
1257 report a mismatch between the setting and capabilities. That situation is handled by a separate alert (see 7.9.30,
1258 DIAG46 – LogStorage mismatch with capabilities).

1259 The variables in this message are:

1260 • Log Storage – Identifies the log storage that was requested.

1261 • Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name
1262 property of the DiagnosticTest instance.

1263 • Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive)
1264 that was specified.

1265 This could be one of the following:

1266 – The Object Path of the element
1267 – The ElementName of the element
1268 – A unique user friendly name not in the model (such as, asset name)

1269 The Element Moniker can be any of these, but whichever one is used shall be used consistently
1270 for all managed elements of the same type within the scoping profile (such as, all disk drives in
1271 a system).

1272 With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType should be set to Log
1273 Storage Not Supported.

1274 With this alert, the PerceivedSeverity shall have the value 3 (Warning).

1275 7.9.14 DIAG16 – LoopControl Parameter invalid

1276 The test ran, but a LoopControlParameter supplied in the DiagnosticSetting parameter of the
1277 RunDiagnosticService method was invalid and ignored.

1278 This alert would be sent if a LoopControlParameter provided on the method is invalid. An invalid
1279 LoopControlParameter could be:

1280 • A string that could not be converted to a number or datetime datatype
1281 • A string that converts to a number or datetime, but is not supported

1282 If multiple LoopControlParameters are invalid, multiple alert messages would be sent, one for each invalid
1283 LoopControlParameter.

1284 The variables in this message are:

1285 • Loop Control Parameter – Identifies the LoopControlParameter value supplied with the
1286 DiagnosticSetting parameter on the RunDiagnosticService request.

1287 • Loop Control – Identifies the LoopControl that was supplied with the DiagnosticSetting
1288 parameter on the RunDiagnosticService request for interpreting the LoopControlParameter

1289 • Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name
1290 property of the DiagnosticTest instance.

1291 • Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive)
1292 that was specified.

1293 This could be one of the following:

1294 – The Object Path of the element
1295 – The ElementName of the element

1296 – A unique user friendly name not in the model (such as, asset name)

1297 The Element Moniker can be any of these, but whichever one is used shall be used consistently
1298 for all managed elements of the same type within the scoping profile (such as, all disk drives in
1299 a system).

1300 With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType should be set to
1301 Parameter Ignored.

1302 With this alert, the PerceivedSeverity shall have the value 3 (Warning).

1303 7.9.15 DIAG17 – VerbosityLevel not supported

1304 The test ran, but the VerbosityLevel requested by the DiagnosticSetting parameter was not supported.

1305 This alert would be sent if the client requested a VerbosityLevel in the DiagnosticSetting parameter of the
1306 RunDiagnosticService method, but that VerbosityLevel is not supported. The default VerbosityLevel was
1307 used instead.

1308 The variables in this message are:

1309 • Verbosity Level Specified – Identifies the VerbosityLevel value supplied with the
1310 DiagnosticSetting parameter on the RunDiagnosticService request.

1311 • Verbosity Level Used – Identifies the VerbosityLevel value used on the RunDiagnosticService
1312 request.

1313 • Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name
1314 property of the DiagnosticTest instance.

1315 • Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive)
1316 that was specified.

1317 This could be one of the following:

1318 – The Object Path of the element
1319 – The ElementName of the element
1320 – A unique user friendly name not in the model (such as, asset name)

1321 The Element Moniker can be any of these, but whichever one is used shall be used consistently
1322 for all managed elements of the same type within the scoping profile (such as, all disk drives in
1323 a system).

1324 With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType should be set to
1325 Parameter Ignored.

1326 With this alert, the PerceivedSeverity shall have the value 3 (Warning).

1327 7.9.16 DIAG18 – PercentOfTestCoverage level was not completed

1328 The test ran, but the PercentOfTestCoverage level that was requested in the DiagnosticSetting parameter
1329 of the RunDiagnosticService method was not completed.

1330 This alert would be sent if the client requested a PercentOfTestCoverage level in the DiagnosticSetting
1331 parameter of the RunDiagnosticService method but the percentage was not achieved.

1332 The variables in this message are:

1333 • PercentRequested – Identifies the percent requested in the DiagnosticSetting parameter.

1334 • PercentCompleted – Identifies the percent completed.

- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

  – The Object Path of the element
  – The ElementName of the element
  – A unique user friendly name not in the model (such as, asset name)

  The Element Moniker can be any of these, but whichever one is used shall be used consistently for all managed elements of the same type within the scoping profile (such as, all disk drives in a system).

With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType should be set to Parameter ignored.

With this alert, the PerceivedSeverity shall have the value 3 (Warning).

7.9.17 DIAG22 – ErrorCount exceeded

The test ran, but the ErrorCount specified in the LoopControlParameter of the DiagnosticSetting was exceeded and the test terminated.

This alert would be sent if the client specified ErrorCount as one of the loop controls in the DiagnosticSetting the client t supplied on the RunDiagnosticService method and the error count (as specified in the LoopControlParameter) has been achieved.

The variables in this message are:

- LoopControl Error Count – Identifies the LoopControlParameter requested (the count that was exceeded).
- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

  – The Object Path of the element
  – The ElementName of the element
  – A unique user friendly name not in the model (such as, asset name)

  The Element Moniker can be any of these, but whichever one is used shall be used consistently for all managed elements of the same type within the scoping profile (such as, all disk drives in a system).

With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType should be set to Loop Control Reached.

With this alert, the PerceivedSeverity shall have the value 3 (Warning).

7.9.18 DIAG23 – LoopControl limit reached as configured by the client

The test ran, but the Count or Error Count specified in the LoopControlParameter of the DiagnosticSetting was reached and the test terminated.

This alert would be sent if the client specified Count or Error Count as one of the loop controls in the DiagnosticSetting that was supplied on the RunDiagnosticService method and the loop count (as specified in the LoopControlParameter) has been achieved. If multiple LoopControl limits are reached, there would be multiple messages.

The variables in this message are:

- Loop Control – Identifies which LoopControl limit was reached.
- LoopControl Parameter Value – Identifies the LoopControlParameter requested (the count that was reached).
- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

  – The Object Path of the element
  – The ElementName of the element
  – A unique user friendly name not in the model (such as, asset name)

  The Element Moniker can be any of these, but whichever one is used shall be used consistently for all managed elements of the same type within the scoping profile (such as, all disk drives in a system).

With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType should be set to Loop Control Reached.

With this alert, the PerceivedSeverity shall have the value 3 (Warning).

7.9.19 DIAG24 – LoopControl Timeout limit reached as configured by the client

The test ran, but the Timer specified in the LoopControlParameter of the DiagnosticSetting was reached and the test terminated.

This alert would be sent if the client specified Timer as one of the loop controls in the DiagnosticSetting that was supplied on the RunDiagnosticService method and the loop time (as specified in the LoopControlParameter) has been achieved.

The variables in this message are:

- LoopControl Parameter Value – Identifies the LoopControlParameter requested (the timer that was reached).
- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

  – The Object Path of the element
  – The ElementName of the element
  – A unique user friendly name not in the model (such as, asset name)

  The Element Moniker can be any of these, but whichever one is used shall be used consistently for all managed elements of the same type within the scoping profile (such as, all disk drives in a system).

With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType should be set to Loop Control Reached.

With this alert, the PerceivedSeverity shall have the value 3 (Warning).

7.9.20 DIAG26 – Test cannot be run with NonDestructive set to true

The test was not run because the client requested NonDestructive=TRUE in the DiagnosticSetting parameter of the RunDiagnosticService method and this function is not supported for the test or the element under test.

This alert would be sent if the client requested a NonDestructive execution, but the implementation does not support this for the test or the element under test.

NOTE This message would not be sent when the NonDesctructive value conflicts with the SupportedServiceModes property of the DiagnosticServiceCapabilities (see 7.9.26 for the message for a mismatch with capabilities). DIAG26 would be sent if the optional DiagnosticServiceCapabilities was not implemented or the capabilities was implemented and the SupportedServiceModes include NonDestructive, but not for the element under test.

The variables in this message are:

- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

  – The Object Path of the element
  – The ElementName of the element
  – A unique user friendly name not in the model (such as, asset name)

  The Element Moniker can be any of these, but whichever one is used shall be used consistently for all managed elements of the same type within the scoping profile (such as, all disk drives in a system).

With this alert, the AlertType shall have the value 4 (Processing Error).

With this alert, the PerceivedSeverity shall have the value 5 (Major).

7.9.21 DIAG27 – Capability to set LoopControl not supported

The test ran, but a LoopControl specified in the DiagnosticSetting parameter of the RunDiagnosticService method does not match any SupportedLoopControl values specified in the DiagnosticServiceCapabilities and was ignored.

This alert would be sent if a DiagnosticServiceCapabilities exists for the DiagnosticTest and the client asked for a LoopControl that was not included in the SupportedLoopControl property. The LoopControl was ignored and the test ran without that control. If multiple LoopControls were specified and missing from the capabilities, there would be one alert message for each LoopControl.

The variables in this message are:

- Loop Control – Identifies the LoopControl value that was ignored.
- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

1459 – The Object Path of the element
1460 – The ElementName of the element
1461 – A unique user friendly name not in the model (such as, asset name)

1462 The Element Moniker can be any of these, but whichever one is used shall be used consistently
1463 for all managed elements of the same type within the scoping profile (such as, all disk drives in
1464 a system).

1465 With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType should be set to
1466 Capabilities Mismatch.

1467 With this alert, the PerceivedSeverity shall have the value 3 (Warning).

1468 7.9.22 DIAG28 – Capability to set LogStorage not supported

1469 The test ran, but a LogStorage specified in the DiagnosticSetting parameter of the RunDiagnosticService
1470 method does not match any SupportedLogStorage values specified in the DiagnosticServiceCapabilities
1471 and was ignored.

1472 This alert would be sent if a DiagnosticServiceCapabilities exists for the DiagnosticTest and the client
1473 asked for a LogStorage that was not included in the SupportedLogStorage property. The LogStorage was
1474 ignored and the test ran without that log option. If multiple LogStorage values were specified and missing
1475 from the capabilities, there would be one alert message for each LogStorage not in the
1476 SupportedLogStorage property.

1477 The variables in this message are:

1478 • LogStorage Option – Identifies the type of log storage that is not supported by the capabilities.

1479 • Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name
1480 property of the DiagnosticTest instance.

1481 • Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive)
1482 that was specified.

1483 This could be one of the following:

1484 – The Object Path of the element
1485 – The ElementName of the element
1486 – A unique user friendly name not in the model (such as, asset name)

1487 The Element Moniker can be any of these, but whichever one is used shall be used consistently
1488 for all managed elements of the same type within the scoping profile (such as, all disk drives in
1489 a system).

1490 With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType should be set to
1491 Capabilities Mismatch.

1492 With this alert, the PerceivedSeverity shall have the value 3 (Warning).

1493 7.9.23 DIAG30 – Capability to set PercentOfTestCoverage not supported

1494 The test ran, but the PercentOfTestCoverage option specified in the DiagnosticSetting parameter of the
1495 RunDiagnosticService method is not included in the SupportedServiceModes specified in the
1496 DiagnosticServiceCapabilities and was ignored.

1497 This alert would be sent if a DiagnosticServiceCapabilities exists for the DiagnosticTest and the client
1498 asked for a PercentOfTestCoverage, but PercentOfTestCoverage was not included in the
1499 SupportedServiceModes property. The PercentOfTestCoverage was ignored and the test ran without that
1500 option.

The variables in this message are:

- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

  – The Object Path of the element
  – The ElementName of the element
  – A unique user friendly name not in the model (such as, asset name)

  The Element Moniker can be any of these, but whichever one is used shall be used consistently for all managed elements of the same type within the scoping profile (such as, all disk drives in a system).

With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType should be set to Capabilities Mismatch.

With this alert, the PerceivedSeverity shall have the value 3 (Warning).

7.9.24 DIAG31 – Capability to set QuickMode not supported

The test ran, but the QuickMode option specified in the DiagnosticSetting parameter of the RunDiagnosticService method is not included in the SupportedServiceModes specified in the DiagnosticServiceCapabilities and was ignored.

This alert would be sent if a DiagnosticServiceCapabilities exists for the DiagnosticTest and the client asked for QuickMode, but QuickMode was not included in the SupportedServiceModes property. The QuickMode was ignored and the test ran without that option.

The variables in this message are:

- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

  – The Object Path of the element
  – The ElementName of the element
  – A unique user friendly name not in the model (such as, asset name)

  The Element Moniker can be any of these, but whichever one is used shall be used consistently for all managed elements of the same type within the scoping profile (such as, all disk drives in a system).

With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType should be set to Capabilities Mismatch.

With this alert, the PerceivedSeverity shall have the value 3 (Warning).

7.9.25 DIAG32 – Capability to set HaltOnError not supported

The test ran, but the HaltOnError option specified in the DiagnosticSetting parameter of the RunDiagnosticService method is not included in the SupportedServiceModes specified in the DiagnosticServiceCapabilities and was ignored.

This alert would be sent if a DiagnosticServiceCapabilities exists for the DiagnosticTest and the client asked for the HaltOnError option, but HaltOnError was not included in the SupportedServiceModes property. The HaltOnError was ignored and the test ran without that option.

The variables in this message are:

- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

  – The Object Path of the element
  – The ElementName of the element
  – A unique user friendly name not in the model (such as, asset name)

  The Element Moniker can be any of these, but whichever one is used shall be used consistently for all managed elements of the same type within the scoping profile (such as, all disk drives in a system).

With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType should be set to Capabilities Mismatch.

With this alert, the PerceivedSeverity shall have the value 3 (Warning).

7.9.26 DIAG33 – Capability to set NonDestructive to true not supported

The test was not run because the DiagnosticSetting NonDestructive was set toTRUE but the DiagnosticServiceCapabilities.SupportedServiceModes does not include NonDestructive.

This alert would be sent if the client supplied a DiagnosticSetting parameter to the RunDiagnosticService with NonDestructive set to TRUE, but the DiagnosticServiceCapabilities.SupportedServiceModes does not include NonDestructive. The test was not run because it might be destructive.

The variables in this message are:

- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

  – The Object Path of the element
  – The ElementName of the element
  – A unique user friendly name not in the model (such as, asset name)

  The Element Moniker can be any of these, but whichever one is used shall be used consistently for all managed elements of the same type within the scoping profile (such as, all disk drives in a system).

With this alert, the AlertType shall have the value 4 (Process Error).

With this alert, the PerceivedSeverity shall have the value 5 (Major).

7.9.27 DIAG43 – The Requested DiagnosticSettings is not supported

The test ran, but the requested DiagnosticSettings property parameter of the RunDiagnosticService method is not supported and was not used.

This alert would be sent if a DiagnosticSettings property requested in the RunDiagnosticService extrinsic method is not supported for the test or the element referenced.

The variables in this message are:

- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

  – The Object Path of the element
  – The ElementName of the element
  – A unique user friendly name not in the model (such as, asset name)

  The Element Moniker can be any of these, but whichever one is used shall be used consistently for all managed elements of the same type within the scoping profile (such as, all disk drives in a system).
- DiagnosticSettings Property – Identifies the DiagnosticSettings property by property name.
- DiagnosticSettings Value – Identifies the value requested.
- DiagnosticSettings Used – Identifies the value used instead of the requested value.

With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType should be set to Parameter Ignored.

With this alert, the PerceivedSeverity shall have the value 3 (Warning).

7.9.28 DIAG44 – The test did not start

The test did not start for one of a variety of reasons.

This alert would be sent as a test completion status message. The reason for why the test did not start would be identified by an earlier alert message (or in the log). For example, DIAG12 (see DSP1119) is an example of a message that might have been sent earlier.

The variables in this message are:

- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

  – The Object Path of the element
  – The ElementName of the element
  – A unique user friendly name not in the model (such as, asset name)

  The Element Moniker can be any of these, but whichever one is used shall be used consistently for all managed elements of the same type within the scoping profile (such as, all disk drives in a system).
- Log Object Path – This would be the Object Path of the CIM_DiagnosticLog instance.

With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType should be set to Test Not Started.

With this alert, the PerceivedSeverity shall have the value 2 (Information).

7.9.29 DIAG45 – The test aborted

The test was not completed for various reasons.

This alert would be sent as a test completion status message. The reason for why the test aborted would be identified by an earlier alert message (or in the log). For example, “The test was killed by the client” (see DIAG19 in DSP1119) and “The test was terminated by client” (see DIAG20 in DSP1119) are two messages that might precede this message.

The variables in this message are:

- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

  – The Object Path of the element
  – The ElementName of the element
  – A unique user friendly name not in the model (such as, asset name)

  The Element Moniker can be any of these, but whichever one is used shall be used consistently for all managed elements of the same type within the scoping profile (such as, all disk drives in a system).
- Log Object Path – Identifies the Object Path of the CIM_DiagnosticLog instance.

With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType should be set to Test Aborted.

With this alert, the PerceivedSeverity shall have the value 2 (Information).

7.9.30 DIAG46 – LogStorage mismatch with capabilities

The test ran, but a LogStorage request was not one that was identified in the DiagnosticServiceCapabilities.

This alert would be sent if the client requested one or more log storage types, but one of them is not identified in the DiagnosticServiceCapabilities.

NOTE If multiple types are not supported by the capabilities, multiple alerts would be sent. This does not report cases where the LogStorage is not supported for other reasons. That situation is handled by a separate alert (see 7.9.13).

The variables in this message are:

- Log Storage Requested – Identifies the LogStorage requested.
- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

  – The Object Path of the element
  – The ElementName of the element
  – A unique user friendly name not in the model (such as, asset name)

The Element Moniker can be any of these, but whichever one is used shall be used consistently for all managed elements of the same type within the scoping profile (such as, all disk drives in a system).

With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType should be set to Parameter Ignored.

With this alert, the PerceivedSeverity shall have the value 3 (Warning).

7.9.31 DIAG47 – Capability to set the DiagnosticSettings parameter not supported

The test ran, but a property in the DiagnosticSettings input to the RunDiagnosticService method was not supported and was ignored.

This alert would be sent if client attempted to set a DiagnosticSettings property that cannot be set.

The variables in this message are:

- Diag Setting Property – Identifies the property that was set, but not supported.
- Diag Setting Property Value – Identifies the value supplied for the property.
- Diagnostic Test Name – Identifies the Diagnostic Test instance that was run. This is the Name property of the DiagnosticTest instance.
- Element Moniker – Identifies a unique name for the element under test (such as, Disk Drive) that was specified.

  This could be one of the following:

  – The Object Path of the element
  – The ElementName of the element
  – A unique user friendly name not in the model (such as, asset name)

  The Element Moniker can be any of these, but whichever one is used shall be used consistently for all managed elements of the same type within the scoping profile (such as, all disk drives in a system).

With this alert, the AlertType shall have the value 1 (Other). The OtherAlertType should be set to Parameter Ignored.

With this alert, the PerceivedSeverity shall have the value 3 (Warning).

8 Methods

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

8.1 CIM_DiagnosticService.RunDiagnosticService( ) extrinsic method

The RunDiagnosticService( ) method is invoked to commence execution of a diagnostic service on a specific instance of a 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_DiagnosticSettingData and CIM_JobSettingData instances to apply as input parameters. The RunDiagnosticService( ) method shall capture the attributes of CIM_DiagnosticSettingData in an instance of CIM_DiagnosticSettingDataRecord. This information is useful for post-mortem analysis of diagnostic results.

1704 A job shall be instantiated to run and monitor the diagnostic service. The job shall also provide useful
1705 accounting and status information when the diagnostic service has been completed.

1706 RunDiagnosticService( ) return values are specified in Table 2 and parameters are specified in Table 3.
1707 No standard messages are defined.

1708 **Table 2 – RunDiagnosticService( ) method: Return code values**

Value	Description
0	Job completed with no error
2	Unknown or unspecified error
3	Cannot complete within the timeout period
4	Failed
5	Invalid parameter
0x8000..0xFFFF	Vendor specific

1709 **Table 3 – RunDiagnosticService( ) method: Parameters**

Qualifiers	Name	Type	Description/Values
IN	ManagedElement	CIM_ManagedElement	A reference that specifies the element upon which to run the diagnostic service
IN	DiagnosticSettings	[EmbeddedInstance(CIM_ DiagnosticSettingData)] string	A string (encoding a CIM_DiagnosticSettingData instance) that specifies the settings to be applied to the diagnostic service. If NULL, the diagnostic service's defaults are used.
IN	JobSettings	[EmbeddedInstance(CIM_ JobSettingData)] string	A string (encoding a CIM_ JobSettingData instance) that specifies the settings to be applied to the resulting job. If NULL, the job's defaults are used.
OUT	Job	CIM_ConcreteJob	A reference to the resulting job

1710 ## 8.2 CIM_Log.ClearLog( ) extrinsic method

1711 The ClearLog( ) method is invoked to delete all records (instances of CIM_DiagnosticRecord subclasses)
1712 that are associated with the log instance through the CIM_LogManagesRecord association. This method
1713 has no parameters, and no standard messages are defined.

1714 ClearLog return values are specified in Table 4.

1715 **Table 4 – ClearLog( ) method: Return code values**

Value	Description
0	Request was successfully executed
2	Unknown or unspecified error
3	Cannot be completed within the timeout period
4	Failed
5	Invalid parameter
0x8000..0xFFFF	Vendor specific

8.3 CIM_HelpService.GetHelp( ) extrinsic method

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 6 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 5 and parameters are specified in Table 6. No standard messages are defined.

Table 5 – GetHelp( ) method: Return code values

Value	Description
0	Request was successfully executed
2	Unknown or unspecified error
3	Cannot be completed within the timeout period
4	Failed
5	Invalid parameter
0x1000	Busy — indicates that the method cannot be invoked "at this time" It is not an error condition, but signals that the implementation is doing something else and cannot respond.
0x1001	Requested document not found
0x8000..0xFFFF	Vendor Reserved

Table 6 – 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.

8.4 Profile conventions for operations

For each profile class (including associations), the implementation requirements for operations, including those in the following default list, are specified in class-specific subclauses of this clause.

1729 The default list of operations is as follows:

1730 • GetInstance
1731 • EnumerateInstances
1732 • EnumerateInstanceNames
1733 • Associators
1734 • AssociatorNames
1735 • References
1736 • ReferenceNames

1737 8.5 CIM_DiagnosticTest

1738 Table 7 lists implementation requirements for operations. If implemented, these operations shall be
1739 implemented as defined in DSP0200. In addition, and unless otherwise stated in Table 7, all operations in
1740 the default list in 8.4 shall be implemented as defined in DSP0200.

1741 NOTE Related profiles may define additional requirements on operations for the profile class.

1742 **Table 7 – Operations: CIM_DiagnosticTest**

Operation	Requirement	Messages
GetInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None
InvokeMethod	Mandatory	None
ExecQuery	Optional	None
Associators	Mandatory	None
AssociatorNames	Mandatory	None
References	Optional	None
ReferenceNames	Optional	None

1743 8.6 CIM_AvailableDiagnosticService

1744 Table 8 lists implementation requirements for operations. If implemented, these operations shall be
1745 implemented as defined in DSP0200. In addition, and unless otherwise stated in Table 8, all operations in
1746 the default list in 8.4 shall be implemented as defined in DSP0200.

1747 NOTE Related profiles may define additional requirements on operations for the profile class.

1748 **Table 8 – Operations: CIM_AvailableDiagnosticService**

Operation	Requirement	Messages
GetInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None

8.7 CIM_ServiceAffectsElement

Table 9 lists implementation requirements for operations. If implemented, these operations shall be implemented as defined in DSP0200. In addition, and unless otherwise stated in Table 9, all operations in the default list in 8.4 shall be implemented as defined in DSP0200.

NOTE Related profiles may define additional requirements on operations for the profile class.

Table 9 – Operations: CIM_ServiceAffectsElement

Operation	Requirement	Messages
GetInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None

8.8 CIM_SoftwareIdentity

Table 10 lists implementation requirements for operations. If implemented, these operations shall be implemented as defined in DSP0200. In addition, and unless otherwise stated in Table 10, all operations in the default list in 8.4 shall be implemented as defined in DSP0200.

NOTE Related profiles may define additional requirements on operations for the profile class.

Table 10 – Operations: CIM_SoftwareIdentity

Operation	Requirement	Messages
GetInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None
ExecQuery	Optional	None
Associators	Mandatory	None
AssociatorNames	Mandatory	None
References	Optional	None
ReferenceNames	Optional	None

8.9 CIM_ElementSoftwareIdentity

Table 11 lists implementation requirements for operations. If implemented, these operations shall be implemented as defined in DSP0200. In addition, and unless otherwise stated in Table 11, all operations in the default list in 8.4 shall be implemented as defined in DSP0200.

NOTE Related profiles may define additional requirements on operations for the profile class.

Table 11 – Operations: CIM_ElementSoftwareIdentity

Operation	Requirement	Messages
GetInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None

8.10 CIM_HelpService

Table 12 lists implementation requirements for operations. If implemented, these operations shall be implemented as defined in DSP0200. In addition, and unless otherwise stated in Table 12, all operations in the default list in 8.4 shall be implemented as defined in DSP0200.

NOTE Related profiles may define additional requirements on operations for the profile class.

Table 12 – Operations: CIM_HelpService

Operation	Requirement	Messages
GetInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None
InvokeMethod	Mandatory	None
ExecQuery	Optional	None
Associators	Mandatory	None
AssociatorNames	Mandatory	None
References	Optional	None
ReferenceNames	Optional	None

8.11 CIM_ServiceAvailableToElement

Table 13 lists implementation requirements for operations. If implemented, these operations shall be implemented as defined in DSP0200. In addition, and unless otherwise stated in Table 13, all operations in the default list in 8.4 shall be implemented as defined in DSP0200.

NOTE Related profiles may define additional requirements on operations for the profile class.

Table 13 – Operations: CIM_ServiceAvailableToElement

Operation	Requirement	Messages
GetInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None

8.12 CIM_DiagnosticSettingData

Table 14 lists implementation requirements for operations. If implemented, these operations shall be implemented as defined in DSP0200. In addition, and unless otherwise stated in Table 14, all operations in the default list in 8.4 shall be implemented as defined in DSP0200.

NOTE Related profiles may define additional requirements on operations for the profile class.

1784 **Table 14 – Operations: CIM_DiagnosticSettingData**

Operation	Requirement	Messages
GetInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None
ExecQuery	Optional	None
Associators	Mandatory	None
AssociatorNames	Mandatory	None
References	Optional	None
ReferenceNames	Optional	None

1785 ## 8.13 CIM_DiagnosticServiceCapabilities

1786 Table 15 lists implementation requirements for operations. If implemented, these operations shall be
1787 implemented as defined in DSP0200. In addition, and unless otherwise stated in Table 15, all operations
1788 in the default list in 8.4 shall be implemented as defined in DSP0200.

1789 NOTE Related profiles may define additional requirements on operations for the profile class.

1790 **Table 15 – Operations: CIM_DiagnosticServiceCapabilities**

Operation	Requirement	Messages
GetInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None
ExecQuery	Optional	None
Associators	Mandatory	None
AssociatorNames	Mandatory	None
References	Optional	None
ReferenceNames	Optional	None

1791 ## 8.14 CIM_ElementCapabilities

1792 Table 16 lists implementation requirements for operations. If implemented, these operations shall be
1793 implemented as defined in DSP0200. In addition, and unless otherwise stated in Table 16, all operations
1794 in the default list in 8.4 shall be implemented as defined in DSP0200.

1795 NOTE Related profiles may define additional requirements on operations for the profile class.

1796 **Table 16 – Operations: CIM_ElementCapabilities**

Operation	Requirement	Messages
GetInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None

8.15 CIM_ElementSettingData

Table 17 lists implementation requirements for operations. If implemented, these operations shall be implemented as defined in DSP0200. In addition, and unless otherwise stated in Table 17, all operations in the default list in 8.4 shall be implemented as defined in DSP0200.

NOTE Related profiles may define additional requirements on operations for the profile class.

Table 17 – Operations: CIM_ElementSettingData

Operation	Requirement	Messages
GetInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None

8.16 CIM_DiagnosticLog

Table 18 lists implementation requirements for operations. If implemented, these operations shall be implemented as defined in DSP0200. In addition, and unless otherwise stated in Table 18, all operations in the default list in 8.4 shall be implemented as defined in DSP0200.

NOTE Related profiles may define additional requirements on operations for the profile class.

Table 18 – Operations: CIM_DiagnosticLog

Operation	Requirement	Messages
DeleteInstance	Optional	None
GetInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None
InvokeMethod	Mandatory	None
ExecQuery	Optional	None
Associators	Mandatory	None
AssociatorNames	Mandatory	None
References	Optional	None
ReferenceNames	Optional	None

8.16.1 DeleteInstance

DeleteInstance shall be supported if the implementation supports CIM_DiagnosticLog and allows the CIM_DiagnosticSettingData.ResultPersistence to be set to 0xFFFFFFFF (“Persist Forever”). This allows the client to delete the log and all its records with one operation on the log.

8.17 CIM_UseOfLog

Table 19 lists implementation requirements for operations. If implemented, these operations shall be implemented as defined in DSP0200. In addition, and unless otherwise stated in Table 19, all operations in the default list in 8.4 shall be implemented as defined in DSP0200.

NOTE Related profiles may define additional requirements on operations for the profile class.

1818 **Table 19 – Operations: CIM_UseOfLog**

Operation	Requirement	Messages
GetInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None

1819 ## 8.18 CIM_DiagnosticServiceRecord

1820 Table 20 lists implementation requirements for operations. If implemented, these operations shall be
1821 implemented as defined in DSP0200. In addition, and unless otherwise stated in Table 20, all operations
1822 in the default list in 8.4 shall be implemented as defined in DSP0200.

1823 NOTE Related profiles may define additional requirements on operations for the profile class.

1824 **Table 20 – Operations: CIM_DiagnosticServiceRecord**

Operation	Requirement	Messages
GetInstance	Mandatory	None
DeleteInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None
ExecQuery	Mandatory	None
Associators	Mandatory	None
AssociatorNames	Mandatory	None
References	Optional	None
ReferenceNames	Optional	None

1825 ### 8.18.1 DeleteInstance

1826 DeleteInstance shall be supported if the implementation supports DiagnosticServiceRecord and wants to
1827 give the client the ability to delete records after it has read them and stored them in client storage. This
1828 may be required if the test generates a lot of records and the test is at risk of running out of resources.

1829 ## 8.19 CIM_DiagnosticCompletionRecord

1830 Table 21 lists implementation requirements for operations. If implemented, these operations shall be
1831 implemented as defined in DSP0200. In addition, and unless otherwise stated in Table 21, all operations
1832 in the default list in 8.4 shall be implemented as defined in DSP0200.

1833 NOTE Related profiles may define additional requirements on operations for the profile class.

1834 **Table 21 – Operations: CIM_DiagnosticCompletionRecord**

Operation	Requirement	Messages
GetInstance	Mandatory	None
DeleteInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None
ExecQuery	Mandatory	None
Associators	Mandatory	None
AssociatorNames	Mandatory	None
References	Optional	None
ReferenceNames	Optional	None

1835 ### 8.19.1 DeleteInstance

1836 DeleteInstance shall be supported if the implementation supports DiagnosticCompletionRecord and
1837 wants to give the client the ability to delete records after it has read them and stored them in client
1838 storage. This may be required if the test generates a lot of records and the test is at risk of running out of
1839 resources.

1840 ## 8.20 CIM_DiagnosticSettingDataRecord

1841 Table 22 lists implementation requirements for operations. If implemented, these operations shall be
1842 implemented as defined in DSP0200. In addition, and unless otherwise stated in Table 22, all operations
1843 in the default list in 8.4 shall be implemented as defined in DSP0200.

1844 NOTE Related profiles may define additional requirements on operations for the profile class.

1845 **Table 22 – Operations: CIM_DiagnosticSettingDataRecord**

Operation	Requirement	Messages
GetInstance	Mandatory	None
DeleteInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None
ExecQuery	Mandatory	None
Associators	Mandatory	None
AssociatorNames	Mandatory	None
References	Optional	None
ReferenceNames	Optional	None

1846 ### 8.20.1 DeleteInstance

1847 DeleteInstance shall be supported if the implementation supports DiagnosticSettingDataRecord and
1848 wants to give the client the ability to delete records after it has read them and stored them in client
1849 storage. This may be required if the test generates a lot of records and the test is at risk of running out of
1850 resources.

8.21 CIM_LogManagesRecord

Table 23 lists implementation requirements for operations. If implemented, these operations shall be implemented as defined in DSP0200. In addition, and unless otherwise stated in Table 23, all operations in the default list in 8.4 shall be implemented as defined in DSP0200.

NOTE Related profiles may define additional requirements on operations for the profile class.

Table 23 – Operations: CIM_LogManagesRecord

Operation	Requirement	Messages
GetInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None

8.22 CIM_RecordAppliesToElement

Table 24 lists implementation requirements for operations. If implemented, these operations shall be implemented as defined in DSP0200. In addition, and unless otherwise stated in Table 24, all operations in the default list in 8.4 shall be implemented as defined in DSP0200.

NOTE Related profiles may define additional requirements on operations for the profile class.

Table 24 – Operations: CIM_RecordAppliesToElement

Operation	Requirement	Messages
GetInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None

8.23 CIM_CorrespondingSettingDataRecord

Table 25 lists implementation requirements for operations. If implemented, these operations shall be implemented as defined in DSP0200. In addition, and unless otherwise stated in Table 25, all operations in the default list in 8.4 shall be implemented as defined in DSP0200.

NOTE Related profiles may define additional requirements on operations for the profile class.

Table 25 – Operations: CIM_CorrespondingSettingDataRecord

Operation	Requirement	Messages
GetInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None

8.24 CIM_ServiceComponent

Table 26 lists implementation requirements for operations. If implemented, these operations shall be implemented as defined in DSP0200. In addition, and unless otherwise stated in Table 26, all operations in the default list in 8.4 shall be implemented as defined in DSP0200.

NOTE Related profiles may define additional requirements on operations for the profile class.

1874 **Table 26 – Operations: CIM_ServiceComponent**

Operation	Requirement	Messages
GetInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None

1875

9 Use cases

This clause 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.

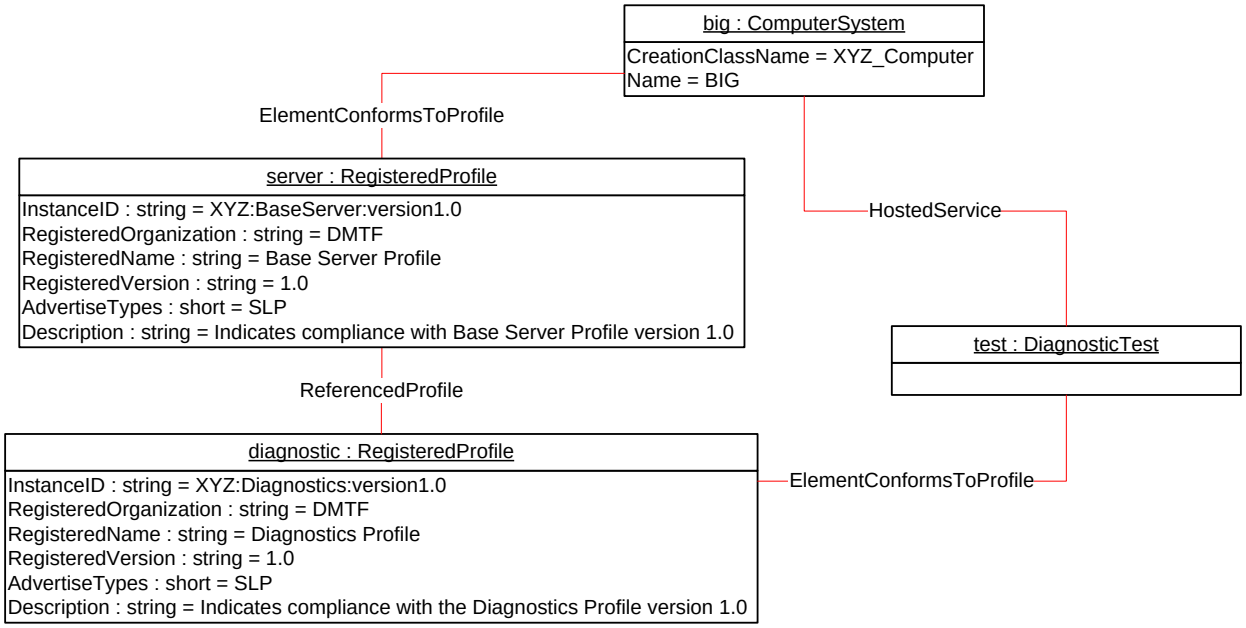
With CIM_ComputerSystem as the Scoping Class of this profile, the object diagram in Figure 2 shows how instances of CIM_RegisteredProfile may be used to identify the version of the *Diagnostics Profile* to which an instance of CIM_DiagnosticTest and its associated instances conform. In this example (using 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 v2.0*."

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 v2.0*.

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.

1903 The CIM_ prefix has been omitted from the class names in Figure 2 for simplicity and readability.



1904

1905 **Figure 2 – Registered profile**

1906 9.2 Use case summary

1907 Table 27 summarizes the use cases that are described in this clause. The use cases are categorized and
1908 named, and references are provided to the body text that describes the use case.

1909 NOTE Although use case names follow the convention for naming classes, properties, and methods in the schema,
1910 this naming was done for readability only and does not imply any functionality attached to the name.

1911 The CIM_ prefix has been omitted from the class names in the use cases for readability.

1912 **Table 27 – Diagnostics Profile use cases**

Category	Name	Description
Discover Available Diagnostics See 9.4.	GetAllDiagnostics	Find all diagnostics available on a system. See 9.4.1.
	GetAllDiagnosticMEPairs	Find all diagnostic/managed elements pairs available on a system. See 9.4.2.
	GetDiagnosticsForME	Find all the diagnostics available on a system for a managed element. See 9.4.3.
	GetMEsForDiagnostic	Find all the managed elements that support a particular diagnostic. See 9.4.4.
	GetCapabilitiesOfDiagnostic	Find the capabilities of a particular diagnostic. See 9.4.5.
	GetCharacteristicsOfDiagnostic	Find the characteristics of a particular diagnostic. See 9.4.6.

Category	Name	Description
	GetDiagnosticsWithCharacteristicsForME	Find all the diagnostics available on a system, for a managed element, with certain characteristics. See 9.4.7.
	GetDiagnosticsWithCapabilitiesForME	Find all the diagnostics available on a system, for a managed element, with certain capabilities. See 9.4.8.
	GetPackageSubtests	Find the subtests for a diagnostic test with the value of the DiagnosticTest.Characteristics property set to Is Package. See 9.4.9.
Configure Diagnostic See 9.5.	GetDefaultDiagnosticSettings	Find the default diagnostic settings for a diagnostic. See 9.5.1.
	CreateDiagnosticSettings	Create a unique setting for a diagnostic. See 9.5.2.
	GetDefaultJobSettings	Find the default job settings for a diagnostic. See 9.5.3.
	CreateJobSettings	Create a unique setting for a diagnostic job. See 9.5.4.
Execute and Control Diagnostic See 9.6.	RunDiagnostic	Run a diagnostic with default and unique settings. See 9.6.1.
	SuspendDiagnostic	Suspend a running diagnostic. See 9.6.2.
	ResumeDiagnostic	Resume a suspended diagnostic. See 9.6.3.
	AbortDiagnostic	Abort a running diagnostic. See 9.6.4.
	KillDiagnostic	Abort a running diagnostic immediately, with no attempt to perform a clean shutdown. See 9.6.5.
Discover Diagnostic Executions See 9.7.	GetAffectedMEs	Find all the managed elements affected by a running diagnostic. See 9.7.1.
	GetAllDiagnosticExecutionsForME	Find all the diagnostic executions on a system for a managed element. See 9.7.2.
	GetSpecificDiagnosticExecutions	Find all the executions of a specific diagnostic. See 9.7.3
	GetSpecificDiagnosticExecutionsForME	Find all the executions of a specific diagnostic for a particular managed element. See 9.7.4.
Discover Diagnostic Results (in-progress and final) See 9.8.	GetLogRecordsForDiagnostic	Find all the diagnostic log records for a particular diagnostic. See 9.8.1.
	GetLogRecordsForME	Find all the diagnostic log records for a particular managed element. See 9.8.2.

Category	Name	Description
	GetLogRecordsForMEAndDiagnostic	Find all the diagnostic log records for a particular diagnostic run on a particular managed element. See 9.8.3.
	GetDiagnosticExecutionFinalResults	Determine the final result of a diagnostic execution. See 9.8.4.
	GetDiagnosticExecutionResults	Find all diagnostic log records for a particular execution (job). See 9.8.5.
	GetDiagnosticExecutionSettings	Find the settings used in a diagnostic execution. See 9.8.6.
	GetDiagnosticProgress	Get the progress of a running diagnostic. See 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 clause.

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

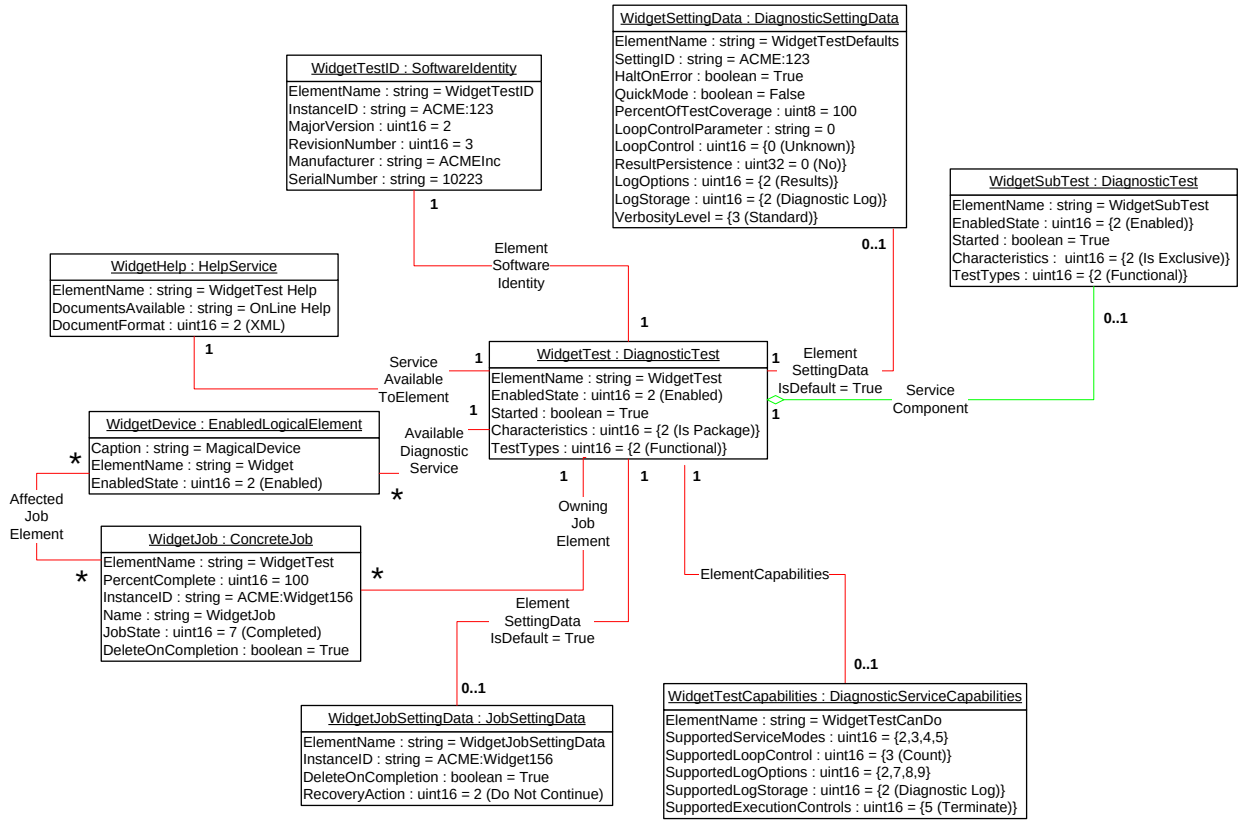


Figure 3 – Diagnostic services object diagram

9.4 Discover available diagnostics

The use cases in this clause 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:

The client calls the EnumerateInstances (or EnumerateInstanceNames) operation by using the DiagnosticTest class. 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.

The client calls the EnumerateInstances (or EnumerateInstanceNames) operation by using the AvailableDiagnosticService class. 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.

From the ManagedElement instance, the client calls the Associators operation by using AvailableDiagnosticService as the association class. 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.From the DiagnosticTest instance, the client calls the Associators operation by using AvailableDiagnosticService as the association class. 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 through a DiagnosticServiceCapabilities instance. A client can use the information in DiagnosticServiceCapabilities to generate an instance of DiagnosticSettingData that is passed as the DiagnosticSettings argument of the RunDiagnosticService extrinsic method of DiagnosticTest. The client can find the capabilities of a diagnostic as follows. Assume that the client starts at a known DiagnosticTest instance.

From the DiagnosticTest instance, the client calls the Associators operation by using ElementCapabilities as the association class and DiagnosticServiceCapabilities as the result class. 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. See the MOF file class definition for DiagnosticTest for further information.

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 (see 9.4.3) describes the necessary steps.

2) For each DiagnosticTest instance, the client checks the diagnostic characteristics. The GetCharacteristicsOfDiagnostic use case (see 9.4.6) describes the necessary steps.

   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 (see 9.4.3) describes the necessary steps.

2) For each DiagnosticTest instance, the client checks the diagnostic capabilities. The GetCapabilitiesOfDiagnostic use case (see 9.4.5) describes the necessary steps.

   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 by 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 by using ServiceComponent as the association class and DiagnosticTest as the result class.

   The operation returns the DiagnosticTest instances that are subtests of the known DiagnosticTest.

9.5 Configure diagnostic

The use cases in this clause 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 by using ElementSettingData as the association class and DiagnosticSettingData as the result class. The operation returns DiagnosticSettingData instances.

2) For each DiagnosticSettingData instance, the client calls the References operation by 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 DiagnosticTest name and the value of the ElementSettingData.IsDefault property is 1 (Is Default). If so, the DiagnosticSettingData instance represents the default diagnostic settings. The name of this DiagnosticSettingData instance may also be retrieved from ElementSettingData.SettingData property.

NOTE Because the implementation of DiagnosticSettingData is optional, it may not be available.

9.5.2 CreateDiagnosticSettings

To run a diagnostic test, the client invokes the RunDiagnosticService extrinsic method of DiagnosticTest. The DiagnosticSettings argument may be an empty string, NULL, or a string representing an embedded instance of DiagnosticSettingData. When DiagnosticSettings is an empty string or NULL, the test runs using the default settings which may or may not have been published by the implementation.

Note that the diagnostic default settings are represented by a DiagnosticSettingData 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. The client may use their own diagnostic settings as follows

1) The client discovers the diagnostic capabilities of the DiagnosticTest instance. The GetCapabilitiesOfDiagnostic use case (9.4.5) describes the necessary steps.

2) If Step 1 does not return an instance, the client can attempt to discover the default diagnostic settings of the DiagnosticTest instance. The GetDefaultDiagnosticSettings use case (9.5.1) describes the necessary steps.

3) If Step 2 does not return an instance or if the client chooses to create an instance of the DiagnosticSettingData class, a GetClass operation for DiagnosticSettingData can be performed and then used to create an instance locally in the client scope (for example, IwbemClassObject or CIMInstance object) based on the class definition.

4) The client modifies the created DiagnosticSettingData instance as necessary. However, the client should consider the diagnostic capabilities during the changes. If test capabilities are published, the client should set the values in DiagnosticSettingData instance based on the published capabilities (if any) because any setting for an unsupported capability shall be ignored.

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 by using ElementSettingData as the association class and JobSettingData as the result class.

2) For each JobSettingData instance that is returned, the client calls the References operation by using ElementSettingData as the result class.

3) For each ElementSettingData instance that is returned, the client determines whether the value of the ElementSettingData.ManagedElement property matches the DiagnosticTest name 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.

9.5.4 CreateJobSettings

To run a diagnostic test, the client invokes the RunDiagnosticService extrinsic method of DiagnosticTest. The JobSettings argument may be an empty string, NULL, or a string representing an embedded instance of JobSettingData. When JobSettings is an empty string or NULL, the job runs using the default settings which may or may not have been published by the implementation.

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. The client may use their own job settings as follows:

1) The client can attempt to discover the default job settings of the DiagnosticTest instance. The GetDefaultJobSettings use case (see 9.5.3) describes the necessary steps.
2) If Step 1 does not return an instance or if the client chooses to create an instance of the JobSettingData class, a GetClass operation for JobSettingData can be performed and then used to create an instance locally in the client scope (for example, IwbemClassObject or CIMInstance object) based on the class definition.
3) The client modifies the created JobSettingData instance as necessary.

9.6 Execute and control diagnostic

The RunDiagnosticService( ) method is invoked to start the diagnostic service. Input parameters are the ManagedElement being tested, the test settings, and the job settings to be used for the test execution. The test settings and job settings arguments are optional. If the settings argument is NULL or an empty string, the default settings are used. A reference to a ConcreteJob instance shall be returned.

An instance of ConcreteJob is created by the diagnostic implementation 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.

NOTE To expedite test data retrieval, the InstanceID values of ConcreteJob, DiagnosticSettingDataRecord, DiagnosticServiceRecord, and DiagnosticCompletionRecord are closely related to each other. For further information, see the Discover Diagnostic Results use cases in 9.8.

Figure 4 is an object diagram that shows the state of instances when a DiagnosticTest RunDiagnosticService( ) 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; that is, DiagnosticTest.Characteristics has the value of 2 (Is Exclusive). If this had been the case in this example, the DiagnosticTest would have returned an error on the second RunDiagnosticService( ) method call to run a test on ManagedElement2.

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

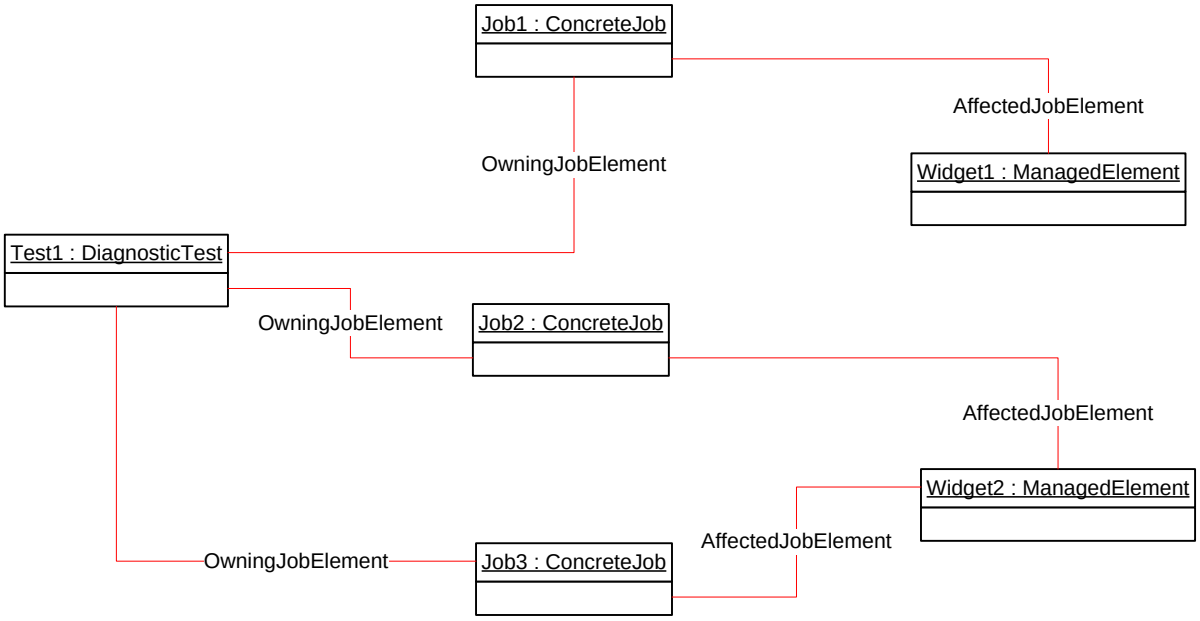


Figure 4 – Job example

9.6.1 RunDiagnostic

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

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

   The diagnostic service creates a Job instance to represent that test running on that managed element and shall return a reference to it in the return call from RunDiagnosticService( ). 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.

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 RunDiagnosticService( ) method, as shown in the following procedure. Assume that the client starts at a known ConcreteJob instance.

1) The client follows the OwningJobElement association from the ConcreteJob to the DiagnosticTest

2) The client follows the ElementCapabilities association from the DiagnosticTest to the DiagnosticServiceJobCapabilities for the service.

   The DiagnosticServiceJobCapabilities.RequestedStatesSupported property indicates the permitted values of the RequestedState input parameter for the ConcreteJob.RequestStateChange( ) extrinsic method. Because DiagnosticServiceJobCapabilities is an optional class, a client may not be able to examine an instance to determine which values of RequestedState to use. If a client invokes

ConcreteJob.RequestStateChange( ) to change to an unsupported state, the extrinsic method shall return 4097 (Invalid State Transition).

3) The client checks the DiagnosticServiceJobCapabilities.RequestedStatesSupported property for the value of 3 (Suspend).

   If the value exists, the Job supports suspension.

4) The client should cache the capabilities for the DiagnosticTest for future reference.

5) Assuming the job supports the suspend operation, the client calls the RequestStateChange( ) method for the ConcreteJob instance, passing in a RequestedState value of 3 (Suspend).

   After the transition is completed successfully, the ConcreteJob that represents the test will set the value of the JobState property to 5 (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 RunDiagnosticService( ) 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 DiagnosticServiceJobCapabilities for the service.

   The DiagnosticServiceJobCapabilities.RequestedStatesSupported property indicates the permitted values of the RequestedState input parameter for the ConcreteJob.RequestStateChange( ) extrinsic method. Because DiagnosticServiceJobCapabilities is an optional class, a client may not be able to examine an instance to determine which values of RequestedState to use. If a client invokes ConcreteJob.RequestStateChange( ) to change to an unsupported state, the extrinsic method shall return 4097 (Invalid State Transition).

2) The client checks the DiagnosticServiceJobCapabilities.RequestedStatesSupported property for the value of 2 (Start).

   If the value exists, the Job supports resumption.

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

4) The client calls the RequestStateChange( ) method of DiagnosticTest, passing in a RequestedState value of 2 (Start).

   After the transition is completed successfully, the ConcreteJob that represents the test will set the value of the JobState property to 4 (Running) and set the value of TimeOfLastStateChange to the current time.

NOTE The JobState property may transition from the value 3 (Starting) before the final transition to the value of 4 (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 RunDiagnosticService( ) 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 DiagnosticServiceJobCapabilities for the service.

   If no DiagnosticServiceJobCapabilities is returned, proceed to step 3. Support for Terminate is mandatory.

2) The client checks the DiagnosticServiceJobCapabilities.RequestedStatesSupported property for the value of 4 (Terminate).

   If the value exists, the Job supports termination.

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

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

   After the transition is completed successfully, the ConcreteJob that represents the test will set the value of the JobState property to 8 (Terminated) and set the value of TimeOfLastStateChange to the current time.

NOTE The JobState property may transition to Number (Shutting Down) before the final transition to 8 (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 RunDiagnosticService( ) 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 DiagnosticServiceJobCapabilities for the service.

   If no DiagnosticServiceJobCapabilities is returned, proceed to step 3. Support for Kill is mandatory.

2) The client checks the DiagnosticServiceJobCapabilities.RequestedStatesSupported property for the value of 5 (Kill).

   If the value exists, the Job supports kill.

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

4) The client calls the RequestStateChange( ) method, passing in a RequestedState value of 5 (Kill).

   After the transition is completed successfully, the ConcreteJob that represents the test will set the value of the JobState property to 9 (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 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 by 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 by 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 by 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 by 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. 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 by using OwningJobElement as the association class.

The operation returns the ConcreteJob instances launched by the DiagnosticTest. 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 by 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 by 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 by 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 by 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 by 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.

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 required logging implementation, using the DiagnosticLog class. DiagnosticLog is a special subclass of RecordLog that supports a standard mechanism for organizing and retrieving (using ExecQuery) the records that diagnostics services generate. Use of this common logging mechanism can substantially increase interoperability and simplify client design.

NOTE A separate DiagnosticLog instance shall be created each time the RunDiagnosticService method of DiagnosticTest is invoked.

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

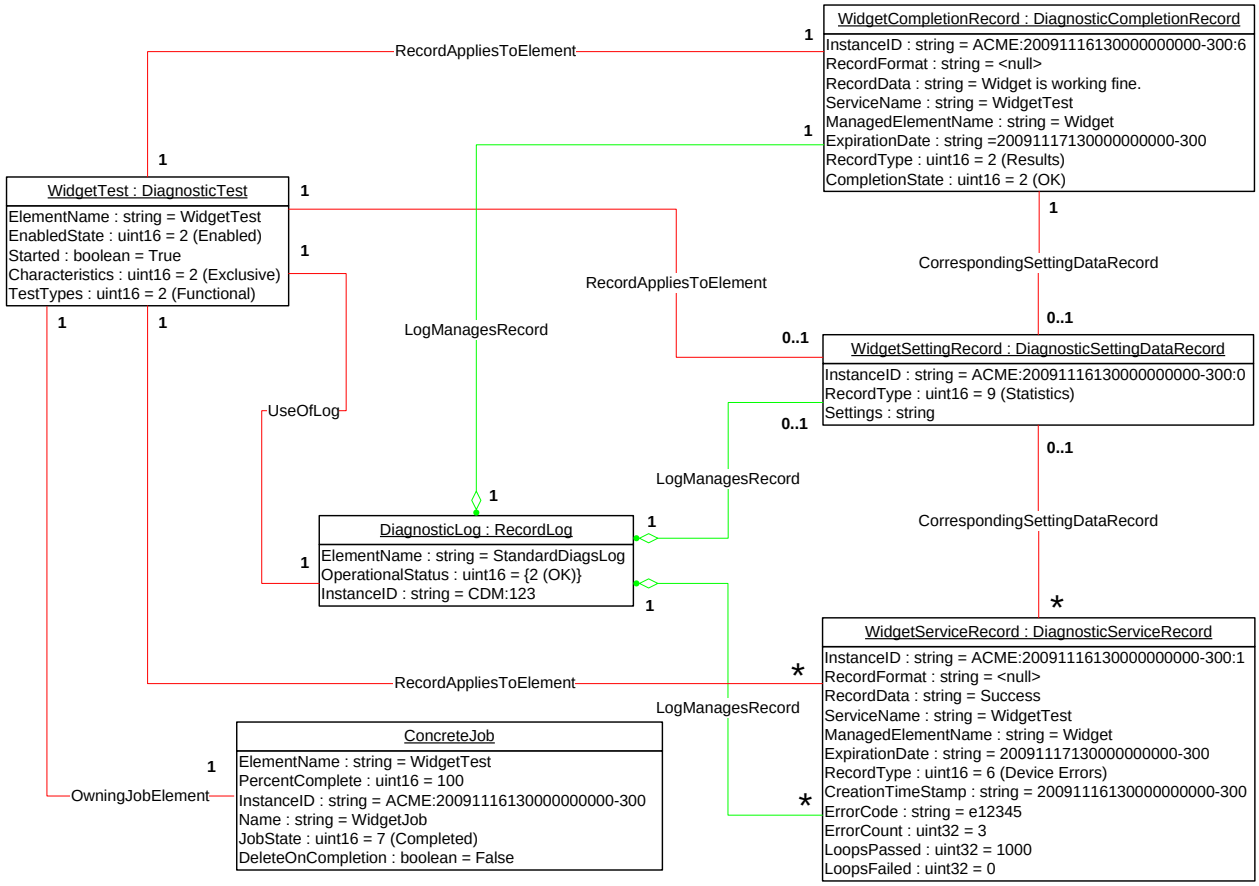


Figure 5 – Diagnostic logging object diagram

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>'

   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 is as follows:

Assume that the client starts at the known DiagnosticTest instance.

1) From the DiagnosticTest instance, the client calls the Associators operation by using UseOfLog as the association class and DiagnosticsLog as the result class.

   The operation returns the DiagnosticLog instances that contain records for the DiagnosticTest.

2) For each DiagnosticLog instance, the client calls the Associators operation by 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 whether 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>'

   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 is as follows:

Assume that the client starts at the known ManagedElement instance.

1) From the ManagedElement instance, the client calls the Associators operation by 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 by using UseOfLog as the association class and DiagnosticLog as the result class.

   The operation returns the DiagnosticLog instances that contain records for the DiagnosticTest.

3) For each DiagnosticLog instance, the client calls the Associators operation by 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 whether 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>'

   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 is as follows:

Assume that the client starts at the known DiagnosticTest instance.

1) From the DiagnosticTest instance, the client calls the Associators operation by using UseOfLog as the association class and DiagnosticLog as the result class.

   The operation returns the DiagnosticLog instances that contain records for the DiagnosticTest.

2) For each DiagnosticLog instance, the client calls the Associators operation by 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.ElementName to determine whether the instance is one created for the specific DiagnosticTest and ManagedElement.

9.8.4 GetDiagnosticExecutionFinalResults

The client can determine the final result of 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 is completed and removed if the client knows the original ConcreteJob.InstanceID.

1) The client calls the ExecQuery operation as follows:

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

   The operation returns the DiagnosticCompletionRecord instance created for the specific ConcreteJob.

   NOTE Only one DiagnosticCompletionRecord shall be returned.

2) The client reads the DiagnosticCompletionRecord.CompletionState property, which shows the final result (Passed, Warning, Failed, Aborted, Incomplete, and so on) of the diagnostic execution.

An alternate method without using ExecQuery is as follows:

Assume that the client starts at the known DiagnosticTest instance.

1) From the DiagnosticTest instance, the client calls the Associators operation by using UseOfLog as the association class and DiagnosticLog as the result class.

   The operation returns the DiagnosticLog instances that contain records for the DiagnosticTest.

2) For each DiagnosticLog instance, the client calls the Associators operation by using LogManagesRecord as the association class and DiagnosticCompletionRecord as the result class.

   The operation returns the DiagnosticCompletionRecord instances in the Log.

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

9.8.5 GetDiagnosticExecutionResults

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

The diagnostic implementation will store the results of running the diagnostic in the manner selected through the LogStorage setting. The most common mechanism is for the implementation 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 DiagnosticSettingDataRecord for preserving the test settings. The implementations for these classes will implement ExecQuery to simplify the retrieval of records.

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

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 is completed and 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>%'

   The operation returns the DiagnosticRecord instances created for the specific ConcreteJob which may either be DiagnosticServiceRecord or DiagnosticSettingDataRecord instances.

NOTE Only one DiagnosticSettingDataRecord shall be returned, while one or more DiagnosticServiceRecord instances may be returned.

9.8.6 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 is completed and removed if the client knows the original ConcreteJob.InstanceID.

1) The client calls the ExecQuery operation as follows:

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

   The operation returns the DiagnosticSettingDataRecord instance created for the specific ConcreteJob.

   NOTE Only one DiagnosticSettingDataRecord instance shall be returned.

2) The client reads the DiagnosticSettingDataRecord.Settings property, which is a DiagnosticSettingData embedded instance that contains the settings of the diagnostic execution.

An alternate method without using ExecQuery is as follows:

Assume that the client starts at the known DiagnosticTest instance.

1) From the DiagnosticTest instance, the client calls the Associators operation by 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 by using LogManagesRecord as the association class and DiagnosticSettingDataRecord as the result class.

   The operation returns the DiagnosticSettingDataRecord 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 whether the instance is one created for the specific execution of the DiagnosticTest.

4) The client reads the DiagnosticSettingDataRecord.Settings property, which is a DiagnosticSettingData embedded instance that contains the settings of the diagnostic execution.

Another alternate method without using ExecQuery is as follows:

NOTE This alternative use case depends on 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 by using UseOfLog as the association class and DiagnosticLog as the result class.

   The operation returns the DiagnosticLog instances that contain records for the DiagnosticTest.

2) For each DiagnosticLog instance, the client calls the Associators operation by using LogManagesRecord as the association class and DiagnosticSettingDataRecord as the result class.

   The operation returns the DiagnosticSettingRecord instances in the Log.

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

4) From the DiagnosticSettingDataRecord instance, the client calls the Associators operation by 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

9.8.7 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 implementations 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 (see 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 (see 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

The indication received will notify the client that the PercentComplete property for the specific ConcreteJob has changed.

3) The client can use the SourceInstance property in the indication to see the actual PercentComplete value to determine test progress.

10 CIM Elements

Table 28 shows the instances of CIM Elements for this profile. Instances of the CIM Elements shall be implemented as described in Table 28. Clauses 7 ("Implementation") and 8 ("Methods") may impose additional requirements on these elements.

Table 28 – CIM Elements: Diagnostics Profile

Element Name	Requirement	Description
Classes		
CIM_AvailableDiagnosticService	Mandatory	Association to link diagnostic services that can be launched against managed elements See 10.1.
CIM_CorrespondingSettingDataRecord (DiagnosticServiceRecord)	Optional	Association to link a settings record to its corresponding service records. If CIM_DiagnosticSettingDataRecord is implemented, this class is Mandatory. See 10.2.
CIM_CorrespondingSettingDataRecord (DiagnosticCompletionRecord)	Optional	Association to link a settings record to its corresponding completion records. If CIM_DiagnosticSettingDataRecord is implemented, this class is Mandatory. See 10.3.
CIM_DiagnosticCompletionRecord	Mandatory	Records that contain serviced completion information See 7.6 and 10.4.
CIM_DiagnosticLog	Mandatory	Although several legitimate mechanisms for logging results exist (see CIM_DiagnosticSettingData.LogStorage), aggregation of diagnostic records to a diagnostic log is Mandatory. See 7.5 and 10.5.
CIM_DiagnosticServiceCapabilities	Optional	See 7.3 and 10.6.
CIM_DiagnosticServiceRecord	Mandatory	See 7.6 and 10.7.
CIM_DiagnosticSettingData (Default)	Optional	See 7.4 and 10.8.
CIM_DiagnosticSettingData (Client)	Optional	See 7.4 and 10.9.
CIM_DiagnosticSettingDataRecord	Optional	See 7.6 and 10.10.
CIM_DiagnosticTest	Mandatory	See 7.1 and 10.11.
CIM_ElementCapabilities	Optional	See 10.12.
CIM_ElementSettingData (JobSettingData)	Optional	See 10.13.
CIM_ElementSettingData (DiagnosticSettingData)	Optional	See 10.14.

Element Name	Requirement	Description
CIM_ElementSoftwareIdentity	Mandatory	See 10.15.
CIM_FilterCollection	Optional	See 10.16
CIM_HelpService	Optional	See 10.16.
CIM_HostedService	Mandatory	See 10.18 and 9.1.
CIM_IndicationFilter	Mandatory	See 10.19
CIM_LogManagesRecord	Mandatory	See 10.20.
CIM_MemberOfCollection	Optional	See 10.21
CIM_OwningCollectionElement	Optional	See 10.22
CIM_RecordAppliesToElement	Optional	See 10.23.
CIM_RegisteredProfile	Mandatory	See 10.24.
CIM_ServiceAffectsElement	Mandatory	See 10.25.
CIM_ServiceAvailableToElement	Mandatory	See 10.26.
CIM_ServiceComponent	Optional	See 10.27.
CIM_SoftwareIdentity	Mandatory	See 10.28.
CIM_UseOfLog	Mandatory	See 10.29.
Indications		
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG0"	Mandatory	Query Language="DMTF:CQL" Name="DMTF:Diagnostics:DIAG0" See 7.9.1
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG1"	Optional	Query Language="DMTF:CQL" Name="DMTF:Diagnostics:DIAG1" See 7.9.2
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG3"	Mandatory	Query Language="DMTF:CQL" Name="DMTF:Diagnostics:DIAG3" See 7.9.3
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG4"	Mandatory	Query Language="DMTF:CQL" Name="DMTF:Diagnostics:DIAG4" See 7.9.4
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG5"	Optional	Query Language="DMTF:CQL" Name="DMTF:Diagnostics:DIAG5" See 7.9.5
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG6"	Optional	Query Language="DMTF:CQL" Name="DMTF:Diagnostics:DIAG6" See 7.9.6
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG7"	Optional	Query Language="DMTF:CQL" Name="DMTF:Diagnostics:DIAG7" See 7.9.7
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG8"	Optional	Query Language="DMTF:CQL" Name="DMTF:Diagnostics:DIAG8" See 7.9.8

Element Name	Requirement	Description
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG10"	Optional	Query Language=”DMTF:CQL” Name=”DMTF:Diagnostics:DIAG10” See 7.9.9
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG11"	Optional	Query Language=”DMTF:CQL” Name=”DMTF:Diagnostics:DIAG11” See 7.9.10
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG13"	Optional	Query Language=”DMTF:CQL” Name=”DMTF:Diagnostics:DIAG13” See 7.9.11
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG14"	Optional	Query Language=”DMTF:CQL” Name=”DMTF:Diagnostics:DIAG14” See 7.9.12
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG15"	Optional	Query Language=”DMTF:CQL” Name=”DMTF:Diagnostics:DIAG15” See 7.9.13
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG16"	Optional	Query Language=”DMTF:CQL” Name=”DMTF:Diagnostics:DIAG16” See 7.9.14
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG17"	Optional	Query Language=”DMTF:CQL” Name=”DMTF:Diagnostics:DIAG17” See 7.9.15
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG18"	Optional	Query Language=”DMTF:CQL” Name=”DMTF:Diagnostics:DIAG18” See 7.9.16
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG22"	Optional	Query Language=”DMTF:CQL” Name=”DMTF:Diagnostics:DIAG22” See 7.9.17
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG23"	Optional	Query Language=”DMTF:CQL” Name=”DMTF:Diagnostics:DIAG23” See 7.9.18
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG24"	Optional	Query Language=”DMTF:CQL” Name=”DMTF:Diagnostics:DIAG24” See 7.9.19
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG26"	Optional	Query Language=”DMTF:CQL” Name=”DMTF:Diagnostics:DIAG26” See 7.9.20
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG27"	Optional	Query Language=”DMTF:CQL” Name=”DMTF:Diagnostics:DIAG27” See 7.9.21
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG28"	Optional	Query Language=”DMTF:CQL” Name=”DMTF:Diagnostics:DIAG28” See 7.9.22

Element Name	Requirement	Description
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG30"	Optional	Query Language="DMTF:CQL" Name="DMTF:Diagnostics:DIAG30" See 7.9.23
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG31"	Optional	Query Language="DMTF:CQL" Name="DMTF:Diagnostics:DIAG31" See 7.9.24
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG32"	Optional	Query Language="DMTF:CQL" Name="DMTF:Diagnostics:DIAG32" See 7.9.25
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG33"	Optional	Query Language="DMTF:CQL" Name="DMTF:Diagnostics:DIAG33" See 7.9.26
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG43"	Optional	Query Language="DMTF:CQL" Name="DMTF:Diagnostics:DIAG43" See 7.9.27
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG44"	Mandatory	Query Language="DMTF:CQL" Name="DMTF:Diagnostics:DIAG44" See 7.9.28
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG45"	Mandatory	Query Language="DMTF:CQL" Name="DMTF:Diagnostics:DIAG45" See 7.9.29
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG46"	Optional	Query Language="DMTF:CQL" Name="DMTF:Diagnostics:DIAG46" See 7.9.30
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="DMTF" and MessageID="DIAG47"	Optional	Query Language="DMTF:CQL" Name="DMTF:Diagnostics:DIAG47" See 7.9.31

10.1 CIM_AvailableDiagnosticService

CIM_AvailableDiagnosticService is used to discover the diagnostic services that are installed for a particular managed element. Table 29 provides information about the properties of CIM_AvailableDiagnosticService.

Table 29 – 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.
EstimatedDurationOfService	Mandatory	See 7.2.1.
EstimatedDurationQualifier	Optional	See 7.2.2.

10.2 CIM_CorrespondingSettingDataRecord (DiagnosticServiceRecord)

CIM_CorrespondingSettingDataRecord is used to associate a service record with the corresponding setting data record. Table 30 provides information about the properties of CIM_CorrespondingSettingDataRecord.

Table 30 – Class: CIM_CorrespondingSettingDataRecord

Properties	Requirement	Notes
DataRecord	Mandatory	**Key**: This property shall be a reference to an instance of CIM_DiagnosticServiceRecord.
SettingsRecord	Mandatory	**Key**: This property shall be a reference to an instance of CIM_DiagnosticSettingDataRecord. Cardinality 1

10.3 CIM_CorrespondingSettingDataRecord (DiagnosticCompletionRecord)

CIM_CorrespondingSettingDataRecord is used to associate a completion record with the corresponding setting data record. Table 31 provides information about the properties of CIM_CorrespondingSettingDataRecord.

Table 31 – Class: CIM_CorrespondingSettingDataRecord

Properties	Requirement	Notes
DataRecord	Mandatory	**Key**: This property shall be a reference to an instance of CIM_DiagnosticCompletionRecord.
SettingsRecord	Mandatory	**Key**: This property shall be a reference to an instance of CIM_DiagnosticSettingDataRecord. Cardinality 1

10.4 CIM_DiagnosticCompletionRecord

CIM_DiagnosticCompletionRecord is used to report the final state of diagnostic execution (OK, Failed, Incomplete, Aborted, and so on). Table 32 provides information about the properties of CIM_DiagnosticCompletionRecord.

Table 32 – Class: CIM_DiagnosticCompletionRecord

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]*$")
CreationTimeStamp	Mandatory	None.
RecordData	Mandatory	None.
RecordFormat	Mandatory	None.
ServiceName	Mandatory	The ServiceName property shall be constructed as follows: <OrgID>:<TestName>. (pattern "^.*[:].*$")
ManagedElementName	Mandatory	This property will be formatted as a free-form string of variable length. (pattern ".*")
RecordType	Mandatory	The record type shall be 2 (Results).
ExpirationDate	Mandatory	See 7.6.1.
CompletionState	Mandatory	None.
OtherCompletionStateDescription	Conditional	If CompletionState has the value 1 (Other), this property is Mandatory.
LoopsPassed	Optional	If looping is supported, this property is Mandatory.
LoopsFailed	Optional	If looping is supported, this property is Mandatory.
ErrorCode	Mandatory	This property shall be an array that contains the error codes of all errors generated by the diagnostic service execution. If there are no errors, this property may have the value NULL.

Properties	Requirement	Notes
ErrorCount	Mandatory	This property shall be an array where each position should contain the number of times that an error (which can be identified by the same position of the ErrorCode array) happened. If there are no errors, this property may have the value NULL.

10.5 CIM_DiagnosticLog

CIM_DiagnosticLog represents a log that aggregates all of the results (records) that the execution of a diagnostic generates. Table 33 provides information about the properties of CIM_DiagnosticLog.

Table 33 – Class: CIM_DiagnosticLog

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 8.2.

10.6 CIM_DiagnosticServiceCapabilities

CIM_DiagnosticServiceCapabilities publishes the diagnostic service's capabilities, such as settings and execution controls that are supported. Table 34 provides information about the properties of CIM_DiagnosticServiceCapabilities.

Table 34 – 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.) <LocalID> should be set to the Name property value of the Service to which these capabilities apply. (pattern "^.*[:].*$")
ElementName	Mandatory	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 ".*")

Properties	Requirement	Notes
SupportedServiceModes	Optional	If service modes are supported, they shall be published using this property.
OtherSupportedServiceModesDescriptions	Conditional	If SupportedServiceModes includes the value of 1 (Other), this property is Mandatory.
SupportedLoopControl	Optional	If looping is supported, its controls shall be published using this property.
OtherSupportedLoopControlDescriptions	Conditional	If SupportedLoopControl includes 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 includes 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 includes the value 1 (Other), this property is Mandatory.
SupportedExecutionControls	Optional	**Deprecated:** If any execution controls are supported, they shall be published using this property.
OtherSupportedExecutionControls Descriptions	Conditional	**Deprecated:** If SupportedExecutionControls includes the value 1 (Other), this property is Mandatory.

10.7 CIM_DiagnosticServiceRecord

CIM_DiagnosticServiceRecord is used to report diagnostic service messages, such as results, errors, warnings, and status. Table 35 provides information about the properties of CIM_DiagnosticServiceRecord.

Table 35 – 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.
RecordFormat	Mandatory	None.
LoopsPassed	Mandatory	None.

Properties	Requirement	Notes
LoopsFailed	Mandatory	None.
ErrorCode	Conditional	If the RecordType value is 7(Device Errors) or 8 (Service Errors), this property shall be an array that contains only one error code number. 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. If the RecordType value is not 2 (Results) or 7(Device Errors) or 8 (Service Errors), this property may be NULL. The property will be formatted as a free-form string of variable length. (pattern ".*")
ErrorCount	Conditional	If the RecordType value is 7(Device Errors) or 8 (Service Errors), this property shall be an array that has just one element whose value is 1. If the RecordType value is 2 (Results), this property should be an array where each position should contain the number of times that an error occurred that can be identified by the same position in the ErrorCode array. If the RecordType value is not 2 (Results) or 7(Device Errors) or 8 (Service Errors), this property may be NULL.
ServiceName	Mandatory	This property shall be constructed as follows: <OrgID>:<TestName>. (pattern "^.*[:].*$")
ManagedElementName	Mandatory	This property shall be formatted as a free-form string of variable length. (pattern ".*")
RecordType	Mandatory	A RecordType value of 2 (Results) shall be used to log interim results from the diagnostic service execution (for example, results from a subtest).
OtherRecordTypeDescription	Conditional	If RecordType has the value 1 (Other), this property is Mandatory.
ExpirationDate	Mandatory	See 7.6.1.

10.8 CIM_DiagnosticSettingData (Default)

Diagnostic services use CIM_DiagnosticSettingData to publish default settings by using CIM_ElementSettingData where the IsDefault property has the value of TRUE. Table 36 provides information about the properties of CIM_DiagnosticSettingData.

Table 36 – Class: CIM_DiagnosticSettingData

Properties	Requirement	Notes
InstanceID	Mandatory	**Key**: InstanceID should be constructed using the following preferred algorithm: <OrgID>:<LocalID> (See the MOF file for more detail.) <LocalID> should be set to a time stamp (CIM DateTime). For example: ACME:19980525133015.0000000-300 (pattern "^.*[:].*$")
ElementName	Mandatory	This property shall be formatted as a free-form string of variable length. (pattern ".*")
HaltOnError	Optional	If the DiagnosticServiceCapabilities.SupportedServiceModes includes a value of 4 (HaltOnError), this property can be used to affect test behavior. When this property is TRUE, the service should halt after finding the first error.
QuickMode	Optional	If the DiagnosticServiceCapabilities.SupportedServiceModes includes a value of 3 (QuickMode), this property can be used to affect test behavior. When this property is TRUE, the service should attempt to run in an accelerated fashion either by reducing the coverage or by reducing the number of tests performed.
PercentOfTestCoverage	Optional	If the DiagnosticServiceCapabilities.SupportedServiceModes includes a value of 2 (PercentOfTestCoverage), this property can be used to affect test behavior. This property requests that the service reduce test coverage to the specified percentage.
NonDestructive	Optional	If the DiagnosticServiceCapabilities.SupportedServiceModes includes a value of 7 (NonDestructive), this property can be used to affect test behavior. When this property is TRUE, the service should not run destructive tests.

Properties	Requirement	Notes
LoopControl	Optional	This property is used in combination with LoopControlParameter to set one or more loop control mechanisms that limit the number of times that a test should be repeated.
LoopControlParameter	Conditional	If a LoopControl includes the value of 3 (Count) or 5 (ErrorCount), the corresponding LoopControlParameter array element shall represent a uint32 numeric value. If a LoopControl includes the value of 4 (Timer), the corresponding LoopControlParameter array element shall represent a datetime value. (pattern "^b[01]* | ^d[0123456789]* | ^x[0123456789ABCDEFabcdef]* | ^[0123456789]*")
OtherLoopControlDescriptions	Conditional	If LoopControl includes the value 1 (Other), this property is Mandatory.
ResultPersistence	Mandatory	If the DiagnosticServiceCapabilities.SupportedServiceModes includes a value of 5 (ResultPersistence), this property can be used to affect test behavior. This property specifies how many seconds the records should persist after service execution finishes. 0 (zero) indicates “no persistence” and 0xFFFFFFFF indicates “persist forever”. See 7.6.1.
LogOptions	Optional	This property specifies the types of data that should be logged by the diagnostic service.
OtherLogOptionsDescriptions	Conditional	If LogOptions includes the value 1 (Other), this property is Mandatory.
LogStorage	Optional	This property specifies the logging mechanism to store the diagnostic results. This property must be one of the values in DiagnosticServiceCapabilities.LogStorage
OtherLogStorageDescriptions	Conditional	If LogStorage includes 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_DiagnosticSettingData (Client)

A client uses CIM_DiagnosticSettingData to override the defaults settings and run a diagnostic service using specific settings. Such settings are passed as the DiagnosticSettings argument when the RunDiagnosticService( ) extrinsic method of CIM_DiagnosticTest is invoked. Table 37 provides information about the properties of CIM_DiagnosticSettingData.

Table 37 – Class: CIM_DiagnosticSettingData

Properties	Requirement	Notes
InstanceID	Mandatory	**Key**: InstanceID should be constructed using the following preferred algorithm: <OrgID>:<LocalID> (See the MOF file for more detail.) <LocalID> should be set to a time stamp (CIM DateTime). For example: ACME:19980525133015.0000000-300 (pattern "^.*[:].*$")
ElementName	Mandatory	This property shall be formatted as a free-form string of variable length. (pattern ".*")
HaltOnError	Optional	If the DiagnosticServiceCapabilities.SupportedServiceModes includes a value of 4 (HaltOnError), this property can be used to affect test behavior. When this property is TRUE, the service should halt after finding the first error.
QuickMode	Optional	If the DiagnosticServiceCapabilities.SupportedServiceModes includes a value of 3 (QuickMode), this property can be used to affect test behavior. When this property is TRUE, the service should attempt to run in an accelerated fashion either by reducing the coverage or by reducing the number of tests performed.
PercentOfTestCoverage	Optional	If the DiagnosticServiceCapabilities.SupportedServiceModes includes a value of 2 (PercentOfTestCoverage), this property can be used to affect test behavior. This property requests that the service reduce test coverage to the specified percentage.
NonDestructive	Optional	If the DiagnosticServiceCapabilities.SupportedServiceModes includes a value of 7 (NonDestructive), this property can be used to affect test behavior. When this property is TRUE, the service should not run destructive tests.

Properties	Requirement	Notes
LoopControl	Optional	This property is used in combination with LoopControlParameter to set one or more loop control mechanisms that limit the number of times that a test should be repeated.
LoopControlParameter	Conditional	If a LoopControl includes the value of 3 (Count) or 5 (ErrorCount), the corresponding LoopControlParameter array element shall represent a uint32 numeric value. If a LoopControl includes the value of 4 (Timer), the corresponding LoopControlParameter array element shall represent a datetime value. (pattern "^b[01]* | ^d[0123456789]* | ^x[0123456789ABCDEFabcdef]* | ^[0123456789]*")
OtherLoopControlDescriptions	Conditional	If LoopControl includes the value 1 (Other), this property is Mandatory.
ResultPersistence	Mandatory	If the DiagnosticServiceCapabilities.SupportedServiceModes array contains a value of 5 (ResultPersistence), this property can be used to affect test behavior. This property specifies how many seconds the records should persist after service execution finishes. 0 (zero) indicates “no persistence” and 0xFFFFFFFF indicates “persist forever”. See 7.6.1.
LogOptions	Optional	This property specifies the types of data that should be logged by the diagnostic service.
OtherLogOptionsDescriptions	Conditional	If LogOptions includes the value 1 (Other), this property is Mandatory.
LogStorage	Optional	This property specifies the logging mechanism to store the diagnostic results. This property must be one of the values in DiagnosticServiceCapabilities.LogStorage
OtherLogStorageDescriptions	Conditional	If LogStorage includes the value 1 (Other), this property is Mandatory.
VerbosityLevel	Optional	This property specifies the desired volume or detail logged by a diagnostic service.

10.10 CIM_DiagnosticSettingDataRecord

CIM_DiagnosticSettingDataRecord stores the settings used in a specific diagnostic service execution. Table 38 provides information about the properties of CIM_DiagnosticSettingDataRecord.

Table 38 – Class: CIM_DiagnosticSettingDataRecord

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]*$")
CreationTimeStamp	Mandatory	None.
ServiceName	Mandatory	This property shall be constructed as follows: <OrgID>:<TestName>. (pattern "^.*[:].*$")
ManagedElementName	Mandatory	This property will be formatted as a free-form string of variable length. (pattern ".*")
RecordType	Mandatory	A RecordType value of 9 (Results) shall be used to log a DiagnosticSettingDataRecord.
ExpirationDate	Mandatory	See 7.6.1.
Settings	Conditional	This property is set to a string that encodes a DiagnosticSettingData instance. If an instance of CIM_DiagnosticSettingData is associated through CIM_ElementSettingData to the instance of CIM_DiagnosticTest at the time the Diagnostic Service is run, this property is Mandatory.

10.11 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 RunDiagnosticService( ) method, a standard mechanism for invoking the test.

Table 39 provides information about the properties of CIM_DiagnosticTest.

Table 39 – Class: CIM_DiagnosticTest

Properties	Requirement	Notes
SystemCreationClassName	Mandatory	**Key**
SystemName	Mandatory	**Key**
CreationClassName	Mandatory	**Key**
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	See 7.1.3.
OtherCharacteristicsDescriptions	Conditional	If Characteristics includes the value of 1 (Other), this property is Mandatory.
TestTypes	Optional	See 7.1.4
OtherTestTypesDescriptions	Optional	See 7.1.5
RunDiagnosticService()	Mandatory	See 8.1.

10.12 CIM_ElementCapabilities

CIM_ElementCapabilities associates a diagnostic service with its capabilities. Table 40 provides information about the properties of CIM_ElementCapabilities.

Table 40 – 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

10.13 CIM_ElementSettingData (JobSettingData)

CIM_ElementSettingData associates the job settings with the job used to run a diagnostic test. Table 41 provides information about the properties of CIM_ElementSettingData.

2535 **Table 41 – 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	If the instance of CIM_JobSettingData is the default setting, this property shall have the value of TRUE. Otherwise, this property shall have the value of FALSE.

2536 ## 10.14 CIM_ElementSettingData (DiagnosticSettingData)

2537 CIM_ElementSettingData associates the diagnostic service with its default. Table 42 provides information
2538 about the properties of CIM_ElementSettingData.

2539 **Table 42 – 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_DiagnosticSettingData. Cardinality 0..1
IsDefault	Mandatory	If the instance of CIM_DiagnosticSettingData is the default setting, this property shall have the value of TRUE. Otherwise, this property shall have the value of FALSE.

2540 ## 10.15 CIM_ElementSoftwareIdentity

2541 CIM_ElementSoftwareIdentity associates the diagnostic service with its version information. Table 43
2542 provides information about the properties of CIM_ElementSoftwareIdentity.

2543 **Table 43 – Class: CIM_ElementSoftwareIdentity**

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

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

10.16 CIM_FilterCollection

CIM_FilterCollection represents a ProfileSpecificFilterCollection as defined in DSP1054. It defines the collection of all the alert indications of the Diagnostics profile. Table 44 contains the requirements for elements of this class.

Table 44 - Class: CIM_FilterCollection

Properties	Requirement	Notes
InstanceID	Mandatory	**Key**: See DSP1054.
CollectionName	Mandatory	The property shall be “DMTF:Diagnostics: ProfileSpecifiedAlertIndicationFilterCollection”.

10.17 CIM_HelpService

CIM_HelpService is the preferred way for a service to publish online help information. Table 45 provides information about the properties of CIM_HelpService.

Table 45 – 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 of 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 of 1 (Other), this property is Mandatory.
GetHelp()	Mandatory	See 8.3.

10.18 CIM_HostedService

CIM_HostedService is used to associate an instance of CIM_DiagnosticTest with an instance of CIM_ComputerSystem to which the CIM_DiagnosticTest is scoped and to associate an instance of CIM_HelpService with an instance of CIM_ComputerSystem to which the CIM_HelpService is scoped. Table 46 provides information about the properties of CIM_HostedService.

Table 46 – Class: CIM_HostedService

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..*

10.19 CIM_IndicationFilter

CIM_IndicationFilter represents a StaticIndicationFilter as defined in DSP1054. It defines the format of all the alert indication filters of the Diagnostics profile. Table 47 contains the requirements for elements of this class.

Table 47 - Class: CIM_IndicationFilter

Properties	Requirement	Notes
Name	Mandatory	**Key**: See the Name values as identified in Table 28.
CreationClassName	Mandatory	**Key**: See DSP1054.
SystemName	Mandatory	**Key**: See DSP1054.
SystemCreationClassName	Mandatory	**Key**: See DSP1054.
SourceNamespaces[ ]	Mandatory	See DSP1054.
IndividualSubscriptionSupported	Mandatory	See DSP1054.
Query	Mandatory	See the Query values as identified in Table 28.
QueryLanguage	Mandatory	See the QueryLanguage values as identified in Table 28.

10.20 CIM_LogManagesRecord

CIM_LogManagesRecord associates a log with its records (service records, setting records, or completion records). Table 48 provides information about the properties of CIM_LogManagesRecord.

Table 48 – Class: CIM_LogManagesRecord

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

10.21 CIM_MemberOfCollection

CIM_MemberOfCollection represents an association between the profile specific FilterCollection and the CIM_IndicationFilters for the alert indications. Table 49 contains the requirements for elements of this class.

Table 49 - Class: CIM_MemberOfCollection

Properties	Requirement	Notes
Collection	Mandatory	**Key**: Value shall reference the profile specific FilterCollection instance representing a filter collection containing the alert indication filters.
Member	Mandatory	**Key**: Value shall reference an Alert IndicationFilter instance representing a contained alert indication filter.

10.22 CIM_OwningCollectionElement

CIM_OwningCollectionElement represents an association between the IndicationService that controls the profile specific FilterCollection and the profile specific CIM_FilterCollection for the alert indication filters. Table 50 contains the requirements for elements of this class.

Table 50 - Class: CIM_OwningCollectionElement

Properties	Requirement	Notes
OwningElement	Mandatory	**Key**: See DSP1054.
OwnedElement	Mandatory	**Key**: Value shall reference the profile specific Alert Indication FilterCollection instance

10.23 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.

10.24 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 DSP1033 Profile Registration Profile. With the exception of the mandatory values specified in Table 52, the behavior of the CIM_RegisteredProfile instance is in accordance with DSP1033 *DSP1033 Profile Registration Profile*.

Table 52 – 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 "2.0.0".
RegisteredOrganization	Mandatory	This property shall have a value of 2 (DMTF).

10.25 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 53 provides information about the properties of CIM_ServiceAffectsElement.

Table 53 – 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.

10.26 CIM_ServiceAvailableToElement

CIM_ServiceAvailableToElement associates the diagnostic service with its help service information. Table 54 provides information about the properties of CIM_ServiceAvailableToElement.

Table 54 – 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

10.27 CIM_ServiceComponent

CIM_ServiceComponent associates a test that is also part of another test. This class is used when DiagnosticTest.Characteristics includes the value 6 (Is Package) and subtests are implemented as separate instances of DiagnosticTest. Table 55 provides information about the properties of CIM_ServiceComponent.

Table 55 – 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.

10.28 CIM_SoftwareIdentity

CIM_SoftwareIdentity is used to publish version information about the diagnostic service. Table 56 provides information about the properties of CIM_SoftwareIdentity.

Table 56 – Class: CIM_SoftwareIdentity

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 "^.*[:].*$")
MajorVersion	Mandatory	None.
MinorVersion	Mandatory	None.
RevisionNumber	Mandatory	None.
VersionString	Mandatory	None.
Manufacturer	Mandatory	This property will be formatted as a free-form string of variable length. (pattern ".*")

10.29 CIM_UseOfLog

CIM_UseOfLog associates a log with a managed element (a device or diagnostic service) whose information is stored in the log. Table 57 provides information about the properties of CIM_UseOfLog.

Table 57 – Class: CIM_UseOfLog

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

ANNEX A
(informative)

Change log

Version	Date	Description
1.0.0a	2006-04-17	Preliminary
1.0.1	2009-09-23	Final Standard
2.0.0	2010-08-13	DMTF Draft Standard
2.1.0	2013-05-09	Changed the version Changed the Date Changed the Document Status Edited the Normative References Added Diagnostic Job Control and Indications to the Related Profile table of the Synopsis Clause 7 changes - Added a clause "7.1.4 CIM_DiagnosticTest.TestType" to define TestType - Greatly expanded clause “7.3 CIM_DiagnosticServiceCapabilities” - Greatly expanded clause 7.4 “CIM_DiagnosticServiceSettingData” - Deleted clause "7.5 CIM_ConcreteJob" (it‘s moved to DSP1119) - Added a clause "7.8 Diagnostics Profile Indications Support" - Added a clause "7.9 Diagnostics Alert Indications and Standard Messages" Clause 8 Methods - Removed references to CreateInstance, since it left too many questions unanswered Clause 9 Use Cases - Minor editing of the use cases Clause 10 CIM Elements changes - Deleted the classes moved to Diagnostic Job Control (CIM_AffectedJobElement, CIM_ConcreteJob, CIM_JobSettingData and CIM_OwningJobElement) - Added the entries for the Alert Indications
2.1.0a	2013-06-13	Released as Work in Progress

Bibliography

DMTF DSP2000, *CIM Diagnostic Model White Paper 1.0*,
http://www.dmtf.org/standards/published_documents/DSP2000.pdf