System Diagnostic Model White Paper

DSP0138 ***Status: Preliminary***

A
Diagnostic Model in CIM

January 6th, 2000

Abstract

Diagnostics under a multitasking operating system, like Windows NT®, are inherently complex. Items like Hardware Abstraction Layers and accelerated hardware with advanced device drivers place an ever increasing amount of software between the applications and the hardware, there by making diagnostics less effective. Kernel features that allow "hot swapping" of devices, (including processors, and memory) can be extended to allow "Suspend / Resume" features for diagnostics. Under the realm of concurrent diagnostics, idle devices, e.g. processors, network cards, memory, ... could be "suspended" (taken off line) and testing performed, if the resource were needed or the testing complete, the device is "resumed" (brought back on line).

Change History

Version	Date	Author	Comments
0.001	01/27/99	RDW/IBM	First Draft
0.002	02/08/99	RDW/IBM	Added CIM detail
0.003	03/22/99	RDW/IBM	Added detail about modeling coverage, and intro/vision
0.004	04/15/99	RDW/IBM	Word smithing
0.005	04/22/99	RDW/IBM	Changing CIM_DevDiag to subclass from CIM_Dependency
0.006	04/29/99	RDW/IBM	Add subclass example, change name to CIM_Diag interface
0.007	05/12/99	RDW/IBM	Changed CIM_Diag to subclass logical system element
0.008	06/20/99	RDW/IBM	Changed CIM_Diag to subclass CIM_Service, added information about settings. Moved many properties and qualifiers to the new Settings class.
0.009	07/05/99	RDW/IBM	Deleted settings class replaced with a results class that stores settings information along with results
1.0	07/19/99	RDW/IBM	First draft to the DMTF, matches Visio diagram, and MOF
1.001	07/28/99	DM/IBM	Corrected typos
1.002	08/08/99	RDW/IBM	Revised document with DMTF comments: Package class replaced with DiagnosticTestInPackage aggregation. Separated Settings, and results. Several new associations added.
1.003	08/29/98	RDW/IBM	Revised document with DMTF comments: DiagnosticResult class uses an embedded DiagnosticSetting object, rather than copying all the properties of DiagnosticSetting. DiagnosticResults no longer subclasses off of LogicalElement.
1.004	09/09/99	RDW/IBM	Corrections and clarifications based on feed back from DMTF
1.005	09/14/99	RDW/IBM	All valuemaps are now uint16, Results: TestStartTime made key; TestState no key; TimeStamp not key, DiagnosticTest: ClearResults now takes a ref to a ManagedSystemElement; DiscontinueTest does not output a result object. Added some usage examples for setting for test associations.
1.006	9/16/99	RDW/IBM	Changed cardinality on diagnosticResultForTest to 0..1 from 1..1. Renamed stuff to Is, e.g. IsPackage, IsInUse,… Changed method returns to uint32
1.007	9/25/99	RDW/IBM	Typos corrected, new ones created.
1.008	10/05/99	RDW/IBM WHK/IBM	Changes to make the Mof CIM Compliant. Added an association to SoftwareElement
1.009	10/18/19	WHK/IBM	Remove the imbedded settings object in Result and add a copy of the setting properties into Result
1.009a	10/20/99	RDW/IBM JSB/IBM	Cleaned up association inheritance differences with the MOF
1.010	11/18/99	RDW/IBM	Removed the copy of settings from the result object. In theory the settings in results problem will be address in CIM 2.4
1.011	11/29/99	RDW/IBM	Change the wording on PercentComplete in diagnosticResult
1.012	01/06/00	RDW/IBM	Final cleanup of typo's for DMTF publishing

Editor

Rod Waltermann, et al.

A joint effort of IBM, Intel, Watergate Software, And Dell.

The editor would like to thank the following contributors and reviewers (in alphabetical order): Dell: Sidney Smith; IBM: Alfred Burress, Bill Klump, Ray Pedersen, Ken Young; Intel: Russ Carr, Eric Grimm, Arvind Kumar, Reza Nassib, Ioan Scumpu; Watergate Software: R. Marc Browning, Aki Korhonen, Mark Proudfoot

For further information contact: DMTF diagnostic sub group: tm-diag@dmtf.org

Table of Contents

Introduction

Diagnostics under a multitasking operating system, like Windows NT®, is inherently complex. Items like Hardware Abstraction Layers and accelerated hardware with advanced device drivers place an ever increasing amount of software between the applications and the hardware, there by making diagnostics less effective. Kernel features that allow "hot swapping" of devices, (including processors, and memory) can be extended to allow "Suspend / Resume" features for diagnostics. Under the realm of concurrent diagnostics, idle devices, e.g. processors, network cards, memory, ... could be "suspended" (taken off line) and testing performed, if the resource were needed or the testing complete, the device is "resumed" (brought back on line).

For example: the customer is installing a new card to allow connection to a digital camera. When they install the device driver for the camera, a diagnostic for the hardware is installed along with the driver, this diagnostic is refered to as a diagnostic data provider. This diagnostic provider will register its set of diagnostic methods and data, called tests, into CIM (Common Information Model). These tests describe what methods and features of the standard are supported. When the customer runs their diagnostic program it will query CIM for a list of registered devices that have tests. The program will then build a menu of tests for the installed devices. Even though the diagnostic program was purchased when the computer was new and never updated, the new digital camera option shows up in the menu.

As we strive to move into the realm of concurrent / predictive diagnostics the primary requirements are:

- Diagnostics from multiple sources, that interface through a common framework.
- The ability to place idle resources in a "diagnostic mode" for diagnostic testing.
- Diagnostics can be remotely administered through CIM, or an enterprise management system.
- A diagnostic consumer can access the cost of performing a given diagnostic, based on system load, or resources used during the diagnosing.
- Abnormal device conditions can trigger diagnostic events

During the development of this model we focused on three main users of diagnostics: the customer, service personnel, and manufacturing. All three have somewhat different requirements

The customer would like to have a high level view of the system, i.e. they would like a pass/fail information about the device. Additionally we would like to link with other data in CIM for example inventory and FRU (Field replaceable unit) data, and predictive failure data for hard disks, memory, etc. This combined with diagnostic logs stored on the machine, allow an IT (Information Technology) manager to gain a system health view over their machines. Another requirement is the ability to access and test the machines remotely. CIM also allows the customer to log the data for trend analysis, or to make it available to the service personnel.

Service needs to be able to access the machine or the test results remotely, allowing them to validate what parts to ship/bring along to complete the repair. The servicer needs to be able to recreate the error, this implies viewing the settings for the diagnostic at the time the error was logged.

The manufacturing environment needs are somewhat different. During burn-in, a period of extensive testing of a randomly selected machine for quality control, the OS and all components are installed. This is a point where this model seems to fit; however during other phases of the assembly, diagnostic requirements are very different. Having the same test modules work in a manufacturing only environment and a common diagnostics model is desirable, both for repeatability, and to help reduce cost of development.

Along with the three users, we tried to stay with a model that focused on modeling the results of the diagnostic being performed on the device (a customer centered model), instead of modeling the methods/tests.

Instrumented Diagnostics through CIM

A diagnostic is broken into two parts. A diagnostic provider, and a diagnostic consumer. The diagnostic consumer is the test module that will request a test, or call the methods in a diagnostic provider, to perform a single test or a set of tests on the device. The diagnostic provider registers itself through CIM to export the events and methods

supported. When a diagnostic test module queries CIM for diagnostics supported on a given device, CIM will report about the instance of the diagnostic provider. This effectively establishes communication between the provider and the consumer. The consumer can now enable events, or execute methods registered by the provider.

Vision

Modeling diagnostics further enhances the remote manageability of a machine. A diagnostic is composed of a collection of methods and their resulting pass/fail and logged data. This document further breaks the model down into classes, and associations. The classes being for the diagnostic, its' methods, and results from their execution. The associations relate the tests to the devices they support.

The diagnostic test class models a single test, e.g. a linear scan disk test. A test has properties that describe its function. Properties that show if the test is currently in use, will destroy data (a hard drive test that writes to the disk could overwrite data), and so on. A test is designed with a main entry point to run the test. Along with running the test, there is a method to stop its execution.

The test needs to be associated with the hardware it can diagnose. As an example you have a machine with two Ethernet cards (Network Interface Cards NICs)) from the same manufacturer. You only need to have one diagnostic exposed through CIM, one diagnostic, with two associations, one for each NIC.

Now we have tests, and devices they can act upon. It is expected that there will be more than one test for a given device. A way to collect the tests that share a common device is needed. A diagnostic package describes a collection of tests. This collection will perform basically the same way that an individual test would perform, i.e. you can start testing, stop testing, check if testing is in progress, etc.

The setting and result classes store all the information to rerun a test on the same device. For example you have a machine with a hard drive. There are two tests for the hard drive and one package containing both tests. Given this example, we call one of the tests. This test, with its setting object, will create a result object as it's output.

If instead we call the package, there will still be only one setting object; however each test will create it's own result object that is associated to the package's result object. In effect creating a package of results from all the tests. The way to associate these results back to the device is defined in a manner similar to the way a test is associated to the device.

Road map

The first release of this specification covers on-demand diagnostics. Diagnostics will run when invoked by a method call.

Release two is scheduled to address many new areas including: real time (concurrent) and event driven diagnostics, tests that need to interact with someone, e.g. a floppy disk test needs someone to insert the disk, data logging, and parameter validation. Other topics in no particular order include: watch-dog timers, synchronous method execution, diagnostic mode, and pre/post diagnostic processing.

The third version expects to cover self-diagnostics. Utilizing the base built in the first and second versions, a AI (Artificial Intelligence) based data consumer could use the diagnostic results along with other CIM based data to provide a "self-healing" function. A hypothetical example is a server with multiple network cards. One of the NICs is causing problems on the machine. The fault detection built into the card throws an event that a diagnostic manager will use to: launch the concurrent diagnostics built into the device driver, trigger an event which would send requests to have the offending card powered down, and the network traffic rerouted. The system admin would be informed, and an entry in the system log stating the problem and time are recorded.

CIM Schema for diagnostics

Diagnostic class registration

Diagnostic providers need to subclass from three classes: CIM_DiagnosticTest, CIM_DiagnosticSetting, and CIM_DiagnosticResult. The provider will also need to subclass from seven associations: CIM_DiagnosticTestForMSE, CIM_DiagnosticSettingForTest, CIM_DiagnosticTestInPackage, CIM_DiagnosticResultInPackage, CIM_DiagnosticResultForTest , CIM_DiagnosticResultForMSE, and CIM_DiagnosticTestDependency.

CIM_DiagnosticTest : CIM_Service

The actual diagnostic test instances and subclasses are here. It is expected that there will be a subclass for every manufacturer of diagnostics and one or more instances of diagnostic tests.

CIM_DiagnosticSetting : CIM_Setting

This class is used to store the default or run-specific settings for a given test. This class works in conjunction with the result class, see the example in CIM_DiagnosticResult.

CIM_DiagnosticResult

For every RunTest method from a diagnostic instance of CIM_DiagnosticTest executed against an instance of managed system element, there will be an instance of a subclass of CIM_DiagnosticResult. Along with the results there will be an association instance of a subclass in CIM_DiagnosticResultsForMSE. For example, we have a subclass of CIM_DiagnosticTest titled vendorX_ScanDiskDiag, which has one instance called vendorX_ScanDiskDiag_1. Now we have a subclass of CIM_DiagnosticTestForMSE titled vendorX_ScanDiskDiagAssoc. There is one instance of this association for the hard drive, vendorX_HardDiskAssoc_1. Finally a subclass of CIM_DiagnosticResult titled vendorX_ScanDisk_1DiagResult is created to store all the data for each running of the test.

When the RunTest method is invoked it will require one of the associations, and optionally an instance of vendorX_ScanDisk_1DiagSetting. If the consumer wants to change any of the default settings, it will create its' own instance of vendorX_ScanDisk_1DiagSetting, adjust the test settings, and pass this instance to the RunTest method when it is invoked. If an instance of vendorX_ScanDisk_1DiagSetting is not provided the default setting object will be used. When the testing begins an instance of vendorX_ScanDisk_1DiagResult is created which stores a copy of the settings and all the resulting test information.

CIM_DiagnosticTestForMSE : CIM_ProvidesServiceToElement

This is an association class to relate a device under CIM_ManagedSystemElement with the diagnostic test instance provider in CIM_DiagnosticTest : CIM_Service. If there is no device registered for which a diagnostic test can service, that diagnostic will not register an association. Diagnostic consumers will query this association for instances of devices with a diagnostic test provider.

CIM_DiagnosticTestSoftware : CIM_Dependency

This is an association class to relate a diagnostic test under CIM_DiagnosticTest : CIM_Service with the SoftwareElement that provides this test. SoftwareElement describes information about the test, for example: vendor, version and other deployment information.

CIM_DiagnosticTestInPackage : CIM_Component

This is an association class to relate a diagnostic test under CIM_DiagnosticTest with the diagnostic test package instance also under CIM_DiagnosticTest. Diagnostic consumers are expected to query this association in order to discover which tests are in a given package.

CIM_DiagnosticResultInPackage

This is an association class to relate a diagnostic test's results under CIM_DiagnosticResult : CIM_LogicalElement with the diagnostic result package instance also under CIM_DiagnosticResult. Diagnostic consumers are expected to query this association in order to dis cover which results are in a given package.

CIM_DiagnosticResultForMSE

This is an association class to relate a device under CIM_ManagedSystemElement with the diagnostic result instance provider in CIM_DiagnosticResult. If there are no results, do not re gister an association. Results for a given execution of a diagnostic are stored in the diagnostic result class. Diagnostic consumers will query this association for instances of diagnostic result for devices with a diagnostic provider.

CIM_DiagnosticResultForTest

This is an association class to relate a diagnostic result under CIM_DiagnosticResult with the diagnostic test instance under CIM_DiagnosticTest. Diagnostic consumers are expected to query this association in order to discover which tests have results from previous runs.

CIM_ServiceServiceDependency

This is an association class used to order execution of tests with services or other tests that may be running on the device. For example: A network interface card ping-test requires that the TCP/IP service be running on the card before it can execute the test. Diagnostic consumers are expected to query this association to discover which tests have dependencies on other services or tests.

CIM_DiagnosticSettingForTest : CIM_ElementSetting

This is an association class to relate a diagnostic test's settings under CIM_DiagnosticSetting : CIM_Setting with the diagnostic test instance under CIM_DiagnosticTest. Diagnostic consumers are expected to query this association in order to discover settings for a given test.

The data model, a picture:

Figure 1, below, shows the relationship between the classes and figure 2 shows the associations.

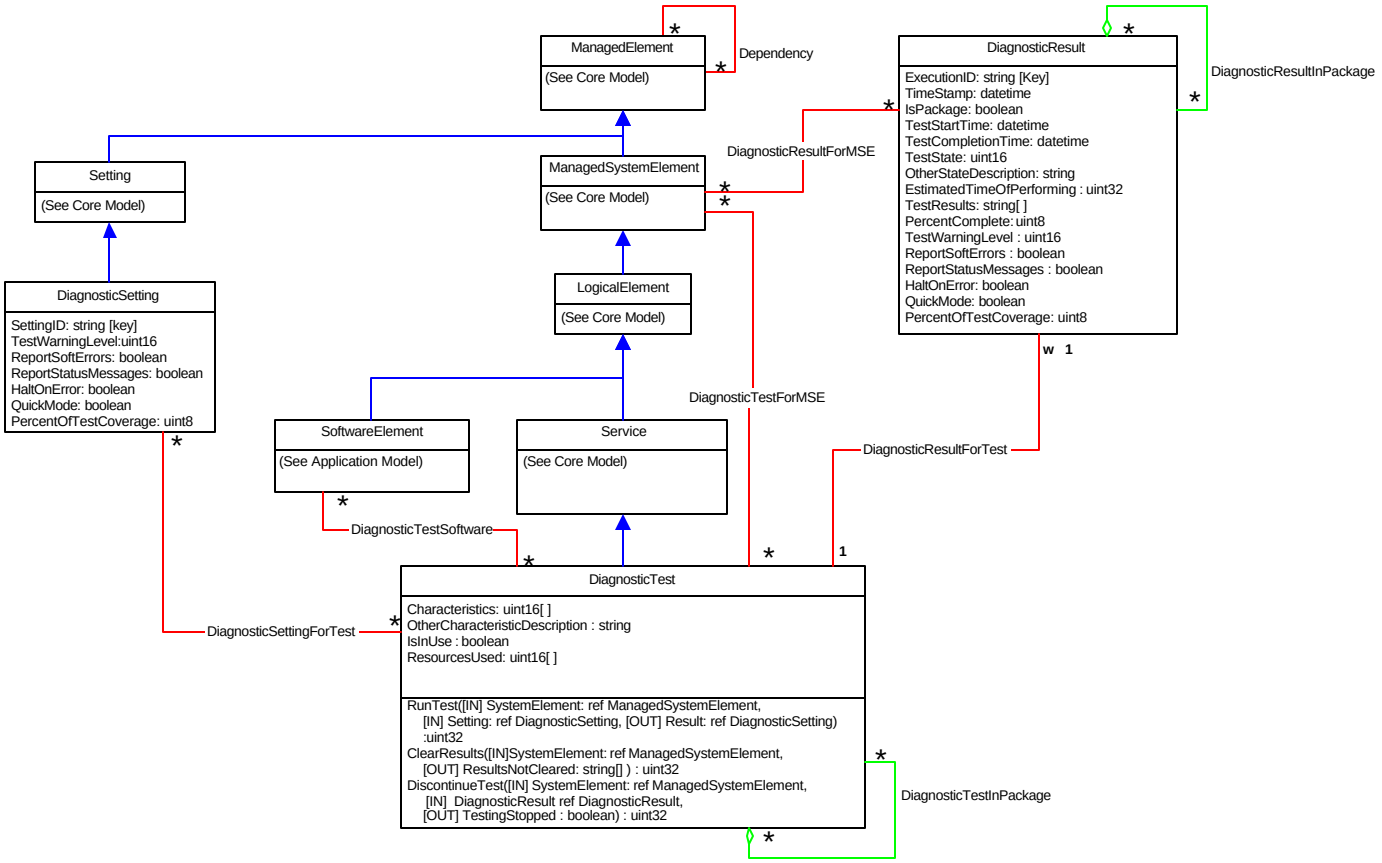

Figure 1: UML diagnostic Classes

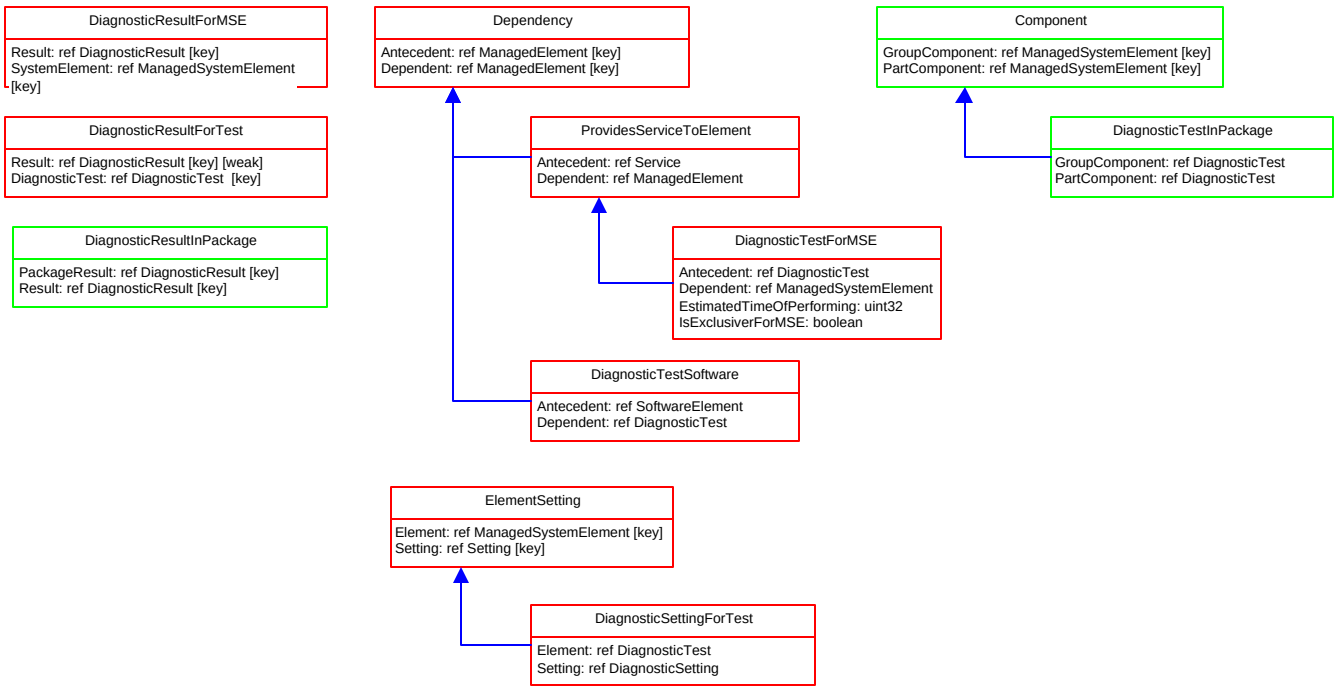

Figure 2: UML Diagnostic Associations

Class definitions

CIM_DiagnosticTest : CIM_Service

The CIM_DiagnosticTest class represents the ability to execute a test. Specific diagnostic tests may be defined by subclassing and/or instantiating this object. To provide more detail for a type of test (additional properties and methods), subclassing is appropriate. To indicate that a test exists and may be applied to a specific ManagedSystemElement, instantiation of the DiagnosticTest class may suffice.

Properties

- Characteristics[]
  - Uint16 array
  - Values supported
    - Unknown
    - Other
      - When other is defined, additional detail may be found in the OtherCharacteristicDescription property of this class.
    - IsExclusive
      - If exclusive is specified for the test module only if the diagnostic cannot run more than one test at a time, regardless of how many SystemElements are supported. Typically, this occurs when hardware or software constraints prevent the test from running as multiple, concurrent instances. If the diagnostic can be run against multiple SystemElements, but only once per Element, then set the IsExclusiveForMSE boolean property on the appropriate instances of DiagnosticTestForMSE.
    - IsInteractive
      - IsInteractive is set if the diagnostic displays a message either before, during or after the testing.
    - IsDestructive
      - IsDestructive is set if the diagnostic will destroy data or reconfigure the Element that is being tested.
    - IsRisky
      - Is Risky indicates that data loss may occur if the test is interrupted. Some tests make copies of the data, perform the test, and restore the data returning the tested entity to its previous configuration. If the test is interrupted, then loss of data or reconfiguration of the tested ManagedSystenElement may occur.
    - IsPackage
      - If IsPackage is set, this test is actually a set of lower level diagnostics, that are packaged together .
    - SupportsPrecentOfCoverage
      - SupportsPrecentOfCoverage indicates that a request for reduced coverage can be specified using the PrecentOfCoverage property of this class.

- OtherCharacteristicDescription;
  - String
  - Provides additional information for the Characteristic when its value is set to "Other".

- IsInuse
  - Boolean
  - If this test is being performed, the InUse property is set to TRUE. To determine which ManagedSystemElement is being tested, query the DiagnosticResults objects associated with this test (query DiagnosticResultForTest), and for which the TrestState equals InProgress, The DiagnosticResult object is associated with the SystemElement under test, using DiagnosticResultForMSE..

- ResourcesUsed []
  - Uint16 Array, type “bag”, read only

- The Expensive qualifier can be applied to this class and its RunTest method. If so, the ResourcesUsed property describes the resources that are capitalized, based on a default setup for the test. Multiple resources can be specified since this is an array.
- Currently recognized values are:
  - CPU
  - Memory
  - Hard Disk, CDROM, Floppy
  - PCI Bus, USB Bus , IEEE 1394 Bus, SCSI Bus, IDE Bus
  - ISA Bus, EISA Bus, VESA Bus, PCMCIA Bus, CardBus
  - Access.bus, NuBus, AGP, VME Bus
  - Sbus IEEE 1396-1993, MCA Bus, GIO Bus, XIO Bus, HIO Bus, PMC Bus, SIO Bus
  - Network
- This property uses value mapping with the above keywords

Methods

Diagnostic providers should register for the following methods. Access these methods through the functionality built-into the CIM Object manager.

- Uint32 RunTest

The RunTest method executes this DiagnosticTest for the specified ManagedSystemElement (defined using the SystemElement input parameter). Results of the test are stored in a CIM_DiagnosticResult object, a reference to which is returned as the Result output parameter. How the test should execute, i.e. its settings, is defined in a CIM_DiagnosticSetting object (or by a subclass of DiagnosticSetting). A reference to a Setting object is specified using the "Setting" input parameter. If a reference is not passed into the method, then a default DiagnosticSetting may be specified and used. This default Setting is associated with the DiagnoticTest using the DefaultSetting relationship of the Core Model.

- [IN] Ref CIM_ManagedSystemElement
  - The Path to the logical element you want to test.
- [IN] Ref CIM_DiagnosticSetting
  - Default for this parameter is “”. If the consumer wants to change the default settings, it should instantiate an instance of CIM_DiagnosticSetting or the correct subclass, make any changes needed, then pass the reference in this parameter.
- [OUT] String Ref CIM_DiagnosticResult
  - The instance that contains the settings used and the results of this run
- Method Returns: uint32 and supports the following values:
  These return codes are expected to adhere to the XML return codes as they are introduced.
  - 0 = OK (function succeeded. Note: the hardware may have failed the test even though the function succeeded.)
  - 1 = UnspecifiedError (unexpected failure, reason unknown/unspecified)
  - 2 = NotImplemented (member function is no implemented)
  - 3 = OutOfResources (component could not allocate required resources)

- Uint32 ClearResults
  Execution of this method will delete all instances of the CIM_DiagnosticResultForMSE object, for this DiagnosticTest and the specified ManagedSystemElement (defined using the SystemElement input parameter). Also all of the association refereeing to the result object will be deleted. One output parameter is defined - ResultsNotCleared - which is a string array that lists the keys of the DiagnosticResults which could not be deleted. This information enables those Results to be revisited and either manually removed, or other corrective action taken.
  - [IN] Ref CIM_ManagedSystemElement
    A reference to the managed system element that you want the test results cleared.
  - [OUT] String ResultsNotCleared []
    - Any instances of the correct subclass of CIM_DiagnosticResult that were not deleted, will have their name stored in this array.

- An empty array implies that all of the results were deleted.
- Method Returns: uint32 and supports the following values:
  These return codes are expected to adhere to the XML return codes as they are introduced.
  - 0 = OK (function succeeded. Note: the hardware may have failed the test even though the function succeeded.)
  - 1 = UnspecifiedError (unexpected failure, reason unknown/unspecified)
  - 2 = NotImplemented (member function is no implemented)
  - 3 = OutOfResources (component could not allocate required resources)

- Uint32 DiscontinueTest
  After invocation of this method and its completion, the specified test(s) will be discontinued for the indicated ManagedSystemElement (defined by the SystemElement input parameter). The test to discontinue is specified using the Result input parameter. If all instances of this test should be stopped for the SystemElement, then the Result reference should be NULL. Upon completion of the method, test status and other information (such as PercentComplete) will be stored in the DiagnosticResult instance defined by the Result input parameter. The output parameter, TestingStopped, is used as follows:
  - Set to TRUE if testing was successfully stopped.
  - Set to FALSE if the current test(s) can not be stopped.
    - If set to FALSE, testing will stop when the diagnostic is able to do so safely. To determine if/when the testing is stopped, check the TestState property in the DiagnosticResult instance defined by the Result parameter. TestState will change from In Progress to Stopped.
  - [IN] Ref CIM_ManagedSystemElement
    - The reference to the logical element on which you want testing to stop.
  - [IN] Ref CIM_DiagnosticResult
    - The instance that contains the settings used and the results of this run
  - [OUT] Boolean TestingStopped
    - TRUE if all testing was successfully stopped.
    - FALSE if the current test cannot be stopped. Testing will stop when the diagnostic is able to do so safely.
  - Method Returns: uint32 and supports the following values:
    These return codes are expected to adhere to the XML return codes as they are introduced.
    - 0 = OK (function succeeded. Note: the hardware may have failed the test even though the function succeeded.)
    - 1 = UnspecifiedError (unexpected failure, reason unknown/unspecified)
    - 2 = NotImplemented (member function is no implemented)
    - 3 = OutOfResources (component could not allocate required resources)

- Any additional methods added should support the following parameters.
  - [IN] Ref CIM_ManagedSystemElement
    - A reference to the logical element you want to test.
  - [IN] Ref CIM_DiagnosticSetting
    - Default for this parameter is "". If the consumer wants to change the default settings, it should instantiate an instance of CIM_DiagnosticSetting or the correct subclass, make any changes needed, then pass the reference in this parameter.
  - [OUT] String Ref CIM_DiagnosticResult
    - subclass instance that contains the settings used and the results of this run
  - Method Returns: uint32 and supports the following values:
    - OK (function succeeded. Note: the hardware may have failed the test even though the function succeeded.)
    - UnspecifiedError (unexpected failure, reason unknown/unspecified)
    - NotImplemented (member function is no implemented)
    - OutOfResources (component could not allocate required resources)

CIM_DiagnosticSetting : CIM_Setting

Specific diagnostic test parameters and execution instructions are defined by subclassing and/or instantiating the DiagnosticSetting object. To provide more detailed Settings for a type of test (i.e., additional properties), subclassing is appropriate. When only the generic Setting information is required, instantiation of the Diagnostic Setting class may suffice. The following properties are copied to the diagnostics results class. Any new properties added to diagnostic setting must be replicated in diagnostic results.

- SettingID
  - String
  - Key property
  - The identifier by which the DiagnosticSetting object is known and uniquely named. If there is a one to one mapping between the DiagnosticSetting and its DiagnosticTest (and there will always be only a one to one mapping), then the DiagnosticCreationClassName and DiagnosticName key pair can be used as the SettingID..

- TestWarningLevel
  - Uint16
  - Sets the level of warning messages to be logged. If for example no warning information is required, the level would be set to No Warnings. Using MissingResources will cause warnings to be generated when required resources or hardware are not found. Setting the value to Testing Impacts, results in both missing resources and 'test impact' warnings (for example, multiple retries required) to be reported.
  - This property uses value mapping, and values to define the following keywords
    - No Warnings
    - Missing Resources
    - Testing Impacts
    - All Warnings

- ReportSoftErrors
  - Boolean
  - A soft error, in this context, is a message from the diagnostic software that relates to a known defect in the hardware or driver and has a valid workaround. For example:
    - Not enough memory
    - Driver IOCTL not implemented
    - Video ram compare failed during polygon fill test (A known defect in the video chipset).

- ReportStatusMessges
  - Boolean
  - When this flag is true, the diagnostic test will report status information. Examples are:
    - Diagnostic state information
    - Debug or trap information

- HaltOnError
  - Boolean
  - When this flag is TRUE, the test will halt after finding the first error.

- QuickMode
  - Boolean
  - When this flag is true, the test software should attempt to run in an accelerated fashion either by reducing the coverage or number of tests performed.

- PercentOfTestCoverage
  - uint8

- Requests the diagnostic software to reduce test coverage to the specified percentage. For example, a hard drive scan test could be asked to run at 50%. The most effective way to accomplish this is for the test software to scan every other track, as opposed to only scanning the first half of a drive. It is assumed that the effectiveness of the test is impacted proportional to the percentage of testing requested. Permissible values for this property range from 0 to 100.

  This property may not be applicable to all tests. If it can be set for a test, the value Supports PercentOf TestCoverage should be entered into the Diagnostic Test's Characteristics array.

CIM_DiagnosticResult

When a DiagnosticTest Service is running, test results are reported using a DiagnosticResult object, or one of its subclasses. A DiagnosticTest may be running because its Service is Started or due to an invocation of the RunTest method. DiagnosticResults are related to their Test via an instance of the DiagnosticResultsForMSE association.

An instance of the settings could be changed without the DiagnosticResults indicating that data has been changed. Therefore the DiagnosticSetting used is copied in the DiagnosticResults.

Properties

- DiagnosticName
  - String
  - Key property
  - Propagated from CIM_DiagnosticTest
  - The scoping test's name. Must be a unique name
    - Use: companyName_TestName_N
    - N is a counter for each instance of the diagnostic result method provider.

- DiagnosticCreationClassName
  - String
  - Key property
  - The scoping test's CreationClassName

- ExecutionID
  - String
  - Key property
  - The Unique identifier for an instance of DiagnosticResults.

- TimeStamp
  - DateTime, read only
  - The date and time the result was last updated.

- IsPackage
  - Boolean, read only
  - If this property is TRUE, then this DiagnosticResult summarizes the results from the execution of a packaged set of DiagnosticTests. The Tests in the package can be identified by following the DiagnosticResultForTest association to the test and then using the DiagnosticTestInPackage aggregation. The individual Results can be broken out by instantiating DiagnosticResults for the individual lower level tests and aggregating into the 'summary' Result using the DiagnosticResultInPackage ..

- TestStartTime
  - DateTime, read only
  - The time and date when this test started.

- TestCompletionTime
  - DateTime, read only
  - The time and date when this test completed.

- TestState
  - Uint16
  - Describes how the test is progressing. For example, if the test was discontinued, the TestState will be Stopped, or if testing is currently executing, TestState will be In Progress.
  - This property uses value mapping, and values to define the following keywords
    - Unknown
    - Other
    - Passed
    - Failed
    - InProgress
    - Stopped

- OtherStateDescription
  - String
  - When Other is entered in the TestState property, OtherStateDescription can be used to describe the test's state.

- TestResults []
  - String Array, type “Ordered”, read only.
  - One entry is considered a cell location in the array.
  - TestResults stores one or more textual results from the execution of the DiagnosticTest(s) referenced by the DiagnosticCreationClassName and DiagnosticName properties. One entry is considered a cell location in the array. Each entry is time stamped and contains the following information, in the following format: yyymmddhhttssoutc|DiagnosticName|Textual message Where:
    - yyyy = year, e.g. 2000
    - mm = month (01 – 12)
    - dd = day (01 – 31)
    - hh = hour (00 – 24)
    - tt = minute (00-59)
    - ss = second (00-59)
    - o = "+" or "-", indicating the sign of the UTC (Universal Coordinated Time; for all intents and purposes the same as Greenwich Mean Time) correction field.
    - utc is the offset from UTC in minutes (using the sign indicated by *o*).
    - A '|' followed by the name of either the diagnostic package, or the diagnostic test that is making the entry.
    - Every thing following the second '|' is the textual message.

- PercentComplete

The percentage of the test that has executed thus far, if the TestState property is set to \"In Progress\" or the percentage of the complete test that was executed if the TestState property is set to any of the completed states (\"Passed\", \"Failed\" or \"Stopped\"). Final results may be based on less than 100% coverage due to the parameters defined in DiagnosticSetting (such as QuickMode, PercentOfTestCoverage or HaltOnError).

  - uint8, read only
  - This value varies from 0 to 100 during execution of the DiagnosticTest. If the value is 0, the test has not started. If the value is 100, then testing is complete.
  - In the case of a package, this is the sum of the percentage of contribution for each method

- EstimatedTimeOfPerforming
  - uint32
  - Estimated number of seconds to perform the Diagnostic Test indicated by the DiagnosticCreationClassName and DiagnosticName properties. After the test has completed, the actual elapsed time can be determined by subtracting the TestStartTime from the TestCompletionTime. A similar property is defined in the association, DiagnosticTestForMSE. The difference between the two properties is that the value stored in the association is a generic test execution time for the Element and the Test. But, the value here (in DiagnosticResult) is the estimated time that this instance of testing would run.

  - The following properties are duplicated from DiagnosticSettings
    - TestWarningLevel
    - ReportSoftErrors
    - ReportStatusMessages
    - HaltOnError
    - QuickMode
    - PercentOfTestCoverage

CIM_DiagnosticTestForMSE

This is an association class that relates a DiagnosticTest to a ManagedSystemElement. Consumers wishing to 'diagnose' a particular Element could query this association, for the Element, to determine what tests are available.

Properties

- Antecedent
  - The test that may be run against a ManagedSystemElement
  - Must be a unique name.
  - Path to the diagnostic.

- Dependent
  - The MSE(ManagedSystemElement) that can be tested.
  - Must be a unique name.
  - Path to the managed system element

- EstimatedTimeOfPerforming
  Estimated number of seconds to perform the referenced DiagnosticTest against the ManagedSystemElement. Since execution times could vary by Element, this property is located in the association between the two entities. It is also captured in DiagnosticResult, in the Estimated TimeOfPerforming property.
  - uint32
  - Estimated number of seconds to perform. To find out the real elapsed time Subtract TestStartTime from TestCompletionTime in the CIM_DiagnosticResult object.

- IsExclusioveForMSE
  - Boolean
  - If the DiagnosticTest referenced in this object can be run concurrently against multiple SystemElements, but only run one at a time for the referenced ManagedSystemElement, then this boolean is set to TRUE. Alternately, if the test can NOT be run concurrently irregardless of the SystemElements being tested, then the more general IsExclusive enumerated value should be set in DiagnosticTest.Characteristics.

CIM_DiagnosticResultsForMSE

This is an association class relating diagnostic test results to the ManagedSystemElement that is/was tested..

Properties

- Result
  - This is the diagnostic result.
  - Key property.
  - Reference to the diagnostic result.

- SystemElement
  - The MSE (ManagedSystemElement) to which the results applies.
  - Key property.

CIM_DiagnosticResultsForTest

This is an association class to relate the results of a test to the test itself.

Properties

- DiagnosticResult
  - This is the diagnostic result.
  - Key property.
  - Reference to the diagnostic result.

- DiagnosticTest
  - The tests that generated the result.
  - Key property.

CIM_DiagnosticTestInPackage : CIM_Componet

This is an association class that identifies a Diagnostic Test as made up of lower level Tests. In this case, the Test identified as the GroupComponent reference (i.e, the higher level test) would have the Is Package enumerated value specified in DiagnosticTest.

Properties

- GroupComponet
  - The DiagnosticTest object that acts as the container for all the tests in the package.
  - Reference to CIM_DiagnosticTest.
- PartComponet
  - The DiagnosticTest object that is one of the elements of the package.
  - Reference to CIM_DiagnosticTest.

CIM_DiagnosticResultInPackage

This is an association class that identifies a Diagnostic Result as made up of lower level Results. In this case, the Result identified as the PackageResult reference (i.e., the higher level result) would have its IsPackage property set to TRUE.

Properties

- PackageResult
  - The DiagnosticResult object that acts as the container for all the results of the package.
  - Reference to CIM_DiagnosticResult
  - Key property.
- Result
  - The DiagnosticResult object that is one of the elements of the package.
  - Reference to CIM_DiagnosticResult.
  - Key peoperty.

CIM_DiagnosticTestSoftware : CIM_Dependency

This is an association class relating DiagnosticTest to the SoftwareElements that provide this test. SoftwareElement describes vendor/version information and other deployment data..

Properties

- Antecedent
  - Vendor/Version and other information about the software the runs as the DiagnosticTest.
  - Reference to CIM_SoftwareElement.
- Dependent
  - The DiagnosticTest whose software is described.
  - Reference to CIM_DiagnosticTest.

CIM_DiagnosticSettingForTest : CIM_ElementSetting

This is an association class to relate test settings with diagnostic tests.

Properties

- Element
  - The diagnostic test that can use the Ssetting object.
  - Reference to CIM_DiagnosticTest.
- Setting
  - The Setting the can be applied to the execution fo the DiagnosticTest.
  - Reference to CIM_DiagnosticSetting

Other features of CIM to be used

CIM Events

Events under diagnostics are used for the following.
Since all the events are asynchronous and time of delivery is not guaranteed, some care must be taken to ensure that enough data is reported/logged with the event to resolve the condition being diagnosed.

- Message passing, for test messages that the operator needs to see.
- Abnormal condition reporting.
- Checksum errors in a data stream
- Parity errors in local cache.
- Buffer overrun flags (software diagnostics).
- Data logging
- Some event-logging model is needed to support postmortem operations, and interactive testing.

CIM Collection classes

Collection classes allow an association to be made to groups of CIM objects. For example: we have a diagnostic that applies to a part of the machine that is represented by many CIM objects, the best way to associate to all the different CIM objects is through a collection.

SDK Information

MOF Class Definitions

See the file CIM_Diagnostic.MOF

Qualifiers

This is a selection of the qualifiers CIM has defined. To see all of them visit http://www.dmtf.org/
Excerpts of this were taken from the DMTF web site.

- ABSTRACT
  - Indicates that the class is abstract and serves only as a base for new classes. It is not possible to create instances of such classes.
- AGGREGATION
  - Indicates that the association is an aggregation.
- ARRAYTYPE
  - Indicates the type of the qualified array. Valid values are "Bag", "Indexed" and "Ordered".
- DESCRIPTION
  - A human readable description of the element
- IN
  - Applies to method parameters
  - Indicates this parameter is an input parameter.
- KEY
  - Indicates that the property is part of the namespace handle. If more than one property has the KEY qualifier, then all such properties collectively form the key (a compound key).
    - **Usage Rule:** Keys are written once at object instantiation and must not be modified thereafter. It does not make sense to apply a default value to a KEY-qualified property
- MAXLEN
  - Indicates the maximum number of values a given multi-valued reference can have. A value of NULL implies no limit.
- OUT
  - Applies to method parameters
  - Indicates this parameter is an output parameter
- READ
  - The property is marked readable, read-only if write is not present.
- REQUIRED
  - A non-null value must be specified.
- REVISION
  - Provides the minor revision number of the schema object.
    - **Usage Rule:** The VERSION qualifier must be present to supply the major version number when the REVISION qualifier is used.
- Units
  - Define the units a given property is in.
  - See DMTF web site for a complete listing of the key unit names.
- VALUEMAP
  - Defines the set of permissible values for this property. The ValueMap can be used alone, or in combination with the Values qualifier. When used in combination with the Values qualifier, the location of the property value in the ValueMap array provides the location of the corresponding entry in the Values array.
  - ValueMap may only be used with string and integer values.
  - For example: We have a string array that only supports a limited set of key names the qualifier would look like:
    - ValueMap{"Key1", "Key2", "Key3"}

- VALUES
  - Provides translation between an integer value and an associated string. If a ValueMap qualifier is not present, the Values array is indexed (zero relative) using the value in the associated property. If a ValueMap qualifier is present, the Values index is defined by the location of the property value in the ValueMap.
  - For example: We have an integer that we want to relate a limited set of key names to the integer values the qualifiers would look like:
  - ValueMap {“0”, “1”, “2”}, Values {"Key1", "Key2", "Key3"}
- VERSION
  - Provides the major version number of the schema object. This is incremented when changes are made to the schema that alter the interface.
- WRITE
  - The property is writable by anyone
- EXPENSIVE
  - Indicates the property is expensive to compute.
    - Diagnostics uses this in conjunction with the ResourcesUsed property of CIM_DiagnosticTest to state what is expensive in terms of system usage.
- LARGE
  - This property requires a large amount of storage space.
    - For example an instance of a subclass of CIM_DiagnosticTestSettingAndResult may have a large amount of data in it, and may need to be qualified as large.
- User-defined Qualifiers
  - The user can define any additional arbitrary named qualifiers. However, it is recommended that only defined qualifiers be used, and that the list of qualifiers be extended only if there is no other way to accomplish a particular objective.