Document Number: DSP0200

Date: 2009-07-29

Version: 1.3.1

CIM Operations over HTTP

Document Type: Specification

Document Status: DMTF Standard

Document Language: E

CONTENTS

Tables

Foreword

CIM Operations over HTTP (DSP0200) was prepared by the DMTF CIM-XML Working Group.

Introduction

This document defines a mapping of CIM messages to the Hypertext Transfer Protocol (HTTP) so that implementations of CIM can operate in an open, standardized manner. It also defines the notion of *conformance* in the context of this mapping, and it describes the behavior an implementation of CIM shall exhibit to be a conforming CIM implementation.

This document is structured as follows:

- Clause 5 describes the CIM messages that form the HTTP payload using XML. It specifies the syntax and semantics of the message requests and their corresponding responses.
- Clause 6 describes the encapsulation of these messages in HTTP request and response messages, with examples of each. It also describes the extension headers used to convey additional CIM-specific semantics in the HTTP Header.
- Clause 7 presents details of other aspects of the encapsulation:
  - HTTP version support
  - Use of standard HTTP headers
  - HTTP error codes
  - Security considerations

Requirements

There are many different ways CIM messages can be represented in XML and encapsulated within HTTP messages. To attain interoperability among different implementations of CIM, both the XML representation and the HTTP encapsulation must be standardized. The XML representation is defined in DSP0201 and DSP0203. This document uses that XML representation to define the HTTP encapsulation.

The following criteria are applied to the representation of CIM messages in XML using DSP0201 and DSP0203:

- Each CIM message is completely described in XML; completeness is favored over conciseness.
- The set of CIM messages provides enough functionality to enable implementations of CIM to communicate effectively for management purposes. This release of the mapping does not provide a *complete* set of messages. Rather, the goal is to define the mapping so that it admits straightforward extension (by the addition of further features) in future versions.
- The set of CIM messages is classified into functional profiles to accommodate a range of implementations varying from complete support of all messages to support of a minimal subset. The number of functional profiles is kept as small as possible to encourage interoperability, and mechanisms provided by different CIM implementations can declare their level of support.

The following criteria are applied to the HTTP encapsulation of CIM Messages herein:

- In recognition of the large installed base of HTTP/1.0 systems, the encapsulation is designed to support both HTTP/1.0 and HTTP/1.1.
- The encapsulation does not introduce requirements that conflict with those stated in HTTP/1.0 or HTTP/1.1.
- Use of the encapsulation should be straightforward over the current base HTTP infrastructures. Some features anticipate and exploit enhancements to this base, but no aspects of the encapsulation require such enhancements as mandatory.

198 • The encapsulation avoids the use of pure HTTP tunneling or URL munging (for example, the
199 use of the "?" character) in favor of a mechanism that allows existing HTTP infrastructures to
200 control content safely.

201 • The encapsulation exposes key CIM message information in headers to allow efficient
202 firewall/proxy handling. The information is limited to essentials so that it does not have a
203 significant impact on the size of the header. All CIM-specific information in a header also
204 appears within the CIM message.

205 • There is a clear and unambiguous encapsulation of the CIM message payload within the HTTP
206 message. Conciseness of the encapsulation is of secondary importance.

CIM Operations over HTTP

1 Scope

The Common Information Model (CIM) (DSP0004) is an object-oriented information model defined by the Distributed Management Task Force (DMTF) that provides a conceptual framework for describing management data.

The Hypertext Transfer Protocol (HTTP) (RFC 1945, RFC 2068, RFC 2616) is an application-level protocol for distributed, collaborative, hypermedia information systems. This generic stateless protocol can be used for many tasks through extension of its request methods, error codes, and headers.

The Extensible Markup Language (XML) is a simplified subset of SGML that offers powerful and extensible data modeling capabilities. An *XML document* is a collection of data represented in XML. An *XML schema* is a grammar that describes the structure of an XML document.

This document defines a mapping of CIM messages onto HTTP that allows implementations of CIM to interoperate in an open, standardized manner. It is based on the *CIM XML DTD* (DSP0201 and DSP0203) that defines the XML schema for CIM objects and messages.

2 Normative References

The following referenced documents are indispensable for applying the information in this document while developing an implementation of CIM. For dated references, only the edition cited applies. For undated references, the latest edition applies, including any amendments.

DMTF DSP0004, *Common Information Model (CIM) Infrastructure 2.5*, http://www.dmtf.org/standards/published_documents/DSP0004_2.5.pdf

DMTF DSP0201, *Specification for the Representation of CIM in XML 2.3*, http://www.dmtf.org/standards/published_documents/DSP0201_2.3.pdf

DMTF DSP0203, *CIM XML DTD 2.3*, http://www.dmtf.org/standards/published_documents/DSP0203_2.3.dtd

IETF RFC 1766, *Tags for the Identification of Languages*, March 1995, http://www.ietf.org/rfc/rfc1766.txt

IETF RFC 1945, *Hypertext Transfer Protocol – HTTP/1.0*, May 1996, http://www.ietf.org/rfc/rfc1945.txt

IETF RFC 2068, *Hypertext Transfer Protocol – HTTP/1.1*, January 1997, http://www.ietf.org/rfc/rfc2068.txt

IETF RFC 2069, *An Extension to HTTP: Digest Access Authentication*, January 1997, http://www.ietf.org/rfc/rfc2069.txt

IETF RFC 2277, *IETF Policy on Character Sets and Languages*, January 1998, http://www.ietf.org/rfc/rfc2277.txt

IETF RFC 2279, *UTF-8, a transformation format of Unicode and ISO 10646*, January 1998, http://www.ietf.org/rfc/rfc2279.txt

IETF RFC 2376, *XML Media Types*, July 1998, http://www.ietf.org/rfc/rfc2376.txt

IETF RFC 2396, *Uniform Resource Identifiers (URI): Generic Syntax*, August 1998, http://www.ietf.org/rfc/rfc2396.txt

IETF RFC 2616, *Hypertext Transfer Protocol – HTTP/1.1*, June 1999, http://www.ietf.org/rfc/rfc2616.txt

IETF RFC 2617, *HTTP Authentication: Basic and Digest Access Authentication*, June 1999, http://www.ietf.org/rfc/rfc2617.txt

IETF RFC 2774, *HTTP Extension Framework*, February 2000, http://www.ietf.org/rfc/rfc2774.txt

W3C Recommendation, *Extensible Markup Language (XML)*, Version 1.0, August 2006, http://www.w3.org/TR/REC-xml-names/

W3C Recommendation, *Namespaces in XML*, Jan 1999, http://www.w3.org/TR/1999/REC-xml-names-19990114/

W3C, *XML Schema Part 1: Structures*, May 2001, http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

W3C, *XSL Transformations (XSLT)*, Version 1.0, November 1999, http://www.w3.org/TR/xslt

2.1 Other References

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

3 Terms and Definitions

Throughout this document, the following terms and definitions apply. This specification uses the same notational conventions and basic parsing constructs that are defined in RFC 2068.

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

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

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

3.4
mandatory
indicates requirements that must be strictly followed to conform to the document, with no permitted deviations

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

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

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

3.8
shall
indicates requirements that must be strictly followed to conform to the document, with no permitted deviations

3.9
shall not
indicates requirements that must be strictly followed to conform to the document, with no permitted deviations

3.10
should
indicates that among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required

3.11
should not
indicates that a certain possibility or course of action is deprecated but not prohibited

3.12
unspecified
indicates that this profile does not define any constraints for the referenced CIM element or operation

3.13
CIM Message
a well-defined request or response data packet used to exchange information among CIM products

3.14
CIM Operation Message
a CIM message used to invoke an operation on the target namespace

3.15
CIM Export Message
a CIM message used to communicate information about a CIM namespace or element that is foreign to the target

3.16
Operation Request Message
an XML document that is loosely valid with respect to the CIM XML DTD and that contains a <MESSAGE> subelement under the root <CIM> node that has a <MULTIREQ> or <SIMPLEREQ> subelement under this subelement

4 Abbreviated Terms and Document Conventions

4.1 Abbreviated Terms

The following symbols and abbreviations are used in this document.

4.1.1
CIM
Common Information Model

4.1.2
DTD
document type definition

4.1.3
HTTP
Hypertext Transfer Protocol

4.1.4
XML
Extensible Markup Language

4.2 Document Conventions

Throughout this document, any deprecated element is indicated by one of the following labels:

- The "**DEPRECATION NOTE:**" label preceding a paragraph indicates that the paragraph explains a deprecated element.
- The "**DEPRECATED.**" label before a list item indicates that the information in that list item is deprecated.
- The "(DEPRECATED)" label after a heading applies to the entire clause for that heading.
- The "(DEPRECATED)" label at the end of a line in a code fragment or an example indicates that the particular line of the code fragment or example is deprecated.

5 CIM Message Syntax and Semantics

This specification defines all interactions among CIM products as CIM messages. A *CIM message* is a well-defined request or response data packet for exchanging information among CIM products. The two types of *CIM messages* are as follows:

- *CIM operation message*. This type of message is used to invoke an operation on the target namespace.
- *CIM export message.* This type of message is used to communicate information about a CIM namespace or element that is foreign to the target. It is informational only and does not define an operation on the target CIM namespace or even imply the existence of a target namespace.

This clause describes the syntax and semantics of CIM messages independently of their encapsulation within a particular protocol such as HTTP. XML is used as the basis for this description, and in particular the CIM XML DTD (DSP0201 and DSP0203).

5.1 Well-Formed, Valid, and Loosely Valid Documents

In this discussion, any reference to well-formed or valid XML documents has the standard meaning defined in *Extensible Markup Language (XML)*.

XML document type definitions (DTDs) are restricted to be either well-formed or valid. However, this document also uses the term *loosely valid* to apply to XML that removes any attributes or elements in the XML document that do not appear in the *CIM XML DTD*. The resulting document is valid with respect to the *CIM XML DTD* and is therefore loosely valid.

In effect, a loosely valid document is valid with respect to the *CIM XML DTD* apart from having additional attributes or elements not defined by that DTD. The concept is very similar to that of an *open content model* as defined by the working draft on XML Schemas, expressed within the more limited scope of DTDs. One corollary of this definition is that any XML document that is valid with respect to the *CIM XML DTD* is also loosely valid.

The motivation for introducing the loosely valid class of XML documents is to relax the restrictions on a DefCIMClient, CIM server, or CIM listener when parsing received XML documents defined within the scope of this mapping. Not all clients (including their respective CIM servers or CIM listeners) should be required to validate each received CIM message response (or its respective CIM message request) because such a requirement would place too heavy a processing burden on the validating entity at the expense of footprint and performance, most notably in communication between robust and conformant implementations of this mapping.

Instead, the following requirements are set forth in this specification. In all cases, a CIM client has a respective alternative CIM server or CIM listener, and a received CIM message response has a respective alternative CIM message request:

- A CIM client may include a DOCTYPE element in a CIM message request. If so, an external declaration should be used. In-lining of the complete DTD within a message is discouraged.
- A CIM client may elect to validate a received CIM message response.
- If a CIM client elects not to validate a received CIM message, then loose validation shall be enforced.

The behavior of a CIM server or CIM listener with respect to a received CIM message request is covered in detail in 7.3.

5.2 Operational Semantics

The CIM XML DTD (DSP0201 and DSP0203) defines a subelement under the root <CIM> element called <MESSAGE>, which contains one of the following subelements:

- CIM operation message subelements
  - <SIMPLEREQ>
  - <SIMPLERSP>
  - <MULTIREQ>
  - <MULTIRSP>
- CIM export message subelements
  - <SIMPLEXPREQ>
  - <SIMPLEXPRSP>
  - <MULTIEXPREQ>
  - <MULTIEXPRSP>

In the remainder of this document, the following terms denote an XML document that is loosely valid with respect to the CIM XML DTD:

- *Operation request message.* Contains under the root <CIM> node a <MESSAGE> subelement that has a <MULTIREQ> or <SIMPLEREQ> subelement under it.
- *Operation response message.* Contains under the root <CIM> node a <MESSAGE> subelement that has a <MULTIRSP> or <SIMPLERSP> subelement under it.
- *Export request message*. Contains under the root <CIM> node a <MESSAGE> subelement that has a <MULTIEXPREQ> or <SIMPLEEXPREQ> subelement under it.
- *Export response message*. Contains under the root <CIM> node a <MESSAGE> subelement that has a <MULTIEXPRSP> or <SIMPLEEXPRSP> subelement under it.

The phrase *CIM message request* refers to either an operation request message or an export request message. The phrase *CIM message response* refers to either an operation response message or an export response message.

A CIM message request shall contain a non-empty value for the ID attribute of the <MESSAGE> element. The corresponding CIM message response shall supply the same value for that attribute. Clients should employ a message ID scheme that minimizes the chance of receiving a stale CIM message response.

Any CIM message conforming to this specification shall have a minimum value of "1.0" and a maximum value that is equal to the latest version of this specification for the `PROTOCOLVERSION` attribute of the <MESSAGE> element.

An operation response message sent in response to an operation request message shall specify the same value for the `ID` attribute of the <MESSAGE> element that appears in the request message and contain one of the following:

- A <MULTIRSP> subelement, if the operation request message contains a <MULTIREQ> subelement.
- A <SIMPLERSP> subelement, if the operation request message contains a <SIMPLEREQ> subelement.

A *simple operation request* is an operation request message that contains a <SIMPLEREQ> subelement. A *simple operation response* is an Operation Response Message that contains a <SIMPLERSP> subelement.

A *multiple operation request* is an operation request message that contains a <MULTIREQ> subelement. A *multiple operation response* is an operation response message that contains a <MULTIRSP> subelement.

An export response message sent in response to an export request message shall specify the same value for the ID attribute of the <MESSAGE> element that appears in the export request message and shall contain one of the following:

- A <MULTIEXPRSP> subelement if the export request message contained a <MULTIEXPREQ> subelement, or
- A <SIMPLEEXPRSP> subelement if the export request message contained a <SIMPLEEXPREQ> subelement.

A simple export request is an export request message that contains a <SIMPLEEXPREQ> subelement. A simple export response is an export response message that contains a <SIMPLEEXPRSP> subelement.

A multiple export request is an export request message that contains a <MULTIEXPREQ> subelement. A multiple export response is an export response message that contains a <MULTIEXPRSP> subelement.

5.3 CIM Operation Syntax and Semantics

This clause describes method invocations, intrinsic methods, and namespace manipulation.

5.3.1 Method Invocations

All CIM operation requests defined for this CIM-to-HTTP mapping are defined as invocations of one or more methods. A method can be either:

- An *intrinsic* method, which is defined for the purposes of modeling a CIM operation.
- An *extrinsic* method, which is defined on a CIM class in a schema.

In addition, intrinsic methods are made against a CIM namespace. Extrinsic methods are invoked on a CIM class (if static) or instance otherwise. Intrinsic methods are defined in 5.3.2.

An extrinsic method call is represented in XML by the <METHODCALL> element, and the response to that call is represented by the <METHODRESPONSE> element.

An intrinsic method call is represented in XML by the <IMETHODCALL> element, and the response to that call is represented by the <IMETHODRESPONSE> element. An input parameter has an IN qualifier (with a value of `true`) in the method definition, and an output parameter has an OUT qualifier (with a value of `true`). A parameter can be both an input and an output parameter.

The <METHODCALL> or <IMETHODCALL> element names the method to be invoked and supplies any input parameters to the method call. Note the following rules about parameters:

- Each input parameter shall be named using the name assigned in the method definition.
- Input parameters may be supplied in any order.
- Each input parameter of the method, and no others, shall be present in the call, unless it is optional.

The <METHODRESPONSE> or <IMETHODRESPONSE> element defines either an <ERROR> or an optional return value and output parameters if it is decorated with the OUT qualifier in the method definition. In the latter case, the following rules about parameters apply:

- Each output parameter shall be named using the name assigned in the method definition.
- Output parameters may be supplied in any order.
- Each output parameter of the method, and no others, shall be present in the response, unless it is optional.
- The method invocation process can be thought of as the binding of the input parameter values specified as subelements of the <METHODCALL> or <IMETHODCALL> element to the input parameters of the method. This binding is followed by an attempt to execute the method using the bound input parameters with one of the following results:
  - If the attempt to call the method is successful, the return value and output parameters are bound to the subelements of the <METHODRESPONSE> or <IMETHODRESPONSE> element.
  - If the attempt to call the method is unsuccessful, an error code and optional human-readable description of that code is bound to the <METHODRESPONSE> or <IMETHODRESPONSE> element.

5.3.1.1 Simple Operations

A simple operation invokes a single method. A simple operation request is represented by a <SIMPLEREQ> element, and a simple operation response is represented by a <SIMPLERSP> element.

If the method is intrinsic, then the <SIMPLEREQ> element shall contain an <IMETHODCALL> element, which in turn contains a <LOCALNAMESPACEPATH> subelement identifying the local CIM namespace against which the method is to execute. If the method is extrinsic, then the <SIMPLEREQ> element shall contain a <METHODCALL> element that in turn contains one of the following subelements:

- A <LOCALCLASSPATH> subelement identifying the CIM class on which the method is to be invoked if the method is static.
- A <LOCALINSTANCEPATH> subelement identifying the CIM instance on which the method is otherwise to be invoked.

5.3.1.2 Multiple Operations

A multiple operation requires the invocation of more than one method. A multiple operation request is represented by a <MULTIREQ> element, and a multiple operation response is represented by a <MULTIRSP> element.

A <MULTIREQ> (or its respective <MULTIRSP>) element is a sequence of two or more <SIMPLEREQ> (or their respective <SIMPLERSP>) elements.

A <MULTIRSP> element shall contain a <SIMPLERSP> element for every <SIMPLEREQ> element in the corresponding multiple operation response. These <SIMPLERSP> elements shall be in the same order as their <SIMPLEREQ> counterparts so that the first <SIMPLERSP> in the response corresponds to the first <SIMPLEREQ> in the request, and so forth.

Multiple operations conveniently allow multiple method invocations to be batched into a single HTTP message. Batching reduces the number of roundtrips between a CIM client and a CIM server and allows the CIM server to make internal optimizations if it chooses. Note that multiple operations do not confer any transactional capabilities in processing the request. For example, the CIM server does not have to guarantee that the constituent method calls either all fail or succeed, only that the entity make a "best effort" to process the operation. However, servers shall finish processing each operation in a batched operation before executing the next one. Clients shall recognize that the order of operations within a batched operation is significant.

Not all CIM servers support multiple operations; the way they declare support for this feature is defined in 7.5.

5.3.1.3 Status Codes

This clause defines the status codes and detailed error information that a conforming CIM server application can return. The value of an <ERROR> subelement within a <METHODRESPONSE> or <IMETHODRESPONSE> element includes the following parts:

- a mandatory status code
- an optional human-readable description of the status code
- zero or more CIM_Error instances

Table 1 defines the status codes that a conforming CIM server application can return as the value of the CODE attribute of an <ERROR> subelement. In addition to a status code, a conforming CIM server may return zero or more <INSTANCE> subelements as part of an <ERROR> element. Each <INSTANCE> subelement shall be an instance of CIM_Error. For each instance of CIM_Error, the value of CIMStatusCode shall comply with the definition of expected error codes for the CIM operation request. A CIM client may ignore any <INSTANCE> subelements.

The symbolic names defined in Table 1 do not appear on the wire. They are used here solely for convenient reference to an error in other parts of this specification.

521 Not all methods are expected to return all the status codes listed in Table 1. For intrinsic methods, the
522 relevant clause on each method in this specification defines the error codes expected to be returned. For
523 extrinsic methods, 5.3.5 specifies which of the codes in Table 1 can be used.

524 **Table 1 – Status Codes Returned by an <Error> Subelement**

Symbolic Name	Code	Definition
CIM_ERR_FAILED	1	A general error occurred that is not covered by a more specific error code.
CIM_ERR_ACCESS_DENIED	2	Access to a CIM resource is not available to the client.
CIM_ERR_INVALID_NAMESPACE	3	The target namespace does not exist.
CIM_ERR_INVALID_PARAMETER	4	One or more parameter values passed to the method are not valid.
CIM_ERR_INVALID_CLASS	5	The specified class does not exist.
CIM_ERR_NOT_FOUND	6	The requested object cannot be found. The operation can be unsupported on behalf of the CIM server in general or on behalf of an implementation of a management profile.
CIM_ERR_NOT_SUPPORTED	7	The requested operation is not supported on behalf of the CIM server, or on behalf of a provided class. If the operation is supported for a provided class but is not supported for particular instances of that class, then CIM_ERR_FAILED shall be used.
CIM_ERR_CLASS_HAS_CHILDREN	8	The operation cannot be invoked on this class because it has subclasses.
CIM_ERR_CLASS_HAS_INSTANCES	9	The operation cannot be invoked on this class because one or more instances of this class exist.
CIM_ERR_INVALID_SUPERCLASS	10	The operation cannot be invoked because the specified superclass does not exist.
CIM_ERR_ALREADY_EXISTS	11	The operation cannot be invoked because an object already exists.
CIM_ERR_NO_SUCH_PROPERTY	12	The specified property does not exist.
CIM_ERR_TYPE_MISMATCH	13	The value supplied is not compatible with the type.
CIM_ERR_QUERY_LANGUAGE_NOT_SUPPORTED	14	The query language is not recognized or supported.
CIM_ERR_INVALID_QUERY	15	The query is not valid for the specified query language.
CIM_ERR_METHOD_NOT_AVAILABLE	16	The extrinsic method cannot be invoked.
CIM_ERR_METHOD_NOT_FOUND	17	The specified extrinsic method does not exist.
CIM_ERR_NAMESPACE_NOT_EMPTY	20	The specified namespace is not empty.

Symbolic Name	Code	Definition
CIM_ERR_INVALID_ENUMERATION_CONTEXT	21	The enumeration identified by the specified context cannot be found, is in a closed state, does not exist, or is otherwise invalid.
CIM_ERR_INVALID_OPERATION_TIMEOUT	22	The specified operation timeout is not supported by the CIM Server.
CIM_ERR_PULL_HAS_BEEN_ABANDONED	23	The Pull operation has been abandoned due to execution of a concurrent CloseEnumeration operation on the same enumeration.
CIM_ERR_PULL_CANNOT_BE_ABANDONED	24	The attempt to abandon a concurrent Pull operation on the same enumeration failed. The concurrent Pull operation proceeds normally.
CIM_ERR_FILTERED_ENUMERATION_NOT_SUPPORTED	25	Using a filter in the enumeration is not supported by the CIM server.
CIM_ERR_CONTINUATION_ON_ERROR_NOT_SUPPORTED	26	The CIM server does not support continuation on error.
CIM_ERR_SERVER_LIMITS_EXCEEDED	27	The CIM server has failed the operation based upon exceeding server limits.
CIM_ERR_SERVER_IS_SHUTTING_DOWN	28	The CIM server is shutting down and cannot process the operation.

5.3.2 Intrinsic Methods

This clause describes the Intrinsic methods defined outside the schema for CIM operations. These methods can only be called on a CIM namespace, rather than on a CIM class or instance.

The notation used in the following subclauses to define the signatures of the intrinsic methods is a pseudo-MOF notation that extends the standard MOF BNF (DSP0004) for describing CIM methods with several pseudo-parameter types enclosed within angle brackets (< and >).

This notation decorates the parameters with pseudo-qualifiers (IN, OUT, OPTIONAL, and NULL) to define their invocation semantics. These qualifiers are for description purposes only within the scope of this specification; in particular, a CIM client shall not specify them in intrinsic method invocations.

This notation uses the IN qualifier to denote that the parameter is an input parameter.

This notation uses the OUT qualifier to denote that the parameter is an output parameter.

A CIM client may omit an optional parameter by not specifying an <IPARAMVALUE> element for that parameter if the required value is the specified default. It shall not omit a parameter that is not marked as optional. A CIM server may omit support for an optional parameter. Any attempt to call a method with an optional parameter that is not supported shall return either CIM_ERR_NOT_SUPPORTED or CIM_ERR_INVALID_PARAMETER.

This notation uses the NULL qualifier for parameters whose values can be specified as NULL in a method call. A NULL (unassigned) value for a parameter is specified by an <IPARAMVALUE> or <PARAMVALUE> element with no subelement. For parameters without the NULL qualifier, the CIM client shall specify a value by including a suitable subelement for the <IPARAMVALUE> or <PARAMVALUE> element.

All parameters shall be uniquely named and shall correspond to a valid parameter name for that method as described by this specification. The order of the parameters is not significant.

The non-NULL values of intrinsic method parameters or return values modeled as standard CIM types (such as string and Boolean or arrays thereof) are represented as follows:

- Simple values use the <VALUE> subelement within an <IPARAMETER> element for method parameters or within an <IRETURNVALUE> element for method return values.
- Array values use the <VALUE.ARRAY> subelement within an <IPARAMETER> element for method parameters or within an <IRETURNVALUE> element for method return values.

Table 2 shows how each pseudo-type used by the intrinsic methods shall be mapped to an XML element described in DSP0201 in the context of both a parameter value (subelement of <IPARAMVALUE>) and a return value (subelement of <IRETURNVALUE>).

Table 2 – Mapping of Intrinsic Method Pseudo-Types to XML Elements

Type	XML Element
<object>	(VALUE.OBJECT|VALUE.OBJECTWITHLOCALPATH|VALUE.OBJECTWITHPATH)
<class>	CLASS
<instance>	INSTANCE
<className>	CLASSNAME
<namedInstance>	VALUE.NAMEDINSTANCE
<instanceName>	INSTANCENAME
<instancePath>	INSTANCEPATH
<objectWithPath>	VALUE.OBJECTWITHPATH
<instanceWithPath>	VALUE.INSTANCEWITHPATH
<objectName>	(CLASSNAME|INSTANCENAME)
<objectPath>	OBJECTPATH
<propertyValue>	(VALUE|VALUE.ARRAY|VALUE.REFERENCE)
<qualifierDecl>	QUALIFIER.DECLARATION
<enumerationContext>	ENUMERATIONCONTEXT

5.3.2.1 GetClass

The GetClass operation returns a single CIM class from the target namespace:

```
    GetClass
    <class>GetClass (
        [IN] <className> ClassName,
        [IN,OPTIONAL] boolean LocalOnly = true,
        [IN,OPTIONAL] boolean IncludeQualifiers = true,
        [IN,OPTIONAL] boolean IncludeClassOrigin = false,
        [IN,OPTIONAL,NULL] string PropertyList [] = NULL
)
```

The `ClassName` input parameter defines the name of the class to be retrieved.

If the `LocalOnly` input parameter is `true`, any CIM elements (properties, methods, and qualifiers), except those added or overridden in the class as specified in the classname input parameter, shall not be included in the returned class. If it is false, no additional filtering is defined.

If the `IncludeQualifiers` input parameter is `true`, all qualifiers for that class (including qualifiers on the class and on any returned properties, methods, or method parameters) shall be included as <QUALIFIER> elements in the response. If it is `false`, no <QUALIFIER> elements are present in the returned class.

If the `IncludeClassOrigin` input parameter is `true`, the `CLASSORIGIN` attribute shall be present on all appropriate elements in the returned class. If it is `false`, no `CLASSORIGIN` attributes are present in the returned class.

If the `PropertyList` input parameter is not `NULL`, the members of the array define one or more property names. The returned class shall not include elements for properties missing from this list. Note that if `LocalOnly` is specified as `true`, it acts as an additional filter on the set of properties returned. For example, if property A is included in the `PropertyList` but `LocalOnly` is set to `true` and A is not local to the requested class, it is not included in the response. If the `PropertyList` input parameter is an empty array, no properties are included in the response. If the `PropertyList` input parameter is NULL, no additional filtering is defined.

If the `PropertyList` contains duplicate elements, the server shall ignore them but otherwise process the request normally. If the `PropertyList` contains elements that are invalid property names for the target class, the server shall ignore them but otherwise process the request normally.

If GetClass is successful, the return value is a single CIM class that shall include all CIM elements (properties, methods, and qualifiers) defined in or inherited by that class, reduced by any elements excluded as a result of using the `LocalOnly` or `PropertyList` filters.

If GetClass is unsuccessful, this method shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional method-specific interpretation of the error is enclosed in parentheses:

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_NOT_FOUND (The request CIM class does not exist in the specified namespace.)
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.2 GetInstance

The GetInstance operation returns a single CIM instance from the target namespace:

```
GetInstance
<instance>GetInstance (
    [IN] <instanceName> InstanceName,
    [IN,OPTIONAL] boolean LocalOnly = true,  (DEPRECATED)
    [IN,OPTIONAL] boolean IncludeQualifiers = false,  (DEPRECATED)
    [IN,OPTIONAL] boolean IncludeClassOrigin = false,
    [IN,OPTIONAL,NULL] string PropertyList [] = NULL
)
```

The `InstanceName` input parameter defines the name of the instance to be retrieved.

DEPRECATION NOTE: With the 1.2 release of this specification, the LocalOnly parameter is DEPRECATED. LocalOnly filtering, as defined in 1.1, will not be supported in the next major revision of this specification. In the 1.1 version of this specification, the definition of the LocalOnly parameter was incorrectly modified. This change introduced a number of interoperability and backward compatibility

problems for CIM clients using the LocalOnly parameter to filter the set of properties returned. The DMTF strongly recommends that CIM clients set LocalOnly to `false` and do not use this parameter to filter the set of properties returned. To minimize the impact of this recommendation on CIM clients, a CIM server may choose to treat the value of the LocalOnly parameter as `false` for all requests. A CIM server shall consistently support a single interpretation of the LocalOnly parameter. Refer to ANNEX B for additional details.

DEPRECATION NOTE: The use of the `IncludeQualifiers` parameter is DEPRECATED and it may be removed in a future version of this specification. The `IncludeQualifiers` parameter definition is ambiguous and when it is set to `true`, CIM clients cannot be assured that any qualifiers will be returned. A CIM client should always set `IncludeQualifiers` to `false`. To minimize the impact of this recommendation on CIM clients, a CIM server may choose to treat the value of the `IncludeQualifiers` parameter as `false` for all requests. The preferred behavior is to use the class operations to receive qualifier information and not depend on any qualifiers existing in this response. If the `IncludeQualifiers` input parameter is `true`, all qualifiers for that instance (including qualifiers on the instance and on any returned properties) shall be included as <QUALIFIER> elements in the response. If it is `false`, no <QUALIFIER> elements are present.

If the `IncludeClassOrigin` input parameter is `true`, the `CLASSORIGIN` attribute shall be present on all appropriate elements in the returned instance. If it is `false`, no `CLASSORIGIN` attributes are present.

If the `PropertyList` input parameter is not `NULL`, the members of the array define one or more property names. The returned instance shall not include elements for properties missing from this list. Note that if `LocalOnly` is `true`, this acts as an additional filter on the set of properties returned. For example, if property A is included in the `PropertyList` but `LocalOnly` is set to `true` and A is not local to the requested instance, it is not included in the response. If the `PropertyList` input parameter is an empty array, no properties are included in the response. If the `PropertyList` input parameter is NULL, no additional filtering is defined.

If the `PropertyList` contains duplicate elements, the server shall ignore the duplicates but otherwise process the request normally. If the `PropertyList` contains elements that are invalid Property names for the target instance, the server shall ignore them but otherwise process the request normally.

Properties with the NULL value may be omitted from the response, even if the CIM client has not requested the exclusion of the property through the `LocalOnly` or `PropertyList` filters. The CIM client shall interpret such omitted properties as NULL. Note that the CIM client cannot make any assumptions about properties omitted as a result of using `LocalOnly` or `PropertyList` filters.

If GetInstance is successful, the return value is a single CIM instance with all properties defined in and inherited by its class reduced by any properties excluded as a result of using the `LocalOnly` or `PropertyList` filters and further reduced by any NULL valued properties omitted from the response.

If GetInstance is unsuccessful, the method shall return one of the following status codes where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional method-specific interpretation of the error is enclosed in parentheses:

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_INVALID_CLASS (The CIM class does not exist in the specified namespace.)
- CIM_ERR_NOT_FOUND (The CIM class does exist, but the requested CIM instance does not exist in the specified namespace.)

- CIM_ERR_FAILED (some other unspecified error occurred)

5.3.2.3 DeleteClass

The DeleteClass operation deletes a single CIM class from the target namespace:

```
DeleteClass
void  DeleteClass (
   [IN] <className> ClassName
)
```

The `ClassName` input parameter defines the name of the class to be deleted.

If DeleteClass is successful, the CIM server removes the specified class, including any subclasses and any instances. The operation shall fail if any one of these objects cannot be deleted.

If DeleteClass is unsuccessful, this method shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional method-specific interpretation of the error is enclosed in parentheses:

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_NOT_FOUND (The CIM class to be deleted does not exist.)
- CIM_ERR_CLASS_HAS_CHILDREN (The CIM class has one or more subclasses that cannot be deleted.)
- CIM_ERR_CLASS_HAS_INSTANCES (The CIM class has one or more instances that cannot be deleted.)
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.4 DeleteInstance

The DeleteInstance operation deletes a single CIM instance from the target namespace.

```
DeleteInstance
void  DeleteInstance (
   [IN] <instanceName> InstanceName
)
```

The `InstanceName` input parameter defines the name (model path) of the instance to be deleted.

Deleting the instance may or may not cause the automatic deletion of additional instances. For example, the deletion of an instance may cause the automatic deletion of all associations that reference that instance. Or the deletion of an instance may cause the automatic deletion of instances (and their associations) that have a Min(1) relationship to that instance.

If DeleteInstance is successful, the CIM server removes the specified instance.

If DeleteInstance is unsuccessful, this method shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional method-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED

- CIM_ERR_NOT_SUPPORTED (by the CIM server for this operation)
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_INVALID_CLASS (The CIM class does not exist in the specified namespace.)
- CIM_ERR_NOT_SUPPORTED (This operation is not supported for the class of the specified instance, if provided.)
- CIM_ERR_NOT_FOUND (The CIM class does exist, but the requested CIM instance does not exist in the specified namespace.)
- CIM_ERR_FAILED (This operation is not supported for the specified instance, or some other unspecified error occurred.)

5.3.2.5 CreateClass

The CreateClass operation creates a single CIM class in the target namespace. The class shall not already exist:

```
CreateClass
void CreateClass (
   [IN] <class> NewClass
)
```

The `NewClass` input parameter defines the new class. The proposed definition shall be a correct class definition according to DSP0004.

In processing the creation of the new class, the CIM server shall conform to the following rules:

- The server shall ignore any `CLASSORIGIN` and `PROPAGATED` attributes in the `NewClass`.
- If the new class has no superclass, the `NewClass` parameter defines a new base class. The server shall ensure that all properties and methods of the new class have a `CLASSORIGIN` attribute whose value is the name of the new class.
- If the new class has a superclass, the `NewClass` parameter defines a new subclass of that superclass. The superclass shall exist. The server shall ensure that the following conditions are met:
  - Any properties, methods, or qualifiers in the subclass not defined in the superclass are created as new elements of the subclass. In particular, the server shall set the `CLASSORIGIN` attribute on the new properties and methods to the name of the subclass and ensure that all others preserve their `CLASSORIGIN` attribute value from that defined in the superclass.
  - If a property is defined in the superclass and in the subclass, the value assigned to that property in the subclass (including NULL) becomes the default value of the property for the subclass.
  - If a property or method of the superclass is not specified in the subclass, then it is inherited without modification by the subclass.
  - Any qualifiers defined in the superclass with a `TOSUBCLASS` attribute value of `true` shall appear in the resulting subclass. Qualifiers in the superclass with a `TOSUBCLASS` attribute value of `false` shall not be propagated to the subclass.
  - Any qualifier propagated from the superclass cannot be modified in the subclass if the `OVERRIDABLE` attribute of that qualifier is set to `false` in the superclass. It is a client error

to specify such a qualifier in the `NewClass` with a definition different than that in the superclass (where definition encompasses the name, type, and flavor attribute settings of the <QUALIFIER> element and the value of the qualifier).

If CreateClass is successful, the CIM server creates the specified class.

If CreateClass is unsuccessful, this method shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional method-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_ALREADY_EXISTS (The CIM class already exists.)
- CIM_ERR_INVALID_SUPERCLASS (The putative CIM class declares a non-existent superclass.)
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.6 CreateInstance

The CreateInstance operation creates a single CIM Instance in the target namespace. The instance shall not already exist:

```
CreateInstance
<instanceName>CreateInstance (
    [IN] <instance> NewInstance
)
```

DEPRECATION NOTE: The use of qualifiers on instances is DEPRECATED and may be removed in a future version of this specification. A CIM client cannot rely on any qualifiers included in the NewInstance to have any impact on the operation. It is recommended that the CIM server ignore any qualifiers included in the instance. The `NewInstance` input parameter defines the new instance. The proposed definition shall be a correct instance definition for the underlying CIM class according to DSP0004.

In creating the new instance, the CIM server shall conform to the following rules and ensure that they are applied:

- The server shall ignore any `CLASSORIGIN` and `PROPAGATED` attributes in the `NewInstance`.
- **DEPRECATED.** Any qualifiers in the instance not defined in the class are created as new elements of the instance.
- All properties of the instance preserve their `CLASSORIGIN` attribute value from that defined in the class.
- If a property is specified in the `NewInstance` parameter, the value assigned to that property in the instance (including NULL) becomes the value of the property for the instance. Note that it is a client error to specify a property that does not belong to the class.
- If a property of the class is not specified in the instance, then the instance inherits that property without modification.
- **DEPRECATION NOTE:** Use of the TOINSTANCE attribute is DEPRECATED**.** Servers may choose to ignore TOINSTANCE. Servers that do not ignore TOINSTANCE shall interpret it so

that any qualifiers defined in the class with a TOINSTANCE attribute value of `true` appear in the instance. Qualifiers in the class with a value of `false` shall not be propagated to the instance.

- **DEPRECATED.** Any Qualifier propagated from the class cannot be modified in the instance if the `OVERRIDABLE` attribute of that qualifier is set to `false` in the class. It is a client error to specify such a qualifier in the `NewInstance` with a definition different than that in the class (where definition encompasses the name, type, and flavor attribute settings of the `<QUALIFIER>` element and the value of the qualifier).

If CreateInstance is successful, the return value defines the object path of the new CIM instance relative to the target namespace created by the CIM server (that is, the model path as defined by DSP0004). It is returned if one or more of the new keys of the instance are dynamically allocated during creation rather than specified in the request.

If CreateInstance is unsuccessful, this method shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional method-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED (by the CIM server for this operation)
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_INVALID_CLASS (The CIM class for the new instance does not exist.)
- CIM_ERR_NOT_SUPPORTED (This operation is not supported for the class of the specified instance, if provided.)
- CIM_ERR_ALREADY_EXISTS (The CIM instance already exists.)
- CIM_ERR_FAILED (This operation is not supported for the specified instance or some other unspecified error occurred.)

5.3.2.7 ModifyClass

The ModifyClass operation modifies an existing CIM class in the target namespace. The class shall already exist:

```
ModifyClass
void ModifyClass (
   [IN] <class> ModifiedClass
)
```

The `ModifiedClass` input parameter defines the set of changes to be made to the current class definition, which shall be correct amendments to the CIM class as defined by DSP0004.

In modifying the class, the CIM server shall conform to the following rules:

- The CIM server shall ignore any `CLASSORIGIN` and `PROPAGATED` attributes in the `ModifiedClass`.
- If the modified class has no superclass, the `ModifiedClass` parameter defines modifications to a base class. The server shall ensure that the following conditions are met:
  - All properties and methods of the modified class have a `CLASSORIGIN` attribute whose value is the name of this class.

828 – Any properties, methods, or qualifiers in the existing class definition that do not appear in
829 the `ModifiedClass` parameter are removed from the resulting modified class.

830 • If the modified class has a superclass, the `ModifiedClass` parameter defines modifications to
831 a subclass of that superclass. The superclass shall exist, and the client shall not change the
832 name of the superclass in the modified subclass. The server shall ensure that the following
833 conditions are met:

834 – Any properties, methods, or qualifiers in the subclass not defined in the superclass are
835 created as elements of the subclass. In particular, the server shall set the `CLASSORIGIN`
836 attribute on the new properties and methods to the name of the subclass and shall ensure
837 that all other others preserve their `CLASSORIGIN` attribute value from that defined in the
838 superclass.

839 – Any property, method, or qualifier previously defined in the subclass but not defined in the
840 superclass, and which is not present in the `ModifiedClass` parameter, is removed from
841 the subclass.

842 – If a property is specified in the `ModifiedClass` parameter, the value assigned to that
843 property (including NULL) becomes the default value of the property for the subclass.

844 – If a property or method of the superclass is not specified in the subclass, then the subclass
845 inherits it without modification. Any previous changes to such an element in the subclass
846 are lost.

847 – If a qualifier in the superclass is not specified in the subclass and the qualifier is defined in
848 the superclass with a `TOSUBCLASS` attribute value of `true`, then the qualifier shall still be
849 present in the resulting modified subclass. A propagated qualifier cannot be removed from
850 a subclass.

851 – Any qualifier propagated from the superclass cannot be modified in the subclass if the
852 `OVERRIDABLE` attribute of that qualifier is set to `false` in the superclass. It is a client error
853 to specify such a qualifier in the `ModifiedClass` with a definition different than that in the
854 superclass (where definition encompasses the name, type, and flavor attribute settings of
855 the <QUALIFIER> element and the value of the qualifier).

856 – Any qualifiers defined in the superclass with a `TOSUBCLASS` attribute value of `false` shall
857 not be propagated to the subclass.

858 If ModifyClass is successful, the CIM server updates the specified class. The request to modify the class
859 shall fail if the server cannot consistently update any existing subclasses or instances of that class.

860 If ModifyClass is unsuccessful, this method shall return one of the following status codes, where the error
861 returned is the first applicable error in the list, starting with the first element and working down. Any
862 additional method-specific interpretation of the error is enclosed in parentheses.

863 • CIM_ERR_ACCESS_DENIED

864 • CIM_ERR_NOT_SUPPORTED

865 • CIM_ERR_INVALID_NAMESPACE

866 • CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise
867 incorrect parameters)

868 • CIM_ERR_NOT_FOUND (The CIM class does not exist.)

869 • CIM_ERR_INVALID_SUPERCLASS (The putative CIM class declares a non-existent or
870 incorrect superclass.)

871 • CIM_ERR_CLASS_HAS_CHILDREN (The modification could not be performed because the
872 subclasses of the class could not be updated consistently.)

- CIM_ERR_CLASS_HAS_INSTANCES (The modification could not be performed because the instances of the class could not be updated consistently.)
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.8 ModifyInstance

The ModifyInstance operation modifies an existing CIM instance in the target namespace. The instance shall already exist:

```
ModifyInstance
void ModifyInstance (
   [IN] <namedInstance> ModifiedInstance,
   [IN, OPTIONAL] boolean IncludeQualifiers = true,  (DEPRECATED)
   [IN, OPTIONAL, NULL] string propertyList[] = NULL
)
```

The `ModifiedInstance` input parameter identifies the name of the instance to be modified and provides the new property values.

DEPRECATION NOTE: Use of the `IncludeQualifiers` parameter is DEPRECATED**,** and it may be removed in a future version of this specification. The behavior of the `IncludeQualifiers` parameter is not specified. A CIM client cannot rely on `IncludeQualifiers` to have any impact on the operation. It is recommended that the CIM server ignore any qualifiers included in ModifiedInstance. If the `IncludeQualifiers` input parameter is `true`, the qualifiers are modified as specified in ModifiedInstance. If the parameter is `false`, qualifiers in ModifiedInstance are ignored and no qualifiers are explicitly modified.

The set of properties to be modified shall be determined as follows:

If the PropertyList input parameter is not NULL, the members of the array define one or more property names. Only properties specified in the PropertyList are modified. Properties of the ModifiedInstance that are missing from the PropertyList are ignored. If the PropertyList is an empty array, no properties are modified. If the PropertyList is NULL, the set of properties to be modified consists of those of ModifiedInstance with values different from the current values in the instance to be modified.

If the PropertyList contains duplicate elements, the server shall ignore them but otherwise process the request normally. If the PropertyList contains elements that are invalid property names for the instance to be modified, the server shall reject the request. If a property to be modified as previously defined cannot be modified because it is a key property, non-writable, or cannot be modified at this time for any other reason, the server shall reject the request. Non-writable properties are those defined to be non-writable in the schema implemented for the creation class of the instance to be modified. The value of the WRITE qualifier as defined in the schema for the creation class (or any of its superclasses) has no effect on the behavior of ModifyInstance.

In modifying the instance, the CIM server shall conform to the following rules and ensure their application:

- The server shall ignore any CLASSORIGIN and PROPAGATED attributes in the ModifiedInstance.
- The class shall exist, and the client shall not change its name in the instance to be modified.
- **DEPRECATED.** Any qualifiers in the instance not defined in the class are created as new elements of the instance if `IncludeQualifiers` is true.
- All properties of the instance to be modified preserve their CLASSORIGIN attribute value from that defined in the class.

- **DEPRECATED.** Any qualifier previously defined in the instance to be modified but not defined in the class, and which is not present in the ModifiedInstance parameter, is removed from the instance if `IncludeQualifiers` is true.
- If a property is to be modified as previously defined, its value in the ModifiedInstance (including NULL) becomes its new value in the instance to be modified.
- If a property is not specified in the ModifiedInstance but is specified in the PropertyList, then the class-defined default value (or NULL if none is defined) becomes its new value in the instance to be modified.
- **DEPRECATION NOTE:** The use of the TOINSTANCE qualifier attribute is DEPRECATED**.** Servers may choose to ignore TOINSTANCE. Servers that do not ignore TOINSTANCE shall interpret it so that any qualifiers defined in the class with a TOINSTANCE attribute value of `true` appear in the instance. A propagated qualifier cannot be removed from an instance. qualifiers in the class with a TOINSTANCE attribute value of `false` shall not be propagated to the instance
- **DEPRECATED.** Any qualifier propagated from the class cannot be modified in the instance if the OVERRIDABLE attribute of that qualifier is set to `false` in the class. It is a client error to specify such a qualifier in ModifiedInstance with a definition different than that in the class (where definition encompasses the name, type, and flavor attribute settings of the <QUALIFIER> element and the value of the qualifier).

If ModifyInstance is successful, all properties to be modified are updated in the specified instance.

If ModifyInstance is unsuccessful, the specified Instance is not updated, and the method shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED (by the CIM server for this operation)
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters and invalid properties to be modified)
- CIM_ERR_INVALID_CLASS (The CIM class of the instance to be modified does not exist.)
- CIM_ERR_NOT_SUPPORTED (This operation is not supported for the class of the specified instance, if provided.)
- CIM_ERR_NOT_FOUND (The CIM instance to be modified does not exist.)
- CIM_ERR_FAILED (This operation is not supported for the specified instance or some other unspecified error occurred, including a request for non-writable properties to be modified or a property that cannot be modified at this time.)

5.3.2.9 EnumerateClasses

The EnumerateClasses operation enumerates subclasses of a CIM class in the target namespace:

```
EnumerateClasses
<class>*EnumerateClasses (
    [IN,OPTIONAL,NULL] <className> ClassName=NULL,
    [IN,OPTIONAL] boolean DeepInheritance = false,
    [IN,OPTIONAL] boolean LocalOnly = true,
    [IN,OPTIONAL] boolean IncludeQualifiers = true,
    [IN,OPTIONAL] boolean IncludeClassOrigin = false
)
```

The `ClassName` input parameter defines the class that is the basis for the enumeration.

If the `DeepInheritance` input parameter is `true`, all subclasses of the specified class should be returned. If the `ClassName` input parameter is absent, this implies that all classes in the target namespace should be returned. If `DeepInheritance` is `false`, only immediate child subclasses are returned. If the `ClassName` input parameter is NULL, this implies that all base classes in the target namespace should be returned. This definition of `DeepInheritance` applies only to the EnumerateClasses and EnumerateClassName operations.

If the `LocalOnly` input parameter is `true`, any CIM elements (properties, methods, and qualifiers) except those added or overridden in the class as specified in the classname input parameter shall not be included in the returned class. If it is `false`, this parameter defines no additional filtering.

If the `IncludeQualifiers` input parameter is `true`, all qualifiers for each class (including qualifiers on the class and on any returned properties, methods, or method parameters) shall be included as `<QUALIFIER>` elements in the response. If it is `false`, no `<QUALIFIER>` elements are present.

If the `IncludeClassOrigin` input parameter is `true`, the `CLASSORIGIN` attribute shall be present on all appropriate elements in each returned class. If it is `false`, no `CLASSORIGIN` attributes are present.

If EnumerateClasses is successful, the method returns zero or more classes that meet the required criteria. These classes shall include all CIM elements (properties, methods, and qualifiers) defined in or inherited by each class, reduced by any elements excluded as a result of using the `LocalOnly` filter.

If EnumerateClasses is unsuccessful, this method shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional method-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_INVALID_CLASS (The CIM class for this enumeration does not exist.)
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.10 EnumerateClassNames

The EnumerateClassNames operation enumerates the names of subclasses of a CIM class in the target namespace:

```
EnumerateClassNames
<className>*EnumerateClassNames (
    [IN,OPTIONAL,NULL] <className> ClassName = NULL,
    [IN,OPTIONAL] boolean DeepInheritance = false
)
```

The `ClassName` input parameter defines the class that is the basis for the enumeration.

If the `DeepInheritance` input parameter is `true`, the names of all subclasses of the specified class should be returned. If the `ClassName` input parameter is absent, this implies that the names of all classes in the target namespace should be returned. If `DeepInheritance` is `false`, only the names of immediate child subclasses are returned. If the `ClassName` input parameter is NULL, this implies that the

names of all base classes in the target namespace should be returned. This definition of `DeepInheritance` applies only to the EnumerateClasses and EnumerateClassName operations.

If EnumerateClassNames is successful, the method returns zero or more names of classes that meet the requested criteria.

If EnumerateClassNames is unsuccessful, this method returns one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional method-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_INVALID_CLASS (The CIM class that is the basis for this enumeration does not exist.)
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.11 EnumerateInstances

The EnumerateInstances operation enumerates instances of a CIM class in the target namespace, including instances in the class and any subclasses in accordance with the polymorphic nature of CIM objects:

```
EnumerateInstances
<namedInstance>*EnumerateInstances (
    [IN] <className> ClassName,
    [IN,OPTIONAL] boolean LocalOnly = true,  (DEPRECATED)
    [IN,OPTIONAL] boolean DeepInheritance = true,
    [IN,OPTIONAL] boolean IncludeQualifiers = false,  (DEPRECATED)
    [IN,OPTIONAL] boolean IncludeClassOrigin = false,
    [IN,OPTIONAL,NULL] string PropertyList [] = NULL
)
```

The `ClassName` input parameter defines the class that is the basis for the enumeration.

DEPRECATION NOTE: With the 1.2 release of this specification, the `LocalOnly` parameter is DEPRECATED. LocalOnly filtering, as defined in 1.1, will not be supported in the next major revision of this specification. In the 1.1 version of this specification, the definition of the `LocalOnly` parameter was incorrectly modified. This change introduced a number of interoperability and backward compatibility problems for CIM clients using the `LocalOnly` parameter to filter the set of properties returned. The DMTF strongly recommends that CIM clients set `LocalOnly` to `false` and do not use this parameter to filter the set of properties returned. To minimize the impact of this recommendation on CIM clients, a CIM server may choose to treat the value of the `LocalOnly` parameter as `false` for all requests. A CIM server shall consistently support a single interpretation of the `LocalOnly` parameter. Refer to ANNEX B for details.

If the `DeepInheritance` input parameter is `false`, each returned instance shall not include any properties added by subclasses of the specified class. If it is `true`, no additional filtering is defined.

DEPRECATION NOTE: The use of the `IncludeQualifiers` parameter is DEPRECATED and it may be removed in a future version of this specification. The definition of `IncludeQualifiers` is ambiguous and when this parameter is set to `true`, CIM clients cannot be assured that any qualifiers will be

returned. A CIM client should always set this parameter to `false`. To minimize the impact of this recommendation on CIM clients, a CIM server may choose to treat the value of `IncludeQualifiers` as `false` for all requests. The preferred behavior is to use the class operations to receive qualifier information and not depend on any qualifiers in this response. If the `IncludeQualifiers` input parameter is `true`, all qualifiers for the instance, (including qualifiers on the instance and on any returned properties, shall be included as <QUALIFIER> elements in the response. If it is `false`, no <QUALIFIER> elements are present in the returned instance.

If the `IncludeClassOrigin` input parameter is `true`, the `CLASSORIGIN` attribute shall be present on all appropriate elements in each returned Instance. If it is `false`, no `CLASSORIGIN` attributes are present.

If the `PropertyList` input parameter is not NULL, the members of the array define one or more property names of the designated class. This definition may include inherited property names or property names explicitly defined in the designated class. However, it may not include property names added in subclasses of the designated class. Each returned instance shall not include elements for any properties missing from this list. Note that `PropertyList` acts as an additional filter on the properties defined by the `LocalOnly` and `DeepInheritance` input parameters; if the PropertyList includes a property that is not in the set defined by the `LocalOnly` and `DeepInheritance` combination, the element for the property shall not be returned. If `PropertyList` is an empty array, no elements for properties are included in the returned instances. If `PropertyList` is NULL, no additional filtering is defined.

If the `PropertyList` contains duplicate elements, the server shall ignore the duplicates but otherwise process the request normally. If the `PropertyList` contains elements which are invalid Property names for any target instance, the server shall ignore such entries but otherwise process the request normally.

Properties with the NULL value may be omitted from the response, even if the CIM client has not requested the exclusion of the property through the `LocalOnly`, `DeepInheritance`, or `PropertyList` filters. The CIM client shall interpret such omitted properties as NULL. Note that the CIM client cannot make any assumptions about properties omitted as a result of using any `LocalOnly`, `DeepInheritance`, or `PropertyList` filters.

If EnumerateInstances is successful, the method returns zero or more named instances that meet the required criteria. These instances shall have all properties defined in and inherited by their respective classes, reduced by any properties excluded as a result of using the `LocalOnly`, `DeepInheritance`, or `PropertyList` filters and further reduced by any NULL-valued properties omitted from the response.

If EnumerateInstances is unsuccessful, this method shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional method-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED (by the CIM server for this operation)
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_INVALID_CLASS (The CIM class that is the basis for this enumeration does not exist.)
- CIM_ERR_NOT_SUPPORTED (This operation is not supported for the specified class and all of its subclasses, if provided.)
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.12 EnumerateInstanceNames

The EnumerateInstanceNames operation enumerates the names (model paths) of the instances of a CIM class in the target namespace, including instances in the class and any subclasses in accordance with the polymorphic nature of CIM objects:

```
EnumerateInstanceNames
<instanceName>*EnumerateInstanceNames (
    [IN] <className> ClassName
)
```

The `ClassName` input parameter defines the class that is the basis for the enumeration.

If EnumerateInstanceNames is successful, the method returns zero or more InstanceNames (referred to in DSP0004 as a model path) that meet the requested criteria. The `InstanceName` shall specify the class from which the instance is instantiated, not any of its base classes. Note that this class may be different from the class specified as input.

If EnumerateInstanceNames is unsuccessful, this method shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional method-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED (by the CIM server for this operation)
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_INVALID_CLASS (The CIM class that is the basis for this enumeration does not exist.)
- CIM_ERR_NOT_SUPPORTED (This operation is not supported for the specified class and all of its subclasses, if provided.)
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.13 ExecQuery

The ExecQuery operation executes a query against the target namespace:

```
ExecQuery
<object>*ExecQuery (
    [IN] string QueryLanguage,
    [IN] string Query
)
```

The `QueryLanguage` input parameter defines the query language in which the query parameter is expressed.

The `Query` input parameter defines the query to be executed. The results of the query shall be constrained to contain only CIM classes that exist in the target namespace or CIM instances whose classes exist in the target namespace. Note that any instances in the result set may or may not exist in any namespace. Note that for query languages supporting select-lists and from-clauses, this implies that all select-list entries resolve to disjoint properties exposed by one CIM class named in the from-clause. This rule does not prevent such queries from using joins.

Neither the query language nor the format of the query is defined by this specification. It is anticipated that query languages will be submitted to the DMTF as separate proposals.

CIM servers can declare which query languages they support (if any) using a mechanism defined in 7.5.

If ExecQuery is successful, the method returns zero or more CIM classes or instances that correspond to the results of the query.

If ExecQuery is unsuccessful, the method shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional method-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_QUERY_LANGUAGE_NOT_SUPPORTED (The requested query language is not recognized.)
- CIM_ERR_INVALID_QUERY (The query is not a valid query in the specified query language.)
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.14 Associators

The Associators operation enumerates CIM objects (classes or instances) associated with a particular source CIM object:

```
Associators
<objectWithPath>*Associators (
    [IN] <objectName> ObjectName,
    [IN,OPTIONAL,NULL] <className> AssocClass = NULL,
    [IN,OPTIONAL,NULL] <className> ResultClass = NULL,
    [IN,OPTIONAL,NULL] string Role = NULL,
    [IN,OPTIONAL,NULL] string ResultRole = NULL,
    [IN,OPTIONAL] boolean IncludeQualifiers = false, (DEPRECATED)
    [IN,OPTIONAL] boolean IncludeClassOrigin = false,
    [IN,OPTIONAL,NULL] string PropertyList [] = NULL
)
```

The `ObjectName` input parameter defines the source CIM object whose associated objects are to be returned. This may be either a class name or instance name (model path).

The `AssocClass` input parameter, if not `NULL`, shall be a valid CIM association class name. It acts as a filter on the returned set of objects by mandating that each returned object shall be associated to the source object through an instance of this class or one of its subclasses.

The `ResultClass` input parameter, if not `NULL`, shall be a valid CIM class name. It acts as a filter on the returned set of objects by mandating that each returned object shall be either an instance of this class (or one of its subclasses) or be this class (or one of its subclasses).

The `Role` input parameter, if not `NULL`, shall be a valid property name. It acts as a filter on the returned set of objects by mandating that each returned object shall be associated with the source object through an association in which the source object plays the specified role. That is, the name of the property in the association class that refers to the source object shall match the value of this parameter.

The `ResultRole` input parameter, if not `NULL`, shall be a valid property name. It acts as a filter on the returned set of objects by mandating that each returned object shall be associated to the source object

through an association in which the returned object plays the specified role. That is, the name of the property in the association class that refers to the returned object shall match the value of this parameter.

DEPRECATION NOTE: The use of the `IncludeQualifiers` parameter is DEPRECATED and it may be removed in a future version of this specification. The preferred behavior is to use the class operations to receive qualifier information and not depend on any qualifiers in this response. If `IncludeQualifiers` is `true`, all qualifiers for each object (including qualifiers on the object and on any returned properties) shall be included as <QUALIFIER> elements in the response. If it is `false`, no <QUALIFIER> elements are present.

If the `IncludeClassOrigin` input parameter is `true`, the `CLASSORIGIN` attribute shall be present on all appropriate elements in each returned object. If it is `false`, no `CLASSORIGIN` attributes are present.

If the `PropertyList` input parameter is not NULL, the members of the array define one or more property names. Each returned object shall not include elements for any properties missing from this list. If `PropertyList` is an empty array, no properties are included in each returned object. If it is NULL, no additional filtering is defined.

If `PropertyList` contains duplicate elements, the server shall ignore them but otherwise process the request normally. If `PropertyList` contains elements that are invalid property names for any target object, the server shall ignore such entries but otherwise process the request normally.

Clients should not explicitly specify properties in the `PropertyList` parameter unless they specify a non-NULL value for the `ResultClass` parameter.

If instances are returned, properties with the NULL value may be omitted from the response, even if the CIM client has not requested the exclusion of the through the `PropertyList` filter. The CIM client shall interpret such omitted properties as NULL. Note that the CIM client cannot make any assumptions about properties omitted as a result of using the `PropertyList` filter. If classes are returned, the CIM server cannot make this choice, and only the CIM client can cause properties to be excluded by using the `PropertyList` filter.

If Associators is successful, the method returns zero or more CIM classes or instances meeting the requested criteria. Because it is possible for CIM objects from different hosts or namespaces to be associated, each returned object includes location information. If the ObjectName refers to a class, then classes are returned. These classes shall have all CIM elements (properties, methods, and qualifiers) defined in and inherited by that class, reduced by any properties excluded as a result of using the `PropertyList` filter. If the ObjectName refers to an instance, then instances are returned. These instances shall have all properties defined in and inherited by its class, reduced by any properties excluded as a result of using the `PropertyList` filter and further reduced by any NULL valued properties omitted from the response.

If Associators is unsuccessful, this method shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional method-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED (by the CIM server for this operation)
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_NOT_SUPPORTED (This operation is not supported for the class of the specified instance, if provided.)

- CIM_ERR_FAILED (This operation is not supported for the specified instance, or some other unspecified error occurred.)

5.3.2.15 AssociatorNames

The AssociatorNames operation enumerates the names of CIM Objects (classes or instances) that are associated with a particular source CIM object:

```
AssociatorNames
<objectPath>*AssociatorNames (
    [IN] <objectName> ObjectName,
    [IN,OPTIONAL,NULL] <className> AssocClass = NULL,
    [IN,OPTIONAL,NULL] <className> ResultClass = NULL,
    [IN,OPTIONAL,NULL] string Role = NULL,
    [IN,OPTIONAL,NULL] string ResultRole = NULL
)
```

The `ObjectName` input parameter defines the source CIM object whose associated names are to be returned. This is either a class or instance name (model path).

The `AssocClass` input parameter, if not `NULL`, shall be a valid CIM association class name. It acts as a filter on the returned set of names by mandating that each returned name identify an object that shall be associated to the source object through an instance of this class or one of its subclasses.

The `ResultClass` input parameter, if not `NULL`, shall be a valid CIM class name. It acts as a filter on the returned set of names by mandating that each returned name identify an object that shall be either an instance of this class (or one of its subclasses) or be this class (or one of its subclasses).

The `Role` input parameter, if not `NULL`, shall be a valid property name. It acts as a filter on the returned set of names by mandating that each returned name identify an object that shall be associated to the source object through an association in which the source object plays the specified role. That is, the name of the property in the association class that refers to the source object shall match the value of this parameter.

The `ResultRole` input parameter, if not `NULL`, shall be a valid property name. It acts as a filter on the returned set of names by mandating that each returned name identify an object that shall be associated to the source object through an association in which the named returned object plays the specified role. That is, the name of the property in the association class that refers to the returned object shall match the value of this parameter.

If AssociatorNames is successful, the method returns zero or more CIM class paths or instance paths meeting the requested criteria. Because CIM objects from different hosts or namespaces can be associated, each returned object includes location information. If the ObjectName refers to a class path, then class paths are returned. Otherwise, the ObjectName refers to an instance path, and instance paths are returned.

If AssociatorNames is unsuccessful, one of the following status codes shall be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED (by the CIM server for this operation)
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)

- CIM_ERR_NOT_SUPPORTED (This operation is not supported for the class of the specified instance, if provided.)
- CIM_ERR_FAILED (This operation is not supported for the specified instance, or some other unspecified error occurred.)

5.3.2.16 References

The References operation enumerates the association objects that refer to a particular target CIM object (class or instance).

```
References
<objectWithPath>*References (
    [IN] <objectName> ObjectName,
    [IN,OPTIONAL,NULL] <className> ResultClass = NULL,
    [IN,OPTIONAL,NULL] string Role = NULL,
    [IN,OPTIONAL] boolean IncludeQualifiers = false,  (DEPRECATED)
    [IN,OPTIONAL] boolean IncludeClassOrigin = false,
    [IN,OPTIONAL,NULL] string PropertyList [] = NULL
)
```

The `ObjectName` input parameter defines the target CIM object whose referring objects are to be returned. This is either a class or instance name (model path).

The `ResultClass` input parameter, if not `NULL`, shall be a valid CIM class name. It acts as a filter on the returned set of objects by mandating that each returned object shall be an instance of this class (or one of its subclasses) or this class (or one of its subclasses).

The `Role` input parameter, if not `NULL`, shall be a valid property name. It acts as a filter on the returned set of objects by mandating that each returned object shall refer to the target object through a property with a name that matches the value of this parameter.

DEPRECATION NOTE: The use of the `IncludeQualifiers` parameter is DEPRECATED and it may be removed in a future version of this specification. The preferred behavior is to use the class operations to receive qualifier information and not depend on any qualifiers in this response. If `IncludeQualifiers` is `true`, all qualifiers for each object (including qualifiers on the object and on any returned properties) shall be included as <QUALIFIER> elements in the response. If this parameter is `false`, no <QUALIFIER> elements are present in each returned Object.

If the `IncludeClassOrigin` input parameter is `true`, the `CLASSORIGIN` attribute shall be present on all appropriate elements in each returned object. If it is `false`, no `CLASSORIGIN` attributes are present.

If the `PropertyList` input parameter is not NULL, the members of the array define one or more property names. Each returned object shall not include elements for any properties missing from this list. If `PropertyList` is an empty array, no properties are included in each returned object. If `PropertyList` is NULL, no additional filtering is defined.

If `PropertyList` contains duplicate elements, the server shall ignore them but otherwise process the request normally. If `PropertyList` contains elements that are invalid property names for any target object, the server shall ignore them but otherwise process the request normally.

Clients should not explicitly specify properties in the `PropertyList` parameter unless they specify a non-NULL value for the `ResultClass` parameter.

If instances are returned, properties with the NULL value may be omitted from the response, even if the CIM client has not requested the exclusion of the property through the `PropertyList` filter. The CIM client must interpret such omitted properties as NULL. Note that the CIM client cannot make any

assumptions about properties omitted as a result of using the `PropertyList` filter. If classes are returned, the CIM server cannot make this choice, and only the CIM client can cause properties to be excluded by using the `PropertyList` filter.

If References is successful, the method returns zero or more CIM classes or instances meeting the requested criteria. Because CIM objects from different hosts or namespaces can be associated, each returned object includes location information. If the ObjectName refers to a class, then classes are returned. These classes shall have all CIM elements (properties, methods, and qualifiers) defined in and inherited by that class, reduced by any properties excluded as a result of using the `PropertyList` filter. If the ObjectName refers to an instance, then instances are returned. These instances shall have all properties defined in and inherited by their respective classes, reduced by any properties excluded as a result of using the `PropertyList` filter and further reduced by any NULL valued properties omitted from the response.

If References is unsuccessful, this method shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional method-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED (by the CIM server for this operation)
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_NOT_SUPPORTED (This operation is not supported for the class of the specified instance, if provided.)
- CIM_ERR_FAILED (This operation is not supported for the specified instance, or some other unspecified error occurred.)

5.3.2.17 ReferenceNames

The ReferenceNames operation enumerates the association objects that refer to a particular target CIM object (class or instance):

```
ReferenceNames
<objectPath>*ReferenceNames (
    [IN] <objectName> ObjectName,
    [IN,OPTIONAL,NULL] <className> ResultClass = NULL,
    [IN,OPTIONAL,NULL] string Role = NULL
)
```

The `ObjectName` input parameter defines the target CIM object with the referring object names to be returned. It may be either a class or an instance name (model path).

The `ResultClass` input parameter, if not `NULL`, shall be a valid CIM class name. It acts as a filter on the returned set of object names by mandating that each returned Object Name identify an instance of this class (or one of its subclasses) or this class (or one of its subclasses).

The `Role` input parameter, if not `NULL`, shall be a valid property name. It acts as a filter on the returned set of object names by mandating that each returned object name shall identify an object that refers to the target instance through a property with a name that matches the value of this parameter.

If ReferenceNames is successful, the method returns zero or more CIM class paths or instance paths meeting the requested criteria. Because CIM objects from different hosts or namespaces can be associated, each returned object includes location information. If the ObjectName refers to a class path,

then class paths are returned. Otherwise, the ObjectName refers to an instance path, and instance paths are returned.

If ReferenceNames is unsuccessful, this method shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional method-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED (by the CIM server for this operation)
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_NOT_SUPPORTED (This operation is not supported for the class of the specified instance, if provided.)
- CIM_ERR_FAILED (This operation is not supported for the specified instance, or some other unspecified error occurred.)

5.3.2.18 GetProperty

The GetProperty operation retrieves a single property value from a CIM instance in the target namespace:

```
GetProperty
<propertyValue>GetProperty (
    [IN] <instanceName> InstanceName,
    [IN] string PropertyName
)
```

The `InstanceName` input parameter specifies the name of the instance (model path) from which the property value is requested.

The `PropertyName` input parameter specifies the name of the property with the value to be returned.

If GetProperty is successful, the return value specifies the value of the requested property. If the value is NULL, no element is returned.

If GetProperty is unsuccessful, this method shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional method-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_INVALID_CLASS (The CIM class does not exist in the specified namespace.)
- CIM_ERR_NOT_FOUND (The CIM class exists, but the requested CIM instance does not exist in the specified namespace.)
- CIM_ERR_NO_SUCH_PROPERTY (The CIM instance exists, but the requested property does not.)
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.19 SetProperty

The SetProperty operation sets a single property value in a CIM instance in the target namespace:

```
SetProperty
void SetProperty (
   [IN] <instanceName> InstanceName,
   [IN] string PropertyName,
   [IN,OPTIONAL,NULL] <propertyValue> NewValue = NULL
```

The `InstanceName` input parameter specifies the name of the instance (model path) with the property value to be updated.

The `PropertyName` input parameter specifies the name of the property with the value to be updated.

The `NewValue` input parameter specifies the new value for the property (which may be NULL).

If SetProperty is unsuccessful, this method shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional method-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED (by the CIM server for this operation)
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_INVALID_CLASS (The CIM class does not exist in the specified namespace.)
- CIM_ERR_NOT_FOUND (The CIM class exists, but the requested CIM instance does not exist in the specified namespace.)
- CIM_ERR_NOT_SUPPORTED (This operation is not supported for the class of the specified instance, if provided.)
- CIM_ERR_NO_SUCH_PROPERTY (The CIM instance exists, but the requested property does not.)
- CIM_ERR_TYPE_MISMATCH (The supplied value is incompatible with the type of the property.)
- CIM_ERR_FAILED (This operation is not supported for the specified instance, or some other unspecified error occurred.)

5.3.2.20 GetQualifier

The GetQualifier operation retrieves a single qualifier declaration from the target namespace.

```
GetQualifier
<qualifierDecl>GetQualifier (
   [IN] string QualifierName
)
```

The `QualifierName` input parameter identifies the qualifier with the declaration to be retrieved.

If GetQualifier is successful, the method returns the qualifier declaration for the named qualifier.

If GetQualifier is unsuccessful, this method shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional method-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_NOT_FOUND (The requested qualifier declaration does not exist.)
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.21 SetQualifier

The SetQualifier operation creates or updates a single qualifier declaration in the target namespace. If the qualifier declaration already exists, it is overwritten:

```
SetQualifier
void SetQualifier (
    [IN] <qualifierDecl> QualifierDeclaration
)
```

The `QualifierDeclaration` input parameter defines the qualifier declaration to add to the namespace.

If SetQualifier is successful, the qualifier declaration is added to the target namespace. If a qualifier declaration with the same qualifier name already exists, the new declaration replaces it.

If SetQualifier is unsuccessful, this method returns one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional method-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.22 DeleteQualifier

The DeleteQualifier operation deletes a single qualifier declaration from the target namespace.

```
DeleteQualifier
void DeleteQualifier (
    [IN] string QualifierName
)
```

The `QualifierName` input parameter identifies the qualifier with the declaration to be deleted.

If DeleteQualifier is successful, the specified qualifier declaration is deleted from the namespace.

If DeleteQualifier is unsuccessful, this method shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional method-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_NOT_FOUND (The requested qualifier declaration does not exist.)
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.23 EnumerateQualifiers

The EnumerateQualifiers operation enumerates qualifier declarations from the target namespace.

```
EnumerateQualifiers
<qualifierDecl>*EnumerateQualifiers (
)
```

If EnumerateQualifiers is successful, the method returns zero or more qualifier declarations.

If EnumerateQualifiers is unsuccessful, this method shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional method-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.24 Pulled Enumeration Operations

This clause defines a set of operations that return CIM instances or instance paths in portions controlled by the CIM client. These operations are called *pulled enumerations*. Usually, an enumeration session is established through an Open operation, and subsequent repeated executions of a Pull operation on the enumeration session are used to retrieve them. Optionally, the Open operation can also pull a first set of items.

Pulled enumeration operations consist of the following individual operations:

- Open operations open an enumeration of the following elements:
  - OpenEnumerateInstances (instances of a class)
  - OpenEnumerateInstancePaths (instance paths of instances of a class)
  - OpenReferenceInstances (association instances referencing a target instance)
  - OpenReferenceInstancePaths (the instance paths of association instances referencing a target instance)
  - OpenAssociatorInstances (instances associated with a source instance)

1507 – OpenAssociatorInstancePaths (the instance paths of instances associated to a source
1508 instance)

1509 – OpenQueryInstances (the rows resulting from a query)

1510 • Pull operations are for the following cases:

1511 – PullInstances (Instances are enumerated, and instance paths are either not available, for
1512 example as in for OpenQueryInstances, or not desired.)

1513 – PullInstancesWithPath (Instances with paths are enumerated.)

1514 – PullInstancePaths (Instance paths are enumerated.)

1515 • Other operations are as follows:

1516 – CloseEnumeration (closes an open enumeration)

1517 – EnumerationCount (estimates the number of items in an open enumeration)

1518 **5.3.2.24.1 Behavioral Rules for Pulled Enumeration Operations**

1519 A central concept of pulled enumeration operations is the "enumeration session," which provides a
1520 context in which the operations perform their work and which determines the set of elements to be
1521 returned. To process the operations of an enumeration session, some parameters of the Open operation
1522 need to be maintained as long as the enumeration session is open. In addition, some state data about
1523 where the enumeration session is with regard to elements already returned must be maintained.

1524 From a CIM client perspective, an enumeration session is an enumeration context value. A successful
1525 Open operation establishes the enumeration session and returns an enumeration context value
1526 representing it. This value is used as an input/output parameter in subsequent Pull operations on that
1527 enumeration session. The enumeration context value shall uniquely identify the open enumeration
1528 session within the target CIM namespace of the Open operation that established the enumeration
1529 session. It is valid for a CIM server to use NULL as an enumeration context value representing a closed
1530 enumeration session, but a CIM client shall not rely on that.

1531 Defining the enumeration context value in Pull operations as both an input parameter and an output
1532 parameter allows the CIM server to change the enumeration context value during the execution of a pull
1533 operation. This ability to change allows different implementation approaches on the CIM server side,
1534 which are transparent for the CIM client. Example approaches are as follows:

1535 • Maintain any state data describing the enumeration session internally in the CIM server. The
1536 enumeration context value does not need to change in subsequent Pull operations. The CIM
1537 server uses this value only to identify the internal state data for the open enumeration session. It
1538 does not use the value to store any state data. A variation of this approach is to hand back
1539 modified enumeration context values for additional CIM server-side sequence checking.

1540 • Maintain any state data describing the enumeration session only on the CIM client side. All state
1541 data is stored in the enumeration context value, and the CIM server does not maintain any state
1542 data about the enumeration session, essentially being completely stateless with regard to the
1543 enumeration session.

1544 • A combination of the two previous approaches.

1545 A CIM server may support keeping enumeration sessions open across connection terminations and
1546 shutdowns of the server. Elements may be created, deleted, or modified concurrently with an
1547 enumeration session that involves these elements. Such changes may or may not be reflected in the
1548 enumeration set. Therefore, there is no guarantee to the CIM client that the enumeration set represents a
1549 consistent snapshot of its elements at a point in time. However, the CIM server should make a best effort
1550 attempt for the returned enumeration set to represent a consistent snapshot of its elements at a point in
1551 time. The order of elements in the enumeration set is undefined.

This specification does not restrict the number of enumeration sessions that can be established or executed concurrently in the same CIM server or client. This remains true even if the enumeration sets of such concurrently established enumeration sessions contain the same elements.

Except for CloseEnumeration, all operations on a particular enumeration session shall be executed sequentially. An enumeration session can be open or closed. It is considered open if operations using its enumeration context value as an input parameter can be executed successfully. It is opened by the successful completion of an Open operation and closed by one of the following events:

- Successful completion of a CloseEnumeration operation
- Successful completion of an open or pull operation with the EndOfSequence output parameter set to `true`
- Unsuccessful completion of a pull operation when ContinueOnError is not requested
- CIM server-side decision to close the enumeration session based upon an operation timeout
- CIM server-side decision to close an enumeration session during an operation on that enumeration session based upon exceeding server limits

A conformant CIM server may support closure of enumeration sessions based upon exceeding server limits. Example situations for such a decision are:

- Pull operations with no objects requested that are repeated with a high frequency on the same enumeration session
- EnumerationCount operations repeated with a high frequency on the same enumeration session

A mechanism by which CIM servers can declare support for closure of enumeration sessions based upon exceeding server limits is defined in 7.5. If a CIM server supports such closure of enumeration sessions, it shall make the decision to close during an operation on that enumeration session. There is no way to indicate the reason for the closure if the decision is made elsewhere. If a CIM server closes an enumeration session based upon exceeding server limits, it shall return failure on the operation on that enumeration session with the status code CIM_ERR_SERVER_LIMITS_EXCEEDED.

5.3.2.24.2 Common Parameters for the Open Operations

This clause defines commonly used parameters for the Open operations. The description of the individual Open operations references these parameters as appropriate. Note that not every Open operation uses every one of these common parameters:

- EnumerationContext
  - This output parameter is the enumeration context value representing the enumeration session. If the EndOfSequence is true, the EnumerationContext value may be NULL.
- EndOfSequence
  - This output parameter indicates to the CIM client whether the enumeration session is exhausted. If EndOfSequence is `true` upon successful completion of an Open operation, no more elements are available and the CIM server closes the enumeration session, releasing any allocated resources related to the enumeration session. If the enumeration set is empty, it is valid for a CIM server to set EndOfSequence to `true`, even if MaxObjectCount is 0. In this case, the enumeration session is closed upon successful completion of the Open operation. If EndOfSequence is `false`, additional elements may be available and the CIM server shall not close the enumeration session.

- IncludeClassOrigin
  - This input parameter is used only on Open operations that enumerate CIM instances. It controls whether information about the class origin of properties, references or methods is included in any enumerated CIM instances. If IncludeClassOrigin is `true`, the CLASSORIGIN attribute shall be present on all appropriate elements in each CIM instance returned by any subsequent PullInstance operations on this enumeration session. If IncludeClassOrigin is `false`, any CLASSORIGIN attributes shall not be present in any enumerated instances.
- FilterQueryLanguage and FilterQuery
  - These input parameters act as an additional restricting filter on the set of enumerated elements (for example, instances or instance paths).
  - If FilterQueryLanguage is not NULL, it shall specify a query language and FilterQuery shall be a valid query in that query language. Neither the query language nor the format of the query is defined by this specification. It is anticipated that query languages will be submitted to the DMTF as separate proposals. A mechanism by which CIM servers can declare the query languages they support for filtering in Pulled enumerations (if any) is defined in 7.5.
  - The query specified in FilterQuery shall address (for example, in its FROM list) the class specified in ClassName and shall not address any other classes. The result set specified by the query (for example, SELECT list) shall be ignored. The query shall not define any ordering criteria or any grouping of objects.
  - If the CIM server does not support filtered enumerations and FilterQueryLanguage is not NULL, the CIM server shall return a failure with the status code CIM_ERR_FILTERED_ENUMERATION_NOT_SUPPORTED.
- OperationTimeout
  - This input parameter determines the minimum time the CIM server shall maintain the open enumeration session after the last Open or Pull operation (unless the enumeration session is closed during the last operation). If the operation timeout is exceeded, the CIM server may close the enumeration session at any time, releasing any resources allocated to the enumeration session.
  - An OperationTimeout of 0 means that there is no operation timeout. That is, the enumeration session is never closed based on time.
  - If OperationTimeout is NULL, the CIM server shall choose an operation timeout.
  - All other values for OperationTimeout specify the operation timeout in seconds.
  - A CIM server may restrict the set of allowable values for OperationTimeout. Specifically, the CIM server may not allow 0 (no timeout). If the specified value is not an allowable value, the CIM server shall return failure with the status code CIM_ERR_INVALID_OPERATION_TIMEOUT. A mechanism by which CIM servers can declare the allowable values for OperationTimeout is defined in 7.5.
- ContinueOnError
  - This input parameter, if `true`, requests a continuation on error, which is the ability to resume an enumeration session successfully after a Pull operation returns an error. A mechanism by which conformant CIM servers can declare support for continuation on error is defined in 7.5.
  - If a CIM server does not support continuation on error and ContinueOnError is `true`, it shall return a failure with the status code CIM_ERR_CONTINUATION_ON_ERROR_NOT_SUPPORTED.

  - If a CIM server supports continuation on error and ContinueOnError is `true`, the enumeration session shall remain open when a Pull operation fails, and any subsequent successful Pull operations shall return the set of elements that would have been returned if the failing Pull operations were successful. This behavior is subject to the consistency rules defined for pulled enumerations. If ContinueOnError is `false`, the enumeration session shall be closed when a Pull operation returns a failure.

- MaxObjectCount
  - This input parameter defines the maximum number of elements that this Open operation can return. Any uint32 number is valid, including 0. The CIM server may deliver any number of elements up to MaxObjectCount but shall not deliver more than MaxObjectCount elements. A conformant CIM server implementation may choose to never return any elements during an Open operation, regardless of the value of MaxObjectCount. Note that a CIM client can use a MaxObjectCount value of 0 to specify that it does not want to retrieve any instances in the Open operation.

- Return Value (array of enumerated elements)
  - The return value of a successful Open operation is an array of enumerated elements with a number of entries from 0 up to a maximum defined by MaxObjectCount. These entries meet the criteria defined in the Open operation. Note that returning no entries in the array does not imply that the enumeration session is exhausted. Only the EndOfSequence output parameter indicates whether the enumeration session is exhausted.

5.3.2.24.3 OpenEnumerateInstances

The OpenEnumerateInstances operation establishes and opens an enumeration session of the instances of a CIM class (including instances of its subclasses) in the target namespace. Optionally, it retrieves a first set of instances.

```
OpenEnumerateInstances
<instanceWithPath>* OpenEnumerateInstances (
   [OUT] <enumerationContext> EnumerationContext,
   [OUT] Boolean EndOfSequence,
   [IN] <className> ClassName,
   [IN,OPTIONAL] boolean DeepInheritance = true,
   [IN,OPTIONAL] boolean IncludeClassOrigin = false,
   [IN,OPTIONAL,NULL] string PropertyList [] = NULL,
   [IN,OPTIONAL,NULL] string FilterQueryLanguage = NULL,
   [IN,OPTIONAL,NULL] string FilterQuery = NULL,
   [IN,OPTIONAL,NULL] uint32 OperationTimeout = NULL,
   [IN,OPTIONAL] Boolean ContinueOnError = false,
   [IN,OPTIONAL] uint32 MaxObjectCount = 0
)
```

The OpenEnumerateInstances operation shall comply with the behavior defined in 5.3.2.24.1.

The `EnumerationContext` output parameter is defined in 5.3.2.24.2.

The `EndOfSequence` output parameter is defined in 5.3.2.24.2.

The `ClassName` input parameter defines the class that is the basis for the enumeration. The enumeration set shall consist of all instances of that specified class, including any instances of any of its subclasses, in accordance with the polymorphic nature of CIM objects.

The `DeepInheritance` input parameter acts as a filter on the properties included in any enumerated CIM instances. If the `DeepInheritance` input parameter is `true`, all properties of each enumerated

instance of the class shall be present (subject to constraints imposed by the other parameters), including any added by subclassing the specified class. If `DeepInheritance` is `false`, each enumerated instance includes only properties defined for the class specified by `ClassName`.

The `IncludeClassOrigin` input parameter is defined in 5.3.2.24.2.

The `PropertyList` input parameter acts as a filter on the properties in any enumerated CIM instances. If `PropertyList` is not NULL, the members of the array define zero or more property names of the specified class. This array may include inherited property names or property names explicitly defined in the specified class. However, it shall not include property names defined in subclasses of the specified class. Each enumerated instance shall not include elements for properties missing from this list. Note that `PropertyList` acts as an additional filter on the properties defined by the `DeepInheritance` input parameter. If `PropertyList` includes a property that is not in the set defined by `DeepInheritance`, the element for the property shall not be included. If `PropertyList` is an empty array, no elements for properties are included in the enumerated instances. If `PropertyList` is NULL, no additional filtering is defined. If `PropertyList` contains duplicate elements, the CIM server shall ignore them but otherwise process the request normally. If `PropertyList` contains elements that are invalid property names for any target instance, the CIM server shall ignore such entries but otherwise process the request normally.

The `FilterQueryLanguage` and `FilterQuery` input parameters are defined in 5.3.2.24.2.

The `OperationTimeout` input parameter is defined in 5.3.2.24.2.

The `ContinueOnError` input parameter is defined in 5.3.2.24.2.

The `MaxObjectCount` input parameter is defined in 5.3.2.24.2.

If OpenEnumerateInstances is successful, the return value shall be an array of enumerated instances as defined in 5.3.2.24.2.

The PullInstancesWithPath operation shall be used to pull instances for an enumeration session opened using OpenEnumerateInstances. If any other operation is used to pull instances, the CIM server shall return failure with the status code CIM_ERR_FAILED.

If OpenEnumerateInstances is unsuccessful, this operation shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional operation-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_SERVER_IS_SHUTTING_DOWN
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_OPERATION_TIMEOUT
- CIM_ERR_CONTINUATION_ON_ERROR_NOT_SUPPORTED
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_INVALID_CLASS (The CIM class that is the basis for this enumeration does not exist.)
- CIM_ERR_FILTERED_ENUMERATION_NOT_SUPPORTED
- CIM_ERR_QUERY_LANGUAGE_NOT_SUPPORTED (The requested filter query language is not recognized.)

- CIM_ERR_INVALID_QUERY (The filter query is not a valid query in the specified filter query language.)
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.24.4 OpenEnumerateInstancePaths

The OpenEnumerateInstancePaths operation establishes and opens an enumeration session of the instance paths of the instances of a CIM class (including instances of its subclasses) in the target namespace. Optionally, it retrieves a first set of instance paths:

```
OpenEnumerateInstancePaths
<instancePath>* OpenEnumerateInstancePaths (
    [OUT] <enumerationContext> EnumerationContext,
    [OUT] Boolean EndOfSequence,
    [IN] <className> ClassName,
    [IN,OPTIONAL,NULL] string FilterQueryLanguage = NULL,
    [IN,OPTIONAL,NULL] string FilterQuery = NULL,
    [IN,OPTIONAL,NULL] uint32 OperationTimeout = NULL,
    [IN,OPTIONAL] Boolean ContinueOnError = false,
    [IN,OPTIONAL] uint32 MaxObjectCount = 0
)
```

The OpenEnumerateInstancePaths operation shall comply with the behavior defined in 5.3.2.24.1.

The `EnumerationContext` output parameter is defined in 5.3.2.24.2.

The `EndOfSequence` output parameter is defined in 5.3.2.24.2.

The `ClassName` input parameter defines the class that is the basis for the enumeration. The enumeration set shall consist of the instance paths of all instances of the specified class, including any instances of any of its subclasses, in accordance with the polymorphic nature of CIM objects.

The `FilterQueryLanguage` and `FilterQuery` input parameters are defined in 5.3.2.24.2.

The `OperationTimeout` input parameter is defined in 5.3.2.24.2.

The `ContinueOnError` input parameter is defined in 5.3.2.24.2.

The `MaxObjectCount` input parameter is defined in 5.3.2.24.2.

If OpenEnumerateInstancePaths is successful, the return value shall be an array of enumerated instance paths as defined in 5.3.2.24.2.

The PullInstancePaths operation shall be used to pull instances for an enumeration session opened using OpenEnumerateInstancePaths. If any other operation is used to pull instances, the CIM server shall return failure with the status code CIM_ERR_FAILED.

If OpenEnumerateInstancePaths is unsuccessful, this operation shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional operation-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_SERVER_IS_SHUTTING_DOWN
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE

- CIM_ERR_INVALID_OPERATION_TIMEOUT
- CIM_ERR_CONTINUATION_ON_ERROR_NOT_SUPPORTED
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_INVALID_CLASS (The CIM class that is the basis for this enumeration does not exist.)
- CIM_ERR_FILTERED_ENUMERATION_NOT_SUPPORTED
- CIM_ERR_QUERY_LANGUAGE_NOT_SUPPORTED (The requested filter query language is not recognized.)
- CIM_ERR_INVALID_QUERY (The filter query is not a valid query in the specified filter query language.)
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.24.5 OpenReferenceInstances

The OpenReferenceInstances operation establishes and opens the enumeration session of association instances that refer to a particular target CIM instance in the target namespace. Optionally, it retrieves a first set of instances:

```
OpenReferenceInstances
<instanceWithPath>* OpenReferenceInstances (
    [OUT] <enumerationContext> EnumerationContext,
    [OUT] Boolean EndOfSequence,
    [IN] <instanceName> InstanceName,
    [IN,OPTIONAL,NULL] <className> ResultClass = NULL,
    [IN,OPTIONAL,NULL] string Role = NULL,
    [IN,OPTIONAL] boolean IncludeClassOrigin = false,
    [IN,OPTIONAL,NULL] string PropertyList [] = NULL,
    [IN,OPTIONAL,NULL] string FilterQueryLanguage = NULL,
    [IN,OPTIONAL,NULL] string FilterQuery = NULL,
    [IN,OPTIONAL,NULL] uint32 OperationTimeout = NULL,
    [IN,OPTIONAL] Boolean ContinueOnError = false,
    [IN,OPTIONAL] uint32 MaxObjectCount = 0
)
```

The OpenReferenceInstances operation shall comply with the behavior defined in 5.3.2.24.1.

The `EnumerationContext` output parameter is defined in 5.3.2.24.2.

The `EndOfSequence` output parameter is defined in 5.3.2.24.2.

The `InstanceName` input parameter specifies an instance name (model path) that identifies the target CIM instance with the referring association instances to be enumerated. Unless restricted by any of the filter parameters of this operation, the enumeration set shall consist of all association instances that reference the target instance.

The `ResultClass` input parameter, if not NULL, shall be a CIM class name. It acts as a filter on the enumerated set of instances by mandating that each enumerated instance shall be an instance of this class or one of its subclasses. The CIM server shall not return an error if the `ResultClass` input parameter value is an invalid class name or if the class does not exist in the target namespace,

The `Role` input parameter, if not `NULL`, shall be a property name. It acts as a filter on the enumerated set of instances by mandating that each enumerated instance shall refer to the target instance through a

property with a name that matches the value of this parameter. The CIM server shall not return an error if the `Role` input parameter value is an invalid property name or if the property does not exist,

The `IncludeClassOrigin` input parameter is defined in 5.3.2.24.2.

The `PropertyList` input parameter acts as a filter on the properties included in any enumerated CIM instances. If `PropertyList` is not `NULL`, the members of the array define zero or more property names. Each enumerated instance shall not include elements for any properties missing from this list. If `PropertyList` is an empty array, no properties are included in each enumerated instance. If `PropertyList` is `NULL`, all properties are included in each enumerated instance, subject to the conditions expressed by the other parameters. If `PropertyList` contains duplicate elements, the CIM server shall ignore them but otherwise process the request normally. If `PropertyList` contains elements that are invalid property names for any target instance, the CIM server shall ignore them but otherwise process the request normally. CIM clients should not specify properties in `PropertyList` unless they specify a non-NULL value for the `ResultClass` parameter.

The `FilterQueryLanguage` and `FilterQuery` input parameters are defined in 5.3.2.24.2.

The `OperationTimeout` input parameter is defined in 5.3.2.24.2.

The `ContinueOnError` input parameter is defined in 5.3.2.24.2.

The `MaxObjectCount` input parameter is defined in 5.3.2.24.2.

If OpenReferenceInstances is successful, the return value shall be an array of enumerated instances as defined in 5.3.2.24.2.

The PullInstancesWithPath operation shall be used to pull instances for an enumeration session opened using OpenReferenceInstances. If any other operation is used to pull instances, the CIM server shall return failure with the status code CIM_ERR_FAILED.

If OpenReferenceInstances is unsuccessful, this operation shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element of and working down. Any additional operation-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_SERVER_IS_SHUTTING_DOWN
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_OPERATION_TIMEOUT
- CIM_ERR_CONTINUATION_ON_ERROR_NOT_SUPPORTED
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters)
- CIM_ERR_NOT_FOUND (The target instance was not found.)
- CIM_ERR_FILTERED_ENUMERATION_NOT_SUPPORTED
- CIM_ERR_QUERY_LANGUAGE_NOT_SUPPORTED (The requested filter query language is not recognized.)
- CIM_ERR_INVALID_QUERY (The filter query is not a valid query in the specified filter query language.)
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.24.6 OpenReferenceInstancePaths

The OpenReferenceInstancePaths operation establishes and opens an enumeration session of the instance paths of the association instances that refer to a particular target CIM instance in the target namespace. Optionally, it retrieves a first set of instance paths.

```
OpenReferenceInstancePaths
<instancePath>* OpenReferenceInstancePaths (
    [OUT] <enumerationContext> EnumerationContext,
    [OUT] Boolean EndOfSequence,
    [IN] <instanceName> InstanceName,
    [IN,OPTIONAL,NULL] <className> ResultClass = NULL,
    [IN,OPTIONAL,NULL] string Role = NULL,
    [IN,OPTIONAL,NULL] string FilterQueryLanguage = NULL,
    [IN,OPTIONAL,NULL] string FilterQuery = NULL,
    [IN,OPTIONAL,NULL] uint32 OperationTimeout = NULL,
    [IN,OPTIONAL] Boolean ContinueOnError = false,
    [IN,OPTIONAL] uint32 MaxObjectCount = 0
)
```

The OpenReferenceInstancePaths operation shall comply with the behavior defined in 5.3.2.24.1.

The `EnumerationContext` output parameter is defined in 5.3.2.24.2.

The `EndOfSequence` output parameter is defined in 5.3.2.24.2.

The `InstanceName` input parameter specifies an instance name (model path) that identifies the target CIM instance with the referring association instances (respectively, their instance paths) to be enumerated. Unless restricted by any filter parameters of this operation, the enumeration set shall consist of the instance paths of all association instances that reference the target instance.

The `ResultClass` input parameter, if not NULL, shall be a CIM class name. It acts as a filter on the enumerated set of instance paths by mandating that each enumerated instance path shall identify an instance of this class or one of its subclasses. The CIM server shall not return an error if the `ResultClass` input parameter value is an invalid class name or if the class does not exist in the target namespace.

The `Role` input parameter, if not NULL, shall be a property name. It acts as a filter on the enumerated set of instance paths by mandating that each enumerated instance path shall identify an instance that refers to the target instance through a property with a name that matches the value of this parameter. The CIM server shall not return an error if the `Role` input parameter value is an invalid property name or if the property does not exist,

The `FilterQueryLanguage` and `FilterQuery` input parameters are defined in 5.3.2.24.2.

The `OperationTimeout` input parameter is defined in 5.3.2.24.2.

The `ContinueOnError` input parameter is defined in 5.3.2.24.2.

The `MaxObjectCount` input parameter is defined in 5.3.2.24.2.

If OpenReferenceInstancePaths is successful, the return value shall be an array of enumerated instance paths as defined in 5.3.2.24.2.

The PullInstancePaths operation shall be used to pull instances for an enumeration session opened using OpenReferenceInstancePaths. If any other operation is used to pull instances, the CIM server shall return failure with the status code CIM_ERR_FAILED.

If OpenReferenceInstancePaths is unsuccessful, this operation shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional operation-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_SERVER_IS_SHUTTING_DOWN
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_OPERATION_TIMEOUT
- CIM_ERR_CONTINUATION_ON_ERROR_NOT_SUPPORTED
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_NOT_FOUND (The target instance was not found.)
- CIM_ERR_FILTERED_ENUMERATION_NOT_SUPPORTED
- CIM_ERR_QUERY_LANGUAGE_NOT_SUPPORTED (The requested filter query language is not recognized.)
- CIM_ERR_INVALID_QUERY (The filter query is not a valid query in the specified filter query language.)
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.24.7 OpenAssociatorInstances

The OpenAssociatorInstances operation establishes and opens an enumeration session of the instances associated with a particular source CIM instance in the target namespace. Optionally, it retrieves a first set of instances.

```
OpenAssociatorInstances
<instanceWithPath>* OpenAssociatorInstances (
    [OUT] <enumerationContext> EnumerationContext,
    [OUT] Boolean EndOfSequence,
    [IN] <instanceName> InstanceName,
    [IN,OPTIONAL,NULL] <className> AssocClass = NULL,
    [IN,OPTIONAL,NULL] <className> ResultClass = NULL,
    [IN,OPTIONAL,NULL] string Role = NULL,
    [IN,OPTIONAL,NULL] string ResultRole = NULL,
    [IN,OPTIONAL] boolean IncludeClassOrigin = false,
    [IN,OPTIONAL,NULL] string PropertyList [] = NULL,
    [IN,OPTIONAL,NULL] string FilterQueryLanguage = NULL,
    [IN,OPTIONAL,NULL] string FilterQuery = NULL,
    [IN,OPTIONAL,NULL] uint32 OperationTimeout = NULL,
    [IN,OPTIONAL] Boolean ContinueOnError = false,
    [IN,OPTIONAL] uint32 MaxObjectCount = 0
)
```

The OpenAssociatorInstances operation shall comply with the behavior defined in 5.3.2.24.1.

The `EnumerationContext` output parameter is defined in 5.3.2.24.2.

The `EndOfSequence` output parameter is defined in 5.3.2.24.2.

The InstanceName input parameter specifies an instance name (model path) that identifies the source CIM instance with the associated instances to be enumerated. Unless restricted by any filter parameters of this operation, the enumeration set shall consist of all instances associated with the source instance.

The AssocClass input parameter, if not NULL, shall be a CIM association class name. It acts as a filter on the enumerated set of instances by mandating that each enumerated instance shall be associated with the source instance through an instance of this class or one of its subclasses. The CIM server shall not return an error if the AssocClass input parameter value is an invalid class name or if the class does not exist in the target namespace.

The ResultClass input parameter, if not NULL, must be a CIM class name. It acts as a filter on the enumerated set of instances by mandating that each enumerated instance shall be an instance of this class or one of its subclasses. The CIM server shall not return an error if the ResultClass input parameter value is an invalid class name or if the class does not exist in the target namespace.

The Role input parameter, if not NULL, shall be a property name. It acts as a filter on the enumerated set of instances by mandating that each enumerated instance shall be associated with the source instance through an association in which the source instance plays the specified role. That is, the name of the property in the association class that refers to the source instance shall match the value of this parameter. The CIM server shall not return an error if the Role input parameter value is an invalid property name or if the property does not exist.

The ResultRole input parameter, if not NULL, shall be a property name. It acts as a filter on the enumerated set of instances by mandating that each enumerated instance shall be associated with the source instance through an association in which the enumerated instance plays the specified role. That is, the name of the property in the association class that refers to the enumerated instance shall match the value of this parameter. The CIM server shall not return an error if the ResultRole input parameter value is an invalid property name or if the property does not exist.

The IncludeClassOrigin input parameter is defined in 5.3.2.24.2.

The PropertyList input parameter acts as a filter on the properties included in any enumerated CIM instances. If PropertyList is not NULL, the members of the array define zero or more property names. Each enumerated instance shall not include elements for any properties missing from this list. If PropertyList is an empty array, no properties are included in each enumerated instance. If PropertyList is NULL, all properties are included in each enumerated instance, subject to the conditions expressed by the other parameters. If PropertyList contains duplicate elements, the CIM server shall ignore them but otherwise process the request normally. If PropertyList contains elements that are invalid property names for any target instance, the CIM server shall ignore them but otherwise process the request normally. CIM clients should not specify properties in PropertyList unless they specify a non-NULL value for the ResultClass parameter.

The FilterQueryLanguage and FilterQuery input parameters are defined in 5.3.2.24.2.

The OperationTimeout input parameter is defined in 5.3.2.24.2.

The ContinueOnError input parameter is defined in 5.3.2.24.2.

The MaxObjectCount input parameter is defined in 5.3.2.24.2.

If OpenAssociatorInstances is successful, the return value shall be an array of enumerated instances as defined in 5.3.2.24.2.

The PullInstancesWithPath operation shall be used to pull instances for an enumeration session opened using OpenAssociatorInstances. If any other operation is used to pull instances, the CIM server shall return failure with the status code CIM_ERR_FAILED.

If OpenAssociatorInstances is unsuccessful, this operation shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional operation-specific interpretation of the error is given in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_SERVER_IS_SHUTTING_DOWN
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_OPERATION_TIMEOUT
- CIM_ERR_CONTINUATION_ON_ERROR_NOT_SUPPORTED
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_NOT_FOUND (The source instance was not found.)
- CIM_ERR_FILTERED_ENUMERATION_NOT_SUPPORTED
- CIM_ERR_QUERY_LANGUAGE_NOT_SUPPORTED (The requested filter query language is not recognized.)
- CIM_ERR_INVALID_QUERY (The filter query is not a valid query in the specified filter query language.)
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.24.8 OpenAssociatorInstancePaths

The OpenAssociatorInstancePaths operation establishes and opens an enumeration session of the instance paths of the instances associated with a particular source CIM instance in the target namespace. Optionally, it retrieves a first set of instance paths.

```
OpenAssociatorInstancePaths
<instancePath>* OpenAssociatorInstancePaths (
    [OUT] <enumerationContext>EnumerationContext,
    [OUT] Boolean EndOfSequence,
    [IN] <instanceName> InstanceName,
    [IN,OPTIONAL,NULL] <className>AssocClass= NULL,
    [IN,OPTIONAL,NULL] <className>ResultClass = NULL,
    [IN,OPTIONAL,NULL] string Role = NULL,
    [IN,OPTIONAL,NULL] string ResultRole = NULL,
    [IN,OPTIONAL,NULL] string FilterQueryLanguage = NULL,
    [IN,OPTIONAL,NULL] string FilterQuery = NULL,
    [IN,OPTIONAL,NULL] uint32 OperationTimeout = NULL,
    [IN,OPTIONAL] Boolean ContinueOnError = false,
    [IN,OPTIONAL] uint32 MaxObjectCount = 0
)
```

This operation shall comply with the behavior defined in 5.3.2.24.1.

The `EnumerationContext` output parameter is defined in 5.3.2.24.2.

The `EndOfSequence` output parameter is defined in 5.3.2.24.2.

The `InstanceName` input parameter specifies an instance name (model path) that identifies the source CIM instance with the associated instances (respectively, their instance paths) to be enumerated. Unless

restricted by any filter parameters of this operation, the enumeration set shall consist of the instance paths of all instances associated with the source instance.

The `AssocClass` input parameter, if not `NULL`, shall be a CIM association class name. It acts as a filter on the enumerated set of instance paths by mandating that each instance path identify an instance that shall be associated with the source instance through an instance of this class or one of its subclasses. The CIM server shall not return an error if the `AssocClass` input parameter value is an invalid class name or if the class does not exist in the target namespace.

The `ResultClass` input parameter, if not NULL, shall be a CIM class name. It acts as a filter on the enumerated set of instance paths by mandating that each instance path identify an instance that shall be an instance of this class or one of its subclasses. The CIM server shall not return an error if the `ResultClass` input parameter value is an invalid class name or if the class does not exist in the target namespace.

The `Role` input parameter, if not NULL, shall be a property name. It acts as a filter on the enumerated set of instance paths by mandating that each instance path identify an instance that shall be associated with the source instance through an association in which the source instance plays the specified role. That is, the name of the property in the association class that refers to the source instance shall match the value of this parameter. The CIM server shall not return an error if the `Role` input parameter value is an invalid property name or if the property does not exist.

The `ResultRole` input parameter, if not NULL, shall be a property name. It acts as a filter on the enumerated set of instance paths by mandating that each instance path identify an instance that shall be associated with the source instance through an association in which the instance identified by the enumerated instance path plays the specified role. That is, the name of the property in the association class that refers to the instance identified by the enumerated instance path shall match the value of this parameter. The CIM server shall not return an error if the `ResultRole` input parameter value is an invalid property name or if the property does not exist.

The `FilterQueryLanguage` and `FilterQuery` input parameters are defined in 5.3.2.24.2.

The `OperationTimeout` input parameter is defined in 5.3.2.24.2.

The `ContinueOnError` input parameter is defined in 5.3.2.24.2.

The `MaxObjectCount` input parameter is defined in 5.3.2.24.2.

If OpenAssociatorInstancePaths is successful, the return value shall be an array of enumerated instance paths as defined in 5.3.2.24.2.

The PullInstancePaths operation shall be used to pull instances for an enumeration session opened using OpenAssociatorInstancePaths. If any other operation is used to pull instances, the CIM server shall return failure with the status code CIM_ERR_FAILED.

If OpenAssociatorInstancePaths is unsuccessful, this operation shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional operation-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_SERVER_IS_SHUTTING_DOWN
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_OPERATION_TIMEOUT
- CIM_ERR_CONTINUATION_ON_ERROR_NOT_SUPPORTED

- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_NOT_FOUND (The source instance was not found.)
- CIM_ERR_FILTERED_ENUMERATION_NOT_SUPPORTED
- CIM_ERR_QUERY_LANGUAGE_NOT_SUPPORTED (The requested filter query language is not recognized.)
- CIM_ERR_INVALID_QUERY (The filter query is not a valid query in the specified filter language.)
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.24.9 Common Parameters for the Pull Operations

This clause defines commonly used parameters for the Pull operations. The description of the individual Pull operations references these parameters as appropriate. Note that not every Pull operation uses every one of these common parameters.

- EnumerationContext
  - This parameter is the enumeration context value representing the enumeration session to be used.
  - When the Pull operation is invoked, the enumeration session represented by the `EnumerationContext` input parameter shall be open. The first enumeration session shall use one of the Open operations with a type of enumerated element that matches the Pull operation. For the first Pull operation on an enumeration session, the value of the `EnumerationContext` input parameter shall be the enumeration context value returned by a successful Open operation. For subsequent Pull operations on that enumeration session, the value of the `EnumerationContext` input parameter shall be the value of the `EnumerationContext` output parameter returned by the previous Pull operation on the same enumeration session.
  - After the Pull operation is completed, the enumeration session represented by the `EnumerationContext` output parameter shall be open or closed.
- EndOfSequence
  - This output parameter indicates to the CIM client whether the enumeration session is exhausted. If `EndOfSequence` is `true` upon successful completion of a Pull operation, no more elements are available and the CIM server shall close the enumeration session, releasing any allocated resources related to the session. If `EndOfSequence` is `false`, additional elements may be available, and the CIM server shall not close the session.
- MaxObjectCount
  - This input parameter defines the maximum number of elements that may be returned by this Pull operation. Any uint32 number is valid, including 0. The CIM server may deliver any number of elements up to `MaxObjectCount` but shall not deliver more than `MaxObjectCount` elements. The CIM client may use a MaxObjectCount value of 0 to restart the OperationTimeout for the enumeration session when it does not need to not retrieve any elements.
- Return Value (array of enumerated elements)
  - The return value of a Pull operation upon successful completion is an array of enumerated elements with a number of entries from 0 up to a maximum defined by `MaxObjectCount`. These entries meet the criteria defined in the Open operation that established this enumeration session. Note that returning no entries in the array does not imply that the

enumeration session is exhausted. Only the `EndOfSequence` output parameter indicates whether the enumeration session is exhausted.

5.3.2.24.10PullInstancesWithPath

The PullInstancesWithPath operation retrieves instances including their instance paths from an open enumeration session represented by an enumeration context value:

```
PullInstancesWithPath
<instanceWithPath>* PullInstancesWithPath (
    [IN,OUT] <enumerationContext> EnumerationContext,
    [OUT] Boolean EndOfSequence,
    [IN] uint32 MaxObjectCount
)/td>
```

The PullInstancesWithPath operation shall comply with the behavior defined in 5.3.2.24.1.

The `EnumerationContext` input/output parameter is defined in 5.3.2.24.9. The enumeration session shall be established using one of the OpenEnumerateInstances, OpenReferenceInstances, or OpenAssociatorInstances operations.

The `EndOfSequence` output parameter is defined in 5.3.2.24.9.

The `MaxObjectCount` input parameter is defined in 5.3.2.24.9.

If PullInstancesWithPath is successful, the return value shall be an array of enumerated instances including their instance paths as defined in 5.3.2.24.9.

If PullInstancesWithPath is unsuccessful, this operation shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional operation-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_SERVER_IS_SHUTTING_DOWN
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_INVALID_ENUMERATION_CONTEXT
- CIM_ERR_PULL_HAS_BEEN_ABANDONED
- CIM_ERR_SERVER_LIMITS_EXCEEDED
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.24.11PullInstancePaths

The PullInstancePaths operation retrieves instance paths from an open enumeration session represented by an enumeration context value:

```
PullInstancePaths
<instancePath>* PullInstancePaths (
    [IN,OUT] <enumerationContext> EnumerationContext,
    [OUT] Boolean EndOfSequence,
    [IN] uint32 MaxObjectCount
)
```

The PullInstancePaths operation shall comply with the behavior defined in 5.3.2.24.1.

The `EnumerationContext` input/output parameter is defined in 5.3.2.24.9. The enumeration session shall have been established using one of the OpenEnumerateInstancePaths, OpenReferenceInstancePaths, or OpenAssociatorInstancePaths operations.

The `EndOfSequence` output parameter is defined in 5.3.2.24.9.

The `MaxObjectCount` input parameter is defined in 5.3.2.24.9.

If PullInstancePaths is successful, the return value shall be an array of enumerated instance paths as defined in 5.3.2.24.9.

If PullInstancePaths is unsuccessful, this operation shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional operation-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_SERVER_IS_SHUTTING_DOWN
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_INVALID_ENUMERATION_CONTEXT
- CIM_ERR_SERVER_LIMITS_EXCEEDED
- CIM_ERR_PULL_HAS_BEEN_ABANDONED
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.24.12CloseEnumeration

The CloseEnumeration operation closes an open enumeration session, performing an early termination of an enumeration sequence:

```
CloseEnumeration
void CloseEnumeration (
    [IN] <enumerationContext> EnumerationContext
)
```

The `EnumerationContext` parameter is the value representing the enumeration session to be closed. The enumeration session shall be open and shall be established using one of the Open operations. This implies that this operation is not to close an enumeration sequence already indicated by `EndOfSequence` because the sequence has already been closed. The value of the `EnumerationContext` parameter shall be the value of the `EnumerationContext` output parameter returned by the previous Pull operation on the enumeration session to be closed.

If CloseEnumeration is successful, the CIM server shall close the enumeration session represented by `EnumerationContext`, releasing any allocated resources. Any subsequent use of the `EnumerationContext` value is unsuccessful.

CloseEnumeration may be executed concurrently with a Pull operation or an EnumerationCount operation on the same enumeration session. If a CIM server receives a CloseEnumeration operation request while it is processing a Pull operation on the same enumeration session, the server shall attempt to abandon that Pull operation. If the Pull operation can be abandoned, it shall return a failure with the status code CIM_ERR_PULL_HAS_BEEN_ABANDONED and the CloseEnumeration operation shall return success.

If the Pull operation cannot be abandoned, it shall proceed as if the CloseEnumeration operation has not been issued, and the CloseEnumeration operation shall return a failure with the status code CIM_ERR_PULL_CANNOT_BE_ABANDONED.

If CloseEnumeration is unsuccessful, this operation shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional operation-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_SERVER_IS_SHUTTING_DOWN
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_INVALID_ENUMERATION_CONTEXT
- CIM_ERR_PULL_CANNOT_BE_ABANDONED
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.24.13EnumerationCount

The EnumerationCount operation provides an estimated count of the total number of objects in an open enumeration session represented by an EnumerationContext:

```
EnumerationCount
uint64 EnumerationCount (
    [IN] <enumerationContext>EnumerationContext
)
```

The `EnumerationContext` parameter identifies the enumeration session for the EnumerationCount operation. It shall be established using any of the Open operations and shall be open at the time of the CloseEnumeration request. A conformant CIM server may support this operation. A CIM server that does not support this operation should respond with the CIM_ERR_NOT_SUPPORTED status.

If EnumerationCount is successful, the operation returns an approximate count of the number of objects in the enumeration session. This is the number of items remaining to be sent with subsequent Pull operations. Thus, executing this operation immediately after the open may provide an approximate estimate of the total number of objects to be returned in the enumeration set. The returned count is only an estimate of the number of objects to be pulled in the enumeration sequence. This mechanism is intended to assist CIM clients in determining the overall size of an enumeration set and the number of objects remaining in the enumeration session. It should not be used instead of the `EndOfSequence` parameter to determine the end of an enumeration sequence.

If the CIM server cannot or will not return an estimate of the number of objects to be returned for the enumerationContext, it may return success and the NULL value.

If EnumerationCount is unsuccessful, this operation shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional operation-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_SERVER_IS_SHUTTING_DOWN
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE

- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_INVALID_ENUMERATION_CONTEXT
- CIM_ERR_SERVER_LIMITS_EXCEEDED
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.24.14OpenQueryInstances

The OpenQueryInstances operation establishes and opens an enumeration session of the instances of a CIM class (including instances of its subclasses) in the target namespace. Optionally, it retrieves a first set of instances:

```
OpenQueryInstances
<instance>* OpenQueryInstances (
    [IN] string FilterQuery,
    [IN] string FilterQueryLanguage,
    [IN,OPTIONAL] Boolean ReturnQueryResultClass = false,
    [IN,OPTIONAL,NULL] uint32 OperationTimeout = NULL,
    [IN,OPTIONAL] Boolean ContinueOnError = false,
    [IN,OPTIONAL] uint32 MaxObjectCount = 0,
    [OUT, OPTIONAL, NULL] <class> QueryResultClass,
    [OUT] <enumerationContext> EnumerationContext,
    [OUT] Boolean EndOfSequence
)
```

The OpenQueryInstances shall comply with the behavior defined in 5.3.2.24.1.

The `FilterQuery` and `FilterQueryLanguage` input parameters specify the set of enumerated instances.

`FilterQueryLanguage` shall specify a query language and the value of `FilterQuery` shall be a valid query in that query language. This specification defines neither the query language nor the format of the query. It is anticipated that query languages will be submitted to the DMTF as separate proposals. A mechanism by which CIM servers can declare the query languages they support for filtering in Pulled enumerations (if any) is defined in 7.5.

The `ReturnQueryResultClass` input parameter controls whether a class definition is returned in `QueryResultClass`. If it is set to `false`, `QueryResultClass` shall be set to NULL on output. If it is set to `true`, the value of the `QueryResultClass` on output shall be a class definition that defines the properties (columns) of each row of the query result.

The `OperationTimeout` input parameter is defined in 5.3.2.24.2.

The `ContinueOnError` input parameter is defined in 5.3.2.24.2.

The `MaxObjectCount` input parameter is defined in 5.3.2.24.2.

The `QueryResultClass` output parameter shall be set to NULL if the `ReturnQueryResultClass` input parameter is set to `false`. Otherwise, it shall return a class definition where each property of the class corresponds to one entry of the query select list. The class definition corresponds to one row of the query result. The class name of this returned class shall be "CIM_QueryResult." This class definition is valid only in the context of this enumeration.

The `EnumerationContext` output parameter is defined in 5.3.2.24.2.

The `EndOfSequence` output parameter is defined in 5.3.2.24.2.

If OpenQueryInstances is successful, the return value shall be an array of enumerated instances as defined in 5.3.2.24.2. Such instances are available only in the context of the enumeration and do not return an instance path. The PullInstancesWithPath operation may not be used to continue an enumeration started by the OpenQueryInstances operation.

The PullInstances operation shall be used to pull instances for an enumeration session opened using If OpenQueryInstances. If any other operation is used to pull instances, the CIM server shall return failure with the status code CIM_ERR_FAILED.

If OpenQueryInstances is unsuccessful, this operation shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional operation-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_SERVER_IS_SHUTTING_DOWN
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_OPERATION_TIMEOUT
- CIM_ERR_CONTINUATION_ON_ERROR_NOT_SUPPORTED
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_QUERY_LANGUAGE_NOT_SUPPORTED (The requested filter query language is not recognized.)
- CIM_ERR_INVALID_QUERY (The filter query is not a valid query in the specified filter query language.)
- CIM_ERR_QUERY_FEATURE_NOT_SUPPORTED (The query requires support for features that are not supported.)
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.2.24.15PullInstances

The PullInstances operation retrieves instances from an OpenQueryInstances session represented by an enumeration context value:

```
PullInstances
<instance>* PullInstances (
    [IN,OUT] <enumerationContext> EnumerationContext,
    [OUT] Boolean EndOfSequence,
    [IN] uint32 MaxObjectCount
)
```

The PullInstances operation shall comply with the behavior defined in 5.3.2.24.1.

The `EnumerationContext` input/output parameter is defined in 5.3.2.24.9. The enumeration session shall be established using the OpenQueryInstances operation.

The `EndOfSequence` output parameter is defined in 5.3.2.24.9.

The `MaxObjectCount` input parameter is defined in 5.3.2.24.9.

If PullInstances is successful, the return value shall be an array of enumerated instances as defined in 5.3.2.24.9.

If PullInstances is unsuccessful, this operation shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional operation-specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_SERVER_IS_SHUTTING_DOWN
- CIM_ERR_NOT_SUPPORTED
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_INVALID_ENUMERATION_CONTEXT
- CIM_ERR_SERVER_LIMITS_EXCEEDED
- CIM_ERR_PULL_HAS_BEEN_ABANDONED
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.3.3 Namespace Manipulation Using the CIM_Namespace Class

No intrinsic methods are defined specifically to manipulate namespaces. Namespaces shall be manipulated using intrinsic methods on the CIM_Namespace class.

5.3.3.1 Namespace Creation

A namespace is created by calling the intrinsic method CreateInstance for the CIM_Namespace class. A value is specified for the new instance parameter that defines a valid instance of the CIM_Namespace class and that has a name property that is the desired name of the new namespace.

The proposed definition shall be a correct namespace definition according to DSP0004. Despite the naming conventions used in the CIM specifications (use of / in namespaces such as root/CIMV2 and root/CIMV2/test), there is no hierarchy implied among different namespaces. Each namespace is independent of all others. The namespaces are to be considered flat, and there is no defined behavior for navigating namespaces.

In creating the new namespace, the CIM server shall conform to the following rules:

- The namespace defined by name property shall not already exist in the CIM server.
- The <LOCALNAMESPACEPATH> defined for the operation defines the namespace in which the CIM_Namespace instance associated with this new namespace is created.

It is recommended that instances of CIM_Namespace be created in root unless there is a specific reason to define them in another namespace. The inclusion of a CIM_Namespace instance within a namespace other than root is allowed.

In addition to creating instances of CIM_Namespace, compliant implementations shall also create an instance of the association class CIM_NamespaceInManager defining the linking of the namespace created to the current CIM_ObjectManager.

If CreateInstance is successful, the CIM server creates the specified namespace. In addition, the CIM server shall return information about the namespace as an instance of the class CIM_Namespace and of returning instances of the association class CIM_NamespaceInManager for each CIM_Namespace instance created.

5.3.3.2 Namespace Deletion

If the CIM server supports the CIM_Namespace class, all valid namespaces shall be represented by an instance of the CIM_Namespace class. A namespace is deleted using the intrinsic method DeleteInstance to delete the instance of the class CIM_Namespace that represents the namespace. The namespace to be deleted shall exist.

If DeleteInstance is successful, the CIM server shall remove the specified CIM_Namespace instance.

If DeleteInstance is unsuccessful, one of the status codes defined for the DeleteInstance operation shall be returned. A CIM server may return CIM_ERR_FAILED if a non-empty namespace cannot successfully be deleted.

5.3.3.3 Manipulation and Query of Namespace Information

The query of namespaces is provided through the following means:

- Query of the CIM_Namespace class on an individual namespace
- Use of the CIM_NamespaceInManager association to link the target CIM_ObjectManager and the instances of CIM_Namespace representing all namespaces defined in the target CIM_ObjectManager

5.3.3.4 Use of the __Namespace Pseudo Class (DEPRECATED)

In previous versions of this specification, namespaces were manipulated through the pseudo class __Namespace as follows:

No intrinsic methods are specifically defined for manipulating CIM namespaces. However, modeling a CIM namespace using class __Namespace, together with the requirement that the root namespace be supported by all CIM servers, implies that all namespace operations can be supported.

For example, all child namespaces of a particular namespace are enumerated by calling the intrinsic method EnumerateInstanceNames against the parent namespace, specifying a value for the ClassName parameter of __Namespace. A child namespace is created by calling the intrinsic method CreateInstance against the parent namespace, specifying a value for the NewInstance parameter that defines a valid instance of the class __Namespace and that has a name property that is the desired name of the new namespace.

DEPRECATION NOTE: The use of the __Namespace class is DEPRECATED. In its place, use the CIM_Namespace class.

5.3.4 Functional Profiles

To establish conformance, this clause partitions the intrinsic methods into functional groups.

Support for a particular group does *not* guarantee that all invocations of a method in that group will succeed. Rather, the exclusion of a group is a declaration that any attempt to call a method in that group always returns CIM_ERR_NOT_SUPPORTED.

Mechanisms by which a CIM server may declare the functional groups that it supports are defined in 7.5.

To limit the number of different profiles that a CIM server may support, each functional group has a dependency on another group (with the exception of the Basic Read functional group). If functional group G_1 has a dependency on functional group G_2, then a CIM server that supports G_1 shall also support G_2.

The dependency relation is transitive, so if G_1 depends on G_2, and G_2 depends on G_3, then G_1 depends on G_3. It is also anti-symmetric, so if G_1 depends on G_2, then G_2 cannot depend on G_1.

2401 Using these rules, Table 3 defines a rooted-directed tree of dependencies with the Basic Read
2402 dependency representing the root node.

2403 For example, a CIM server that supports the Schema Manipulation functional group shall also support the
2404 Instance Manipulation, Basic Write, and Basic Read.

2405 A CIM server shall support the Basic Read functional group.

2406 **Table 3 – Root-Directed Tree of Functional Profile Dependencies**

Functional Group	Dependency	Methods
Basic Read	*none*	GetClass EnumerateClasses EnumerateClassNames GetInstance EnumerateInstances EnumerateInstanceNames GetProperty
Pulled Read	*Basic Read*	OpenEnumerateInstances OpenEnumerateInstancePaths OpenReferenceInstances OpenReferenceInstancePaths OpenAssociatorInstances OpenAssociatorInstancePaths PullInstancesWithPath PullInstancePaths CloseEnumeration
PulledReadCount	Pulled Read	EnumerationCount
Pulled Query Execution	Pulled Read	OpenQueryInstances PullInstances
Basic Write	Basic Read	SetProperty
Schema Manipulation	Instance Manipulation	CreateClass ModifyClass DeleteClass
Instance Manipulation	Basic Write	CreateInstance ModifyInstance DeleteInstance
Association Traversal	Basic Read	Associators AssociatorNames References ReferenceNames
Query Execution	Basic Read	ExecQuery
Qualifier Declaration	Schema Manipulation	GetQualifier SetQualifier DeleteQualifier EnumerateQualifiers

5.3.5 Extrinsic Method Invocation

Any CIM server is assumed to support extrinsic methods, which are defined by the schema supported by the CIM server. If a CIM server does not support extrinsic method invocations, it shall return the error code CIM_ERR_NOT_SUPPORTED to any request to execute an extrinsic method (subject to the considerations described in the rest of this clause). This allows a CIM client to determine that all attempts to execute extrinsic methods will fail.

If the CIM server cannot invoke extrinsic methods, it shall return one of the following status codes, where the error returned is the first applicable error in the list, starting with the first element and working down. Any additional specific interpretation of the error is enclosed in parentheses.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED (The CIM server does not support extrinsic method invocations.)
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise incorrect parameters)
- CIM_ERR_NOT_FOUND (The target CIM class or instance does not exist in the specified namespace.)
- CIM_ERR_METHOD_NOT_FOUND
- CIM_ERR_METHOD_NOT_AVAILABLE (The CIM server is unable to honor the invocation request.)
- CIM_ERR_FAILED (Some other unspecified error occurred.)

5.4 CIM Export Syntax and Semantics

This clause focuses on export methods and their invocation, as well as on functional profiles.

5.4.1 Export Method Invocations

All CIM export message requests defined for the CIM-to-HTTP mapping are invocations of one or more export methods. Export methods do not operate against CIM namespaces.

An export method call is represented in XML by the <EXPMETHODCALL> element, and the response to that call is represented by the <EXPMETHODRESPONSE> element.

An input parameter has an IN qualifier with value `true` in the method definition. An output parameter has an OUT qualifier with value `true` in the method definition. A parameter may be both an input parameter and an output parameter.

The <EXPMETHODCALL> element names the method to be invoked and supplies any input parameters to the export method call:

- Each input parameter shall be named using the name assigned in the method definition.
- Input parameters may be supplied in any order.
- Each input parameter of the method, and no others, shall be present in the call unless it is optional.

The <EXPMETHODRESPONSE> element defines either an <ERROR> or a (possibly optional) return value and output parameters, which are decorated with the OUT qualifier in the method definition. In the latter case, the following rules apply:

- Each output parameter shall be named using the name assigned in the method definition.
- Output parameters may be supplied in any order.
- Each output parameter of the method, and no others, shall be present in the response, unless it is optional.

The method invocation process may be thought of as a two-part process:

- Binding the input parameter values specified as subelements of the <EXPMETHODCALL> element to the input parameters of the method.
- Attempting to execute the method using the bound input parameters, with one of the following results:
  - If the attempt to call the method is successful, the return value and output parameters are bound to the subelements of the <EXPMETHODRESPONSE> element.
  - If the attempt to call the method is unsuccessful, an error code and (optional) human-readable description of that code is bound to the <EXPMETHODRESPONSE> element.

5.4.1.1 Simple Export

A simple export requires the invocation of a single export method. A simple export request is represented by a <SIMPLEEXPREQ> element, and a simple export response is represented by a <SIMPLEEXPRSP> element.

A <SIMPLEEXPREQ> shall contain a <EXPMETHODCALL> element.

5.4.1.2 Multiple Export

A multiple export requires the invocation of more than one export method. A multiple export request is represented by a <MULTIEXPREQ> element, and a multiple export response is represented by a <MULTIEXPRSP> element.

A <MULTIEXPREQ> (or its respective <MULTIEXPRSP>) element is a sequence of two or more <SIMPLEEXPREQ> (or its respective <SIMPLEEXPRSP>) elements.

A <MULTIEXPRSP> element shall contain a <SIMPLEEXPRSP> element for every <SIMPLEEXPREQ> element in the corresponding multiple export response. These <SIMPLEEXPRSP> elements shall be in the same order as their <SIMPLEEXPREQ> counterparts. The first <SIMPLEEXPRSP> in the response corresponds to the first <SIMPLEEXPREQ> in the request, and so forth.

Multiple exports conveniently batch the delivery of multiple export method invocations into a single HTTP message, reducing the number of roundtrips between a CIM client and a CIM listener and allowing the CIM listener to make certain internal optimizations. Note that multiple exports do not confer any transactional capabilities in processing the request. For example, the CIM listener does not have to guarantee that the constituent export method calls either all failed or all succeeded. The CIM listener must only make a "best effort" to process the operation. However, CIM listeners shall finish processing each method invocation in a batched message before executing the next method invocation in the batch. Clients shall recognize that the order of method calls within a batched message is significant.

Not all CIM listeners support multiple exports. If a CIM listener does not support multiple exports, it shall return the status code CIM_ERR_NOT_SUPPORTED.

5.4.1.3 Status Codes

This clause defines the status codes and detailed error information that a conforming CIM listener may return.

The value of an <ERROR> subelement within a <EXPMETHODRESPONSE> element includes the following parts:

- mandatory status code
- optional human-readable description of the status code
- zero or more CIM_Error instances

The symbolic names defined in Table 4 do not appear on the wire. They are used here solely for convenient reference to an error in other parts of this specification. Not all methods are expected to return all these status codes.

In addition to returning a status code, a conforming CIM listener may return zero or more <INSTANCE> subelements as part of an <ERROR> element. Each <INSTANCE> subelement shall be an instance of CIM_Error, and the value of CIMStatusCode shall comply with the definition of expected error codes for the CIM export request. A CIM client may ignore any <INSTANCE> subelements.

Table 4 – Symbolic Names for Referencing Error Codes

Symbolic Name	Code	Definition
CIM_ERR_FAILED	1	A general error occurred that is not covered by a more specific error code.
CIM_ERR_ACCESS_DENIED	2	Access was not available to the client.
CIM_ERR_NOT_SUPPORTED	7	The requested operation is not supported.
CIM_ERR_TYPE_MISMATCH	13	The value supplied is incompatible with the type.

5.4.2 Export Methods

This clause describes the methods that can be defined within a CIM export message. These methods operate only on an external data representation of a CIM entity, namespace, or element. Specifically, export methods do not operate on CIM namespaces or elements. The export method defined in this specification is Export an Indication.

The notation used in the following subclauses to define the signatures of the export methods is a pseudo-MOF notation that extends the standard MOF BNF (DSP0004) for describing CIM export methods with a number of pseudo parameter types. The pseudo parameter types are enclosed in angle brackets (< >).

This notation allows parameters to be decorated with pseudo-qualifiers (IN, OPTIONAL, and NULL) to define their invocation semantics. Note that these qualifiers are for description purposes only within the scope of this specification. In particular, a CIM client shall not specify them in export method invocations.

This notation uses the IN qualifier for input parameters.

A CIM client may omit an optional parameter if the required value is the specified default by not specifying an <EXPPARAMVALUE> element for the parameter. It shall not omit a parameter that is not optional.

The NULL qualifier indicates parameters with values that may be specified as NULL in an export method call. A NULL (unassigned) value for a parameter is specified by an <EXPPARAMVALUE> element with no subelement. The CIM client shall specify a value for parameters without the NULL qualifier by including a suitable subelement for the <EXPPARAMVALUE> element.

All parameters shall be uniquely named and shall correspond to a valid parameter name for that method as described by this specification. The order of the parameters is not significant.

The non-NULL values of export method parameters or return values that are modeled as standard CIM types (such as string and Boolean, or arrays thereof) are represented as follows:

- Simple values shall be represented by the <VALUE> subelement in an <EXPPARAMVALUE> element (for export method parameters) or in an <IRETURNVALUE> element (for export method return values).
- Array values shall be represented by the <VALUE.ARRAY> subelement in an <EXPPARAMVALUE> element (for export method parameters) or in an <IRETURNVALUE> element (for export method return values).

Table 5 shows how each pseudo-type used by the export methods shall be mapped to an XML element described in DSP0201 in the context of both a parameter value (subelement of <EXPPARAMVALUE>) and a return value (subelement of <IRETURNVALUE>).

2532 **Table 5 – Mapping of Export Method Pseudo-Types to XML Elements**

Type	XML Element
<object>	(VALUE.OBJECT|VALUE.OBJECTWITHLOCALPATH|VALUE.OBJECTWITHPATH)
<class>	CLASS
<instance>	INSTANCE
<className>	CLASSNAME
<namedInstance>	VALUE.NAMEDINSTANCE
<instanceName>	INSTANCENAME
<objectWithPath>	VALUE.OBJECTWITHPATH
<objectName>	(CLASSNAME|INSTANCENAME)
<propertyValue>	(VALUE|VALUE.ARRAY|VALUE.REFERENCE)
<qualifierDecl>	QUALIFIER.DECLARATION

2533 #### 5.4.2.1 ExportIndication

2534 The ExportIndication operation exports a single CIM indication to the destination CIM listener:

2535 **`ExportIndication`**
2536 `void` **`ExportIndication`** `(`
2537 `[IN] <instance> NewIndication`
2538 `)`

2539 The `NewIndication` input parameter defines the indication to be exported. The proposed definition
2540 should be a correct instance definition for the underlying CIM indication class according to the CIM
2541 specification.

2542 If ExportIndication is unsuccessful, this method shall return one of the following status codes, where the
2543 error returned is the first applicable error in the list, starting with the first element and working down. Any
2544 additional method-specific interpretation of the error is enclosed in parentheses.

2545 • CIM_ERR_ACCESS_DENIED

2546 • CIM_ERR_NOT_SUPPORTED

2547 • CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized, or otherwise
2548 incorrect parameters)

2549 • CIM_ERR_INVALID_CLASS (The CIM class of which this is to be a new instance does not
2550 exist.)

2551 • CIM_ERR_FAILED (Some other unspecified error occurred.)

2552 ### 5.4.3 Functional Profiles

2553 This clause partitions the export methods into functional groups to establish conformance. See Table 6.

2554 Support for a particular group does not guarantee that all invocations of an export method in that group
2555 will succeed. Rather, the exclusion of a group is a declaration that any attempt to call an export method in
2556 that group always returns CIM_ERR_NOT_SUPPORTED.

2557 The dependency relation is transitive, so if group G_1 depends on G_2, and G_2 depends on G_3, then G_1
2558 depends on G_3. It is also anti-symmetric, so if G_1 depends on G_2, then G_2 cannot depend on G_1.

2559 **Table 6 – Functional Groups of Export Methods**

Functional Group	Dependency	Method
Indication	None	ExportIndication

2560 # 6 Encapsulation of CIM Messages

2561 This clause describes how to use CIM messages in HTTP. CIM message requests may be used with or
2562 without the HTTP Extension Framework.

2563 Although CIM messages can be used in combination with a variety of HTTP request methods, this
2564 specification defines CIM messages only within HTTP POST requests. (M-POST may be used in place of
2565 POST. For details on how to use CIM messages with the HTTP Extension Framework, see 6.2.)

2566 All CIM message responses are carried in the corresponding HTTP response. In the remaining
2567 discussion, the following terms are used as convenient shorthand for the definitions provided here:

2568 2569 - *CIM operation request.* An HTTP POST request message with an XML entity body that defines an Operation Request Message.
2570 2571 - *CIM operation response.* An HTTP response message, issued in response to a CIM operation request, with an entity body that defines an Operation Response Message.
2572 2573 - *CIM export request*. An HTTP POST request message with an XML entity body that defines a CIM export message request.
2574 2575 - *CIM export response.* An HTTP response message, issued in response to a CIM export message request, with an entity body that defines a CIM export message response.
2576 2577 - *CIM message request*. An HTTP POST request message with an XML entity body that defines either a CIM operation or export message request.
2578 2579 2580 - *CIM message response.* An HTTP response message, issued in response to a CIM operation message (or CIM export message) request, with an entity body that defines a CIM operation message (or CIM export message) response.

2581 Note that an HTTP response to a CIM request is not always a CIM response. For example, a "505 HTTP
2582 Version Not Supported" response is not a CIM response.

2583 ## 6.1 CIM Clients, CIM Servers, and CIM Listeners

2584 A CIM product is any product that can supply and/or consume management information using the CIM
2585 schema. In particular, CIM clients, CIM servers, and CIM listeners are examples of CIM products:

2586 2587 - A *CIM client* issues CIM operation requests (CIM message requests) and receives and processes CIM operation responses (CIM message responses).
2588 2589 - A *CIM server* receives and processes CIM operation message requests and issues CIM operation message responses.
2590 2591 - A *CIM listener* is a server that receives and processes CIM export message requests and issues CIM export message responses.

2592 A CIM server may be act as any combination of CIM client, CIM server, and CIM listener. For example, a
2593 CIM server that supports indication subscription and generation acts as a CIM client when delivering an
2594 indication, through ExportIndication, to a CIM listener.

2595 Throughout this document, the terms CIM client, CIM server, CIM listener, and CIM product are used as
2596 convenient shorthand to refer to the subset of CIM products that conform to this specification.

6.2 Use of M-POST

A CIM client attempting to invoke a CIM message using the HTTP Extension Framework method "M-POST" shall follow these steps:

- If the M-POST invocation fails with an HTTP status of "501 Not Implemented" or "510 Not Extended," the client should retry the request using the HTTP method "POST" with the appropriate modifications (described in 6.2.2).
- If the M-POST invocation fails with an HTTP status of "405 Method Not Allowed," the client should fail the request.
- For all other status codes, the client shall act in accordance with standard HTTP (RFC 1945, RFC 2068).

This extended invocation mechanism gives Internet proxies and firewalls greater filtering control and administrative flexibility over CIM message invocations.

If a client receives a 501 or 510 status in response to an M-POST request, in subsequent invocations to the same HTTP server, the client may omit the attempt at M-POST invocations for a suitable period. This omission avoids the need for an extra round trip on each and every method invocation. The details of the caching strategy employed by the client are outside the scope of this specification.

6.2.1 Use of the Ext Header

If a CIM server or CIM listener receives a valid M-POST request and has fulfilled all mandatory extension header declarations in the request, it shall include in the response the "Ext" header defined by RFC 2774. This included header shall be protected by the appropriate Cache-Control directive.

6.2.2 Naming of Extension Headers

In M-POST request messages (and their responses), CIM extension headers shall be declared using the name space prefix allotted by the "Man" extension header (in accordance with RFC 2774) that refers to the name space "http://www.dmtf.org/cim/mapping/http/v1.0". The full format of the "Man" header declaration for this specification is:

```
Man           = "Man" ":" "http://www.dmtf.org/cim/mapping/http/v1.0"
                ";" "ns" "=" header-prefix
header-prefix = 2*DIGIT
```

This header-prefix should be generated at random on a per-HTTP message basis, and should not necessarily be a specific number.

In accordance with RFC 2774, all POST request messages (and their responses) shall not include such a mandatory extension declaration. In POST request messages (and their responses), name space prefixes shall not be used.

EXAMPLE 1:

Using M-POST:

```
M-POST /cimom HTTP/1.1
Man: http://www.dmtf.org./cim/mapping/http/v1.0 ; ns=23
23-CIMOperation: MethodCall
...
```

EXAMPLE 2:

Using POST:

```
POST /cimom HTTP/1.1
CIMOperation: MethodCall
```

...

6.3 Extension Headers Defined for CIM Message Requests and Responses

A CIM message contains exactly one CIM operation request, CIM operation response, CIM export request, or CIM export response. This clause describes the extension headers to specify CIM message semantics in the HTTP header of a POST message.

Any CIM Operation Request or CIM Operation Response shall, and only CIM operation requests and responses may, include the following CIM extension header:

- CIMOperation

Any CIM Operation Request shall, and only CIM operation requests may, include one and only one of the following CIM extension header sets:

- CIMMethod and CIMObject, or
- CIMBatch

Any CIM export request or CIM export response shall, and only CIM export requests and responses may, include the following CIM extension header:

- CIMExport

Any CIM export request shall, and only CIM export requests may, include one and only one of the following CIM extension headers:

- CIMExportMethod
- CIMExportBatch

An HTTP response with an error status code to a CIM message request may include the following CIM extension header:

- CIMError

All CIM messages may include the following CIM extension header:

- CIMProtocolVersion

6.3.1 Encoding of CIM Element Names within HTTP Headers and Trailers

CIM element (class, property, qualifier, method, or method parameter) names are natively Unicode, and may use UCS-2 characters unsuitable for inclusion within an HTTP message header or trailer. To encode CIM element names represented in Unicode to values within HTTP headers or trailers, the following two-step mapping process shall be used:

- Encode the full Unicode CIM element name using UTF-8.
- Using the ""%" HEX HEX" convention, apply the standard URI [RFC 2396, section 2] escaping mechanism to the resulting string to escape any characters that are unsafe within an HTTP header or trailer.

In this specification, the token CIMIdentifier represents a CIM element name to which this transformation has been applied.

One characteristic of this mapping is that CIM elements named with an ASCII representation appear in ASCII in the resulting URL.

EXAMPLE:

CIM_LogicalElement is unchanged under this transformation.

The class named using the UCS-2 sequence representing the Hangul characters for the Korean word "hangugo" (D55C, AD6D, C5B4) becomes

%ED%95%9C%EA%B5%AD%EC%96%B4=10

after UTF-8 transformation and escaping all characters with their % HEX HEX equivalent.

6.3.2 Encoding of CIM Object Paths within HTTP Headers and Trailers

This clause describes the mapping that shall be applied to represent CIM object paths, as described within an Operation Request Message using the <LOCALNAMESPACEPATH>, <LOCALCLASSPATH>, or <LOCALINSTANCEPATH> elements, in a format that is safe for representation within an HTTP header or trailer.

If the element to be transformed is a <LOCALNAMESPACEPATH>, the algorithm is as follows:

- For the first NAMESPACE subelement, output the textual content of that element.
- For each subsequent NAMESPACE subelement, output the forward slash character (/) followed by the textual content of that NAMESPACE element.

If the element to be transformed is a <LOCALCLASSPATH>, the algorithm is as follows:

- Transform the <LOCALNAMESPACEPATH> subelement using the rules previously described, and output a colon character (:).
- Output the value of the `NAME` attribute of the <CLASSNAME> subelement.

If the element to be transformed is a <LOCALINSTANCEPATH>, the algorithm is as follows:

- Transform the <LOCALNAMESPACEPATH> subelement using the rules previously described, and output a colon character (:).
- Output the value of the `CLASSNAME` attribute of the <INSTANCENAME> subelement.
- If there is at least one <KEYBINDING> subelement under the <INSTANCENAME> subelement, then for each such subelement:
  - Output a period character (.) if this is the first <KEYBINDING> subelement; otherwise, output a comma character (,).
  - Output the value of the `NAME` attribute, followed by an equal character (=).
  - If there is a <KEYVALUE> subelement, output the textual element content of that element, subject to the following transformation:
    - If the `VALUETYPE` attribute is numeric or Boolean, the output is identical to the content of the element.
    - If the `VALUETYPE` attribute is a string, the output is obtained by enclosing the content of the element in double quote (") characters and escaping any double quote characters or backslash character within the value with a preceding backslash (\) character.
  - If there is a <VALUE.REFERENCE> subelement
    - Output a double quote character (").

2715 2716 2717 ▪ Apply the process recursively to the <CLASSPATH> or <INSTANCEPATH> subelement of the <VALUE.REFERENCE> element, escaping any double quote or backslash character thereby generated with a preceding backslash (\) character.

2718 ▪ Output a closing double quote character (").

2719 2720 • If there is no <KEYBINDING> subelement but there is a <KEYVALUE> or <VALUE.REFERENCE> subelement under the <INSTANCENAME> subelement, then:

2721 – Output an equal character (=).

2722 2723 – Output the transformed value of the <KEYVALUE> or <VALUE.REFERENCE> using the previously-described rules.

2724 2725 2726 • If there are no <KEYBINDING> subelements or no <KEYVALUE> or <VALUE.REFERENCE> subelement, then indicate a singleton instance by outputting the string "=@" under the <INSTANCENAME> subelement.

2727 2728 2729 Finally, after applying these rules to the <LOCALNAMESPACEPATH>, <LOCALCLASSPATH>, or <LOCALINSTANCEPATH> element, transform the entire output string into URI-safe format in the following two-step procedure:

2730 • Encode the string using UTF-8 [RFC 2279] if it is not already in this format.

2731 2732 2733 • Using the ""%" HEX HEX" convention, apply the standard URI [RFC 2396, section 2] escaping mechanism to the resulting string to escape any characters that are unsafe within an HTTP header or trailer.

2734 2735 2736 In this specification, the token CIMObjectPath represents a <LOCALNAMESPACEPATH>, <LOCALCLASSPATH>, or <LOCALINSTANCEPATH> element to which the preceding transformation has been applied.

2737 6.3.3 CIMOperation

2738 2739 The CIMOperation header shall be present in all CIM Operation Request and CIM Operation Response messages. It identifies the HTTP message as carrying a CIM operation request or response.

2740 CIMOperation = "CIMOperation" ":" ("MethodCall" | "MethodResponse")

2741 2742 2743 A CIM client shall include this header, with the value "MethodCall," in all CIM operation requests that it issues. A CIM server shall include this header in all CIM operation responses that it issues, with the value "MethodResponse".

2744 2745 2746 If a CIM server receives a CIM operation request with this header, but with a missing value or a value that is not "MethodCall," then it shall fail the request with status "400 Bad Request". The CIM server shall include a CIMError header in the response with a value of `unsupported-operation`.

2747 2748 2749 If a CIM server receives a CIM operation request without this header, it shall not process it as a CIM operation request. The status code returned by the CIM server in response to such a request is outside the scope of this specification.

2750 2751 2752 2753 If a CIM client receives a response to a CIM operation request without this header (or if this header has a value that is not "MethodResponse"), it should discard the response and take appropriate measures to publicize that it has received an incorrect response. The details as to how this is done are outside the scope of this specification.

2754 2755 The CIMOperation header affords a simple mechanism by which firewall or proxy administrators can make global administrative decisions on all CIM operations.

6.3.4 CIMExport

The CIMExport header shall be present in all CIM export request and response messages. It identifies the HTTP message as carrying a CIM export method request or response.

CIMExport = "CIMExport" ":" ("MethodRequest" | "MethodResponse")

A CIM client shall include this header with the value "MethodRequest" in all CIM export requests that it issues. A CIM listener shall include this header in all CIM export responses that it issues, with the value "MethodResponse".

If a CIM listener receives a CIM export request with this header, but with a missing value or a value that is not "MethodRequest", then it shall fail the request with status "400 Bad Request". The CIM listener shall include a CIMError header in the response with a value of unsupported-operation.

If a CIM listener receives a CIM export request without this header, it shall not process it. The status code returned by the CIM listener in response to such a request is outside of the scope of this specification.

If a CIM client receives a response to a CIM export request without this header (or if this header has a value that is not "MethodResponse"), it should discard the response and take appropriate measures to publicize that it has received an incorrect response. The details as to how this is done are outside the scope of this specification.

The CIMExport header affords a simple mechanism by which firewall or proxy administrators can make global administrative decisions on all CIM exports.

6.3.5 CIMProtocolVersion

The CIMProtocolVersion header may be present in any CIM message. The header identifies the version of the CIM operations over the HTTP specification in use by the sending entity.

CIMProtocolVersion = "CIMProtocolVersion" ":" 1*DIGIT "." 1*DIGIT

If the header is omitted, then a value of 1.0 must be assumed.

The major and minor revision numbers must be treated as independent integers.

The CIMProtocolVersion $x_1.y_1$ is less than CIMProtocolVersion $x_2.y_2$ if and only if one of the following statements is true:

- x_1 is less than x_2
- x_1 equals x_2, and y_1 is less than y_2

The CIMProtocolVersion $x_1.y_1$ is greater than CIMProtocolVersion $x_2.y_2$ if and only if one of the following statements is true:

- x_1 is greater than x_2,
- x_1 equals x_2, and y_1 is greater than y_2

A CIMProtocolVersion $x_1.y_1$ is within tolerance of CIMProtocolVersion $x_2.y_2$ if:

- x_1 equals x_2, and
- y_1 is less than or equal to y_2

If the CIMProtocolVersion of the CIM message received is within tolerance of the CIMProtocolVersion supported for a CIM server or CIM listener implementation, the receiving implementation shall accept that CIM message. Equivalent CIMProtocolVersion values between CIM server or CIM listener and the CIM

client shall be accepted. The CIM server or CIM listener implementation may reject a CIM message in all other cases. For information about how CIM messages are rejected, see 7.3.

Beyond tolerance considerations, the implementation should reject the received CIM message *only* if the design as defined by the CIMProtocolVersion of the receiving implementation has changed in the declaration of the API, method parameters, or behavior since the design defined by the CIMProtocolVersion of the received CIM message.

6.3.6 CIMMethod

The CIMMethod header shall be present in any CIM Operation Request message that contains a Simple Operation Request.

It shall not be present in any CIM Operation Response message nor in any CIM Operation Request message unless it is a simple operation request. It shall not be present in any CIM export request or response message.

The header identifies the name of the CIM method to be invoked, encoded in an HTTP-safe representation. Firewalls and proxies may use this header to carry out routing and forwarding decisions based on the CIM method to be invoked.

The name of the CIM method within a simple operation request is the value of the `NAME` attribute of the <METHODCALL> or <IMETHODCALL> element.

CIMMethod = "CIMMethod" ":" MethodName

MethodName = CIMIdentifier

If a CIM server receives a CIM operation request for which any one of the following statements is true, then it shall fail the request and return a status of "400 Bad Request". Also, it shall include a CIMError header in the response with a value of `header-mismatch`, subject to the considerations specified in 7.3:

- The CIMMethod header is present, but it has an invalid value.
- The CIMMethod header is not present, but the operation request message is a Simple Operation Request.
- The CIMMethod header is present, but the operation request message is not a simple operation request.
- The CIMMethod header is present and the operation request message is a simple operation request, but the CIMIdentifier value (when unencoded) does not match the unique method name within the simple operation request.

Note that this verification provides a *basic* level of assurance that any intermediate firewall or proxy was not acting on misleading information when it decided to forward the request based on the content of the CIMMethod header. Additional securing of HTTP messages against modification in transit (such as the encryption of the payload or appending of a digital signature thereto) would be required to provide a higher degree of integrity.

6.3.7 CIMObject

The CIMObject header shall be present in any CIM Operation Request message that contains a Simple Operation Request.

It shall not be present in any CIM Operation Response message nor in any CIM Operation Request message unless it is a simple operation Request. It shall not be present in any CIM export request or response message.

The header identifies the CIM object on which the method is to be invoked using a CIM object path encoded in an HTTP-safe representation. This object shall be a class or instance for an extrinsic method or a namespace for an intrinsic method. Firewalls and proxies may use this header to carry out routing and forwarding decisions based on the CIM object that is the target of a method invocation.

CIMObject = "CIMObject" ":" ObjectPath

ObjectPath = CIMObjectPath

The ObjectPath value is constructed by applying the algorithm defined in 6.3.2 to either of the following subelements within the CIM operation request:

- The <LOCALNAMESPACEPATH> subelement of the <IMETHODCALL> element.
- The <LOCALCLASSPATH> or <LOCALINSTANCEPATH> subelement of the <METHODCALL> element.

If a CIM server receives a CIM operation request for which any one of the following statements is true, then it shall fail the request and return a status of "400 Bad Request". Also, it shall include a CIMError header in the response with a value of `header-mismatch`, subject to the considerations specified in 7.3:

- The CIMObject header is present, but it has an invalid value.
- The CIMObject header is not present, but the operation request message is a Simple Operation Request.
- The CIMObject header is present, but the operation request message is not a simple operation request.
- The CIMObject header is present and the operation request message is a simple operation request, but the ObjectPath value does not match the operation request message (where a *match* is defined in 6.3.2).

Note that this verification provides a *basic* level of assurance that any intermediate firewall or proxy is not acting on misleading information when it forwards the request based on the content of the CIMObject header. Additional securing of HTTP messages against modification in transit, such as encrypting the payload or appending a digital signature to it, would be required to provide a higher degree of integrity.

6.3.8 CIMExportMethod

The CIMExportMethod header shall be present in any CIM export request message that contains a simple export request.

This header shall not be present in any CIM export response message nor in any CIM export request message unless it is a simple export request. It shall not be present in any CIM operation request or response message.

The CIMExportMethod header identifies the name of the CIM export method to be invoked, encoded in an HTTP-safe representation. Firewalls and proxies may use this header to carry out routing and forwarding decisions based on the CIM export method to be invoked.

The name of the CIM export method within a simple export request is the value of the NAME attribute of the <EXPMETHODCALL> element.

CIMExportMethod = "CIMExportMethod" ":" ExportMethodName

ExportMethodName = CIMIdentifier

If a CIM listener receives a CIM export request for which any one of the following statements is true, then it shall fail the request and return a status of "400 Bad Request". Also, it shall include a CIMError header in the response with a value of header-mismatch, subject to the considerations specified in 7.3:

- The CIMExportMethod header is present, but it has an invalid value.
- The CIMExportMethod header is not present, but the export request message is a simple export request.
- The CIMExportMethod header is present, but the export request message is not a simple export request.
- The CIMExportMethod header is present and the export request message is a simple export request, but the CIMIdentifier value (when unencoded) does not match the unique method name within the simple export request.

Note that this verification provides a basic level of assurance that any intermediate firewall or proxy is not acting on misleading information when it forwards the request based on the content of the CIMExportMethod header. Additional securing of HTTP messages against modification in transit, such as encrypting the payload or appending a digital signature to it, would be required to provide a higher degree of integrity.

6.3.9 CIMBatch

The CIMBatch header shall be present in any CIM Operation Request message that contains a Multiple Operation Request.

This header shall not be present in any CIM Operation Response message nor in any CIM Operation Request message unless it is a multiple operation request. It shall not be present in any CIM export request or response message.

The CIMBatch header identifies the encapsulated operation request message as containing multiple method invocations. Firewalls and proxies may use this header to carry out routing and forwarding decisions for batched CIM method invocations.

CIMBatch = "CIMBatch" ":"

If a CIM server receives a CIM operation request for which any one of the following statements is true, then it must fail the request and return a status of "400 Bad Request". Also it must include a CIMError header in the response with a value of `header-mismatch`, subject to the considerations specified in 7.3:

- The CIMBatch header is present, but it has an invalid value.
- The CIMBatch header is not present, but the operation request message is a multiple operation request.
- The CIMBatch header is present, but the operation request message is not a multiple operation request.

Note that this verification provides a *basic* level of assurance that any intermediate firewall or proxy is not acting on misleading information when it forwards the request based on the content of the CIMBatch header. Additional securing of HTTP messages against modification in transit, such as encrypting the payload or appending a digital signature to it, would be required to provide a higher degree of integrity.

If a CIM server receives a CIM operation request for which the CIMBatch header is present but the server does not support multiple operations, then it shall fail the request and return a status of "501 Not Implemented". Firewalls or Proxies may also employ this mechanism to compel a CIM client to use simple operation requests rather than multiple operation requests.

A CIM client that receives a response of "501 Not Implemented" to a multiple operation request should resubmit that request as a series of simple operation requests.

6.3.10 CIMExportBatch

The CIMExportBatch header shall be present in any CIM export request message that contains a multiple export request.

It shall not be present in any CIM operation request or response message. Also, it shall not be present in any CIM export response message nor in any CIM export request message unless it is a multiple export request.

The header identifies the encapsulated Export Request Message as containing multiple export method invocations. Firewalls and proxies may use this header to carry out routing and forwarding decisions for batched CIM Export method invocations.

CIMExportBatch = "CIMExportBatch" ":"

If a CIM listener receives a CIM export request for which any one of the following statements is true, then it must fail the request and return a status of "400 Bad Request". Also, it must include a CIMError header in the response with a value of header-mismatch, subject to the considerations specified in Errors:

- The CIMExportBatch header is present, but it has an invalid value.
- The CIMExportBatch header is not present, but the export request message is a multiple export request.
- The CIMExportBatch header is present, but the export request message is not a multiple export request.

Note that this verification provides a *basic* level of assurance that any intermediate firewall or proxy is not acting on misleading information when it forwards the request based on the content of the CIMExportBatch header. Additional securing of HTTP messages against modification in transit, such as encrypting the payload or appending a digital signature to it, would be required to provide a higher degree of integrity.

If a CIM listener receives a CIM export request for which the CIMExportBatch header is present, but the CIM listener does not support multiple exports, then it shall fail the request and return a status of "501 Not Implemented". Firewalls or Proxies may also employ this mechanism to compel a CIM client to use simple rather than multiple export requests.

A CIM client that receives a response of "501 Not Implemented" to a multiple export request should resubmit that request as a series of simple export requests.

6.3.11 CIMError

The CIMError header may be present in any HTTP response to a CIM message request that is not a CIM message response.

It shall not be present in any CIM message response or in any CIM message request.

The CIMError header provides further CIM-specific diagnostic information if the CIM server or CIM listener encounters a fundamental error during processing of the CIM operation request and is intended to assist clients to further disambiguate errors with the same HTTP status code:

```
CIMError  = "CIMError" ":" cim-error
cim-error = "unsupported-protocol-version" |
   "multiple-requests-unsupported" |
   "unsupported-cim-version" |
   "unsupported-dtd-version" |
   "request-not-valid" |
   "request-not-well-formed" |
```

```
        "request-not-loosely-valid" |
        "header-mismatch" |
        "unsupported-operation"
```

6.3.12 CIMRoleAuthenticate

A CIM server may return a CIMRoleAuthenticate header as part of the 401 Unauthorized response along with the WWW-Authenticate header. The CIMRoleAuthenticate header must meet the challenge of indicating the CIM server policy on role credentials.

challenge = "credentialrequired" | "credentialoptional" | "credentialnotrequired"

- A challenge of *credentialrequired* indicates that the CIM server requires that a CIM client must present a credential if it seeks to assume a role.
- A challenge of *credentialoptional* indicates that the credential is optional. If a credential is not sent, the CIM server allows the role assumption if it is permitted for the given user. However, certain operations that require the role credential may not succeed.
- A challenge of *credentialnotrequired* indicates that no credential is required to assume the role.

Absence of the CIMRoleAuthenticate header indicates that the CIM server does not support role assumption. A CIM client should handle each of these cases appropriately.

The challenge does not contain any authorization scheme, realm, or other information. A CIM client should extract this information from the WWW-Authenticate header. This implies that for any given request, the role credentials should use the same scheme as those required for the user credentials.

A CIM server allows role assumption to succeed only if the user is allowed to assume the role. Therefore, even if appropriate credentials are presented, role assumption can fail. If either the user authentication or role assumption fails, the entire authentication operation fails.

To maintain backward compatibility, a CIM server that supports role assumption must allow user authentication even if no role is specified.

6.3.13 CIMRoleAuthorization

The CIMRoleAuthorization header is supplied along with the normal authorization header that the CIM client populates to perform user authentication. If the CIM client needs to perform role assumption and the server challenge is credentialrequired, the CIMRoleAuthorization header must be supplied with the appropriate credentials. The credentials supplied as part of the CIMRoleAuthorization header must use the same scheme as those specified for the authorization header, as specified in RFC 2617. Therefore, both Basic and Digest authentication are possible for the role credential.

If the CIM client wishes to assume a role but does not wish to supply role credentials for server challenge credentialoptional or credentialnotrequired, the CIMRoleAuthorization header must set the auth-scheme field as specified in RFC 2617 to be "role". The auth-param must contain the role name.

A CIM server that supports roles must be capable of handling the presence of credentials in the CIMRoleAuthorization header (that is auth-scheme not set to "role") regardless of whether it is expecting credentials or not. It may choose to ignore these credentials.

6.3.14 CIMStatusCodeDescription

If a CIM product includes the CIMStatusCode trailer, it may also include the CIMStatusCodeDescription trailer. The value of this trailer is a string describing the nature of the error. A CIM product shall not include this trailer if the CIMStatusCode trailer is not present.

6.3.15 WBEMServerResponseTime

The WBEMServerResponseTime header may be present in any CIM response message. If it is present, the header shall contain a measure, specified in microseconds, of the elapsed time required by the CIM server to process the request and create a response. Specifically, WBEMServerResponseTime describes the time elapsed since the CIM server received the CIM request message and the associated CIM response message was ready to send to the CIM client.

WBEMServerResponseTime = "WBEMServerResponseTime" ":", where the response time must be representable as a 64-bit unsigned integer value. If the actual elapsed time exceeds the maximum representable value, then the maximum value shall be returned. If the actual elapsed time is less than 1 microsecond, then a 0 shall be returned.

Although a CIM client may ignore the WBEMServerResponseTime header, it shall allow this header to be included in a response.

7 HTTP Requirements and Usage

This clause describes HTTP support and the use of standard headers.

7.1 HTTP Support

It is recommended that CIM clients, CIM servers, and CIM listeners support HTTP/1.1. CIM clients, CIM servers, and CIM listeners may support HTTP/1.0. CIM clients, CIM servers, and CIM listeners shall not be limited to any version of HTTP earlier than 1.0.

CIM products that use extension headers as defined in this specification shall conform to the requirements defined in RFC 2774 for their use.

7.2 Use of Standard Headers

Unless otherwise stated in this specification, CIM products shall comply with the requirements on the use of headers described in RFC 1945, RFC 2068. This clause defines only *additional* requirements on CIM products with respect to the use of standard HTTP headers (RFC 1945, RFC 2068) in a CIM message.

Note that CIM products should not use headers defined in RFC 2068 but deprecated from RFC 2616 (for example, Public, Content-Base).

7.2.1 Accept

If a CIM client includes an Accept header in a request, it shall specify a value that allows the server to return an entity body of "text/xml" or "application/xml" in the response.

A CIM server or CIM listener shall accept any value for this header stating that "text/xml" or "application/xml" is an acceptable type for a response entity. A CIM server or CIM listener should return "406 Not Acceptable" if the Accept header indicates that neither of these content types is acceptable.

If a CIM server or CIM listener accepts a request to return an entity of a type other than "text/xml" or "application/xml", the nature of the response is outside the scope of this specification.

7.2.2 Accept-Charset

If a CIM client includes an Accept-Charset header in a request, it shall specify a value that allows the CIM server or CIM listener to return an entity body using the character set "UTF-8".

A CIM server or CIM listener shall accept any value for this header asserting that "UTF-8" is an acceptable character set for a response entity. If the client does not provide an Accept-Charset, then "UTF-8" should be assumed by the CIM server or CIM listener.

```
Accept-Charset: UTF-8
```

A CIM server or CIM listener shall return "406 Not Acceptable" if the character set requested in the Accept-Charset header is not supported.

If a CIM server or CIM listener accepts a request to return an entity using a character set other than "UTF-8", the behavior of the subsequent CIM client and CIM server interaction is outside the scope of this specification. See 7.8 for details.

7.2.3 Accept-Encoding

If a CIM client includes an Accept-Encoding header in a request, it shall specify a q value that allows the CIM server or CIM listener to use the "Identity" encoding. The value shall be greater than 0 or not specified.

```
Accept-Encoding: Identity
Accept-Encoding: Identity; q=1.0
```

A CIM server or CIM listener shall accept any value for this header asserting that "Identity" is an acceptable encoding for the response entity.

A CIM server or CIM listener shall return "406 Not Acceptable" if the Accept-Encoding header indicates that the requested encoding is not acceptable.

7.2.4 Accept-Language

If a CIM client includes an Accept-Language header in a request, it shall request a language-range, special-range, or both. The CIM client shall also allow any language to be returned if the requested languages cannot be supported. This is accomplished by including the special-range, "*". The CIM client may request multiple languages. Each language has equal priority, unless a q value is provided.

```
Accept-Language: zh, *
Accept-Language: zh;q=1.0, en;q=.7, *
```

Each CIM element in the response should be localized in only one language. A CIM element shall not be duplicated in the response because it is localized in more than one language.

CIM servers may support multiple languages. A CIM product shall interpret the use of the special-range value, "*", as a request to return the response content using the default language defined for the target processing the request. Multiple targets, with different default language settings, may participate in the construction of a response. (See RFC 2616 section 3.10 and ISO 639-1.)

See 7.8 for more information.

7.2.5 Accept-Ranges

CIM clients shall not include the Accept-Ranges header in a request. A CIM server or CIM listener shall reject a request that includes an Accept-Range header with a status of "406 Not Acceptable".

7.2.6 Allow

If a CIM server or CIM listener is returning a "405 Method Not Allowed" response to a CIM message request, then the Allow header shall include either M-POST or POST. Whether it includes any other HTTP methods is outside the scope of this specification.

7.2.7 Authorization

See 7.4 for details.

7.2.8 Cache-Control

Generally, a CIM message request may consist of a mixture of CIM method invocations, some of which may be eminently able to cache (for example, the manufacturer label on a disk drive) and some of which may be decidedly impossible to cache (for example, format a disk drive).

Furthermore, the encapsulation of such multiple method invocations in an HTTP POST or M-POST means that if a CIM message request has any effect on an HTTP cache it is likely to be one of invalidating cached responses for the target CIM server or CIM listener. Indeed, HTTP/1.1 stipulates that by default POST responses cannot be cached unless the server indicates otherwise using an appropriate Cache-Control or Expires header.

For these reasons, CIM message responses should not be considered as able to be cached. A CIM server or CIM listener should not include a Cache-Control header in a CIM message response that might indicate to a cache that the response can be cached.

If the CIM server or CIM listener is responding to a CIM message request conveyed in an M-POST request, then in accordance with RFC 2774 the CIM server or CIM listener shall include a no-cache control directive to prevent inadvertent caching of the "Ext" header, as in the following example:

EXAMPLE

```
HTTP/1.1 200 OK
Ext:
Cache-Control: no-cache
...
```

7.2.9 Connection

The following courses of action are recommended for connections:

- CIM clients should avoid the use of the "Connection: close" header unless it is known in advance that this is the only request likely to be sent out on that connection.
- CIM servers and CIM listener support persistent connections wherever possible.

Timeout mechanisms should be employed to remove idle connections on the CIM client, CIM server, and CIM listener. The details of timeout mechanisms are outside the scope of this specification. Clients should be cautious in retrying requests, especially if they are not idempotent (for example, method invocation).

CIM clients, CIM servers, and CIM listeners should support pipelining (see RFC 2068, section 1.1.2.2) if possible, but be aware of the requirements defined in RFC 2068. In particular, attention is drawn to the requirement from RFC 2068 that clients not pipeline requests using non-idempotent methods or non-idempotent sequences of methods. A client that needs to send a non-idempotent request should wait to send that request until it receives the response status for the previous request.

7.2.10 Content-Encoding

If a CIM client includes a Content-Encoding header in a request, it should specify a value of "identity", unless there is good reason to believe that the server or listener can accept another encoding.

7.2.11 Content-Language

The Content-Language entity-header field of a CIM message describes the natural language(s) of the intended audience of the content.

A CIM message may contain a Content-Language header. The value of the Content-Language header in a CIM response message shall be consistent with the Accept-Language values specified in the corresponding CIM request message. If the CIM server cannot determine one or more of the content languages used to construct the response, then the Content-Language entity shall not be returned.

Multiple targets using different Content-Language values may participate in constructing a response. The Content-Language field shall reflect all Content-Language values used to construct the response. The content of a CIM message may contain elements in languages not listed in the Content-Language field.

```
Content-Language: en
```

See 7.8 for details.

7.2.12 Content-Range

CIM clients, CIM servers, and CIM listeners shall not use this header.

7.2.13 Content-Type

CIM clients, CIM servers, and CIM listeners shall specify (and accept) a media type for the Content-Type header of either "text/xml" or "application/xml" as defined in RFC 2376. In addition, they may specify and shall accept a "charset" parameter as defined in RFC 2616. If a "charset" parameter is specified, it shall have the value "utf-8" either with or without surrounding double quotes. The sending side should use the form without double quotes. The receiving side shall support both forms. If a "charset" parameter is not specified, the receiving side shall assume "utf-8" as a default.

Examples of valid Content-Type headers are:

```
Content-type: text/xml
Content-type: text/xml; charset=utf-8
Content-type: text/xml; charset="utf-8"
Content-type: application/xml
Content-type: application/xml; charset=utf-8
Content-type: application/xml; charset="utf-8"
```

7.2.14 Expires

For the reasons described in 7.2.8, a CIM server or CIM listener shall not include an Expires header in a CIM message response that might indicate to a cache that the response can be cached.

7.2.15 If-Range

CIM clients, CIM servers, and CIM listeners shall not use this header.

7.2.16 Proxy-Authenticate

See 7.4 for details.

7.2.17 Range

CIM clients, CIM servers, and CIM listeners shall not use this header.

7.2.18 WWW-Authenticate

See 7.4 for details.

7.3 Errors and Status Codes

This clause defines how CIM servers and CIM listeners shall handle errors that occur in processing a CIM message request. This specification does not introduce any new HTTP response status codes.

If there is an error in processing the HTTP Request-Line or standard HTTP headers, the CIM server or CIM listener shall take appropriate action as dictated by its conformance to the relevant version of HTTP (RFC 1945, RFC 2068).

Otherwise, if there are any mandatory extension declarations that the server does not support it shall respond with a "510 Not Extended" status according to RFC 2774.

Otherwise, the request shall be processed in accordance with the relevant version of HTTP (RFC 1945, RFC 2068) and the additional rules defined in this document.

Assuming that the HTTP request is otherwise correct, the CIM server or CIM listener shall use the following status codes when processing the CIM extension headers:

- 501 Not Implemented

  This status code indicates that one of the following situations occurred:

  – The CIMProtocolVersion extension header in the request specifies a version of the CIM mapping onto HTTP that is not supported by this CIM server or CIM listener. The CIM server or CIM listener shall include a CIMError header in the response with a value of `unsupported-protocol-version`.

  – The client specified a Multiple Operation Request (or multiple Export Request), and the CIM server (or CIM listener) does not support such requests. The CIM server or CIM listener shall include a CIMError header in the response with a value of `multiple-requests-unsupported`.

  – The CIMVERSION attribute in the message request is not set to a proper value. The CIMVERSION attribute shall be in the form of "M.N", where M is the major revision of the specification in numeric form and N is the minor revision in numeric form. The version shall be at "2.0" or greater (for example, "2.0" or "2.3"). The CIM server or CIM listener shall include a CIMError header in the response with a value of `unsupported-cim-version`.

  – The DTDVERSION attribute in the message request is not set to a proper value. The DTDVERSION attribute shall be in the form of "M.N", where M is the major revision of the specification in numeric form and N is the minor revision in numeric form. The version shall be at "2.0" or greater (for example, "2.0" or "2.1"). The CIM server or CIM listener shall include a CIMError header in the response with a value of `unsupported-dtd-version`.

- 401 Unauthorized

  The CIM server or CIM listener is configured to require that a client authenticate itself before it can issue CIM message requests to the server or listener.

- 403 Forbidden

  The CIM server or CIM listener does not allow the client to issue CIM message requests. The CIM server or CIM listener may alternatively respond with a "404 Not Found" if it does not wish to reveal this information to the client.

- 407 Proxy Authentication Required

  The CIM server or CIM listener is configured to require that the proxy authenticate itself before it can issue CIM message requests on behalf of a CIM client to the server or listener.

Assuming that the CIM extension headers are correct, a validating CIM server or CIM listener (one that enforces the validity of the CIM message request with respect to the CIM XML DTD) shall use the following status code when processing the entity body containing the CIM message request:

- 400 Bad Request

  The entity body defining the CIM message request is not well-formed or not valid with respect to the CIM XML DTD. The CIM server or CIM listener shall include a CIMError header in the response with a value of `request-not-well-formed` or `request-not-valid` (as appropriate).

A loosely-validating CIM server or CIM listener only enforces the CIM message request to be loosely valid. Therefore, it may reject a CIM message request that is not loosely valid with an HTTP status code of 400 (Bad Request) before further processing. In this case, the CIM server or CIM listener shall include a CIMError header in the response with a value of `request-not-loosely-valid`.

A loosely-validating CIM server or CIM listener shall reject a CIM message request that is not well-formed with an HTTP status code of 400 (Bad Request). In this case, the CIM server or CIM listener shall include a CIMError header in the response with a value of `request-not-well-formed`.

A loosely-validating CIM server or CIM listener shall not reject an invalid CIM message request that is loosely valid in the XML sense.

A loosely-validating CIM server or CIM listener shall ultimately signal an error to the CIM client if the CIM message request is not loosely valid. That is, the request is missing required content or the required content is incorrect, such as an attribute with an invalid value according to the CIM XML DTD. It is not mandated to reject a CIM message request before processing, for to do otherwise would compel the server or listener to check the complete request before processing can begin and this would be as expensive as requiring the server or listener to fully validate the request. Therefore, a loosely-validating server or listener may elect to begin processing the request and issuing a response (with an HTTP success status code) before verifying that the entire request is loosely valid.

A CIM client may use the CIMValidation header mechanism to determine whether a CIM server or CIM listener is validating or loosely-validating.

Assuming that the CIM message request is correctly formed as previously described, the CIM server or CIM listener shall process the request accordingly and return a CIM message response.

The entity body shall be a correct CIM message response for that request.

If the CIM message response contains an entity that is a simple message response, then the response status shall be "200 OK". Otherwise, the response status shall be "207 Multistatus".

7.4 Security Considerations

CIM clients, CIM servers, and CIM listeners may elect not to use authentication, but only in environments where lack of security is not an issue.

Basic authentication is described in RFC 1945 and RFC 2068. Digest authentication is defined in RFC 2069. Both authentication schemes are covered in a consolidated document (RFC 2617), which also makes a number of improvements to the original specification of digest authentication.

Basic authentication provides a very rudimentary level of authentication, with the major weakness that the client password is sent over the wire in unencrypted form.

For this reason, CIM clients, CIM servers, and CIM listeners shall not use basic authentication other than in the context of a highly secure environment (for example, in conjunction with SSL or in a physically secure private network). CIM servers and CIM listeners shall not send basic authentication credentials in a WWW-Authenticate header other than in the context of a highly secure environment.

Conforming applications should support the digest authentication scheme. Because digest authentication verifies that both parties share a common secret without having to send that secret in the clear, it is more secure than basic authentication. However, CIM clients, CIM servers, and CIM listeners that require more robust protection should use encryption mechanisms such as SSL or SHTTP.

CIM clients, CIM servers, and CIM listeners using basic or digest authentication shall comply with the requirements set forth in RFC 1945, RFC 2068, RFC 2069, and RFC 2617. This specification describes only *additional* requirements on CIM clients, CIM servers, and CIM listeners when these authentication schemes are used.

CIM servers and CIM listeners should require that CIM clients authenticate themselves. This specification does not mandate this because it is recognized that in some circumstances the CIM server or CIM listener may not require or wish the overhead of employing authentication. CIM servers and CIM listeners should carefully consider the performance/security tradeoffs in determining how often to issue challenges to CIM clients.

A CIM server or CIM listener that returns a "401 Unauthorized" response to a CIM message request should include in the WWW-Authenticate response-header either the "Basic" or "Digest" authentication values (but not both). This specification does not mandate use of basic or digest authentication because it is recognized that in some circumstances the CIM server or CIM listener may use bespoke authentication mechanisms not covered by RFC 2617. Similar considerations apply to the use of the Proxy-Authorization header in "407 Proxy Authentication Required".

7.5 Determining CIM Server Capabilities

If a CIM server can return capabilities information, there are two techniques for returning this information as defined in this specification:

- The preferred technique is through the use of the classes defined in 7.5.1.
- Alternatively, use of the HTTP OPTIONS method as defined in 7.5.2 is allowed because historically it is the original technique defined for requesting capabilities information.

Use of the CIM classes defined in 7.5.1 is strongly encouraged and it is expected that this method will be enhanced and extended in the future to provide more capabilities information. The future use of the HTTP OPTIONS method to determine capabilities of CIM servers is discouraged. It will probably not be expanded significantly and may be reviewed for possible deprecation in the next major revision of this specification.

7.5.1 Determining CIM Server Capabilities through CIM Classes

A set of CIM classes is defined specifically to return CIM server capabilities information as follows:

- CIM_ObjectManager

  This class is a type of CIM_Service that defines the capabilities of the target CIM server.

- CIM_ObjectManagerCommunicationMechanism

  This class describes access to the target CIM server. It defines the capabilities of the CIM server that are available through the target Object Manager Communication mechanism. A CIM server is allowed to support different capabilities through different communication mechanisms.

- CIM_CIMXMLCommunicationMechanism

  This class specializes on ObjectManagerCommunicationMechanism, adding properties specific to the CIM-XML encoding and protocol.

- CIM_CommMechanismForManager

This association between CIM_ObjectManager and CIM_ObjectManagerCommunicationMechanism defines the communications protocols (and corresponding capabilities) available on the target CIM server through the ObjectManagerCommunicationMechanism instances.

A CIM client may use instances of these CIM classes to determine the CIM capabilities (if any) of the target CIM server. A CIM server that supports capabilities determination through these classes shall support at least the Enumerate Instance and Get Instance operations for the classes. The use of other methods of the basic read profile is optional. A CIM server that does not support the determination of CIM capabilities through these classes shall return CIM_ERR_NOT_FOUND to any instance or class request on these classes. These classes shall not be used for reporting any other information than capabilities of the target CIM server.

To provide interoperability, the CIM object manager classes shall exist in a well-known namespace. Because there is no discovery mechanism that can define this well-known namespace to a CIM client, it shall be one or more predefined namespaces. Therefore, to ensure interoperability, we recommend that pending future extensions of the WBEM specifications include discovery tools that define a namespace for these classes in a CIM server; these predefined namespaces should exist in either the root namespace or in the /root/CIMV2 namespace.

A CIM server that supports capabilities reporting through these classes shall correctly report the current actual capabilities of the target CIM server and shall report on all capabilities defined. A CIM server is allowed to report "none" if the capability does not exist or "unknown" if the status of the capability is unknown at the time of the request for those properties where these choices exist in the properties definition. Because the CIM_ObjectManager object provides information on the target CIM server, only a single instance of this class may exist in a CIM server.

The capabilities to be reported through the CIM_ObjectManagerCommunicationMechanism are as follows:

- CommunicationMechanism property, which defines the communication protocol for the CommunicationMechanism object. A compliant CIM server shall include the CIM-XML protocol for at least one ObjectManagerCommunicationMechanism instance.
- ProfilesSupported property, which defines the functional profiles supported as defined in clause 5.3.4. All CIM servers shall support the basic-read functional group. All CIM clients may assume that any CIM server supports the basic-read functional group. The list of functional groups returned by a CIM server shall contain the basic-read group and shall not contain duplicates. CIM clients shall ignore duplicate entries in the functional-group list. If a functional group is included in the list, the CIM client shall assume that all other groups on which it depends (according to the rules defined in 5.3.4) are also supported. A CIM server should not explicitly include a functional group in the list whose presence may be inferred implicitly by a dependency. Support for a functional group does not imply that any method from that group will always succeed. Rather, the absence of the functional group from this list (whether explicit or implied) indicates to the CIM client that methods in that group will never succeed.
- MultipleOperationsSupported property, which defines whether the target CIM server supports multiple operation requests as defined in 5.3.2. `True` in this property indicates that the server can accept and process multiple operation requests. `False` indicates that the CIM server can accept only single operation requests.
- AuthenticationMechanismsSupported property, which defines the authentication mechanisms supported by the target CIM server as defined in 7.4.
- PulledEnumerationClosureOnExceedingServerLimits property, which indicates whether the CIM server supports closure of Pulled Enumeration sessions based upon exceeding server limits.
- PulledEnumerationContinuationOnErrorSupported property, which indicates whether the CIM server supports continuation on error for Pulled enumerations.

- PulledEnumerationMinimumOperationTimeout (PulledEnumerationMaximumOperationTimeout) property, which indicates the minimum (maximum) operation timeout allowed by the CIM server for Pulled enumerations.

Compliant CIM servers may report additional capabilities for the CommunicationMechanism Functional Profiles, QueryLanguageSupported, and AuthenticationMechanismSupported by defining the "other" enumeration in the property and returning additional information in the associated "additional capabilities" property.

7.5.2 Determining CIM Server Capabilities through the HTTP Options

A CIM client may use the OPTIONS method to determine the CIM capabilities (if any) of the target server. A CIM server may support the OPTIONS method (for example, CIM servers supporting only HTTP/1.0 would not support OPTIONS).

To support the ability for a server to declare its CIM capabilities independently of HTTP, the DMTF intends to publish a CIM schema (in a separate document) describing such capabilities. In particular, this mechanism would allow servers that do not support the OPTIONS method to declare their capabilities to a client.

If a CIM server supports the OPTIONS method, it should return the following headers in the response:

- CIM Extension Header CIMProtocolVersion, which provides a way for a client to discover the version of the CIM HTTP mapping supported by the CIM server.
- CIM Extension Header CIMSupportedFunctionalGroups, which provides a way for a client to discover the CIM operations supported by the CIM server.
- CIM Extension Header CIMSupportsMultipleOperations, which provides a way for the client to discover whether the CIM server can support Multiple Operation Requests.

In addition, if the CIM server supports one or more query languages, it should return the following header in the response:

- CIM Extension Header CIMSupportedQueryLanguages, which allows the client to discover the query languages supported by the CIM server.

In addition, if the CIM server runs in a fixed validation mode, it should return the following header in the response:

- CIM Extension Header CIMValidation, which allows the client to determine whether the CIM server is strictly validating or loosely validating.

If the CIMProtocolVersion, CIMSupportedFunctionalGroups, CIMSupportsMultipleOperations, CIMValidation, or CIMSupportedQueryLanguages extension headers are included in the response, the CIM server shall declare them as optional extension headers using the "Opt" header defined in RFC 2774.

The full format of the "Opt" header declaration for this specification is:

```
Opt             = "Opt" ":" "http://www.dmtf.org/cim/mapping/http/v1.0"
                  ";" "ns" "=" header-prefix
header-prefix   = 2*DIGIT
```

This header-prefix should be generated at random on a per-HTTP message basis and should not necessarily be a specific number.

EXAMPLE: The following is a fragment of a legitimate OPTIONS response from a CIM server:

```
HTTP/1.1 200 OK
Opt: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=77
```

```
77-CIMProtocolVersion: 1.0
77-CIMSupportedFunctionalGroups: basic-read
77-CIMBatch
77-CIMSupportedQueryLanguages: wql
...
```

7.5.2.1 CIMSupportedFunctionalGroups

The CIMSupportedFunctionalGroups extension header should be returned by a CIM server in any OPTIONS response. It shall not be returned in any other scenario.

This header is defined as follows:

```
CIMSupportedFunctionalGroups = "CIMSupportedFunctionalGroups" ":" 1#functional-
group
functional-group = "basic-read" |
   "basic-write" |
   "schema-manipulation" |
   "instance-manipulation" |
   "qualifier-declaration" |
   "association-traversal" |
   "query-execution"
```

The functional group definitions correspond directly to those listed in 5.4.3. All CIM servers shall support the basic-read functional group. All CIM clients may assume that any CIM server supports the basic-read functional group.

The list of functional groups returned by a CIM server shall contain the basic-read group and shall not contain any duplicates. CIM clients shall ignore any duplicate entries in the functional-group list.

If a functional group is included in the list, the CIM client shall assume that all other groups on which it depends (according to the rules defined in 5.4.3) are also supported. A CIM server should not explicitly include a functional group in the list if the presence of the group may be implied by a dependency.

EXAMPLE: The following HTTP response message indicates that the CIM server supports instance-manipulation, association-traversal, basic-write, and basic-read.

```
HTTP/1.1 200 OK
Opt: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=77
77-CIMProtocolVersion: 1.0
77-CIMSupportedFunctionalGroups: association-traversal, instance-manipulation
...
```

Support for a functional group does *not* imply that any method from that group will always succeed. Rather, the absence (whether explicit or implied) of the functional group from this header is an indication to the CIM client that methods in that group will *never* succeed.

7.5.2.2 CIMSupportsMultipleOperations

The CIMSupportsMultipleOperations extension header shall be returned in an OPTIONS response by any CIM server that supports Multiple Operation Requests. It shall not be returned in any other circumstances.

This header is defined as follows:

```
CIMSupportsMultipleOperations  = "CIMSupportsMultipleOperations"
```

The presence of this header indicates that the server can accept and process multiple operation requests. The absence of this header indicates that the server can only accept and process Simple Operation Requests.

7.5.2.3 CIMSupportedQueryLanguages

The CIMSupportedQueryLanguages extension header should be returned by a CIM server that supports at least one query language in any OPTIONS response. It shall not be returned in any other scenario.

This header is defined as follows (`token` has the meaning conferred by RFC 2068, section 2.2):

```
CIMSupportedQueryLanguages  = "CIMSupportedQueryLanguages" ":" 1#query-language
query-language     = token
```

The query-language value shall be treated as case-insensitive. It is anticipated that query languages will be submitted for approval to the DMTF, and each submission will define a value for this token to enable it to be specified in this header.

7.5.2.4 CIMValidation

The CIMValidation extension header may be returned by a CIM server to provide information about the level of validation of CIM Operation Request messages.

This header is defined as follows:

```
CIMValidation = "CIMValidation" ":" validation-level
validation-level = "validating" |
   "loosely-validating"
```

A validation-level of `validating` indicates that the CIM server always applies strict validation of each CIM operation request. A validation-level of `loosely-validating` indicates that the CIM server applies loose validation of each CIM operation request.

In the absence of this header, a CIM client should assume that the CIM server operates in strict validation mode.

7.6 Other HTTP Methods

This specification does not in any way define or constrain the way a CIM client, CIM server, or CIM listener uses any HTTP method other than those explicitly cited.

7.7 Discovery and Addressing

The target URI of the CIM Operation Request is defined as the location of the CIM server. This specification does not constrain the format of this URI other than it should be a valid URI (RFC 2396) for describing an HTTP-addressable resource.

An HTTP server that supports the CIM mapping defined in this specification, and which supports the OPTIONS method, should include the following CIM extension header in an OPTIONS response:

- CIMOM

This header is defined as follows:

```
CIMOM        = "CIMOM" ":" (absoluteURI | relativeURI)
```

The terms absoluteURI and relativeURI are taken from RFC 2068; they indicate the location of the CIM server for this HTTP server.

If the CIMOM extension header is included in the response, the CIM server shall declare it an optional extension header as described in 7.5.

A CIM client that needs to communicate with a CIM server on an HTTP server should try an OPTIONS request to that HTTP server. If the OPTIONS request fails or the response does not include the CIM-

CIMOM extension header, the CIM client may assume that the value of CIM-CIMOM is the relative URI **cimom**.

The DMTF recommends the use of the following well-known IP ports in compliant CIM servers. This is a recommendation and not a requirement. The DMTF has registered these port addresses with IANA, so they are for the exclusive use of the DMTF.

- CIM-XML (HTTP) 5988/tcp
- CIM-XML (HTTP) 5988/udp
- CIM-XML (HTTPS) 5989/tcp
- CIM-XML (HTTPS) 5989/udp

Other discovery mechanisms are outside the scope of this version of the specification.

EXAMPLE 1:

This example shows an HTTP server located at http://www.dmtf.org/ issuing an OPTIONS response to an HTTP client to indicate that its CIM server is located at http://www.dmtf.org/access/cimom.

```
HTTP/1.1 200 OK
Opt: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=48
48-CIMOM: /access/cimom
...
```

EXAMPLE 2:

If an HTTP server located at http://www.dmtf.org/ responds with a "501 Not Implemented" to an OPTIONS request from a CIM client, the CIM client may then try to contact the CIM server at http://www.dmtf.org/cimom.

7.8 Internationalization Considerations

This clause defines the capabilities of the CIM HTTP mapping with respect to IETF policy guidelines on character sets and languages (RFC 2277).

In this specification, human-readable fields are contained within a response or request entity body. In all cases, a human-readable content is encoded using XML (which explicitly provides for character set tagging and encoding) and requires that XML processors read XML elements encoded, at minimum, using the UTF-8 (RFC 2279) encoding of the ISO 10646 multilingual plane.

Properties that are not of type string or string array shall not be localized.

Because keys are writeable only on instantiation, key values shall not be localized. See DSP0004 for details.

XML examples in this specification demonstrate the use of the charset parameter of the Content-Type header, as defined in RFC 2616, as well as the XML attribute on the <?xml> processing instruction, which together provide charset identification information for MIME and XML processors. This specification mandates that conforming applications shall support at least the "UTF-8" charset encoding (RFC 2277) in the Content-Type header and shall support the "UTF-8" value for the XML `encoding` attribute.

XML also provides a language tagging capability for specifying the language of the contents of a particular XML element, based on use of IANA registered language tags (RFC 1766) in combination with ISO 639-1, in the xml:lang attribute of an XML element to identify the language of its content and attributes. Section 3.10 of RFC 2616 defines how the two-character ISO 639-1 language code is used as the primary-tag. The language-tag shall be registered by IANA.

DSP0201 and DSP0203 declare this attribute on any XML elements. Therefore, conforming applications should use this attribute when specifying the language in which a particular element is encoded for string and string array attributes and qualifiers. See the usage rules on this element, which are defined by the

World Wide Web Consortium in *XML 1.0, second edition*. The attribute may be scoped by the instance or a class and should not be scoped by a property because instances or classes should be localized in one language.

This specification defines several names of HTTP headers and their values. These names are constructed using standard encoding practices so that they always have an HTTP-safe ASCII representation. Because these headers are not usually visible to users, they do not need to support encoding in multiple character sets.

DSP0201 and DSP0203 introduce several XML element names. Similarly, these names are not visible to an end user and do not need to support multiple character set encodings.

The CIM model (DSP0004) defines the subset of the Unicode character set that can be used to name CIM elements (classes, instances, methods, properties, qualifiers, and method parameters). In general, these characters appear as the value of XML attributes or as element content and are not displayed to end users.

Negotiation and notification of language settings is effected in this mapping using the standard Accept-Language and Content-Language headers defined in RFC 2068.

ANNEX A
(Informative)

Examples of Message Exchanges

This annex illustrates the protocol defined in this document with examples of valid HTTP request/response exchanges. The examples are for illustration purposes only and are not considered part of the specification.

For clarity, additional white space is included in the examples, but such white space is not an intrinsic part of such XML documents.

A.1 Retrieval of a Single Class Definition

The following HTTP request illustrates how a client requests the class CIM_VideoBIOSElement.

```
M-POST /cimom HTTP/1.1
HOST: http://www.myhost.com/
Content-Type: application/xml; charset=utf-8
Content-Length: xxxx
Man: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=73
73-CIMOperation: MethodCall
73-CIMMethod: GetClass
73-CIMObject: root/cimv2

<?xml version="1.0" encoding="utf-8" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.0">
   <MESSAGE ID="87872" PROTOCOLVERSION="1.0">
      <SIMPLEREQ>
         <IMETHODCALL NAME="GetClass">
          <LOCALNAMESPACEPATH>
               <NAMESPACE NAME="root"/>
               <NAMESPACE NAME="cimv2"/>
          </LOCALNAMESPACEPATH>
          <IPARAMVALUE NAME="ClassName"><CLASSNAME
NAME="CIM_VideoBIOSElement"/></IPARAMVALUE>
          <IPARAMVALUE NAME="LocalOnly"><VALUE>FALSE</VALUE></IPARAMVALUE>
         </IMETHODCALL>
      </SIMPLEREQ>
   </MESSAGE>
</CIM>
```

Following is an HTTP response to the preceding request indicating success of the requested operation. For clarity of exposition, the complete definition of the returned <CLASS> element is not shown.

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8
Content-Length: xxxx
Ext:
Cache-Control: no-cache
Man: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=73
73-CIMOperation: MethodResponse
```

```
<?xml version="1.0" encoding="utf-8" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.0">
   <MESSAGE ID="87872" PROTOCOLVERSION="1.0">
      <SIMPLERSP>
         <IMETHODRESPONSE NAME="GetClass">
          <IRETURNVALUE>
             <CLASS NAME="CIM_VideoBIOSElement" SUPERCLASS="CIM_SoftwareElement">
             ...
             </CLASS>
          </IRETURNVALUE>
         </IMETHODRESPONSE>
      </SIMPLERSP>
   </MESSAGE>
</CIM>
```

A.2 Retrieval of a Single Instance Definition

The following HTTP request illustrates how a client requests the instance MyClass.MyKey="S3".

```
M-POST /cimom HTTP/1.1
HOST: http://www.myhost.com/
Content-Type: application/xml; charset=utf-8
Content-Length: xxxx
Man: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=73
73-CIMOperation: MethodCall
73-CIMMethod: GetInstance
73-CIMObject: root%2FmyNamespace
```

```
<?xml version="1.0" encoding="utf-8" ?>
<CIM CIMVERSION="2.0" DTDVERSION="1.1">
   <MESSAGE ID="87855" PROTOCOLVERSION="1.0">
      <SIMPLEREQ>
         <IMETHODCALL NAME="GetInstance">
          <LOCALNAMESPACEPATH>
            <NAMESPACE NAME="root"/>
            <NAMESPACE NAME="myNamespace"/>
          </LOCALNAMESPACEPATH>
          <IPARAMVALUE NAME="InstanceName">
             <INSTANCENAME CLASSNAME="MyClass">
                   <KEYBINDING NAME="MyKey"><KEYVALUE>S3</KEYVALUE></KEYBINDING>
             </INSTANCENAME>
          </IPARAMVALUE>
          <IPARAMVALUE NAME="LocalOnly"><VALUE>FALSE</VALUE></IPARAMVALUE>
         </IMETHODCALL>
      </SIMPLEREQ>
   </MESSAGE>
</CIM>
```

Following is an HTTP response to the preceding request indicating an error because the specified instance is not found.

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8
Content-Length: xxxx
Ext:
```

```
    Cache-Control: no-cache
    Man: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=73
    73-CIMOperation: MethodResponse

    <?xml version="1.0" encoding="utf-8" ?>
    <CIM CIMVERSION="2.0" DTDVERSION="2.0">
       <MESSAGE ID="87885" PROTOCOLVERSION="1.0">
          <SIMPLERSP>
             <IMETHODRESPONSE NAME="GetInstance">
              <ERROR CODE="6" DESCRIPTION="Instance of MyClass not found"/>
             </IMETHODRESPONSE>
          </SIMPLERSP>
       </MESSAGE>
    </CIM>
```

A.3 Deletion of a Single Class Definition

The following HTTP request illustrates how a client deletes the class CIM_VideoBIOSElement.

```
    M-POST /cimom HTTP/1.1
    HOST: http://www.myhost.com/
    Content-Type: application/xml; charset=utf-8
    Content-Length: xxxx
    Man: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=73
    73-CIMOperation: MethodCall
    73-CIMMethod: DeleteClass
    73-CIMObject: root/cimv2

    <?xml version="1.0" encoding="utf-8" ?>
    <CIM CIMVERSION="2.0" DTDVERSION="2.0">
       <MESSAGE ID="87872" PROTOCOLVERSION="1.0">
          <SIMPLEREQ>
             <IMETHODCALL NAME="DeleteClass">
                <LOCALNAMESPACEPATH>
                 <NAMESPACE NAME="root"/>
                 <NAMESPACE NAME="cimv2"/>
                </LOCALNAMESPACEPATH>
                <IPARAMVALUE NAME="ClassName"><CLASSNAME
    NAME="CIM_VideoBIOSElement"/></IPARAMVALUE>
             </IMETHODCALL>
          </SIMPLEREQ>
       </MESSAGE>
    </CIM>
```

Following is an HTTP response to the preceding request indicating failure of the preceding operation due to the inability to delete instances of the class.

```
    HTTP/1.1 200 OK
    Content-Type: application/xml; charset=utf-8
    Content-Length: xxxx
    Ext:
    Cache-Control: no-cache
    Man: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=73
    73-CIMOperation: MethodResponse

    <?xml version="1.0" encoding="utf-8" ?>
    <CIM CIMVERSION="2.0" DTDVERSION="2.0">
```

```
      <MESSAGE ID="87872" PROTOCOLVERSION="1.0">
         <SIMPLERSP>
            <IMETHODRESPONSE NAME="DeleteClass">
             <ERROR CODE="9" DESCRIPTION="Class has non-deletable instances"/>
            </IMETHODRESPONSE>
         </SIMPLERSP>
      </MESSAGE>
   </CIM>
```

A.4 Deletion of a Single Instance Definition

The following HTTP request illustrates how a client deletes the instance MyClass.MyKey="S3".

```
   M-POST /cimom HTTP/1.1
   HOST: http://www.myhost.com/
   Content-Type: application/xml; charset=utf-8
   Content-Length: xxxx
   Man: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=73
   73-CIMOperation: MethodCall
   73-CIMMethod: DeleteInstance
   73-CIMObject: root%2FmyNamespace

   <?xml version="1.0" encoding="utf-8" ?>
   <CIM CIMVERSION="2.0" DTDVERSION="2.0">
      <MESSAGE ID="87872" PROTOCOLVERSION="1.0">
         <SIMPLEREQ>
            <IMETHODCALL NAME="DeleteInstance">
               <LOCALNAMESPACEPATH>
                  <NAMESPACE NAME="root"/>
                  <NAMESPACE NAME="myNamespace"/>
               </LOCALNAMESPACEPATH>
               <IPARAMVALUE NAME="InstanceName">
                     <INSTANCENAME CLASSNAME="MyClass">
                        <KEYBINDING NAME="MyKey">
                         <KEYVALUE>S3</KEYVALUE>
                        </KEYBINDING>
                     </INSTANCENAME>
               </IPARAMVALUE>
            </IMETHODCALL>
         </SIMPLEREQ>
      </MESSAGE>
   </CIM>
```

Following is an HTTP response to the preceding request indicating success of the preceding operation.

```
   HTTP/1.1 200 OK
   Content-Type: application/xml; charset=utf-8
   Content-Length: xxxx
   Ext:
   Cache-Control: no-cache
   Man: http://www.dmtf.org/cim/operation ; ns=73
   73-CIMOperation: MethodResponse

   <?xml version="1.0" encoding="utf-8" ?>
   <CIM CIMVERSION="2.0" DTDVERSION="2.0">
      <MESSAGE ID="87872" PROTOCOLVERSION="1.0">
```

```
            <SIMPLERSP>
             <IMETHODRESPONSE NAME="DeleteInstance"/>
            </SIMPLERSP>
         </MESSAGE>
      </CIM>
```

A.5 Creation of a Single Class Definition

The following HTTP request illustrates how a client creates the class MySchema_VideoBIOSElement as a subclass of CIM_VideoBIOSElement. For clarity of exposition, most of the submitted <CLASS> element is omitted from the example.

```
      M-POST /cimom HTTP/1.1
      HOST: http://www.myhost.com/
      Content-Type: application/xml; charset=utf-8
      Content-Length: xxxx
      Man: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=73
      73-CIMOperation: MethodCall
      73-CIMMethod: CreateClass
      73-CIMObject: root/cimv2

      <?xml version="1.0" encoding="utf-8" ?>
      <CIM CIMVERSION="2.0" DTDVERSION="2.0">
         <MESSAGE ID="87872" PROTOCOLVERSION="1.0">
            <SIMPLEREQ>
               <IMETHODCALL NAME="CreateClass">
                  <LOCALNAMESPACEPATH>
                   <NAMESPACE NAME="root"/>
                   <NAMESPACE NAME="cimv2"/>
                  </LOCALNAMESPACEPATH>
                  <IPARAMVALUE NAME="NewClass">
                   <CLASS NAME="MySchema_VideoBIOSElement"
      SUPERCLASS="CIM_VideoBIOSElement">
                         ...
                   </CLASS>
                  </IPARAMVALUE>
               </IMETHODCALL>
            </SIMPLEREQ>
         </MESSAGE>
      </CIM>
```

Following is an HTTP response to the preceding request indicating success of the preceding operation.

```
      HTTP/1.1 200 OK
      Content-Type: application/xml; charset=utf-8
      Content-Length: xxxx
      Ext:
      Cache-Control: no-cache
      Man: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=73
      73-CIMOperation: MethodResponse

      <?xml version="1.0" encoding="utf-8" ?>
      <CIM CIMVERSION="2.0" DTDVERSION="2.0">
         <MESSAGE ID="87872" PROTOCOLVERSION="1.0">
            <SIMPLERSP>
            <IMETHODRESPONSE NAME="CreateClass"/>
```

```
            </SIMPLERSP>
         </MESSAGE>
      </CIM>
```

A.6 Creation of a Single Instance Definition

The following HTTP request illustrates how a client creates an instance of the class MySchema_VideoBIOSElement. For clarity of exposition, most of the submitted <INSTANCE> element is omitted from the example.

```
      M-POST /cimom HTTP/1.1
      HOST: http://www.myhost.com/
      Content-Type: application/xml; charset=utf-8
      Content-Length: xxxx
      Man: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=73
      73-CIMOperation: MethodCall
      73-CIMMethod: CreateInstance
      73-CIMObject: root/cimv2

      <?xml version="1.0" encoding="utf-8" ?>
      <CIM CIMVERSION="2.0" DTDVERSION="2.0">
         <MESSAGE ID="87872" PROTOCOLVERSION="1.0">
            <SIMPLEREQ>
               <IMETHODCALL NAME="CreateInstance">
                  <LOCALNAMESPACEPATH>
                   <NAMESPACE NAME="root"/>
                   <NAMESPACE NAME="cimv2"/>
                  </LOCALNAMESPACEPATH>
                  <IPARAMVALUE NAME="NewInstance">
                   <INSTANCE CLASSNAME="CIM_VideoBIOSElement">
                         ...
                   </INSTANCE>
                  </IPARAMVALUE>
               </IMETHODCALL>
            </SIMPLEREQ>
         </MESSAGE>
      </CIM>
```

Following is an HTTP response to the preceding request indicating the success of the preceding operation.

```
      HTTP/1.1 200 OK
      Content-Type: application/xml; charset=utf-8
      Content-Length: xxxx
      Ext:
      Cache-Control: no-cache
      Man: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=73
      73-CIMOperation: MethodResponse

      <?xml version="1.0" encoding="utf-8" ?>
      <CIM CIMVERSION="2.0" DTDVERSION="2.0">
         <MESSAGE ID="87872" PROTOCOLVERSION="1.0">
            <SIMPLERSP>
               <IMETHODRESPONSE NAME="CreateInstance">
                  <IRETURNVALUE>
                     <INSTANCENAME CLASSNAME="MySchema_VideoBIOSElement">
```

```
                        <KEYBINDING NAME="Name">
                         <KEYVALUE>S4</KEYVALUE>
                        </KEYBINDING>
                    </INSTANCENAME>
                </IRETURNVALUE>
            </IRETURNVALUE>
        </SIMPLERSP>
    </MESSAGE>
</CIM>
```

A.7 Enumeration of Class Names

The following HTTP request illustrates how a client enumerates the names of all subclasses of the class CIM_SoftwareElement.

```
M-POST /cimom HTTP/1.1
HOST: http://www.myhost.com/
Content-Type: application/xml; charset=utf-8
Content-Length: xxxx
Man: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=73
73-CIMOperation: MethodCall
73-CIMMethod: EnumerateClassNames
73-CIMObject: root/cimv2
```

```
<?xml version="1.0" encoding="utf-8" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.0">
   <MESSAGE ID="87872" PROTOCOLVERSION="1.0">
      <SIMPLEREQ>
         <IMETHODCALL NAME="EnumerateClassNames">
            <LOCALNAMESPACEPATH>
             <NAMESPACE NAME="root"/>
             <NAMESPACE NAME="cimv2"/>
            </LOCALNAMESPACEPATH>
            <IPARAMVALUE NAME="ClassName"><CLASSNAME
               NAME="CIM_SoftwareElement"/></IPARAMVALUE>
            <IPARAMVALUE
               NAME="DeepInheritance"><VALUE>FALSE</VALUE></IPARAMVALUE>
         </IMETHODCALL>
      </SIMPLEREQ>
   </MESSAGE>
</CIM>
```

Following is an HTTP response to the preceding request indicating the success of the preceding operation and returning the names of the requested subclasses.

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8
Content-Length: xxxx
Ext:
Cache-Control: no-cache
Man: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=73
73-CIMOperation: MethodResponse
```

```
<?xml version="1.0" encoding="utf-8" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.0">
   <MESSAGE ID="87872" PROTOCOLVERSION="1.0">
```

```
            <SIMPLERSP>
               <IMETHODRESPONSE NAME="EnumerateClassNames">
                  <IRETURNVALUE>
                   <CLASSNAME NAME="CIM_BIOSElement"/>
                   <CLASSNAME NAME="CIM_VideoBOISElement"/>
                  </IRETURNVALUE>
               </IMETHODRESPONSE>
            </SIMPLERSP>
         </MESSAGE>
      </CIM>
```

A.8 Enumeration of Instances

The following HTTP request illustrates how a client enumerates all instances of the class CIM_LogicalDisk. For clarity of exposition, most of the returned instances are omitted from the example.

```
      M-POST /cimom HTTP/1.1
      HOST: http://www.myhost.com/
      Content-Type: application/xml; charset=utf-8
      Content-Length: xxxx
      Man: http://www.dmtf.org/cim/operation ; ns=73
      73-CIMOperation: MethodCall
      73-CIMMethod: EnumerateInstances
      73-CIMObject: root/cimv2

      <?xml version="1.0" encoding="utf-8" ?>
      <CIM CIMVERSION="2.0" DTDVERSION="2.0">
         <MESSAGE ID="87872" PROTOCOLVERSION="1.0">
            <SIMPLEREQ>
               <IMETHODCALL NAME="EnumerateInstances">
                  <LOCALNAMESPACEPATH>
                   <NAMESPACE NAME="root"/>
                   <NAMESPACE NAME="cimv2"/>
                  </LOCALNAMESPACEPATH>
                  <IPARAMVALUE NAME="ClassName"><CLASSNAME
                     NAME="CIM_LogicalDisk"/></IPARAMVALUE>
                  <IPARAMVALUE NAME="LocalOnly"><VALUE>TRUE</VALUE></IPARAMVALUE>
                  <IPARAMVALUE NAME="DeepInheritance"><VALUE>TRUE</VALUE></IPARAMVALUE>
               </IMETHODCALL>
            </SIMPLEREQ>
         </MESSAGE>
      </CIM>
```

Following is an HTTP response to the preceding request indicating success of the preceding operation, returning the requested instances.

```
      HTTP/1.1 200 OK
      Content-Type: application/xml; charset=utf-8
      Content-Length: xxxx
      Ext:
      Cache-Control: no-cache
      Man: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=73
      73-CIMOperation: MethodResponse

      <?xml version="1.0" encoding="utf-8" ?>
      <CIM CIMVERSION="2.0" DTDVERSION="2.0">
```

```
        <MESSAGE ID="87872" PROTOCOLVERSION="1.0">
           <SIMPLERSP>
              <IMETHODRESPONSE NAME="EnumerateInstances">
                 <IRETURNVALUE>
                    <VALUE.NAMEDINSTANCE>
                       <INSTANCENAME CLASSNAME="Erewhon_LogicalDisk">
                        ...
                       </INSTANCENAME>
                       <INSTANCE CLASSNAME="Erewhon_LogicalDisk">
                        ...
                       </INSTANCE>
                    </VALUE.NAMEDINSTANCE>
                    ...
                    <VALUE.NAMEDINSTANCE>
                       <INSTANCENAME CLASSNAME="Foobar_LogicalDisk">
                        ...
                       </INSTANCENAME>
                       <INSTANCE CLASSNAME="Foobar_LogicalDisk">
                        ...
                       </INSTANCE>
                    </VALUE.NAMEINSTANCE>
                 </IRETURNVALUE>
              </IMETHODRESPONSE>
           </SIMPLERSP>
        </MESSAGE>
     </CIM>
```

A.9 Retrieval of a Single Property

The following HTTP request illustrates how a client retrieves the FreeSpace property from the instance MyDisk.DeviceID="C:".

```
     M-POST /cimom HTTP/1.1
     HOST: http://www.myhost.com/
     Content-Type: application/xml; charset=utf-8
     Content-Length: xxxx
     Man: http://www.dmtf.org/cim/operation ; ns=73
     73-CIMOperation: MethodCall
     73-CIMMethod: GetProperty
     73-CIMObject: root%2FmyNamespace
```

```
     <?xml version="1.0" encoding="utf-8" ?>
     <CIM CIMVERSION="2.0" DTDVERSION="2.0">
        <MESSAGE ID="87872" PROTOCOLVERSION="1.0">
           <SIMPLEREQ>
              <IMETHODCALL NAME="GetProperty">
                 <LOCALNAMESPACEPATH>
                    <NAMESPACE NAME="root"/>
                    <NAMESPACE NAME="myNamespace"/>
                 </LOCALNAMESPACEPATH>
                 <IPARAMVALUE NAME="InstanceName">
                    <INSTANCENAME CLASSNAME="MyDisk">
                     <KEYBINDING NAME="DeviceID"><KEYVALUE>C:</KEYVALUE></KEYBINDING>
                    </INSTANCENAME>
```

```
                </IPARAMVALUE>
                <IPARAMVALUE
    NAME="PropertyName"><VALUE>FreeSpace</VALUE></IPARAMVALUE>
             </IMETHODCALL>
          </SIMPLEREQ>
       </MESSAGE>
    </CIM>
```

Following is an HTTP response to the preceding request indicating success of the preceding operation, returning the requested value.

```
    HTTP/1.1 200 OK
    Content-Type: application/xml; charset=utf-8
    Content-Length: xxxx
    Ext:
    Cache-Control: no-cache
    Man: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=73
    73-CIMOperation: MethodResponse

    <?xml version="1.0" encoding="utf-8" ?>
    <CIM CIMVERSION="2.0" DTDVERSION="2.0">
       <MESSAGE ID="87872" PROTOCOLVERSION="1.0">
          <SIMPLERSP>
             <IMETHODRESPONSE NAME="GetProperty">
                <IRETURNVALUE>
                 <VALUE>6752332</VALUE>
                </IRETURNVALUE>
             </IMETHODRESPONSE>
          </SIMPLERSP>
       </MESSAGE>
    </CIM>
```

A.10 Execution of an Extrinsic Method

The following HTTP request illustrates how a client executes the SetPowerState method on the instance MyDisk.DeviceID="C:".

```
    M-POST /cimom HTTP/1.1
    HOST: http://www.myhost.com/
    Content-Type: application/xml; charset=utf-8
    Content-Length: xxxx
    Man: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=73
    73-CIMOperation: MethodCall
    73-CIMMethod: SetPowerState
    73-CIMObject: root%2FmyNamespace%3AMyDisk.Name%3D%22C%3A%22

    <?xml version="1.0" encoding="utf-8" ?>
    <CIM CIMVERSION="2.0" DTDVERSION="2.0">
       <MESSAGE ID="87872" PROTOCOLVERSION="1.0">
          <SIMPLEREQ>
             <METHODCALL NAME="SetPowerState">
                <LOCALINSTANCEPATH>
                   <LOCALNAMESPACEPATH>
                    <NAMESPACE NAME="root"/>
                    <NAMESPACE NAME="myNamespace"/>
                   </LOCALNAMESPACEPATH>
```

```
                    <INSTANCENAME CLASSNAME="MyDisk">
                     <KEYBINDING NAME="Name"><KEYVALUE>C:</KEYVALUE></KEYBINDING>
                    </INSTANCENAME>
                </LOCALINSTANCEPATH>
                <PARAMVALUE NAME="PowerState"><VALUE>1</VALUE></PARAMVALUE>
                <PARAMVALUE
NAME="Time"><VALUE>00000001132312.000000:000</VALUE></PARAMVALUE>
            </METHODCALL>
        </SIMPLEREQ>
    </MESSAGE>
</CIM>
```

Following is an HTTP response to the preceding request indicating the success of the preceding operation.

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8
Content-Length: xxxx
Ext:
Cache-Control: no-cache
Man: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=73
73-CIMOperation: MethodResponse

<?xml version="1.0" encoding="utf-8" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.0">
    <MESSAGE ID="87872" PROTOCOLVERSION="1.0">
        <SIMPLERSP>
            <METHODRESPONSE NAME="SetPowerState">
                <RETURNVALUE>
                 <VALUE>0</VALUE>
                </RETURNVALUE>
            </METHODRESPONSE>
        </SIMPLERSP>
    </MESSAGE>
</CIM>
```

A.11 Indication Delivery Example

The following HTTP request illustrates the format for sending an indication of type CIM_AlertIndication to a CIM listener.

```
M-POST /cimlistener/browser HTTP/1.1
HOST: http://www.acme.com/
Content-Type: application/xml; charset=utf-8
Content-Length: XXX
Man: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=40
40-CIMExport: MethodRequest
40-CIMExportMethod: ExportIndication

<?xml version="1.0" encoding="utf-8" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.0">
    <MESSAGE ID="1007" PROTOCOLVERSION="1.0">
        <SIMPLEEXPREQ>
            <EXPMETHODCALL NAME="ExportIndication">
                <EXPPARAMVALUE NAME="NewIndication">
                    <INSTANCE CLASSNAME="CIM_AlertIndication" >
```

```
                        <PROPERTY NAME="Description" TYPE="string">
                            <VALUE>Sample CIM_AlertIndication indication</VALUE>
                        </PROPERTY>
                        <PROPERTY NAME="AlertType" TYPE="uint16">
                            <VALUE>1</VALUE>
                        </PROPERTY>
                           <PROPERTY NAME="PerceivedSeverity" TYPE="uint16">
                           <VALUE>3</VALUE>
                        </PROPERTY>
                           <PROPERTY NAME="ProbableCause" TYPE="uint16">
                           <VALUE>2</VALUE>
                        </PROPERTY>
                           <PROPERTY NAME="IndicationTime" TYPE="datetime">
                           <VALUE>20010515104354.000000:000</VALUE>
                        </PROPERTY>
                     </INSTANCE>
                  </EXPPARAMVALUE>
               </EXPMETHODCALL>
            </SIMPLEEXPREQ>
         </MESSAGE>
      </CIM>
```

Following is an HTTP response to the preceding request indicating a successful receipt by the CIM listener.

```
      HTTP/1.1 200 OK
      Content-Type: application/xml; charset=utf-8
      Content-Length: 267
      Ext:
      Cache-Control: no-cache
      Man: http://www.dmtf.org/cim/mapping/http/v1.0; ns=40
      40-CIMExport: MethodResponse

      <?xml version="1.0" encoding="utf-8" ?>
      <CIM CIMVERSION="2.0" DTDVERSION="2.0">
         <MESSAGE ID="1007" PROTOCOLVERSION="1.0">
            <SIMPLEEXPRSP>
               <EXPMETHODRESPONSE NAME="ExportIndication">
                  <IRETURNVALUE>
                  </IRETURNVALUE>
               </EXPMETHODRESPONSE>
            </SIMPLEEXPRSP>
         </MESSAGE>
      </CIM>
```

A.12 Subscription Example

A CIM client application activates a subscription by creating an instance of the CIM_IndicationSubscription class, which defines an association between a CIM_IndicationFilter (a filter) instance and a CIM_IndicationHandler (a handler) instance. The CIM_IndicationFilter instance defines the filter criteria and data project list to describe the desired indication stream. The CIM_IndicationHandler instance defines the desired indication encoding, destination location, and protocol for delivering the indication stream.

The following HTTP request illustrates how a client creates an instance of the class CIM_IndicationFilter. Note that the exact syntax of the WMI Query Language is still under review and is subject to change.

```
    Host: bryce
    Content-Type: application/xml; charset=utf-8
    Content-Length: XXXX
    Man: http://www.dmtf.org/cim/mapping/http/v1.0;ns=20
    20-CIMProtocolVersion: 1.0
    20-CIMOperation: MethodCall
    20-CIMMethod: CreateInstance
    20-CIMObject: root/cimv2

    <?xml version="1.0" encoding="utf-8"?>
    <CIM CIMVERSION="2.0" DTDVERSION="2.0">
       <MESSAGE ID="53000" PROTOCOLVERSION="1.0">
          <SIMPLEREQ>
             <IMETHODCALL NAME="CreateInstance">
                <LOCALNAMESPACEPATH>
                   <NAMESPACE NAME="root"/>
                   <NAMESPACE NAME="cimv2"/>
                </LOCALNAMESPACEPATH>
                <IPARAMVALUE NAME="NewInstance">
                   <INSTANCE CLASSNAME="CIM_IndicationFilter">
                      <PROPERTY NAME="SystemCreationClassName" TYPE="string">
                         <VALUE>CIM_UnitaryComputerSystem</VALUE>
                      </PROPERTY>
                      <PROPERTY NAME="SystemName" TYPE="string">
                         <VALUE>server001.acme.com</VALUE>
                      </PROPERTY>
                      <PROPERTY NAME="CreationClassName" TYPE="string">
                         <VALUE>CIM_IndicationFilter</VALUE>
                      </PROPERTY>
                      <PROPERTY NAME="Name" TYPE="string">
                         <VALUE>ACMESubscription12345</VALUE>
                      </PROPERTY>
                      <PROPERTY NAME="SourceNamespace" TYPE="string">
                         <VALUE>root/cimv2</VALUE>
                      </PROPERTY>
                      <PROPERTY NAME="Query" TYPE="string">
                         <VALUE>
                         SELECT Description, AlertType, PerceivedSeverity,
                            ProbableCause, IndicationTime
                            FROM CIM_AlertIndication
                            WHERE PerceivedSeverity = 3
                         </VALUE>
                      </PROPERTY>
                      <PROPERTY NAME="QueryLanguage" TYPE="string">
                         <VALUE>WQL</VALUE>
                      </PROPERTY>
                   </INSTANCE>
                </IPARAMVALUE>
             </IMETHODCALL>
          </SIMPLEREQ>
       </MESSAGE>
```

```
    </CIM>
```

Following is an HTTP response to the preceding request indicating success of the preceding operation.

```
    HTTP/1.1 200 OK
    Content-Type: application/xml; charset=utf-8
    Content-Length: XXX
    Ext:
    Cache-Control: no-cache
    Man: http://www.dmtf.org/cim/mapping/http/v1.0; ns=28
    28-CIMOperation: MethodResponse

    <?xml version="1.0" encoding="utf-8" ?>
       <CIM CIMVERSION="2.0" DTDVERSION="2.0">
          <MESSAGE ID="53000" PROTOCOLVERSION="1.0">
             <SIMPLERSP>
                <IMETHODRESPONSE NAME="CreateInstance">
                   <IRETURNVALUE>
                   <INSTANCENAME CLASSNAME="CIM_IndicationFilter">
                          <KEYBINDING NAME="SystemCreationClassName">
                             <KEYVALUE VALUETYPE="string">
                                CIM_UnitaryComputerSystem
                             </KEYVALUE>
                          </KEYBINDING>
                          <KEYBINDING NAME="SystemName">
                             <KEYVALUE VALUETYPE="string">
                                server001.acme.com
                             </KEYVALUE>
                          </KEYBINDING>
                          <KEYBINDING NAME="CreationClassName">
                             <KEYVALUE VALUETYPE="string">
                                CIM_IndicationFilter
                             </KEYVALUE>
                          </KEYBINDING>
                          <KEYBINDING NAME="Name">
                             <KEYVALUE VALUETYPE="string">
                                ACMESubscription12345
                             </KEYVALUE>
                       </KEYBINDING>
                       </INSTANCENAME>
                   </IRETURNVALUE>
                </IMETHODRESPONSE>
             </SIMPLERSP>
          </MESSAGE>
       </CIM>
```

The following HTTP request illustrates how a client creates an instance of the class CIM_IndicationHandlerCIMXML.

```
    M-POST /cimom HTTP/1.1
    Host: bryce
    Content-Type: application/xml; charset=utf-8
    Content-Length: XXX
    Man: http://www.dmtf.org/cim/mapping/http/v1.0;ns=20
    20-CIMProtocolVersion: 1.0
```

```
    20-CIMOperation: MethodCall
    20-CIMMethod: CreateInstance
    20-CIMObject: root/cimv2
    <?xml version="1.0" encoding="utf-8"?>
    <CIM CIMVERSION="2.0" DTDVERSION="2.0">
    <MESSAGE ID="54000" PROTOCOLVERSION="1.0">
          <SIMPLEREQ>
             <IMETHODCALL NAME="CreateInstance">
                <LOCALNAMESPACEPATH>
                   <NAMESPACE NAME="root"/>
                   <NAMESPACE NAME="cimv2"/>
                </LOCALNAMESPACEPATH>
                <IPARAMVALUE NAME="NewInstance">
                   <INSTANCE CLASSNAME="CIM_IndicationHandlerCIMXML">
                      <PROPERTY NAME="SystemCreationClassName" TYPE="string">
                         <VALUE>CIM_UnitaryComputerSystem</VALUE>
                      </PROPERTY>
                      <PROPERTY NAME="SystemName" TYPE="string">
                         <VALUE>server001.acme.com</VALUE>
                         </PROPERTY>
                      <PROPERTY NAME="CreationClassName" TYPE="string">
                         <VALUE>CIM_IndicationHandlerCIMXML</VALUE>
                      </PROPERTY>
                      <PROPERTY NAME="Name" TYPE="string">
                         <VALUE>ACMESubscription12345</VALUE>
                      </PROPERTY>
                      <PROPERTY NAME="Owner" TYPE="string">
                         <VALUE>ACMEAlertMonitoringConsole</VALUE>
                      </PROPERTY>
                      <PROPERTY NAME="Destination" TYPE="string">
                         <VALUE>HTTP://www.acme.com/cimlistener/browser</VALUE>
                      </PROPERTY>
                   </INSTANCE>
                </IPARAMVALUE>
             </IMETHODCALL>
          </SIMPLEREQ>
       </MESSAGE>
    </CIM>
```

Following is an HTTP response to the preceding request indicating the success of the preceding operation.

```
    HTTP/1.1 200 OK
    Content-Type: application/xml; charset=utf-8
    Content-Length: XXX
    Ext:
    Cache-Control: no-cache
    Man: http://www.dmtf.org/cim/mapping/http/v1.0; ns=27
    27-CIMOperation: MethodResponse
```

```
    <?xml version="1.0" encoding="utf-8" ?>
       <CIM CIMVERSION="2.0" DTDVERSION="2.0">
          <MESSAGE ID="54000" PROTOCOLVERSION="1.0">
             <SIMPLERSP>
```

```
                <IMETHODRESPONSE NAME="CreateInstance">
                   <IRETURNVALUE>
                      <INSTANCENAME CLASSNAME="CIM_IndicationHandlerCIMXML">
                         <KEYBINDING NAME="SystemCreationClassName">
                            <KEYVALUE VALUETYPE="string">
                               CIM_UnitaryComputerSystem
                            </KEYVALUE>
                         </KEYBINDING>
                         <KEYBINDING NAME="SystemName">
                            <KEYVALUE VALUETYPE="string">
                               server001.acme.com
                            </KEYVALUE>
                         </KEYBINDING>
                         <KEYBINDING NAME="CreationClassName">
                            <KEYVALUE VALUETYPE="string">
                               CIM_IndicationHandlerCIMXML
                            </KEYVALUE>
                         </KEYBINDING>
                         <KEYBINDING NAME="Name">
                            <KEYVALUE VALUETYPE="string">
                               ACMESubscription12345
                            </KEYVALUE>
                         </KEYBINDING>
                      </INSTANCENAME>
                   </IRETURNVALUE>
                </IMETHODRESPONSE>
             </SIMPLERSP>
          </MESSAGE>
       </CIM>
```

The following HTTP request illustrates how a client creates an instance of the class CIM_IndicationSubscription.

```
       M-POST /cimom HTTP/1.1
       Host: bryce
       Content-Type: application/xml; charset=utf-8
       Content-Length: XXXX
       Man: http://www.dmtf.org/cim/mapping/http/v1.0;ns=55
       55-CIMProtocolVersion: 1.0
       55-CIMOperation: MethodCall
       55-CIMMethod: CreateInstance
       55-CIMObject: root/cimv2

       <?xml version="1.0" encoding="utf-8"?>
       <CIM CIMVERSION="2.0" DTDVERSION="2.0">
          <MESSAGE ID="55000" PROTOCOLVERSION="1.0">
             <SIMPLEREQ>
                <IMETHODCALL NAME="CreateInstance">
                   <LOCALNAMESPACEPATH>
                      <NAMESPACE NAME="root"/>
                      <NAMESPACE NAME="cimv2"/>
                   </LOCALNAMESPACEPATH>
                   <IPARAMVALUE NAME="NewInstance">
                      <INSTANCE CLASSNAME="CIM_IndicationSubscription">
```

```
                        <PROPERTY.REFERENCE NAME="Filter"
                           REFERENCECLASS="CIM_IndicationFilter">
                           <VALUE.REFERENCE>
                              <INSTANCENAME CLASSNAME="CIM_IndicationFilter">
                                 <KEYBINDING NAME="SystemCreationClassName">
                                    <KEYVALUE VALUETYPE="string">
                                     CIM_UnitaryComputerSystem
                                    </KEYVALUE>
                                 </KEYBINDING>
                                 <KEYBINDING NAME="SystemName">
                                    <KEYVALUE VALUETYPE="string">
                                     server001.acme.com
                                    </KEYVALUE>
                                 </KEYBINDING>
                                 <KEYBINDING NAME="CreationClassName">
                                    <KEYVALUE VALUETYPE="string">
                                     CIM_IndicationFilter
                                    </KEYVALUE>
                                 </KEYBINDING>
                                 <KEYBINDING NAME="Name">
                                    <KEYVALUE VALUETYPE="string">
                                       ACMESubscription12345
                                    </KEYVALUE>
                                 </KEYBINDING>
                              </INSTANCENAME>
                           </VALUE.REFERENCE>
                           </PROPERTY.REFERENCE>
                           <PROPERTY.REFERENCE NAME="Handler"
   REFERENCECLASS="CIM_IndicationHandler">
                              <VALUE.REFERENCE>
                              <INSTANCENAME CLASSNAME="CIM_IndicationHandlerCIMXML">
                                 <KEYBINDING NAME="SystemCreationClassName">
                                    <KEYVALUE VALUETYPE="string">
                                       CIM_UnitaryComputerSystem
                                    </KEYVALUE>
                                 </KEYBINDING>
                                 <KEYBINDING NAME="SystemName">
                                    <KEYVALUE VALUETYPE="string">
                                       server001.acme.com
                                    </KEYVALUE>
                                 </KEYBINDING>
                                 <KEYBINDING NAME="CreationClassName">
                                    <KEYVALUE VALUETYPE="string">
                                       CIM_IndicationHandlerCIMXML
                                    </KEYVALUE>
                                 </KEYBINDING>
                                 <KEYBINDING NAME="Name">
                                    <KEYVALUE VALUETYPE="string">
                                       ACMESubscription12345
                                    </KEYVALUE>
                                 </KEYBINDING>
                              </INSTANCENAME>
                           </VALUE.REFERENCE>
```

```
                        </PROPERTY.REFERENCE>
                    </INSTANCE>
                </IPARAMVALUE>
            </IMETHODCALL>
        </SIMPLEREQ>
    </MESSAGE>
</CIM>
```

Following is an HTTP response to the preceding request indicating the success of the preceding operation.

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8
Content-Length: XXXX
Ext:
Cache-Control: no-cache
Man: http://www.dmtf.org/cim/mapping/http/v1.0; ns=75
75-CIMOperation: MethodResponse
```

```
<?xml version="1.0" encoding="utf-8" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.0">
   <MESSAGE ID="55000" PROTOCOLVERSION="1.0">
      <SIMPLERSP>
         <IMETHODRESPONSE NAME="CreateInstance">
            <IRETURNVALUE>
               <INSTANCENAME CLASSNAME="CIM_IndicationSubscription">
                  <KEYBINDING NAME="Filter">
                     <VALUE.REFERENCE>
                        <INSTANCENAME CLASSNAME="CIM_IndicationFilter">
                           <KEYBINDING NAME="SystemCreationClassName">
                              <KEYVALUE VALUETYPE="string">
                                 CIM_UnitaryComputerSystem
                              </KEYVALUE>
                           </KEYBINDING>
                           <KEYBINDING NAME="SystemName">
                              <KEYVALUE VALUETYPE="string">
                                 server001.acme.com
                              </KEYVALUE>
                           </KEYBINDING>
                           <KEYBINDING NAME="CreationClassName">
                              <KEYVALUE VALUETYPE="string">
                                 CIM_IndicationFilter
                              </KEYVALUE>
                           </KEYBINDING>
                           <KEYBINDING NAME="Name">
                              <KEYVALUE VALUETYPE="string">
                                 ACMESubscription12345
                              </KEYVALUE>
                           </KEYBINDING>
                        </INSTANCENAME>
                     </VALUE.REFERENCE>
                  </KEYBINDING>
                  <KEYBINDING NAME="Handler">
                     <VALUE.REFERENCE>
```

```
                                <INSTANCENAME CLASSNAME="CIM_IndicationHandlerCIMXML">
                                   <KEYBINDING NAME="SystemCreationClassName">
                                      <KEYVALUE VALUETYPE="string">
                                         CIM_UnitaryComputerSystem
                                      </KEYVALUE>
                                   </KEYBINDING>
                                   <KEYBINDING NAME="SystemName">
                                      <KEYVALUE VALUETYPE="string">
                                       server001.acme.com
                                      </KEYVALUE>
                                   </KEYBINDING>
                                   <KEYBINDING NAME="CreationClassName">
                                      <KEYVALUE VALUETYPE="string">
                                         CIM_IndicationHandlerCIMXML
                                      </KEYVALUE>
                                   </KEYBINDING>
                                   <KEYBINDING NAME="Name">
                                      <KEYVALUE VALUETYPE="string">
                                         ACMESubscription12345
                                      </KEYVALUE>
                                   </KEYBINDING>
                                </INSTANCENAME>
                             </VALUE.REFERENCE>
                          </KEYBINDING>
                       </INSTANCENAME>
                    </IRETURNVALUE>
                 </IMETHODRESPONSE>
              </SIMPLERSP>
           </MESSAGE>
        </CIM>
```

A.13 Multiple Operations Example

The following HTTP request illustrates how a client performs multiple operations. This example batches a GetClass, an EnumerateInstanceNames, and an EnumerateInstance operation on CIM_ObjectManagerAdapter.

```
    POST /CIMOM1 HTTP/1.1
    Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
    Content-Length: XXX
    Host: localhost:5988
    CIMOperation: MethodCall
    CIMProtocolVersion: 1.0
    Content-Type: application/xml; charset=utf-8
    CIMBatch: CIMBatch
    <?xml version="1.0" encoding="UTF-8"?>
    <CIM DTDVERSION="2.0" CIMVERSION="2.0">
       <MESSAGE ID="2004:2:5:1:1:11:41:1" PROTOCOLVERSION="1.0">
          <MULTIREQ>
             <SIMPLEREQ>
                <IMETHODCALL NAME="GetClass">
                   <LOCALNAMESPACEPATH>
                      <NAMESPACE NAME="interop" />
                   </LOCALNAMESPACEPATH>
```

```
                    <IPARAMVALUE NAME="ClassName">
                        <CLASSNAME NAME="CIM_ObjectManagerAdapter" />
                    </IPARAMVALUE>
                    <IPARAMVALUE NAME="LocalOnly">
                        <VALUE>FALSE</VALUE>
                    </IPARAMVALUE>
                    <IPARAMVALUE NAME="IncludeClassOrigin">
                        <VALUE>TRUE</VALUE>
                    </IPARAMVALUE>
                </IMETHODCALL>
            </SIMPLEREQ>
            <SIMPLEREQ>
                <IMETHODCALL NAME="Associators">
                    <LOCALNAMESPACEPATH>
                        <NAMESPACE NAME="interop" />
                    </LOCALNAMESPACEPATH>
                    <IPARAMVALUE NAME="ObjectName">
                        <CLASSNAME NAME="CIM_ObjectManagerAdapter" />
                    </IPARAMVALUE>
                    <IPARAMVALUE NAME="IncludeQualifiers">
                        <VALUE>TRUE</VALUE>
                    </IPARAMVALUE>
                    <IPARAMVALUE NAME="IncludeClassOrigin">
                        <VALUE>TRUE</VALUE>
                    </IPARAMVALUE>
                </IMETHODCALL>
            </SIMPLEREQ>
            <SIMPLEREQ>
                <IMETHODCALL NAME="EnumerateInstanceNames">
                    <LOCALNAMESPACEPATH>
                        <NAMESPACE NAME="interop" />
                    </LOCALNAMESPACEPATH>
                    <IPARAMVALUE NAME="ClassName">
                        <CLASSNAME NAME="CIM_ObjectManagerAdapter" />
                    </IPARAMVALUE>
                </IMETHODCALL>
            </SIMPLEREQ>
            <SIMPLEREQ>
                <IMETHODCALL NAME="EnumerateInstances">
                    <LOCALNAMESPACEPATH>
                        <NAMESPACE NAME="interop" />
                    </LOCALNAMESPACEPATH>
                    <IPARAMVALUE NAME="ClassName">
                        <CLASSNAME NAME="CIM_ObjectManagerAdapter" />
                    </IPARAMVALUE>
                    <IPARAMVALUE NAME="LocalOnly">
                        <VALUE>FALSE</VALUE>
                    </IPARAMVALUE>
                </IMETHODCALL>
            </SIMPLEREQ>
        </MULTIREQ>
    </MESSAGE>
</CIM>
```

Following is the HTTP response to the preceding request indicating the success of the preceding operation.

```
HTTP/1.1 200 OK
CIMOperation: MethodResponse
Content-Length: XXX
<?xml version="1.0" encoding="UTF-8"?>
<CIM DTDVERSION="2.0" CIMVERSION="2.0">
   <MESSAGE ID="2004:2:5:1:1:11:41:1" PROTOCOLVERSION="1.0">
      <MULTIRSP>
         <SIMPLERSP>
            <IMETHODRESPONSE NAME="GetClass">
               <IRETURNVALUE>
                  <CLASS SUPERCLASS="CIM_WBEMService"
                     NAME="CIM_ObjectManagerAdapter">
                     ...
                  </CLASS>
               </IRETURNVALUE>
            </IMETHODRESPONSE>
         </SIMPLERSP>
         <SIMPLERSP>
            <IMETHODRESPONSE NAME="Associators">
               <IRETURNVALUE>
                  <VALUE.OBJECTWITHPATH>
                     ...
                  </VALUE.OBJECTWITHPATH>
                  <VALUE.OBJECTWITHPATH>
                     ...
                  </VALUE.OBJECTWITHPATH>
                     ...
               </IRETURNVALUE>
            </IMETHODRESPONSE>
         </SIMPLERSP>
         <SIMPLERSP>
            <IMETHODRESPONSE NAME="EnumerateInstanceNames">
               <IRETURNVALUE>
                  <INSTANCENAME CLASSNAME="WBEMSolutions_ObjectManagerAdapter">
                     ...
                  </INSTANCENAME>
                  <INSTANCENAME CLASSNAME="WBEMSolutions_ObjectManagerAdapter">
                     ...
                  </INSTANCENAME>
                     ...
               </IRETURNVALUE>
            </IMETHODRESPONSE>
         </SIMPLERSP>
         <SIMPLERSP>
            <IMETHODRESPONSE NAME="EnumerateInstances">
               <IRETURNVALUE>
                  <VALUE.NAMEDINSTANCE>
                     ...
                  </VALUE.NAMEDINSTANCE>
                  <VALUE.NAMEDINSTANCE>
```

```
                            ...
                        </VALUE.NAMEDINSTANCE>
                            ...
                    </IRETURNVALUE>
                </IMETHODRESPONSE>
            </SIMPLERSP>
        </MULTIRSP>
    </MESSAGE>
</CIM>
```

ANNEX B
(informative)

LocalOnly Parameter Discussion

This annex discusses the issues associated with the 1.1 definition of the LocalOnly parameter for the GetInstance and EnumerateInstances operations.

B.1 Explanation of the Deprecated 1.1 Interpretation

In April 2002, two DMTF Change Requests (CRs), CR809 (EnumerateInstances) and CR815 (GetInstance), were approved and incorporated into the 1.1 version of this specification to clarify the interpretation of the LocalOnly flag for the GetInstance and EnumerateInstances operations. With these CRs, the definition of the LocalOnly flag for these operations was modified to align with the interpretation of this flag for the GetClass and EnumerateClasses operations. This change was incorrect, resulted in reduced functionality, and introduced several backward compatibility issues.

To clarify the difference between the 1.0 Interpretation and the 1.1 Interpretation (CR815), consider the following example:

```
class A {
       [Key]
   string name;
   uint32 counter = 3;
};

class B : A {
   uint32 moreData = 4;
};

instance of A {
   name = "Roger";
};

instance of B {
   name = "Karl";
   counter = 3;
   moreData = 5;
};

instance of B {
   name = "Denise";
   counter = 5;
};
```

Assuming PropertyList = NULL and LocalOnly = TRUE, Table B-1 shows the properties returned by a GetInstance operation.

Table B-1 – Comparison of Properties Returned by GetInstance in Versions 1.0 and 1.1

Instance	DSP0200 1.0 Interpretation	DSP0200 1.1 Interpretation
"Roger"	name	name, counter
"Karl"	name, counter, moreData	moreData
"Denise"	name, counter	moreData

The properties returned using the 1.0 interpretation are consistent with the properties specified in the MOF instance definitions, and the properties returned using the 1.1 Interpretation are consistent with the properties defined in the class definitions.

B.2 Risks of Using the 1.1 Interpretation

The risks of using the 1.1 interpretation are as follows:

1) Within the DMTF, promoting a property from a class to one of its superclasses is defined as a backward-compatible change that can be made in a minor revision of the CIM schema. With the 1.1 interpretation, promoting a property to a superclass can cause backward-incompatible changes.

   Suppose, for example, version 1.0 of the schema includes the following definitions:

```
class A {
      [Key]
   string name;
   uint32 counter = 3;
};

class B : A {
   uint32 moreData = 4;
};
```

   Now suppose that the schema is modified in version 1.1 to promote the property moreData from class B to class A.

```
class A {
      [Key]
   string name;
   uint32 counter = 3;
   uint32 moreData = 4;
};

class B : A {

};
```

   Using these examples, Table B-2 shows the properties returned by a call to GetInstance with PropertyList = NULL and LocalOnly = TRUE. With the 1.1 Interpretation, this schema change would affect the list of properties returned. When dealing with a CIM server that complies with the 1.1 interpretation, applications must be designed to treat “promoting properties” as a backward-compatible change.

4646 **Table B-2 – Comparison of Properties Returned by a Call to GetInstance in Versions 1.0 and 1.1**

Instance	Schema Version 1.0	Schema Version 1.1
of A	name, counter	name, counter, moreData
of B	moreData	none

4647 2) The 1.1 Interpretation encourages application developers to use multiple operations to retrieve the
4648 properties of an instance. That is, a commonly-stated use model for the 1.1 interpretation is to
4649 selectively traverse subclasses getting additional properties of an instance. This practice significantly
4650 increases the risk that a client will construct an inconsistent instance. With both Interpretations,
4651 applications should be designed to ensure that dependent properties are retrieved together.

4652 B.3 Techniques for Differentiating between the 1.0 Interpretation and 1.1 4653 Interpretation

4654 For concrete classes, CIM servers that comply with the 1.0 Interpretation return the value of all KEY
4655 properties not explicitly excluded by the PropertyList parameter. CIM servers that comply with the 1.1
4656 interpretation return only the value of KEY properties explicitly defined in the class. Applications can use
4657 this difference to detect which interpretation is supported by a CIM server.

ANNEX C
(informative)

Change Log

Version	Date	Description
Version 1.0	Jun 2, 1999	DMTF Final Standard
Version 1.1	Jan 06, 2003	DMTF Final Standard
Version 1.2	Jan 09, 2007	DMTF Final Standard
Version 1.3.0	Oct 15, 2008	DMTF Final Standard
Version 1.3.1	July 29, 2009	DMTF Standard Release