How WBEM Enables Management Interoperability

Carol Ann Krug Graves
Hewlett-Packard Company
19111 Pruneridge Avenue, MS 44UR
Cupertino CA 95014-0795
Telephone (408) 447-5208
Fax (408) 447-1053
E-mail carolann_graves@hp.com

1. Abstract

Web-Based Enterprise Management (WBEM) is a set of management and Internet standard technologies developed by the Distributed Management Task Force (DMTF) to unify the management of enterprise computing environments. The DMTF Common Information Model (CIM) defines a standard representation of management data. Encoding in the eXtensible Markup Language (XML), and transport over HyperText Transfer Protocol (HTTP) provide a standard communication mechanism. Together, these technologies allow the development of WBEM-based management applications that interoperate in an open, standardized manner. WBEM provides a consistent mechanism to extract data from multiple sources, including existing standard management instrumentation and platform-specific interfaces. The aim of WBEM standards is not to make management applications or instrumentation of managed resources portable from one platform to another, but rather to allow management applications to manage resources in a common way. The WBEM standard has been widely adopted.

This session will benefit system administrators with responsibility for managing HP-UX, Linux, and/or NT systems, as well as developers of management applications for these environments. The presentation will help these users to understand the WBEM technologies that enable interoperability. Using real examples, the flow of CIM management operations from a client management application through the CIM management infrastructure, to the managed resources, and back, will be described. Topics covered will include an overview of the CIM Schemas, CIM operations, representation of CIM operations in XML, CIM management infrastructure, and the XML representation of CIM results.

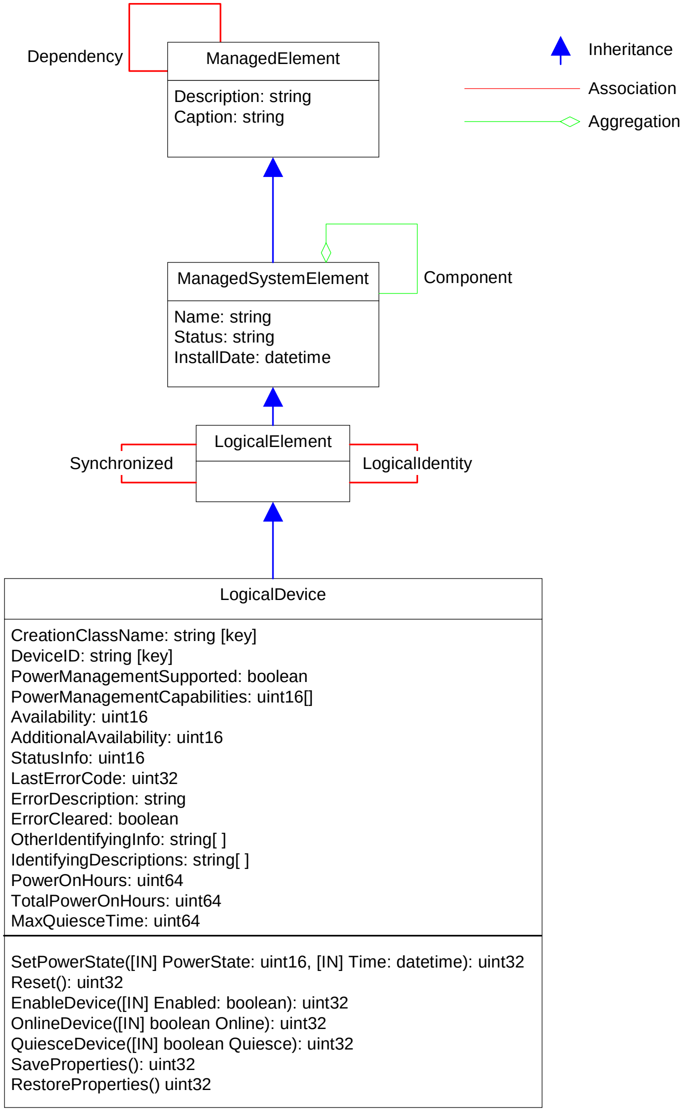
2. Introduction

The Institute of Electrical and Electronics Engineers (IEEE) defines interoperability as “the ability of two or more systems or components to exchange information and to use the information that has been exchanged” [1]. Interoperability is distinct from portability, which is defined, again by the IEEE, as “the ease with which a system or component can be transferred from one hardware or software environment to another” [1]. In the area of management, interoperability might include the ability of a management application on one platform to retrieve data from and perform management functions via a management server on another platform. Portability might include the ability of a management application developed on one platform to run on another platform, or the ability of management instrumentation developed on one platform to be used on another platform. The focus of this paper is how WBEM enables management interoperability. Management interoperability requires the establishment of a single data description mechanism for all data sources, the definition of standard operations used to discover, query and manage resources, and a standard method of encoding and transport of operations for information exchange.

3. CIM Schemas

The DMTF Common Information Model (CIM) [2] defines a standard object-oriented representation of management data to provide a common framework for description of a managed environment. CIM provides a conceptual view of a managed environment that attempts to unify and extend existing instrumentation and management standards, such as Simple Network Management Protocol (SNMP) and Desktop Management Interface (DMI), and to unify data obtained from a number of sources. The object-oriented design of CIM provides the advantages of abstraction and classification, object inheritance, the ability to directly model relationships, and the ability to define standard object behavior via inheritable methods.

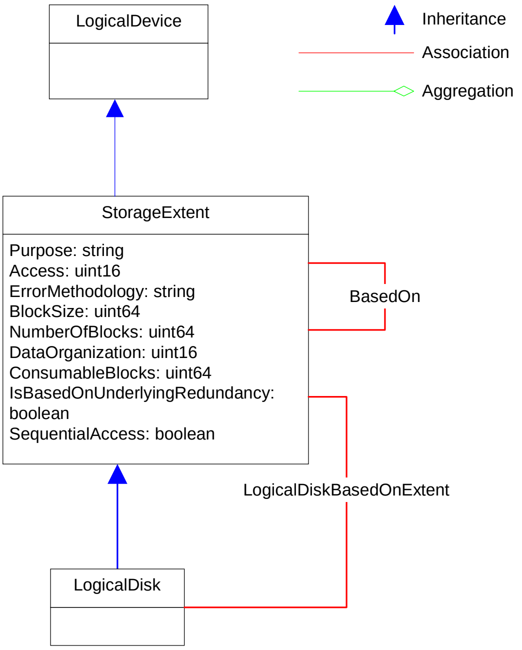
The DMTF provides CIM Schema definitions in both the Unified Modeling Language (UML) and Managed Object Format (MOF). The management model is divided into three conceptual layers, the Core Schema, the Common Schemas, and Extension Schemas. The Core Schema models information applicable to all domains of management. Examples of classes in the Core Schema include CIM_ManagedElement, CIM_ManagedSystemElement, CIM_LogicalElement, and CIM_LogicalDevice. Figure 1 shows the UML representation of a small part of the Core Schema, depicting the derivation of the CIM_LogicalDevice class, an abstraction or emulation of a hardware entity that may or may not be Realized in physical hardware, from


Dependency
ManagedElement
Description: string
Caption: string
Inheritance
Association
Aggregation
ManagedSystemElement
Name: string
Status: string
InstallDate: datetime
Component
LogicalElement
Synchronized
LogicalIdentity
LogicalDevice
CreationClassName: string [key]
DeviceID: string [key]
PowerManagementSupported: boolean
PowerManagementCapabilities: uint16[]
Availability: uint16
AdditionalAvailability: uint16
StatusInfo: uint16
LastErrorCode: uint32
ErrorDescription: string
ErrorCleared: boolean
OtherIdentifyingInfo: string[ ]
IdentifyingDescriptions: string[ ]
PowerOnHours: uint64
TotalPowerOnHours: uint64
MaxQuiesceTime: uint64
SetPowerState([IN] PowerState: uint16, [IN] Time: datetime): uint32
Reset(): uint32
EnableDevice([IN] Enabled: boolean): uint32
OnlineDevice([IN] boolean Online): uint32
QuiesceDevice([IN] boolean Quiesce): uint32
SaveProperties(): uint32
RestoreProperties() uint32

CIM_ManagedElement, an abstract class that provides a common superclass (or top of the inheritance tree) for the non-association classes in the CIM Schema.

Figure 1. Derivation of CIM_LogicalDevice in the Core Schema

The Common Schemas model information applicable to a specific management domain, but independent of any particular technology or implementation. The Common domains all derive from the same fundamental objects and concepts defined in the Core Schema, and are interrelated via associations and subclassing. The Common domains include Systems, Applications, Devices, and Users. Examples of classes in the Devices domain include CIM_StorageExtent and CIM_LogicalDisk. Figure 2 shows the UML representation of a small part of the Common Schema for the Devices domain, depicting the derivation of the CIM_LogicalDisk class, a representation of a contiguous range of logical blocks that is identifiable by a FileSystem via the Disk's DeviceId (key) field, from the CIM_LogicalDevice class. (The



CIM_LogicalDevice class is part of the Core Schema, but is included here to show the inheritance relationship.)

Figure 2. Derivation of CIM_LogicalDisk in the Devices Domain of the Common Schema

The Extension Schemas model technology-specific extensions to the Common Schemas, and are not specified by the DMTF. These classes are typically defined to apply to a specific environment, such as a particular operating system. For example, an HPUX_LogicalDisk class might be defined as a class in an Extension Schema derived from the CIM_LogicalDisk class in the Devices domain. HPUX_LogicalDisk could be used to represent a logical disk in the HP-UX environment, and model any aspects of a logical disk that are specific to the HP-UX environment, and not represented in CIM_LogicalDisk. Figure 3 shows a possible definition of this class represented in UML.

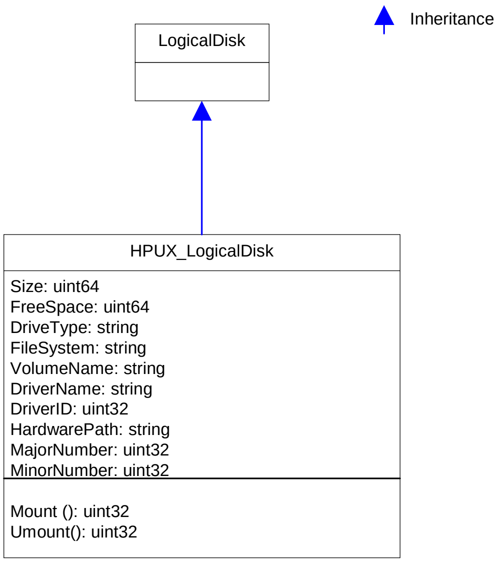


Figure 3. Derivation of HPUX_LogicalDisk in an Extension Schema

4. CIM Operations

While the CIM Schemas define a common representation of management data, a set of standard CIM operations provides a capability to discover and manage resources represented in the Schemas. Compliant CIM implementations must implement some subset or all of the standard CIM operations, must be able to receive and process CIM operation requests represented in XML encapsulated in HTTP, and must be able to issue appropriate operation responses represented in XML encapsulated in HTTP. The set of CIM operations, XML encoding, and transport via HTTP define a standard client interface. Typically, as a convenience, CIM implementations provide a client Application Programming Interface (API), that translates CIM operation requests into XML-encoded CIM requests encapsulated in HTTP (and perform a corresponding translation of the operation responses). However, no standard client API is defined. Thus a client management application would likely not be portable from one platform to another, but should be able to successfully perform CIM operations on different platforms where there are compliant CIM implementations.

All CIM operation requests are defined as invocations of one or more intrinsic or extrinsic methods. An intrinsic method is a method defined to model a CIM operation, such as EnumerateClassNames or GetProperty. The intrinsic methods are listed in Table 1 below. An intrinsic method is invoked on a CIM namespace. (A namespace is a unit for grouping classes and instances to control their scope and visibility. A namespace is not a physical location, but is more like a logical database containing specific classes and instances.) An extrinsic method is a method on a CIM class in some Schema, such as the EnableDevice method of the LogicalDevice class, or the Mount method of the HPUX_LogicalDisk class. An extrinsic method is invoked on a CIM instance, or in the case of a static method, on a CIM class.

The CIM operations are classified into functional profiles (Basic Read, Basic Write, Schema Manipulation, Instance Manipulation, Association Traversal, Query Execution, and Qualifier Declaration) to allow a range of implementations, and the specifications provide mechanisms by which CIM implementations can declare their level of support. The small number of functional profiles, and dependency relationships defined between the functional profiles, limit the number of different profiles that may be supported by a CIM implementation, and encourage interoperability.

Table 1. CIM Functional Profiles

Functional Group	Dependency	Methods
Basic Read	*none*	GetClass EnumerateClasses EnumerateClassNames GetInstance EnumerateInstances EnumerateInstanceNames GetProperty
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

Table 1 shows the partitioning of intrinsic methods into functional groups and the dependency relationships between functional groups. Any CIM implementation is assumed to support extrinsic method invocation; if it does not support extrinsic method invocations, a compliant implementation must return a specified error code to any request to execute an extrinsic method, allowing a client to determine that all attempts to execute extrinsic methods will fail.

For a target managed system running a compliant CIM implementation, a CIM client application might typically use the following process:

1. Discover CIM capabilities of the target server
2. Discover managed resources
3. Query managed resources

4. Manage resources

The client may be able to use the HTTP OPTIONS method (a CIM Server supporting only HTTP/1.0 would not support OPTIONS) to discover CIM capabilities of the target server, including: the version of the CIM HTTP mapping supported (the CIMProtocolVersion extension header), the CIM operations supported (the CIMSupportedFunctionalGroups extension header), whether multiple operation requests are supported (the CIMSupportsMultipleOperations extension header), and the query languages supported (the CIMSupportedQueryLanguages extension header). The client may use CIM operations such as EnumerateClasses, EnumerateInstances, or ExecQuery to discover managed resources on the target server, and may use CIM operations such as GetProperty to query managed resources. Finally, intrinsic methods such as SetProperty, and extrinsic methods such as EnableDevice, may be used to manage resources on the target system.

As an example, a client application could use the EnumerateInstanceNames operation to find instances of the CIM_LogicalDisk class and its subclasses, such as HPUX_LogicalDisk. Once instance names have been retrieved, they can in turn be used in other operations, such as GetProperty, to obtain more information about the instance. Similarly, an instance name can be used to invoke an extrinsic method, such as EnableDevice, to manage the resource.

5. XML Representation of CIM Operations

Representation of CIM operations in XML, and encapsulation in HTTP, provide a standard communication mechanism for CIM management operations, and allow interoperability between different CIM implementations. XML is a ubiquitous method for representing structured data in textual form, and has been standardized by the World Wide Web Consortium (W3C). Because it is based on an open industry standard, implementations of XML parsers are available for many platforms, in many programming languages, making XML an ideal standard medium for transferring data between heterogeneous platforms. Co-operating management platforms need not necessarily be running the same operating environments, but interoperability is enabled because of a common understanding of the XML representation of management information.

The DMTF has defined a standard for the representation of CIM elements and messages in XML in the *Specification for the Representation of CIM in XML* [*3*]. The *CIM XML Document Type Definition* (DTD) [4] defines the XML schema for CIM objects and messages. In the *Specification* *for CIM Operations over HTTP* [5], the DMTF defines a mapping of CIM operations onto HTTP that allows implementations of CIM to operate in an open, standardized manner.

For mapping CIM in XML, the DMTF adopted a metaschema mapping model, in which the XML schema is used to describe the CIM metaschema, and both CIM classes and instances are valid XML documents for the schema. The CIM XML DTD describes the concept of a CIM class or instance in a generic fashion, and CIM element names are mapped to XML attribute or element values, rather than XML element names. One benefit of the metaschema mapping is that changes or extensions to the schema do not require an update of the DTD.

The specification requires the use of extension headers to specify CIM operational semantics in the HTTP header of an M-POST or POST message. The HTTP POST method is used to send data to a server to be processed. In HTTP/1.1 (and later versions), the “M-“ prefix indicates that a POST request includes a mandatory extension declaration. A mandatory extension requires that the recipient consult and adhere to the rules given by the extension, when processing the message or reporting an error. HTTP header field prefixes allow an extension declaration to dynamically reserve a subspace of the header space in a message in order to prevent header field name clashes.

A CIM operation request or response must include the CIMOperation extension header. The CIMOperation extension header identifies the HTTP message as carrying a CIM operation request or response, and provides a simple mechanism by which firewall or proxy administrators can make global administrative decisions on all CIM operations. A CIM operation request must include either the CIMMethod and CIMObject extension headers, or the CIMBatch extension header. The CIMMethod extension header identifies the name of the CIM method to be invoked, and may be used by firewalls and proxies to make routing and forwarding decisions based on the CIM method to be invoked. The CIMObject extension header identifies the CIM object (i.e. a class or instance for an extrinsic method, or a namespace for an intrinsic method) on which the method is to be invoked, and may be used by firewalls and proxies to make routing and forwarding decisions based on the CIM object that is the target of method invocation. The extension headers allow Internet proxies and firewalls greater filtering control and administrative flexibility over CIM operation invocations.

The CIM XML DTD defines a message element that may be either a request or response. A request message must include a message ID value, which is also supplied in the corresponding response message, allowing a client to match a response to the corresponding request.

The following examples demonstrate some of the important features of the XML representation of a CIM operation request. Figure 4 shows an example of the XML representation of a CIM EnumerateInstanceNames operation request encapsulated in HTTP. The EnumerateInstanceNames operation is used to enumerate the names of the instances (i.e. the “model path”, or concatenation of key values, for each instance) of a CIM class in the target namespace. Note the extension headers (e.g. CIMOperation, CIMMethod) that match the XML content of the request, and the header prefix (“32-“) identifying each of these extension headers as belonging to the CIM over HTTP mandatory extension declaration (“Man: http://www.dmtf.org/cim/mapping/http/v1.0;ns=32”).

The CIM element is the root element of every XML document that is valid with respect to the CIM XML DTD. The MESSAGE element models a single CIM message, and is the basis for CIM operations. Note the required ID attribute, which defines an identifier for the message to be used as a correlation mechanism between the CIM client and CIM server. The SIMPLEREQ element defines a simple CIM operation request (i.e. one that requires the invocation of a single method), and contains either an IMETHODCALL element (as in this case) for an intrinsic method call, or a METHODCALL element for an extrinsic method call. The IMETHODCALL element defines a single intrinsic method invocation, and specifies the target namespace (the LOCALNAMESPACEPATH element), and any parameter values to be passed to the method. The IPARAMVALUE element defines an intrinsic method parameter value. In this case, EnumerateInstanceNames takes one parameter, ClassName, so there is one IPARAMVALUE element, containing a CLASSNAME element.

Figure 4. EnumerateInstanceNames request

```
M-POST /cimom HTTP/1.1
Host: www.hpserver1.com
Content-type: application/xml; charset="utf-8"
Content-length: 404
Man: http://www.dmtf.org/cim/mapping/http/v1.0;ns=32
32-CIMOperation: MethodCall
32-CIMProtocolVersion: 1.0
32-CIMMethod: EnumerateInstanceNames
32-CIMObject: root

<?xml version="1.0" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.0">
 <MESSAGE ID="4" PROTOCOLVERSION="1.0">
  <SIMPLEREQ>
   <IMETHODCALL NAME="EnumerateInstanceNames">
    <LOCALNAMESPACEPATH>
     <NAMESPACE NAME="root"> </NAMESPACE>
    </LOCALNAMESPACEPATH>
    <IPARAMVALUE NAME="CLASSNAME">
     <CLASSNAME NAME="CIM_LogicalDisk"/>
    </IPARAMVALUE>
   </IMETHODCALL>
  </SIMPLEREQ>
 </MESSAGE>
</CIM>
```

Figure 5 shows an example of the XML representation of a CIM GetProperty operation request encapsulated in HTTP. The GetProperty operation is used to retrieve a property value from a CIM instance. As in the EnumerateInstanceNames request, the IMETHODCALL element specifies an intrinsic method invocation, and the LOCALNAMESPACEPATH element specifies the target namespace for the intrinsic method. The GetProperty method takes two parameters, InstanceName and PropertyName, so there are two IPARAMVALUE elements. The type of subelement contained in the IPARAMVALUE element depends on the type of each parameter of the intrinsic method. In this case, one is an INSTANCENAME element for an instanceName parameter, and the other is a VALUE element for a string parameter. The INSTANCENAME element specifies the location of an instance in the target namespace, and is comprised of a class name and key binding information (a KEYBINDING element for each key property in the class). The second IPARAMVALUE element includes a VALUE element specifying the name of the property whose value is to be retrieved.

Figure 5. GetProperty Request

```
M-POST /cimom HTTP/1.1
HOST: www.hpserver1.com
Content-type: application/xml; charset="utf-8"
Content-length: 1279
Man: http://www.dmtf.org/cim/mapping/http/v1.0;ns=32
32-CIMOperation: MethodCall
32-CIMProtocolVersion: 1.0
32-CIMMethod: GetProperty
32-CIMObject: root

<?xml version="1.0" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.0">
   <MESSAGE ID="133" PROTOCOLVERSION="1.0">
      <SIMPLEREQ>
         <IMETHODCALL NAME="GetProperty">
            <LOCALNAMESPACEPATH>
               <NAMESPACE NAME="root"> </NAMESPACE>
            </LOCALNAMESPACEPATH>
            <IPARAMVALUE NAME="INSTANCENAME">
               <INSTANCENAME CLASSNAME="HPUX_LogicalDisk">
                  <KEYBINDING NAME="SystemCreationClassName">
                     <KEYVALUE VALUETYPE="string"> CIM_DiskGroup </KEYVALUE>
                  </KEYBINDING>
                  <KEYBINDING NAME="SystemName">
                     <KEYVALUE VALUETYPE="string"> hpserver1 </KEYVALUE>
                  </KEYBINDING>
                  <KEYBINDING NAME="CreationClassName">
                     <KEYVALUE VALUETYPE="string"> HPUX_LogicalDisk </KEYVALUE>
                  </KEYBINDING>
                  <KEYBINDING NAME="DeviceID">
                     <KEYVALUE VALUETYPE="string"> /dev/vg00/lvol3 </KEYVALUE>
                  </KEYBINDING>
               </INSTANCENAME>
            </IPARAMVALUE>
            <IPARAMVALUE NAME="PROPERTYNAME">
               <VALUE> FreeSpace </VALUE>
            </IPARAMVALUE>
         </IMETHODCALL>
      </SIMPLEREQ>
   </MESSAGE>
</CIM>
```

Figure 6 shows an example of extrinsic method invocation. Note that in this example, the METHODCALL and PARAMVALUE elements (as opposed to the IMETHODCALL and IPARAMVALUE elements) appear. The METHODCALL element defines a single method invocation on a class (if static) or instance (otherwise). In this case, a LOCALINSTANCEPATH element specifies an instance, and contains LOCALNAMESPACEPATH and INSTANCENAME elements. As in Figure 5, the INSTANCENAME element contains a KEYBINDING element for each key property of the class. A PARAMVALUE element specifies the value for the Enabled parameter to the EnableDevice method. A value of “FALSE” is specified to request that the device be disabled.

Figure 6. EnableDevice Method Call

```
M-POST /cimom HTTP/1.1
HOST: www.hpserver1.com
Content-type: application/xml; charset="utf-8"
Content-length: 1209
Man: http://www.dmtf.org/cim/mapping/http/v1.0;ns=31
31-CIMOperation: MethodCall
31-CIMProtocolVersion: 1.0
31-CIMMethod: EnableDevice
31-CIMObject: HPUX_LogicalDisk.DeviceID="/dev/vg00/lvol3"

<?xml version="1.0" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.0">
   <MESSAGE ID="134" PROTOCOLVERSION="1.0">
      <SIMPLEREQ>
         <METHODCALL NAME="EnableDevice">
            <LOCALINSTANCEPATH>
               <LOCALNAMESPACEPATH>
                  <NAMESPACE NAME="root"> </NAMESPACE>
               </LOCALNAMESPACEPATH>
               <INSTANCENAME CLASSNAME="HPUX_LogicalDisk">
                  <KEYBINDING NAME="SystemCreationClassName">
                     <KEYVALUE VALUETYPE="string"> CIM_DiskGroup </KEYVALUE>
                  </KEYBINDING>
                  <KEYBINDING NAME="SystemName">
                     <KEYVALUE VALUETYPE="string"> hpserver1 </KEYVALUE>
                  </KEYBINDING>
                  <KEYBINDING NAME="CreationClassName">
                     <KEYVALUE VALUETYPE="string"> HPUX_LogicalDisk </KEYVALUE>
                  </KEYBINDING>
                  <KEYBINDING NAME="DeviceID">
                     <KEYVALUE VALUETYPE="string"> /dev/vg00/lvol3 </KEYVALUE>
                  </KEYBINDING>
               </INSTANCENAME>
            </LOCALINSTANCEPATH>
            <PARAMVALUE NAME="Enabled"> FALSE </PARAMVALUE>
         </METHODCALL>
      </SIMPLEREQ>
   </MESSAGE>
</CIM>
```

6. CIM Management Infrastructure

Processing by the CIM management infrastructure, and interaction between the infrastructure and managed resources is implementation-specific, and does not directly affect interoperability of CIM implementations. However, a brief overview of a typical implementation is provided here, to aid in understanding WBEM. The CIM Object Manager (CIMOM) is a component in the CIM management infrastructure that handles the interaction between management applications and providers. The role of the CIMOM is to provide the infrastructure that allows a system to serve CIM content, not to provide the content itself. The CIMOM stores the CIM schema and some static data in a local repository, but most content is served dynamically by providers. The CIMOM maintains a mapping between CIM content and the providers that serve it, and forwards requests for dynamic data to the appropriate providers. A provider communicates with managed objects to access data from a variety of sources. Providers interact with managed resources via existing standard management protocols, such as SNMP or DMI, or via platform-specific interfaces. Providers forward information to the CIM Object Manager for integration and interpretation. The CIMOM supports services such as remote access and query processing, and also grants access to the CIMOM repository.

Providers can serve several types of content, including instances of classes, properties of instances, and methods. For some classes a single provider may serve instances, properties and methods, while for other classes, each type of content may be served by a separate provider. Two properties of a class may each be served by separate providers, and two methods of a class may likewise each be served by separate providers. For example, for the HPUX_LogicalDisk class, one provider might serve all instances of the class, two separate property providers might serve the FreeSpace and FileSystem properties, and two separate method providers might serve the Mount and Umount methods. Figure 7 shows an excerpt of an HPUX_LogicalDisk MOF, showing the specification of separate instance, property and method providers. In this case, the HPUX_LogicalDisk class also inherits a number of properties, which would be served by the instance provider.

Thus, the operations shown in the previous examples would be carried out by different providers. For example, the EnumerateInstanceNames operation would be performed by the HPUX_Provider, and the GetProperty operation by the HPUX_dfSpaceProvider.

When a CIM operation request comes in, the CIMOM parses the XML-encoded request, and if the requested data is static, retrieves it from the local repository. If the data is served by an object provider, the CIMOM passes the request to the provider, and receives the result of the operation from the provider. In either case, the CIMOM encodes the result in XML and sends it to the client over HTTP.

Figure 7. Excerpt of HPUX_LogicalDisk MOF

```
// ===================================================================
// HPUX_LogicalDisk
// ===================================================================
   [Description (
    "HPUX_LogicalDisk is a representation of a logical disk "
    "device in the HP-UX environment."),
    provider("HPUX_Provider")]

class HPUX_LogicalDisk:CIM_LogicalDisk
{
        [Description (
         "Disk free space"),
         provider("HPUX_dfSpaceProvider")]
   uint64   FreeSpace;

        [Description(
         "File system used"),
         provider("HPUX_FileSystemProvider")]
   string   FileSystem;

        [Description (
         "Mounts the logical disk.  Returns 0 if successful. "
         "Otherwise returns error code."),
         provider("HPUX_MountProvider")]
   uint32   Mount();

        [Description (
         "Unmounts the logical disk.  Returns 0 if successful. "
         "Otherwise returns error code."),
         provider("HPUX_UMountProvider")]
   uint32   Umount();
};
```

7. XML Representation of CIM Results

As described above, the results of a CIM operation are also encoded in XML and encapsulated in HTTP. The following examples demonstrate some of the important features of the XML representation of a CIM operation response. Figure 8 shows an example of the XML representation of a CIM EnumerateInstanceNames operation response encapsulated in HTTP. As in the CIM operation request examples, note the extension headers (e.g. CIMOperation) that match the XML content of the request, and the header prefix (“42-“) identifying each of these extension headers as belonging to the CIM over HTTP mandatory extension declaration (“Man: http://www.dmtf.org/cim/mapping/http/v1.0;ns=42”). The Ext header field is present to indicate that all mandatory extension declarations in the request were fulfilled. The no-cache cache-control directive is required to ensure that the Ext header field is not inadvertently cached in an HTTP/1.1 cache.

The message ID in the response message may be used to match a response to its corresponding request.   Note that the ID in the MESSAGE element of this response matches the ID sent in the MESSAGE element of the corresponding request (Figure 4). The SIMPLERSP element defines a simple CIM operation response, and contains either an IMETHODRESPONSE element (for an intrinsic method), or a METHODRESPONSE element (for an extrinsic method). The IMETHODRESPONSE element defines the response to an intrinsic CIM method invocation. It contains either an ERROR element (if a fundamental error prevented the method from executing at all), or an optional return value (as in this case). None of the intrinsic methods have output parameters. The IRETURNVALUE element specifies the value returned from an intrinsic method call. In this case, the EnumerateInstanceNames method returns the names of instances in the specified class in the target namespace. Thus in this example, the IRETURNVALUE element contains two INSTANCENAME elements. As in Figure 5, the INSTANCENAME elements each contain a KEYBINDING element for each key property of the class.

Figure 8. EnumerateInstanceNames Response

```
HTTP/1.1 200 OK
Content-type: application/xml; charset="utf-8"
Content-length: 1401
Ext:
Cache-Control: no-cache
Man: http://www.dmtf.org/cim/mapping/http/v1.0;ns=06
06-CIMOperation: MethodResponse
06-CIMProtocolVersion: 1.0

<?xml version="1.0" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.0">
 <MESSAGE ID="4" PROTOCOLVERSION="1.0">
  <SIMPLERSP>
   <IMETHODRESPONSE NAME="EnumerateInstanceNames">
    <IRETURNVALUE>
     <INSTANCENAME CLASSNAME="HPUX_LogicalDisk">
      <KEYBINDING NAME="SystemCreationClassName">
       <KEYVALUE VALUETYPE="string"> CIM_DiskGroup </KEYVALUE>
      </KEYBINDING>
      <KEYBINDING NAME="SystemName">
       <KEYVALUE VALUETYPE="string"> hpserver1 </KEYVALUE>
      </KEYBINDING>
      <KEYBINDING NAME="CreationClassName">
       <KEYVALUE VALUETYPE="string"> HPUX_LogicalDisk </KEYVALUE>
      </KEYBINDING>
      <KEYBINDING NAME="DeviceID">
       <KEYVALUE VALUETYPE="string"> /dev/vg00/lvol3 </KEYVALUE>
      </KEYBINDING>
     </INSTANCENAME>
     <INSTANCENAME CLASSNAME="HPUX_LogicalDisk">
      <KEYBINDING NAME="SystemCreationClassName">
       <KEYVALUE VALUETYPE="string"> CIM_DiskGroup </KEYVALUE>
      </KEYBINDING>
      <KEYBINDING NAME="SystemName">
       <KEYVALUE VALUETYPE="string"> hpserver1 </KEYVALUE>
      </KEYBINDING>
      <KEYBINDING NAME="CreationClassName">
       <KEYVALUE VALUETYPE="string"> HPUX_LogicalDisk </KEYVALUE>
      </KEYBINDING>
      <KEYBINDING NAME="DeviceID">
       <KEYVALUE VALUETYPE="string"> /dev/vg00/lvol1 </KEYVALUE>
      </KEYBINDING>
     </INSTANCENAME>
    </IRETURNVALUE>
   </IMETHODRESPONSE>
  </SIMPLERSP>
 </MESSAGE>
</CIM>
```

Figure 9 shows an example of the XML representation of a CIM GetProperty operation response encapsulated in HTTP. As in Figure 8, the CIMOperation extension header indicates that this message is a MethodResponse, and the ID value in the MESSAGE element matches that in the corresponding request message (Figure 5). In this case, the IRETURNVALUE element contains a single VALUE element defining the returned property value.

Figure 9. GetProperty Response

```
HTTP/1.1 200 OK
Content-type: application/xml; charset="utf-8"
Content-length: 337
Ext:
Cache-Control: no-cache
Man: http://www.dmtf.org/cim/mapping/http/v1.0;ns=42
42-CIMOperation: MethodResponse
42-CIMProtocolVersion: 1.0

<?xml version="1.0" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.0">
   <MESSAGE ID="133" PROTOCOLVERSION="1.0">
      <SIMPLERSP>
         <IMETHODRESPONSE NAME="GetProperty">
            <IRETURNVALUE>
               <VALUE> 2980814848 </VALUE>
            </IRETURNVALUE>
         </IMETHODRESPONSE>
      </SIMPLERSP>
   </MESSAGE>
</CIM>
```

Figure 10 shows an example of the XML representation of an extrinsic method response encapsulated in HTTP. Note that in this example, the METHODRESPONSE and RETURNVALUE elements (as opposed to the IMETHODRESPONSE and IRETURNVALUE elements) appear. The METHODRESPONSE element defines the response to a CIM extrinsic method invocation. It contains either an ERROR element (if a fundamental error prevented the method from executing at all), or a combination of an optional return value (as in this case) and zero or more PARAMVALUE elements (depending on whether the method has output parameters). The RETURNVALUE element specifies the value returned from an extrinsic method call. In this case, the EnableDevice method returns 0 when the request has successfully been executed. The EnableDevice method has no output parameters, so no PARAMVALUE elements are present.

Figure 10. EnableDevice Method Response

```
HTTP/1.1 200 OK
Content-type: application/xml; charset="utf-8"
Content-length: 325
Ext:
Cache-Control: no-cache
Man: http://www.dmtf.org/cim/mapping/http/v1.0;ns=43
43-CIMOperation: MethodResponse
43-CIMProtocolVersion: 1.0

<?xml version="1.0" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.0">
   <MESSAGE ID="134" PROTOCOLVERSION="1.0">
      <SIMPLERSP>
         <METHODRESPONSE NAME="EnableDevice">
            <RETURNVALUE>
               <VALUE> 0 </VALUE>
            </RETURNVALUE>
         </METHODRESPONSE>
      </SIMPLERSP>
   </MESSAGE>
</CIM>
```

8. Summary

The key WBEM components that enable interoperability are CIM, XML, and HTTP. CIM provides a standard representation of management data, with the object-oriented advantages of abstraction and classification, object inheritance, the ability to directly model relationships, and the ability to define standard object behavior via inheritable methods. CIM operations provide a standard way to discover, query, and manage resources. Representation in XML, and encapsulation in HTTP, provide a standard communication mechanism for CIM management operations, and allow interoperability between different compliant CIM implementations. Interoperability is possible because WBEM is based on established common standards and technologies not specific to any platform.

9. Acknowledgments

The author would like to thank Susan Campbell, Denise Eckstein, Michael Glantz, Roger Kumpf, and Lyle Wilkinson, who each reviewed an earlier draft of this paper, providing helpful comments and suggestions; and Lihui Zhao, who developed a sample HPUX_LogicalDisk provider, which was used to create some of the examples in this paper.

10. References

1. *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*, IEEE, 1990

2. *Common Information Model (CIM) Specification*, Version 2.2, DMTF, 14 June 1999 (http://www.dmtf.org/spec/cim_spec_v22)

3. *Specification for the Representation of CIM in XML*, Version 2.0, DMTF, 20 July 1999 (http://www.dmtf.org/download/spec/xmls/CIM_XML_Mapping20.htm)

4. *CIM XML Document Type Definition*, Version 2.0, DMTF, 20 July 1999 (http://www.dmtf.org/download/spec/xmls/cim_dtd_V20.txt)

5. *Specification for CIM Operations over HTTP*, Version 1.0, DMTF, 11 August 1999 (http://www.dmtf.org/download/spec/xmls/CIM_HTTP_Mapping10.htm)