http://www.wbem.co.uk

The Web and Web Based Enterprise Management

Technology Whitepaper

Abstract

In Chapters 3 and 5 of **Developing WMI Solutions** we devoted a lot of time to examining the architecture of Windows Management Instrumentation (WMI). We considered the way that the various components, such the CIM Object Manager (CIMOM) and management applications like the WMI CIM Studio could be distributed to provide the flexibility that is so often needed in Enterprise management. This flexibility enables system administrators to access management data and invoke management operations both locally and remotely on WMI compliant machines. There was however one caveat; the management application and CIMOM must be running on the same implementation of WBEM.

In January 2003 the DMTF released the latest version of a series of standards that enable the transfer of CIM data between WBEM implementations regardless of the underlying operating system. These standards allow events to be surfaced, methods invoked, CIM data to be retrieved and sent between any two WBEM compliant management architectures. The purpose of this article is to examine how these standards and the supporting web technologies, XML and HTTP combine to help achieve this goal.

Note: Currently this standard does not apply to providers. For example the standards do not make it possible for a provider developed as part of the WMI architecture to register itself and then interact directly with a CIMOM that has been implemented on a Unix based host.

Introduction

Imagine that you are the administrator on a corporate network populated with a variety of different operating systems such as Unix, Mac OS and Windows. Lets also assume that each of these machines is running a WBEM compliant management architecture. At present the computer running Unix or Mac OS cannot share its management information or have its management methods invoked by a Windows based client application and vice versa. Consequently a network administrator would need three different management tools (e.g. WMI CIM Studio for Windows) to retrieve the management information from each of the different operating systems. In my experience this is problematic for several reasons.

First the administrator needs to be cognisant in multiple management applications. This can lead to increased training costs and more importantly can increase the potential for administrator error.

Second, at a more fundamental level it also leads to a division between the management information generated by different operating systems. For example if we administered a Unix based SQL server running as the back end of a Windows based Web Server this relationship would be difficult to capture using two separate management applications (one for each OS). This could prevent the administrator from resolving faults that span multiple operating systems as each management application could only represent a portion of the overall domain. All of these problems can contribute to an increase in the total cost of ownership which was one of the factors the DMTF's WBEM initiative was aiming to reduce.

Note: In theory it is possible to exchange CIM management information between two different operating systems by exporting and importing MOF files. In practice however this is impractical due to the inherently dynamic nature of much management information. Dynamic data would be completely redundant by the time it had been imported into the destination namespace. The MOF language also does not make provision for triggering events or invoking methods.

Since 1999 the Distributed Management Task Force (DMTF) has been working on a solution to these problems and has developed a protocol that can enable different operating systems to share CIM management information and perform management operations.

The Problem

There are essentially two problems that need to be overcome to enable communication of CIM management information between OS platforms.

Note: By m*anagement information* we refer to meta-data such as CIM qualifiers, to class or instance information and operations such as method invocations.

First you need to provide a way of packaging CIM management information in a platform independent way. This is because all management information needs to packaged up before it is sent and then unpackaged by the recipient and interpreted. The DMTF chose the eXtensible Markup Language (XML) for this task. There are a number of different ways that CIM information could have been represented in XML so the DMTF defined a standard for its use to ensure that all parties used the same approach.

Note: For more information on XML and CIM see, "Representation of CIM in XML" which can be found at http://www.dmtf.org/standards/standard_wbem.php

Second the CIM management information needs to be transported between interacting parties and each party must operate in an open standardised manner. For example tasks such as invoking a method require the recipient to actually take some action. These services require explicit set-up information in order to take place. Existing *communication* protocols such as TCP/IP enable heterogeneous systems to exchange information freely but do not provide a mechanism for specifically conveying or setting up connections between applications that deal in management information. To illustrate this point lets take a look at the OSI 7 layer model. In figure 1.0 we can see that TCP operates at the transport layer (4) and IP at the network layer (3). Protocols operating at the transport layer help ensure reliability and integrity of data whereas the network layer protocols establish, maintain and terminate network connections. Together these protocols provide a way of getting information to a destination but do not set-up Enterprise management specific services such as the retrieval of an instance of a class for an application. In other words they don't carry any information that WMI CIM studio could directly interpret in to CIM related actions. This is the function of the protocols residing at the application layer (7). So in order for the DMTF to convey management information they first needed to select (or define) an appropriate protocol for use at this layer. For this task the DMTF chose the Hyper Text Transfer Protocol (HTTP) which can handle the set-up between CIM products. As we can see from figure 1.1 HTTP operates above TCP/IP.

Note: For more information on CIM and HTTP see, "CIM Operations over HTTP" which can be found at http://www.dmtf.org/standards/standard_wbem.php

Windows Management Instrumentation

When Microsoft developed WMI they also created their own protocol for communicating CIM management information across networks of Windows machines. For example when the WMI CIM Studio connects to a namespace hosted on a remote machine and retrieves an instance it is passing and receiving CIM management information. It does so however using Microsoft's own proprietary format with COM/DCOM as the underlying communication protocol. Because this format does not conform to any open standard it means it cannot communicate CIM management

information between different implementations of WBEM. For example even a simple class definition may be encoded many different ways and different implementations of WBEM may adopt different approaches.

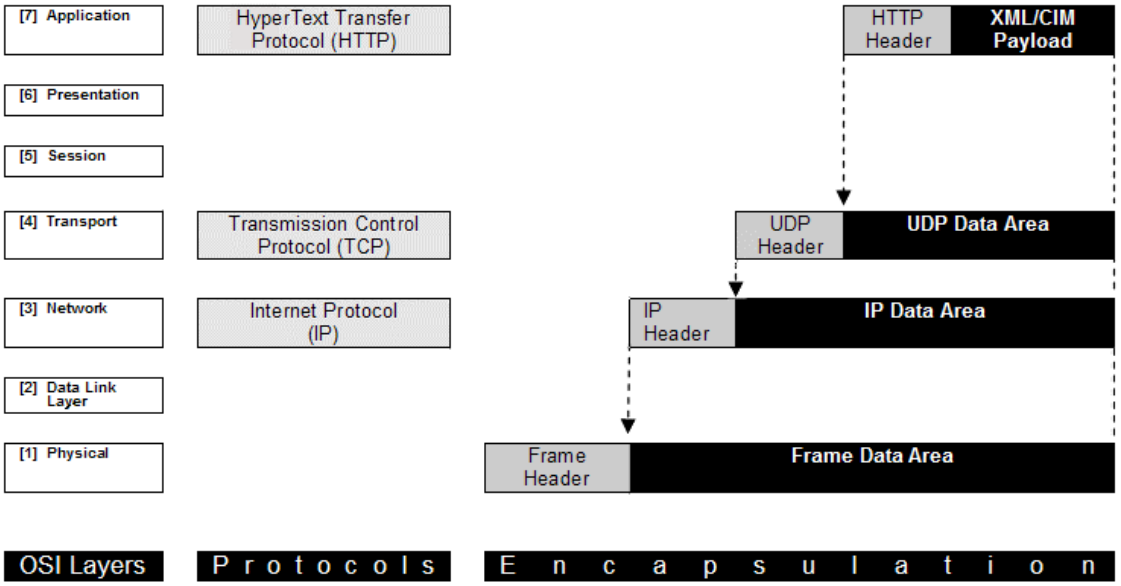

Figure 1.0 Illustrates how XML/CIM data is encapsulated.

Consequently the DMTF's standards handle the encoding of CIM data (e.g. class, instance, meta-data) and the operations that can be performed (e.g. queries, association traversal, method invocation, instance retrieval and so on). By complying with these standards companies implementing WBEM such as Microsoft and Hewlett Packard can all interpret management data from other WBEM implementations. In other words they will be able to use the same client application to manage any devices that conform to the WBEM standard regardless of operating system.

Advantages of using HTTP

One of the primary design goals for enabling communication of CIM information between different networks and operating systems was to ensure that it could be used effectively within existing Enterprise networking infrastructures. In choosing HTTP the DMTF knew that WBEM could take maximum advantage of a protocol that was well established with network administrators. This meant that it could work effectively with commonly found networking devices such as firewalls, proxies etc. Also from a security perspective HTTP can use the secure hypertext transfer protocol (HTTPS) to protect management information from prying eyes. HTTPS uses Secure Socket Layer (SSL) to transfer encrypted information across a network. By using HTTP the DMTF ensured that communicating Enterprise management information over a corporate LAN/WAN/VPN would be as easy as it is to use browsers to access Web pages.

CIM Messages

The DMTF have categorised the different tasks that can be carried out between CIM products as a series of *messages*. A CIM Message is a well-defined request or response data packet used to exchange information between CIM applications. These come in two flavours: -

- A *CIM Operation Message* which invokes an operation on a target namespace.

- A *CIM Export Message* which conveys information about a CIM namespace or element that is foreign to the target. It does not define an operation on the target CIM namespace or even imply the presence of the namespace. This forms part of the CIM Event model and is used to deliver *Indications*.

For the purpose of this document we are only interested in CIM Operation Messages. The DMTF's recommendations specify that CIM Operation Messages adhere to the request-reply paradigm. For each valid request (i.e. when you ask a CIM product to do something) the DMTF asserts that there must be a valid response or an error message if unsuccessful. If your product is to support the standard then it must behave in the way defined in CIM Operations over HTTP in its requests, responses and error messages.

CIM Operations fall into two categories: -

- **Extrinsic** methods operate on a CIM Class in a schema. For example calling the terminate() method on class Win32_Process is an example of an extrinsic method;

- **Intrinsic** methods model a CIM operation that operates on a schema or namespace. For example retrieving a class definition or instance of a class is an intrinsic method.

The DMTF categorise the following operation requests as types of intrinsic method.

Classification	Description	Method(s)
Queries	Provide Methods that deal with queries.	o ExecQuery o Associators o References o Enumerators
Meta-Data	Provide methods that operate on meta-data.	o GetClass o DeleteClass o CreateClass o ModifyClass o EnumerateClasses o EnumerateClassNames o GetQualifier o SetQualifier o DeleteQualifier o EnumerateQualifier
Data	Provide a set of methods that operate on instances of CIM classes.	o GetInstance o DeleteInstance o CreateInstance o ModifyInstance o EnumerateInstances o EnumerateInstanceNames o GetProperty o SetProperty

Table 1.0 Intrinsic Methods classified as being either query, meta-data or data related.

The Hyper Text Transfer Protocol (HTTP)

HTTP is a client/server protocol. This means that an application acting as an HTTP server passively waits for requests from one or more HTTP clients. HTTP is the protocol used to set-up services between Web browsers and Web servers but with a few cunning additions (called extension headers) it is equally well suited to the task of sending and receiving CIM management data. Before we look at the extensions defined by the DMTF lets first look at the three basic roles that CIM products can assume when interacting via HTTP: -

Role	Issues	Receives
A *CIM Client*	CIM Operation Request OR CIM Message Requests	CIM Operation Response OR CIM Message Responses
A *CIM Server*	CIM Operation Message Responses	CIM Operation Message Request
A *CIM Listener*	CIM Export Message Responses.	CIM Export Message Requests

Table 1.1 The three roles for CIM products interacting via HTTP.

A simple HTTP `GET` example

You will most likely have encountered HTTP when you type in the Uniform Resource Locator (URL) of a web site into a Web browser. For example in order to look at this document you would have typed HTTP://www.wbem.co.uk into your Web browser which sends an HTTP GET request to our Web server.

HTTP defines a number of request methods that are of interest to us: -

- `GET` asks to retrieve a document or sends form data in which case the form contents are appended to the URL;
- `POST` as defined in Version 1.0 of HTTP. Commonly used to send form data;
- `M-POST` as defined in Version 1.1 of HTTP. Also used to send form data. This new version however allows a decentralized naming mechanism whereby parties can introduce additional HTTP Headers without fear of conflicting interpretation of a given Header name.

Of the three methods the `M-POST` method is the one of most interest but before we look at it in more detail lets examine `GET` and `POST`.

A simple HTTP `GET` example

First let's look at one of the most basic operations in HTTP; requesting some information. This is relevant to our discussion because it is also one of the most basic operations we will want to do with management information.

When you type in a URL into your web browser the first thing it does is perform some background work such as determining the destination port address and IP address. Then using this information it then constructs an HTTP request. A GET request is then sent to the destination machine and the browser waits for a response. For example:

```
GET index.html HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.02; Windows NT 5.1; .NET
CLR 1.0.3705)
Host: www.wbem.co.uk
Accept: image/gif, image/jpeg, */*
```

The first line identifies the type of request, `GET`, and the version of HTTP being used, which is the most current version 1.1. Next the type of `User Agent` is identified; this is basically the type and version of Web browser which on my machine is Internet Explorer version 6.02. The `Host:` field

identifies where the GET message is being sent to. The `Accept:` field define the data types that be sent in response to the GET request. Notice that there is no data included in the message body as everything is held in the head of the HTTP request. The GET request would then be met with the following response:

```
HTTP/1.1 200 OK
Date: Fri, 17 Jan 2003 14:32:18 GMT
Server: Apache/1.3.27 (Unix) mod_perl/1.27 PHP/4.2.3 mod_fastcgi/2.2.12
FrontPage/5.0.2.2510 mod_jk/1.2.0 mod_ssl/2.8.11 OpenSSL/0.9.6g
Content-Type: text/html
```

This would then be immediately followed by the contents of the response in the message body, which is our webpage:

```
<HTML>
<HEAD>
<TITLE>Developing WMI Solutions</TITLE>
  …
```

Next let's take a look at sending information along with the request.

Sending data with `Get` and `POST`

Having seen how easy it is to request data using the GET method let's look at examples of how the GET and POST methods can be used to transmit data.
The POST message is most commonly used to transmit HTML form data. For an example of an HTML form see http://www.wbem.co.uk/contact.html. Figure 1.1 shows a screenshot of the form on our website with the fields filled in.

Figure 1.1 An HTML form in action. Screenshot taken from http://www.wbem.co.uk/contacts.shtml

Let's see that data as sent using the GET method.

```
GET /cgi-bin/formproc.pl?Name:=Craig%20Tunstall&Email=Craig%2ETunstall
```

```
@wbem%2Eco%2Euk&Occupation=Author&Country=England&Message=This%20is%20a
%20simple%20form HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.02; Windows NT 5.1; .NET
CLR 1.0.3705)
Host: www.wbem.co.uk
Accept: image/gif, image/jpeg, */*
```

Ugly eh? The difference between the `GET` method and the `POST` method is that `GET` places the form data (or any data you wanted to send) in the first line immediately after the URL (i.e. `/cgi-bin/formproc.pl?`). If the information was sent using the POST method it would appear as follows:

```
POST /cgi-bin/post-query HTTP/1.0
Accept: www/source
Accept: text/html
Accept: image/gif, image/jpeg, */*
User-Agent:  Mozilla/4.0 (compatible; MSIE 6.02; Windows NT 5.1; .NET
CLR 1.0.3705)
From:  test@www.wbem.co.uk
Content-type: application/x-www-form-urlencoded
Content-length: 150
     * a blank line *
Name:=Craig%20Tunstall
&Email=Craig%2ETunstall@wbem%2Eco%2Euk
&Occupation=Author
&Country=England
&Message= This%20is%20a%20simple%20form
```

The point of most interest in this example is that the POST message inserts the data after the HTTP header in what's called the message body or *payload*. This removes the length restriction imposed by the GET method. Also notice that each of the fields are paired with the values that I typed in on the HTML form. A typical servers response to this message would be to process the data and then return an HTML page confirming that the form submission was a success.

It may appear that the Get method is an ideal way to retrieve and send CIM data. There are several problems with adopting this approach.
First, GET is restricted to only being able to transmit ASCII characters. This is far from ideal given the diverse nature of Enterprise management information.
Second, there is a length restriction of data passed as part of a GET method. For example Internet Explorer limits its GET data length to 2048 characters, minus the number of characters in the path, to avoid errant or malicious behaviour. This limitation is clearly impractical when dealing with Enterprise management data.
Third the GET method also displays the contents of its query as part of the URL that is submitted which is not very secure.

Note: The DMTF documentation specifies that an attempt should always be made to send data using the M-POST method initially. According to DMTF the M-POST method allows Internet proxies & firewalls greater filtering control and administrative flexibility over CIM Message invocations than POST. If this fails then the client should revert to the POST method. The documentation also states that clients and servers should also be compatible with HTTP V1.0 or greater.

Retrieving an instance using the `M-POST` Header

To enable CIM management information to be passed across a network using HTTP the DMTF have defined an HTTP extension declaration in accordance with RFC 2774. This allows them to define their own set of CIM specific protocol operations. They are also able to define how the message should be processed and also any success/error messages that should be sent back. A definition of these can be found in the DMTF document, '*CIM operations over HTTP'*.
In the following example we construct an HTTP header using these extensions and also a XML/CIM payload for retrieving an instance of class Win32_Process.

The HTTP header for the M-POST method starts as follows:

```
M-POST /cimom HTTP/1.1
```

This declares the header as type M-POST, using V1.1 of HTTP (remember HTTP V1.0 did not support extension headers). The /cimom is there so that the CIM Object Manager knows the following contents are for its attention.

```
HOST:http://www.wbem.co.uk/
```

This line identifies the address of CIMOM. This value could equally be an IP address.

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

This declares the type of the payload which is XML and the character set it uses. UTF-8 is a standard for encoding Unicode characters and stands for UCS-Transformation Format. For more information see RFC-2279.

```
Content-Length: 511
```

This describes the length in bytes of the payload not including the HTTP header.

```
Man:http://www.dmtf.org/cim/mapping/http/v1.0; ns=44
```

All M-POST CIM Messages must use the `Man` extension followed by the namespace given above. `Man` is short for mandatory and tells the recipient that the following Message adheres to the rules defined by the HTTP extension for processing a Message as defined in http://www.dmtf.org/cim/mapping/http/v1.0. Your CIM messages should always include this line.

The value *ns* (short for name space – not to be confused with a CIM Namespace, see http://www.ietf.org/rfc/rfc2774.txt for more details) contains the header prefix which is a random two digit number between 0 and 99 (in this case 44). This value is used by all the following header fields to identify them as belonging to this extension declaration. This value is reassigned for each HTTP message (so the response Message will have a different *ns* value).

```
44-CIMOperation:MethodCall
```

Next we specify the CIMOperation as being a MethodCall. The only other type of operation would be MethodResponse. Notice the value of *ns* precedes CIMOperation.

```
44-CIMMethod: GetInstance
44-CIMObject: root%2fcimv2
```

Next our HTTP header declares CIMMethod to specify the type of intrinsic method we wish to call; in this case GetInstance. Table 1.0 outlines the currently available intrinsic methods. Notice the method definition for GetInstance has been supplied with its associated Input and Output parameters as this information will be useful when we define the XML/CIM payload for our message.

CIMObject identifies the namespace that the operation will be carried out on; root/CIMv2. Notice we use % to denote a hex value and 2f (47 decimal) is forward slash in ASCII.

Purpose	Operation
Get a CIM Class	`GetClass`

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

Delete a CIM Class	`DeleteClass`
Delete a CIM Instance	`DeleteInstance`
Create a CIM Class	`CreateClass`
Create a CIM Instance	`CreateInstance`
Modify a CIM Class	`ModifyClass`
Modify a CIM Instance	`ModifyInstance`
Enumerate subclasses of a CIM Class	`EnumerateClasses`
Enumerate subclass names of a CIM Class	`EnumerateClassNames`
Enumerate instances of a CIM Class	`EnumerateInstances`
Enumerate instance names of a CIM Class	`EnumerateInstanceNames`
Execute a Query	`ExecQuery`
Enumerate associators of a CIM Object	`Associators`
Enumerate names of associators of a CIM Object	`AssociatorNames`
Enumerate references to a CIM Object	`References`
Enumerate names of references to a CIM Object	`ReferenceNames`
Get a CIM Property value from a CIM Instance	`GetProperty`
Set a CIM Property value from a CIM Instance	`SetProperty`
Get a Qualifier declaration	`GetQualifier`
Set a Qualifier declaration	`SetQualifier`
Delete a Qualifier declaration	`DeleteQualifier`
Enumerate Qualifier declarations	`EnumerateQualifier`

Table 1.2 A list of currently defined intrinsic methods and their purpose.

Using XML to represent CIM

At this point in our example we have passed information in the HTTP header that instructs the recipient that we wish to retrieve an instance of a class from the root/CIMV2 namespace. Now we need to specify which class and instance we want to retrieve. To do this we use XML.

XML is not that dissimilar in appearance to HTML. Whereas HTML defines the format of the information displayed on your screen, XML allows users to define a mark-up language for their

own specific type of data. Hence the DMTF used it to define a CIM mark-up language. XML consists of start <> and end </> tags that combine to form an element (like HTML). For example here we declare an element of type INSTANCENAME.

```
<INSTANCENAME CLASSNAME="Win32_Process">  .... </INSTANCENAME>
```

CLASSNAME is an attribute. Attributes are always contained in the start tag of an element.
The main difference is that in HTML these tags are predefined whereas in XML you can define your own. The resulting definitions are held in something called a Document Type Definition (DTD).

A few pointers about XML: -

- Element type names are case sensitive.
- <INSTANCENAME/> denotes an empty element which means no end tag is needed.
- Each (non-empty) element must have a starting and ending tag.
- Tags must maintain their order in nested elements.

So lets have a look at an XML encoded CIM Message.

```
<?xml version="1.0" encoding="utf-8" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.1.1">
<MESSAGE ID="12345" PROTOCOLVERSION="1.0">
<SIMPLEREQ>
```

At the start of every XML message body is a processing instruction that tells the recipient the version number and character encoding of the Message. The next line specifies the CIMVERSION and DTDVERSION number of the Message.

Note: The DMTF specifies that all CIM Messages must be *well defined*. This means that they must use the terms defined in the XML CIM Document Type Definition (DTD). A DTD is an XML document which describes all of the legal expressions that you can use during interactions. Its written in plain text and can be downloaded from the DMTF (http://www.dmtf.org/download/spec/xmls/CIM_DTD_V20.dtd). An explanation of the DTD can be found in the specification for the representation of CIM in XML which is also on the DMTF website (Version 2.01 http://www.dmtf.org/standards/documents/WBEM/DSP0201.zip).

Next we define the ID and PROTOCOLVERSION of the MESSAGE. The ID attribute holds a unique identifier for the Message and helps correlate between two CIM entities so that they can distinguish which responses are for which requests and so on. The PROTOCOLVERSION attribute defines the version of the CIM mapping for the Message.

On the next line we declare the type of message being sent. There are two types of message, simple message requests <SIMPLEREQ> and multiple operation requests <MULTIREQ>. As we are only doing one thing, retrieving an instance of a class, we use <SIMPLEREQ>. Multiple operation requests contain two or more subelements defining the <SIMPLEREQ> elements that make up the request. Multiple requests are ideal for making more efficient use of bandwidth when exchanging many requests.

```
  <IMETHODCALL NAME="GetInstance">
  <LOCALNAMESPACEPATH>
    <NAMESPACE NAME="root"/>
    <NAMESPACE NAME="CIMV2"/>
   </LOCALNAMESPACEPATH>
```

Next we declare the Method we want to invoke, the GetInstance method. This is specified in the NAME attribute in the IMETHODCALL element. IMETHODCALL is used for intrinsic methods and METHODCALL for extrinsic methods.
On the next line we specify the path of the CIM namespace using the LOCALNAMESPACEPATH element. It is specified on the following two lines as root/CIMV2. Notice the use of the forward slash on NAMESPACE to identify it as an empty element. Next the end tag for /LOCALNAMESPACEPATH is given.

The next step is to pass the parameters for the GetInstance method. Before we do that let's take a quick look at the GetInstance Method declaration again:

```
<instance>GetInstance (
[IN] <instanceName> InstanceName,
//input parameter defines the name of the Instance to be retrieved

[IN,OPTIONAL] boolean LocalOnly = true,
// specifies that only CIM Elements (properties, methods and qualifiers) defined or
overridden within the definition of the Class are returned.

[IN,OPTIONAL] boolean IncludeQualifiers = false,
// specifies that all Qualifiers for that Instance (including Qualifiers on the Instance
and on any returned Properties) MUST be included as <QUALIFIER> elements in the response.

[IN,OPTIONAL] boolean IncludeClassOrigin = false,
//specifies that the CLASSORIGIN attribute MUST be present on all appropriate elements in
the returned Instance.

[IN,OPTIONAL,NULL] string PropertyList [] = NULL
//the members of the array define one or more Property names
)
```

From the declaration we can see that method GetInstance returns an object of type *instance*. This is what will be returned in the message response. GetInstance takes five parameters. The keyword OPTIONAL tells us that LocalOnly, IncludeQualifiers, IncludeClassOrigin and PropertyList can be omitted in which case they will assume the default values given. Let's take a look at how we pass these parameters in XML.

```
    <IPARAMVALUE NAME="InstanceName">
      <INSTANCENAME CLASSNAME="Win32_Process">
       <KEYBINDING NAME="Handle"><KEYVALUE>44</KEYVALUE></KEYBINDING>
      </INSTANCENAME>
    </IPARAMVALUE>
    <IPARAMVALUE NAME="LocalOnly"><VALUE>FALSE</VALUE></IPARAMVALUE>
```

The first line passes the parameter InstanceName as an IPARAMVALUE. Next the name of the class we wish to retrieve, Win32_Process, is given. The next line specifies the name of the key property for that class which is Handle. We also pass the unique identifying value for our instance which is 44. Note that this value is not contained within the start or end tag and is referred to as the elements *content*. We then change the settings on LocalOnly parameter to FALSE so that we retrieve all property values including those defined in parent classes.

```
   </IMETHODCALL>
  </SIMPLEREQ>
 </MESSAGE>
</CIM>
```

We end the message with a series of XML closing tags in the reverse order that the start tags were declared. So to recap, Figure 1.2 shows the finished HTTP header with XML/CIM message as its payload.

```
M-POST /cimom HTTP/1.1                                          HTTP Header
HOST:http://www.wbem.co.uk/
Content-Type: application/xml; charset="utf-8"
Content-Length: 511
Man:http://www.dmtf.org/cim/mapping/http/v1.0; ns=44
44-CIMOperation:MethodCall
44-CIMMethod: GetInstance
44-CIMObject: root%2Fcimv2
                                                                XML/CIM Payload
<?xml version="1.0" encoding="utf-8" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.1.1">
  <MESSAGE ID="12345" PROTOCOLVERSION="1.0">
    <SIMPLEREQ>
      <IMETHODCALL NAME="GetInstance">
        <LOCALNAMESPACEPATH>
          <NAMESPACE NAME="root"/>
          <NAMESPACE NAME="CIMV2"/>
        </LOCALNAMESPACEPATH>
        <IPARAMVALUE NAME="InstanceName">
          <INSTANCENAME CLASSNAME="Win32_Process">
            <KEYBINDING NAME="Handle"><KEYVALUE>44</KEYVALUE></KEYBINDING>
          </INSTANCENAME>
        </IPARAMVALUE>
        <IPARAMVALUE NAME="LocalOnly"><VALUE>FALSE</VALUE>
        </IPARAMVALUE>
      </IMETHODCALL>
    </SIMPLEREQ>
  </MESSAGE>
</CIM>
```


Figure 1.2 An example GetInstance Message including HTTP header and XML/CIM payload.

Conclusions

During this paper we have briefly looked at the rationale for adopting HTTP and XML as a means of conveying CIM management information between heterogeneous systems. As a simple example we examined how HTTP and XML could be configured using the standards defined by the DMTF to retrieve an instance of class Win32_Process. Due to space limitations we did not have time to look at the structure of multiple requests, export messages or the invocation of extrinsic methods but I hope to have provided you with a basic introduction.

From a developers perspective these standards mean that management applications and managed devices will be able to send and receive information from any WBEM compliant implementation regardless of its underlying operating system. From an administrators standpoint it means that Enterprise management can realistically become more centralised and that tracing faults to their origin now truly has the potential of Enterprise scope.

http://www.wbem.co.uk

Bibliography

The Distributed Management Task Force
http://www.dmtf.org

List of standards currently published by the DMTF
http://www.dmtf.org/standards/published_documents.php

WBEM HTTP Operations Online Course, Douglas Brown
http://www.dmtf.org/download/presentations/conf1999/w103.pdf

CIM Events, Use and Implementation, Denise Eckstein
http://www.dmtf.org/download/presentations/devcon02/DeniseEckstein-CIMEventsUseandImplementation.pdf

The Hyper Text Transfer Protocol HTTP/V1.1 RFC 2616
http://www.ieft.org/rfc/rfc2616.txt

An HTTP Extension Framework RFC 2774
http://www.ietf.org/rfc/rfc2774.txt

Reasons for namespaces, MSDN
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk30/htm/xmconreasonsfornamespaces.asp

Hypertext Transfer Protocol HTTP/1.0 RFC 1945
http://www.ietf.org/rfc/rfc1945.txt

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

Tags for the Identification of Languages IETF Standards Track RFC 1766
http://www.ietf.org/rfc/rfc1766.txt

XML Schema Part 1: Structures W3C Working Draft 6th
http://www.w3.org/TR/xmlschema-1/

Internetworking With TCP/IP Volume 1: Principles Protocols, and Architecture, 4th edition, 2000. ISBN 0-13-01830-6, Douglas E.Comer

Internetworking With TCP/IP Volume II: Design, Implementation, and Internals (with D. Stevens), Third ed, 1999. ISBN 0-13-973843-6, Douglas E.Comer