OpenWBEM Programmer's Guide

J. Bart Whiteley

December 12, 2002

Contents

1 CIM Client API

The OpenWBEM Client API can be used to create CIM Client applications. A CIM Client application can come in many forms. It could be a management console, or a controlling agent managing multiple CIMOMs. A CIM Client developed with the OpenWBEM CIM Client API can communicate with any CIMOM, not just the OpenWBEM CIMOM.

1.1 Selecting a Protocol

The first step is to define the means of communication between the client and the CIMOM. Currently the only protocol supported is HTTP. An OW_CIMProtocolIFCRef

must be created as this will be passed to the encoding mechanism. This can be done in the following manner:
`OW_CIMProtocolIFCRef httpClient(new OW_HTTPClient(url));`. The constructor of OW_HTTPClient takes a single argument: an OW_String that is the URL to the CIMOM.

A URL takes the form
`[protocol://][username:password@]<hostname>[:port][path]`. The protocol can be http, https, or ipc (Unix Domain Socket). If the protocol is not given, http is assumed. Username and password are optional. If not provided, and authentication is required, the username and password can be obtained through a callback which will be described shortly. The hostname is the name or IP address of the computer where the CIMOM is running. If the port is not given, 5988 is assumed if the protocol is http, and 5989 is assumed if the protocol is https. The path specifies the location of the CIM Server. This is normally /cimom. The OpenWBEM cimom ignores the path, but it may matter to some others.

At this point a callback can be provided to obtain a username and password if the CIM Server requests authentication. A callback for username and password is created by deriving from OW_ClientAuthCBIFC. Suppose the new derived class is called GetLoginInfo. The callback would be provided to the HTTP Client in the following manner:

```
OW_CIMClientAuthCBIFCRef getLoginInfo(new GetLoginInfo);
httpClient->setLoginCallBack(getLoginInfo);
```

1.2 Choosing the Encoding

Now that the protocol is defined, an encoding scheme is selected. Two types of encoding are currently available: CIM-XML (also known as WBEM), and Binary. CIM-XML is supported by all CIMOMs, and Binary is specific to OpenWBEM. A remote CIMOM "handle" can be obtained in the following manner:

```
OW_CIMOMHandleIFCRef handle;
handle = new OW_CIMXMLCIMOMHandle(httpClient);
```

or `handle = new OW_BinaryCIMOMHandle(httpClient);`

1.3 Using the Remote CIMOM Handle

Now the handle can be used to perform CIM operations against the CIMOM. refer to the API documentation for the OW_CIMOMHandleIFC class to see what functions are available on a handle.

2 Adding a new Provider Interface

3 Writing a C++ Provider

Providers are used to provide dynamic (not stored in the CIMOM's internal datastore) data.

3.1 Derive from an abstract base class

To create a C++ provider, simply derive a class from one or more of the abstract base classes for C++ providers. The abstract base classes are: OW_CppAssociatorProviderIFC, OW_CppIndicationExportProviderIFC, OW_CppInstanceProviderIFC, OW_CppMethodProviderIFC, OW_CppPolledProviderIFC, and OW_CppPropertyProviderIFC.

3.1.1 Associator Providers

Associator Providers are used to provide dynamic associations.

3.1.2 Indication Export Providers

Indication Export Providers are used to send events. When an indication is triggered within the CIMOM, it can be exported in multiple ways based on the Indication Export Providers that are loaded. One provider might convert the event to an SNMP trap, while another could send an email alert.

3.1.3 Instance Providers

As you might expect, Instance providers are used to provide dynamic instances.

3.1.4 Method Providers

Method Providers are used to execute "extrinsic" methods on the system.

3.1.5 Polled Providers

Polled Providers can be executed at regular intervals (specified by the polled provider) by the CIMOM, or they can start their own thread and run continuously.

3.1.6 Property Providers

Property Providers are used to handle dynamic properties.

3.2 Multiple Inheritance

It is possible for a C++ provider to be multiple types of providers. Simply derive from multiple base classes, and implement the appropriate virtual functions.

3.3 The C++ Provider Factory Function

Once the class is written, all that is left is to provide a factory function that the CIMOM uses to load the provider. A macro is provided to make this very easy. If your C++ class is called MyInstanceProvider, simply put this line in your file:
`OW_PROVIDERFACTORY(MyInstanceProvider, myinstprov);`
The first argument is the C++ class name. The second argument is the identifier that the provider will be known as in MOF.

3.4 Making a Shared Library

The C++ class representing the provider will need to be compiled into a shared library (.so). Then, just put the .so file in the directory specified by the cppprovifc.prov_location configuration item in openwbem.conf.