OpenWBEM Getting Started Guide

Author: Dan Nuffer

Last update: 12/09/04

Table of Contents

1. OpenWBEM Overview & Concepts

Genesis

OpenWBEM was started at Caldera in early 2001. It was intended to become part of the Volution Manager product. It was open sourced in the desire to help increase the compatiblity of management products, especially for Linux. It is now maintained by a variety of various companies and individuals. Dan Nuffer is the project lead.

Overview

OpenWBEM is a software suite that is a mature implementation of the DMTF CIM and WBEM standards. Components include a CIM Object Manager (CIMOM), a CIM Client API, a CIM Listener API, a WBEM Query Language (WQL) engine.

Features

See openwbem.org for a nice list.

2. Installation

Pre-Requisites

- OpenWBEM is written in C++, and has been developed on gcc. It also can be compiled by most current C++ compilers, include Microsoft Visual C++ 7.1, IBM's VisualAge for C++, and HP's aC++
- pthreads.
- GNU make
- One some OSs you may need GNU versions of standard Unix tools because the vendor provided tools are buggy or have hard-coded limits like 2048 line lengths or other such nonsense.

Optional Feature Pre-Requisites

- flex – Only if you obtained the source from CVS, or if you need to modify the mof or wql lexers.
- bison – Only if you obtained the source from CVS, or if you need to modify the mof or wql parsers.
- perl with embedding headers and libraries – If you want to use the perl provider interface.
- PAM libraries – If you want to use the PAM authentication module. These are part of glibc on Linux.
- PAM development headers – If you want to build the PAM authentication module.
- zlib – If you want to enable & use http compression.

- slp (e.g. OpenSLP) – If you want to use the slp provider to advertise or the client api to discover slp cimom advertisements.
- openssl – If you want to use https. Highly recommended.

The Installation

First download and extract the OpenWBEM source code. Then run the configure script: `./configure` There are quite a few options you can pass to the configure script that are used to enable/disable certain optional features, or to tell it where to find headers and libraries if they aren't in the standard locations. Use the --`help` flag to get a list of the options. If you are building a copy to do development work, it is highly recommended you use the `–enable-debug-mode` option. After configure is finished, you can run `make` to build. Verify that everything is working, and run `make check`. Run `make install` to install OpenWBEM on your system. Having it installed is not required, but will make it easier to use since the binaries and libraries will be in the `PATH`.

There is also an RPM spec file available. You can build the OpenWBEM RPMs by executing: `rpmbuild -ta openwbem-<version>.tar.gz`. Or if you have already extracted the tarball and run configure, run `make rpm`, and the output will be placed under the `rpmbuild` directory. This will generate an openwbem and openwbem-devel rpm. You can then install them by running `rpm -i <rpm filename>`. The RPM provides some additional files that aren't installed otherwise on Linux. These include the init script (`/etc/init.d/owcimomd`), the PAM configuration file (`/etc/pam.d/openwbem`), and some miscellaneous documentation.

3. Configuration

The OpenWBEM cimom (owcimomd) can be passed 3 different command line arguments.

- -h prints the help.
- -d tells owcimomd to run in debug mode. This causes it to **not** fork into the background and detach from the terminal (daemonizing). It will also print all log messages to the console in addition to whatever logger is configured.
- -c <config filename> specifies which config file to use. By default owcimomd will use the config file in `${sysconfdir}/openwbem/openwbem.conf` . `${sysconfdir}` defaults to `/usr/local/etc`, but can be changed by telling the configure script to use a different prefix or sysconfdir (e.g. `--sysconfdir=/etc`). This option allows the user to override the default and manually specify the config file to use.

The owcimomd config file contains many options which can be used to modify it's behavior. Most are set to reasonable defaults. You probably won't need to change any of the options that refer to directories. Some you may wish to configure are those related to authorization, logging, or optional features. The config file contains explanations for all

available options. The OpenWBEM philosophy is that all configuration is stored in one centralized location: the config file. This is why there are only 3 command line options, and no environment variables affect the behavior of owcimomd.

Configuring authentication

Currently there are 3 supported HTTP authentication methods. Basic, Digest and OWLocal.

If Basic is used, then you may choose to use an authentication module. Available modules include: PAM, PAM command line, Simple, AIX authenticator, and non authenticating. OpenWBEM also supports custom authentication modules.

The `owcimomd.allowed_users` config item is the simplest form of access control, and is applied after the authentication module has authenticated a user. It is a space delimited list of the users who are allowed to access the cimom. To allow all users, use `*`.

PAM

This method uses the PAM (Pluggable Authentication Modules) system api. This means that the cimom will use the system's authentication. The client will have to use the same username and password they would use to log in to the system. To use this:

- Set the `owcimomd.authentication_module` config item to point to the `libpamauthentication.so` library file.
- The `pam.allowed_users` config item to a space delimited list of the users who are allowed to access the cimom. To allow all users, put in `*`. This option is deprecated in favor of the more general `owcimomd.allowed_users` option.
- On Linux, install the openwbem PAM config file. Copy `etc/pam.d/openwbem` to `/etc/pam.d/`

To build the PAM authentication module, you must have the PAM development headers installed. They are not installed by default in many versions of Linux. If you find that the `libpamauthentication.so` file was not built, this is because the OpenWBEM configure script will automatically detect if the PAM headers are there or not, and only build the PAM authentication module if they are installed. If you find this happened to you, and you want to remedy the situation, install the headers (commonly in an RPM called pam-devel), then remove the `config.cache` file generated by the configure script, and then re-run configure and rebuild.

When using PAM, the HTTP Basic authentication scheme is used, meaning the password is sent in an insecure fashion over the wire. To prevent sniffers on the Internet from obtaining your credentials you must use SSL together with the PAM authentication module. This may not be a concern in a trusted environment.

PAM Command Line

This method is the same as PAM. The difference is that an external binary (OW_PAMAuth) is called to perform the authentication. The reasons to use this instead

of PAM is if the system's pam APIs are not thread safe or leak memory. Some older versions of Linux (from 2001 and earlier) are known to leak memory. To use this:

- Set the `owcimomd.authentication_module` config item to point to the `libpamauthenticationcl.so` library file.
- The other considerations for PAM also apply.

Simple

The simple authentication module is backed by a file where each line contains a user name and password separated by a colon.

An example file is like this:

```
username1:password1
username2:password2
```

To use this:

- Set the `owcimomd.authentication_module` config item to point to the `libsimpleauthentication.so` library file.
- Set the `simple_auth.password_file` config item to point to the password file you have created.

When using the simple authentication method, the HTTP Basic authentication scheme is used, meaning the password is sent in an insecure fashion over the wire. To prevent sniffers on the Internet from obtaining your credentials you must use SSL together with the simple authentication module. This may not be a concern in a trusted environment.

AIX

The AIX authentication module uses the AIX authenticate(), which is a precursor to PAM. The system's user database will be used. Newer versions of AIX have PAM by default, so you may want to use PAM if possible.

To use this:

- Set the `owcimomd.authentication_module` config item to point to the `libaixauthentication.so` library file.

When using the AIX authentication method, the HTTP Basic authentication scheme is used, meaning the password is sent in an insecure fashion over the wire. To prevent sniffers on the Internet from obtaining your credentials you must use SSL together with the simple authentication module. This may not be a concern in a trusted environment.

Non-Authenticationg

The non-authenticating module authenticates all users. You may wonder what benefit it

has over just setting `owcimomd.allow_anonymous=true`? The difference is that the client is required to provide credentials, and the username and password are stored in the `OperationContext`. This isn't very secure, but if you want to allow anybody to authenticate, but actually need the principal and credential for logging purposes, this may suit your purpose.

To use this:

- Set the `owcimomd.authentication_module` config item to point to the `libnonauthenticatingauthentication.so` library file.

When using the non-authenticating authentication method, the HTTP Basic authentication scheme is used, meaning the password is sent in an insecure fashion over the wire. To prevent sniffers on the Internet from obtaining your credentials you must use SSL together with the simple authentication module. This may not be a concern in a trusted environment.

Custom

Create a class that implements the AuthenticatorIFC interface and compile it into a shared library. Point the config file at it and you're done.

When using a custom authentication method, the HTTP Basic authentication scheme is used, meaning the password is sent in an insecure fashion over the wire. To prevent sniffers on the Internet from obtaining your credentials you must use SSL together with the simple authentication module. This may not be a concern in a trusted environment.

Digest

The digest authentication mechanism is built into the http server, and bypasses the pluggable owcimomd authentication module. To turn on digest authentication, set the `http_server.use_digest` config item to true. Digest uses cryptographic hashing and other mechanisms to prevent discovery of passwords. Because of this, it cannot integrate with system passwords. The digest authentication module requires you create a password file using the `owdigestgenpass` utility. Here is an example of how to use it:

```
owdigestgenpass -l user1 -f /the/password/file
```

Enter the password when prompted. If you aren't running it on the same computer as the cimom, you can also use the -h flag to specify the cimom computer's hostname. To inform owcimomd of the digest password file, set the `http_server.digest_password_file` config item.

Digest authentication protects the password over an unencrypted (non-SSL) connection. Attackers will be unable to obtain the password. Digest also prevents attackers from doing a replay attack. However, the data stream will not be encrypted.

OWLocal

This is an HTTP authentication mechanism that is designed to be used by a client and

server which are on the same system. The mechanism is specified in doc/local_authentication.txt. This allows clients to authenticate without providing a password, and owcimomd trusts the hosting operating system has properly authenticated the user.

To use this:

- Set the config item `http_server.allow_local_authentication = true`

SSL

OpenWBEM will build in SSL client and server support if the configure scripts finds the openssl development headers and libraries. The `http_server.https_port` config item specifies which port the https server will listen on. If set to -1, https will be disabled. If not specified, the default is port 5989 which is the IANA assigned port for CIM-XML over https.

To use SSL you need to setup a SSL host key and certificate. If you are just testing, or doing development, you can use the test file the comes with OpenWBEM: `test/acceptance/testfiles/hostkey+cert.pem`

Otherwise you can generate your own using a SSL key and certificate tool such as openssl, or even get one signed by a your local CA or a recognized CA (e.g. Verisign). The file has to contain both the server key and certificate in pem format.

You may want to use the provided `owgencert` script to create a self signed certificate and key.

Make sure the config item `http_server.SSL_cert` points to the file.

4. Execution

During development I almost always run owcimomd in debug mode (use the -d command line argument). In production, it should be run as a normal system daemon. There is a Linux init script available (`etc/init/owcimomd`) which works on most Linux distributions. The init script assumes openwbem has been `./configure`'d with the following arguments: `--prefix=/usr --sysconfdir=/etc --localstatedir=/var/lib`

Create Namespace(s)

owcimomd always has a namespace named `root`. OpenWBEM supports a hierarchical view of namespaces, using the `__Namespace` class. The DMTF has deprecated `__Namespace` in lieu of `CIM_Namespace`. OpenWBEM supports `CIM_Namespace` if the class has been created, and it provides a flat view of namespaces. Enumerating instances of the class will return an instance for each namespace in the cimom. The CIM Operations over HTTP spec recommends that

CIM_Namespace always be available in the root namespace. It is common practice to use root/cimv2 as the namespace to hold version 2 of the CIM schema. You can create namespaces using any CIM client such as the SNIA browser. OpenWBEM provides a utility named owcreatenamespace which can create a namespace. Here is an example of how to create the root/cimv2 namespace on the cimom running on the same machine:

```
owcreatenamespace -u http://localhost/ -n root/cimv2
```

Import Schema(s)

You use owmofc to import mof classes and instances into the cimom.

It is recommended that the CIM_Interop schema be present in the root namespace. Run the following commands to accomplish this:

```
owmofc -u http://localhost/cimom -n root CIM_Core28.mof
owmofc -u http://localhost/cimom -n root CIM_Event28.mof
owmofc -u http://localhost/cimom -n root
Physical28_Package.mof
owmofc -u http://localhost/cimom -n root
System28_SystemElements.mof
owmofc -u http://localhost/cimom -n root CIM_Interop28.mof
owmofc -u http://localhost/cimom -n root
OpenWBEM_Interop.mof
```

As of this writing the current version of the schema is 2.8. Substitute the version of the schema that you wish to use in the above commands.

For all versions of OpenWBEM:

You will have to import the schema into the namespace you wish to use (root/cimv2 is recommended):

```
owmofc -u http://localhost/cimom -n root/cimv2
CIM_Schema28.mof
```

Substitute the actual schema file you are using. OpenWBEM 3.1.x comes with the CIM 2.8 schema in the schemas/cim28 directory. Other versions can be downloaded from the DMTF at http://dmtf.org/standards/standard_cim.php.

If you wish to use ACLs, create the root/security namespace and import OpenWBEM_Acl1.0.mof. Refer to the ACL.HOWTO file if you want more information about ACLs.

Install Providers

Automatic Provider Registration

When OpenWBEM starts up, it will scan and load all provider interfaces. Each provider interface is given the opportunity to automatically register any class/provider combos that its responsible for with the provider manager. Thus this behavior is dependent on the provider interface. The C++ provider interface will scan all providers at start-up. It is currently not possible to dynamically register a provider that uses this registration scheme. The cimom has to be restarted when one is added or removed. It is not necessary to create any special classes or instances to inform the cimom of the provider, since it will be found when the cimom starts up.

Provider Qualifier

owcimomd has support for the provider qualifier and will continue to support the provider qualifier as long as people still use it.

To use the provider qualifier, you should first find the class in the CIM schema that most closely resembles what you want to model. Then you create a subclass for your object. You attach a provider qualifier to the class that identifies your provider. For instance:

```
[provider("c++::acme_foo")]
class ACME_Foo : CIM_Foo
{
};
```

tells the cimom to look for `libacme_foo.so` when it needs to query the provider for the `ACME_Foo` class. The value of the qualifier `"c++::acme_foo"` identifies both the provider type and the provider library name. `c++` is the type, and `acme_foo` is the name. Other provider interfaces have a different type. To install the provider, simply copy the provider library to the appropriate directory. The config item `cppprovifc.prov_location` specifies the directory. It defaults to `/usr/local/lib/openwbem/c++providers`. Then import your mof into the cimom and your provider will be available. It is **not** necessary to restart the cimom.

Using CIM Clients

OpenWBEM currently has utilities to:

- compile mof (owmofc)
- generate a http digest password file (owdigestgenpass)
- listen for indications (owcimindicationlistener)
- create a namespace (owcreatenamespace)
- delete a namespace (owdeletenamespace)
- enumerate classes (owenumclasses)

- enumerate class names (owenumclassnames)
- enumerate namespaces (owenumnamespace)
- enumerate qualifiers (enumqualifiers)
- execute wql queries (owexecwql)

owmofc and owdigestgenpass have man pages. For usage information on any command, use the --help option.

5. Client API

Overview

OpenWBEM has all the necessary code to facilitate writing a CIM client. All the details of CIM/XML are abstracted. There are also utilities for sockets, threading, etc. See the Doxygen generated API docs if you need more information.

Building/Compiling/Linking

The client API is broken up into various shared libraries to allow developers to be able to pick and choose what subset of functionality they need.

- libopenwbem.so – Common code as well as all CIM meta model classes.
- libowclient.so – CIM client functionality.
- libowxml.so – XML functionality used by the CIM/XML protocol code.
- libowhttpcommon.so – HTTP functionality that is common between client and server.
- libowhttpclient.so – HTTP client code.
- libowhttpxmllistener.so – The CIM/XML Listener.
- libowservicehttp.so – The HTTP server. Used by the CIM/XML Listener.

6. Building Providers

Overview

Instance

Instance providers are responsible for handling the following intrinsic methods for a certain class:

- EnumerateInstanceNames
- EnumerateInstances - The default implementation calls enumInstanceNames() and then getInstance() for each name.
- GetInstance
- CreateInstance

- ModifyInstance
- DeleteInstance

Typically, each instance (uniquely identified by it's keys) represents one object to be managed.

If your provider is read-only, you can derive from CppReadOnlyInstanceProvider and you don't need to implement create, modify or delete. Also you can derive from CppSimpleInstanceProvider and implement doSimpleEnumInstances(). enumInstances(), getInstance() and enumInstanceNames() are all implemented by the base class.

Secondary Instance

Secondary Instance providers aren't responsible to exposing instances like instance providers are, but they are given the opportunity to filter instances returned by enumInstances() or getInstance(). They also get notified on createInstance(), modifyInstance() and deleteInstance().

Associator

Associator providers are responsible for handling the following intrinsic methods for a certain association class:

- Associators
- AssociatorNames
- References
- ReferenceNames

With OpenWBEM, an dynamic association can be implemented with just an instance provider, but it can be much more efficient to implement the associator provider interface. Associator providers should also be instance providers so that the instance functions will work for the association class.

For a simpler (albeit less powerful) provider, you can derive from CppSimpleAssociatorProvider, which only requires the provider to implement one function, and the other 4 standard functions are implemented by the base class.

Method

Method providers are responsible for handling any extrinsic methods defined in a class. It is common for an instance provider to also be a method provider.

Indication

Indication providers are responsible for generating indications. There are two categories of indications, Life-cycle and alert. See the DMTF indication whitepaper for more info. Life-cycle indications are tied to a specific class, and so will usually be implemented as an instance/indication provider. OpenWBEM supports three models of implementing

indication providers:

- Instance Life-cycle: polled by the cimom – This is the easiest to implement, but also is more cpu and memory intensive than the other alternatives. The cimom keeps a cached copy of all the instances of the class and every polling cycle will get the current set. The differences in the sets between the cycles drive the generation of indications.
- Separate thread – This is good for the case when a separate thread can block on an external event (such as a socket or kernel semaphore, etc.) and raise an indication whenever something happens.
- Polled provider – This is good for the case where the provider has to poll for data (e.g. Check the free disk space every 30 seconds)

See the indication provider examples for more detail on these 3 approaches.

Provider Development Process

Extend Schema

The first step is to find the appropriate class in the standard CIM schema that represents the element you want to model. Next you create a subclass for your object. You can add properties and methods that apply. Creating a subclass is done with mof (or you can use the new rose plug-in available from the sblim project and generate the mof). The class may be dependent on others via associations, so you may have multiple iterations of classes that need to be instrumented. If you implement non-association classes, you also usually need to implement associations for them as well.

Use CodeGen

Decide how much functionality your provider will need. Is a read-only instance provider good enough? If you have methods you need to implement, you will need to be an instance/method provider. If your class needs to be monitored for changes, you need to implement the indication provider provider interface to send indications. Once you know what provider interfaces you'll need to implement, you can select the appropriate CodeGen template, and then generate stubs for your provider. See the CodeGen README for more information on how to use it. Unfortunately, as of this writing, the templates for CodeGen are out of date and incomplete. If you write some, please consider contributing it for others to use.

Finish writing provider

Now you get to do the real work, interface with whatever you're modeling and convert it into CIM objects. Link your provider into a shared library and you're set.