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. The Volution Manager technology and staff were transferred to Center 7, Inc. in November of 2002. Center 7 provides commercial support, training, and consulting for OpenWBEM.

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.

This document assumes that you have a working knowledge of CIM and WBEM.

Features

See openwbem.org for a nice list.

2. Compilation & Installation

Pre-Requisites

- OpenWBEM is written in c++, and has been developed on gcc. It may work with other compilers, but this hasn't been attempted.
- pthreads. pth can also be used on Linux, and is required for OpenServer, because it doesn't have a functional pthreads library.

Optional Feature Pre-Requisites

- flex – If you need to modify the mof or wql lexers.
- bison – 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. 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 for Linux) – 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.

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. After configure is finished, you can run `make` to build. If you wish to verify that everything is working, you can 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 OpenWBEM by executing: `rpm -ba openwbem.spec`. This will generate an openwbem and openwbem-devel rpm. You can then install them by running `rpm -i <rpm filename>`.

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 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.

Configuring authentication

Currently there are 3 authentication methods. PAM, Simple, and Digest. OpenWBEM also supports custom authentication modules.

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 use this:

- Set the `owcimomd.authentication_module` config item to point to the `libpamauthentication.so` library file.
- Set the `pam.allowed_users` config item to a space delimited list of the users who are allowed to access the cimom.
- Install the openwbem PAM config file. This isn't necessary if you installed the RPM. 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 they are there or not, and only build the PAM authentication module if they are installed.

When using PAM, the http simple authentication headers are 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. The RPM configuration defaults to using PAM.

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 simple authentication headers are 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 normal openwbem 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 openwbem of the digest password file, set the `http_server.digest_password_file` config item.

Digest authentication is safe to use over an unencrypted (non-SSL) connection. Attackers will be unable to obtain the password or do a replay attack.

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 recognized certificate authority (e.g. Verisign). The file has to contain both the server key and certificate in pem format.

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`. The next version of OpenWBEM will support `CIM_Namespace`. 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 http://localhost/cimom root/cimv2
```

Import Schema(s)

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

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

```
owmofc http://localhost/cimom root/cimv2 CIM_Schema27.mof
```

Substitute the actual schema file you are using. OpenWBEM 2.0.x comes with the CIM 2.5 schema in the schemas/cim25 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

Provider Qualifier

The provider qualifier is the only method of provider registration with OpenWBEM 2.0. It has been deprecated by the DMTF and will be eventually replaced with the Interop schema provider registration classes. OpenWBEM 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++::foo")]
class foo
{
};
```

tells the cimom to look for libfoo.so when it needs to query the provider for the foo class.

The value of the qualifier "`c++::foo`" identifies both the provider type and the provider library name. `c++` is the type, and `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)
- 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)

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
- GetInstance
- CreateInstance
- ModifyInstance
- DeleteInstance

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

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.

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.

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 Rational 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.

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.