Using the WMI Adapter for CIMPLE

Michael E. Brasher, Karl Schopmeyer

November 4, 2008

Version 1.1, 15 December 2008

Table of Contents

1 Introduction

This document explains how to use the WMI adapter for CIMPLE. The WMI Adapter enables CIMPLE providers to work with the Windows Management Instrumentation (WMI) server. The following chapters explain how to:

- install CIMPLE for WMI
- develop a trivial provider
- register the WMI provider
- verify the provider

Except for a few minor details, provider developement for WMI is similar to developing providers for other servers. After reading this guide, you will be able to make existing CIMPLE providers work with WMI.

2 Installing CIMPLE for WMI

The WMI adpater compiles today with a variety of Microsoft compilers including

VC2003 with the service pack VC2005 with the service pack VC2008

It has not been tested with VC 6. Note that the requirement for the platform or windows SDK varies by compiler version. In general the goal is to not require the SDK but the current versions of the wmi adapter depend on the SDK for a few functions.

To build CIMPLE and the WMI adapter, open a Windows terminal session (`cmd.exe`). Run the Visual Studio setup scripts. Verify that the Visual Studio C++ compiler is on your path (`cl.exe`). Obtain GNU make (`make.exe`) and add it to your path. GNU make for Windows is available from the "tools" link on the OpenPegasus home page (see `http://openpegasus.org`). Note that the make provided by Micrososft is not compatible with the cimple make files. You must use GNU make.

Unpack the CIMPLE distribution. From the root of the distribution type.

```
C:\> configure.bat --bindir=c:/windows/system32 --enable-wmi
```

Next build CIMPLE by typing:

```
C:\> make
```

Finally, install CIMPLE as follows:

```
C:\> make install
```

This installs the programs and DLLs under:

```
C:\windows\system32
```

All other CIMPLE files are installed under:

```
C:\cimple
```

You may uninstall CIMPLE later by typing:

```
C:\> make uninstall
```

3 Building a trivial provider

This chapter explains how to build a trivial provider. For the most part, it is like building a CIMPLE provider for other servers, but there are a few minor differences.

We start with the following MOF definition (which we place in `repository.mof`).This example is part of the cimple distribution in in the directory `cimple/src/wmi/person`.

```
[dynamic, provider("Person")]
class Person
{
    [Key] string SSN;
    [Key] string FirstName;
    [Key] string LastName;

    [implemented]
    uint32 foo([in] string arg);
};
```

The `dynamic` and `implemented` qualifiers are unique to WMI. They are not part of the currently defined qualifier set in the DMTF specifications. Therefore, we may be required to add them to the qualifier definitions in qualifiers.mof in the CIM mof schema to get the class definition to compile with genclass. They have been inserted in the schemas provided. The definitions that must be inserted are:

```
qualifiers.mof:Qualifier Dynamic : boolean = false,
qualifiers.mof:Qualifier Implemented : boolean = false,
```

The `provider` qualifier is also required for wmi provider registration but it should be part of the existing DMTF qualifier definitions either in qualifiers.mof or optionalqualifiers.mof. The value for this provider MUST match the name of the provider DLL (without the extension).

Next we use the `genproj` command to generate the classes, provider, and module.

```
C:\> genproj Person Person
Created Person.h
Created Person.cpp
created repository.h
Created repository.cpp
Created Person_Provider.h
Created Person_Provider.cpp
Created module.cpp
Created guid.h
Created register.mof
```

This generates the following files:

- `Person.h` - the Person class declaration
- `Person.cpp` - the Person class definition
- `repository.h` - the class repository declarations
- `repository.cpp` - the class repository definitions
- `Person_Provider.h` - the Person provider declaration

- `Person_Provider.cpp` - the Person provider methods
- `module.cpp` - the WMI entry points.
- `guid.h` - the GUID that uniquely identifies the provider COM server.
- `register.mof` - the WMI registration instances.

Next we must create a `link.def` file shown below.

```
LIBRARY "Person.dll"

EXPORTS
        DllMain PRIVATE
        DllCanUnloadNow PRIVATE
        DllGetClassObject PRIVATE
        DllRegisterServer PRIVATE
        DllUnregisterServer PRIVATE
```

Then we create the following Makefile.

```
## TOP defines location of the cimple mak directory
TOP=../../..
include $(TOP)/mak/config.mak

LIBRARY = Person
SOURCES = Person.cpp Person_Provider.cpp module.cpp repository.cpp
LIBRARIES = cimplewmiadap cimple
EXTRA_LINK_FLAGS = /def:link.def
EXTRA_SYS_LIBS = ole32.lib oleaut32.lib
DEFINES += -DCIMPLE_WMI_MODULE

include $(TOP)/mak/rules.mak
```

Finally, we build the provider as shown below.

```
C:\> make
```

This creates a DLL called `Person.dll`.

4 Registering a WMI provider

This chapter shows how to register a WMI provider using the Microsoft tools. First we must copy the provider DLL to the WMI providers directory, Usually located here:

```
C:\windows\system32\wbem\
```

Second we use the WMI MOF compiler to add our classes to the CIM repository as shown below.

```
mofcomp repository.mof
```

Third we register our provider as follows.

```
mofcomp register.mof
```

Finally, we register our WMI provider as a COM server:

```
     regsvr32 /s C:\windows\system32\wbem\Person.dll
```

In the examples provided with cimple, these operations are integrated into the Makefile provided as the target reg.

5 Verifying a WMI provider

There several tools available help verify the providers once it installed including:

- `cimbrowser.exe` - Part of a wmi toolset available from Micrososft under the name CIMTest. This is a complete graphic wmi cim browser.
- `wbemtest.exe` - Client program that executes wmi cim operations from a set of check boxes.

Either of these tools is helpful to verify the providers you write with CIMPLE. You may also use many other WMI client tools.

In any case, to verify this first provider you should confirm that the Person class was installed in the property namespace (normally Root/cimv2) and that the provider returns two instances of the Person class.

You can validate the response from the defined method as follows:

TBD

For a full discussion of these tools and more information on wmi, see the book "Developing WMI solutions: A Guide to Windows Management Instrumentation" by Craig Tunstall and Gwyn Cole.