CIMPLE 1.0.0 Release Notes

Michael E. Brasher

April 26, 2007

Table of Contents

1 Introduction

The CIMPLE project began over two years ago and has had 55 releases. This document introduces CIMPLE 1.0.0 and explains what has changed since the last public release (CIMPLE 0.99.56). If you are unfamiliar with the major changes introduced by CIMPLE 0.99.56, you might want to review them by clicking here: http://www.cimple.org/downloads.html.

Please read this document carefully if you are an existing CIMPLE user. There are a few major changes in CIMPLE that require some action.

Chapter 1 describes what is new in this release. Chapter 2 discusses bugs fixed by this release. Chapter 3 explains how to migrate providers developed with earlier versions.

2 What's New?

This chapter covers new capabilities introduced by CIMPLE 1.0.0.

2.1 New CIMPLE Users Guide

We recently prepared a new users guide for CIMPLE, which explains how to develop providers with CIMPLE. It also covers features introduced by CIMPLE 1.0.0. This guide is available in PDF format at: http://www.cimple.org/Using_CIMPLE.pdf.

2.2 Platform Support

CIMPLE 1.0.0 officially supports the following platforms.

- Linux-X86 32-bit, GNU C++
- Linux-X86 64-bit, GNU C++
- Linux-IA64 64-bit, GNU C++
- Linux-S390 32-bit, GNU C++
- Linux-S390 64-bit, GNU C++
- Linux-PPC 32-bit, GNU C++
- Linux-PPC 64-bit, GNU C++

2.3 Build Procedures

The CIMPLE build procedures now comply with the GNU Coding Standards (see http://www.gnu.org/prep/standards). For example, the following configures, builds, and installs CIMPLE.

```
$ ./configure [OPTIONS]
$ make
$ make install
```

The configure script supports the standard and custom options listed below.

```
--help
--host=HOST
--prefix=DIR
```

```
--bindir=DIR
--libdir=DIR
--includedir=DIR
--datadir=DIR
--with-pegasus=DIR
--with-pegasus-libdir=DIR
--with-pegasus-includes=DIR
--with-pegasus-env
--with-cmpi=DIR
--with-openwbem=DIR
--with-schema=DIR
--enable-debug
--enable-static
```

CIMPLE can be configured with support for any of the following.

- CMPI (use '`--with-cmpi=DIR`').
- OpenPegasus (use '`--with-pegasus=DIR`').
- OpenPegasus source distribution (use '`--with-pegasus-env`').
- OpenWBEM (use '`--with-openwbem=DIR`').

If you use CIMPLE with an OpenPegasus source distribution, then configure, build and install as follows.

```
./configure --with-pegasus-env
make
make install
```

This configures CIMPLE from the OpenPegasus environment variables (`PEGASUS_PLATFORM`, `PEGASUS_HOME`, `PEGASUS_ROOT`), builds, and then installs CIMPLE under PEGASUS_HOME. **Do not forget to install**.

2.4 New `genprov` Patch Option

The `genprov` tool now patches existing provider sources. That is, it updates the method signatures in the those source files. For example, if the following files exist (in the current directory):

```
CIM_ComputerSystem_Provider.h
CIM_ComputerSystem_Provider.cpp
```

then the following command patches them.

```
$ genprov CIM_ComputerSystem
Patched CIM_ComputerSystem_Provider.h
Patched CIM_ComputerSystem_Provider.cpp
```

This tool should be used in two situations.

- When a MOF class definition changes an extrinsic method.
- When CIMPLE changes a provider method signature (CIMPLE 1.0.0 changed the method signature of `modify_instance` as well as extrinsic method signatures with reference arguments).

We recommend running `genprov` to fix faulty method signatures generated by older versions of `genprov`. **Also after running genprov, you should remove the `proc` method, as it is no longer used.**

2.5 New `genmod` Tool

The new `genmod` tool automatically generates `module.cpp` for one or more providers. The following example generates a module for the `CIM_ComputerSystem` and `CIM_Fan` providers.

```
$ genmod CIM_ComputerSystem CIM_ComputerSystem CIM_Fan
Created module.cpp
```

The first argument is the name of the module. The remaining arguments are the names of classes, for which there are (or will be) providers.

`Genmod` generates entry points for all three provider interfaces, including CMPI, Open-Pegasus, and OpenWBEM. To enable one of these entry points, compile `module.cpp` with one of the following.

```
-DCIMPLE_CMPI_MODULE
-DCIMPLE_PEGASUS_MODULE
-DCIMPLE_OPENWBEM_MODULE
```

`Genmod` also generates the `proc` method, formerly located in the provider sources. **You must remove the `proc` method from the provider sources since it is no longer used. Also, note that running `genmod` is not optional.**

2.6 Multi-Provider Generation in `genprov`

The `genprov` tool now generates provider skeletons for more than one provider. For example, the following command generates provider skeletons for the `CIM_ComputerSystem` and `CIM_Fan` providers.

```
$ genprov CIM_ComputerSystem CIM_Fan
Created CIM_ComputerSystem_Provider.h
Created CIM_ComputerSystem_Provider.cpp
Created CIM_Fan.h
Created CIM_Fan.cpp
```

As mentioned above, `genprov` patches the provider sources if they already exist.

2.7 New `genproj` Tool

The new `genproj` tool (generate project) runs the following tools automatically.

```
genclass
genprov
genmod
```

`Genproj` takes the same arguments as `genmod`. The following command generates a complete set of sources for the for the `CIM_ComputerSystem` and `CIM_Fan` providers.

```
$ genproj CIM_ComputerSystem CIM_ComputerSystem CIM_Fan
==== genclass:
Created CIM_ManagedElement.h
Created CIM_ManagedElement.cpp
Created CIM_ManagedSystemElement.h
Created CIM_ManagedSystemElement.cpp
Created CIM_LogicalElement.h
Created CIM_LogicalElement.cpp
Created CIM_Job.h
Created CIM_Job.cpp
Created CIM_ConcreteJob.h
Created CIM_ConcreteJob.cpp
Created CIM_EnabledLogicalElement.h
Created CIM_EnabledLogicalElement.cpp
Created CIM_System.h
Created CIM_System.cpp
Created CIM_ComputerSystem.h
Created CIM_ComputerSystem.cpp
Created CIM_LogicalDevice.h
Created CIM_LogicalDevice.cpp
Created CIM_CoolingDevice.h
Created CIM_CoolingDevice.cpp
Created CIM_Fan.h
Created CIM_Fan.cpp
created repository.h
Created repository.cpp
==== genprov:
Patched CIM_ComputerSystem_Provider.h
Patched CIM_ComputerSystem_Provider.cpp
Created CIM_Fan_Provider.h
Created CIM_Fan_Provider.cpp
==== genmod:
Created module.cpp
```

Running `genproj` is not required. You can still run `genclass`, `genprov`, and `genmod` separately.

2.8 New `ciminvoke` Tool

The new `ciminvoke` tool is an OpenPegasus client that invokes extrinsic provider methods. Since `ciminvoke` is experimental, it is not linked into the build. To build it, first build CIMPLE, and then type these commands from the root of the CIMPLE distribution (‘`cimple-1.0.0`’).

```
$ cd src/pegasus/ciminvoke
$ make
$ make install
```

The usage follows.

```
ciminvoke [OPTIONS] object-path method-name [param=VALUE]...
```

For more on `ciminvoke`, see `cimple-1.0.0/src/pegasus/ciminvoke/readme.txt`.

2.9 New `cimlisten` Tool

The new `cimlisten` tool is an OpenPegasus client that subscribes to and listens for CIM indications. Since `cimlisten` is experimental, it is not linked into the build. To build it, first build CIMPLE, and then type these commands from the root of the CIMPLE distribution (‘`cimple-1.0.0`’).

```
$ cd src/pegasus/cimlisten
$ make
$ make install
```

The usage is shown below.

```
$ cimlisten [OPTIONS] QUERY
```

The following example, subscribes to and listens for indications of the class `CIM_Indication` on the default namespace (`root/cimv2`).

```
$ cimlisten
```

The command hangs indefinitely, printing indications to standard output as they are received. The following command subscribes to and listens for indications of the class `MyIndication`.

```
cimlisten "select * from MyIndication"
```

For more on `cimlisten`, type:

```
$ cimlisten -h
```

2.10 `Regmod` Is Now an OpenPegasus Client

The `regmod` tool now runs as an OpenPegasus client (rather than modifying the OpenPegasus repository directly). Accordingly, the OpenPegasus server must be started before running `regmod`.

2.11 New `Property::set` and `Property::clear` Methods

The `Property` struct, used in generated classes, now has `set` and `clear` methods to simplify setting and clearing of property values. Before it was necessary to clear the null flag when setting the value as shown in the following example:

```
inst->CreationClassName.value = "LinuxComputerSystem";
inst->CreationClassName.null = false;
```

You can now do this with a single call.

```
inst->CreationClassName.set("LinuxComputerSystem");
```

Conversely, it was necessary to clear the value when setting the null flag as shown in the following example.

```
inst->CreationClassName.value = "";
inst->CreationClassName.null = true;
```

You can now do this with a single call.

```
inst->CreationClassName.clear();
```

We recommend using the `set` and `clear` methods wherever possible to avoid mistakes. Forgetting to clear the null flag is a common mistake.

2.12 Provider Examples

There are now comprehensive provider examples located under:

```
cimple-1.0.0/src/cimple/provider/Employee
```

We suggest looking at these first if you are new to CIMPLE and of course reading the CIMPLE users guide (`http://www.cimple.org/Using_CIMPLE.pdf`).

2.13 New `Datetime::ascii` Method

`Datetime` has a new form of the `ascii` method. The following snippet uses the **old** form:

```
Datetime dt;
...
char buffer[Datetime::BUFFER_SIZE];
dt.ascii(buffer);
```

Whereas, the following using the **new** form.

```
Datetime dt;
...
String buffer = dt.ascii();
```

The former incurs no memory allocation but risks a buffer overrun if `buffer` is too small. The latter eliminates the risk of a buffer overrun but incurs a memory allocation. You can decide for yourself which is better.

2.14 New OpenWBEM Adapter

Bart Whitely and the OpenWBEM team (Novell Inc.) contributed a new adapter enabling CIMPLE providers to work under the native OpenWBEM C++ provider interface. The OpenWBEM adapter is experimental in this release.

2.15 Static Builds

By default CIMPLE installs at most three shared libraries:

- `libcimple.so` — the main CIMPLE library
- `libcimplecmpiadap.so` — the CMPI adapter
- `libcimplepegadap.so` — the OpenPegasus adapter
- `libcimpleowadap.so` — the OpenWBEM adapter

Alternatively, you can build these libraries as static by passing the '`--enable-static`' option to the `configure` script, which eliminates the need to deploy extra libraries.

2.16 MOF Compiler Accepts DMTF CIM Schemas

The CIMPLE MOF compiler (used by `genclass` and `genprov`) now compiles the DMTF CIM schemas as they come from the DMTF. Before two changes were needed:

- Creation of the `CIM_Schema.mof` file.
- Reversing the slashes in the include pragmas.

Now the MOF compiler accepts the schemas "as is".

2.17 MOF Compiler Include Relative

The MOF compiler now handles included files relative to the currently opened file. The following pragma first searches for '`abc.mof`' in the directory containing file that included it.

#pragma include("abc.mof");

2.18 The MOF Search Path

The `genclass` and `genmod` tools now embed the default directory containing the MOF schema files. The CIMPLE_MOF_PATH environment is no longer required, unless you want to override the default. This directory name is set during configuration to `PREFIX/share/cimple/schema/cim214`. You can override it with the '`--with-schema`' option as shown below.

```
./configure --with-schema=DIR
```

The CIMPLE_MOF_PATH overrides the use of this directory. To determine the actual directory name, see the '`WITH_SCHEMA_OPT`' variable in the `config.options` file (located in the root of the CIMPLE source distribution).

2.19 Cross-Namespace Association Support

CIMPLE now supports cross-namespace association providers. The `Instance` class—from which all generated classes derive—defines the following data member.

```
String __name_space;
```

When an association provider builds the references, it may optionally set the `__name_space` member. Take for example, the following MOF definitions.

```
class B
{
    [Key] uint32 Key;
};

[Association]
class A
{
   B REF Left;
   B REF Right;
};
```

The following snippet creates an association of type `A` in which `Left` and `Right` refer to instances in namespaces '`root/left`' and '`root/right`' respectively.

```
B* Left = B::create(true);
Left->__name_space = "root/left";
Left->Key.set(1001);

B* Right = B::create(true);
Left->__name_space = "root/right";
Right->Key.set(1002);
```

```
A* a = A::create(true);
a->Left = Left;
a->Right = Right;
```

If `__name_space` is empty, it defaults to the namespace of the originating request.

2.20 Removed `Meta_Class.crc` Member

The `Meta_Class.crc` member has been removed. **Accordingly, you should regenerate your classes with `genclass`**.

2.21 Upgraded to CIM 2.14 Schema

CIMPLE now includes the CIM 2.14 schema (and the CIM 2.13.1 schema as before). All other schemas have been removed to make the distribution smaller.

3 Bug Fixes

CIMPLE 1.0.0 only fixes two bugs, since the previous release closed most of the outstanding bugs.

3.1 Bug 1: Life-Cycle Indications for the CMPI Adapter

CIMPLE-CMPI Indication providers can now provide instances with embedded objects, making it possible to generate life-cycle indications. The OpenPegasus adapter supported embedded objects before.

3.2 Bug 36: `Modify_instance` Has No Property List

The `modify_instance` provider method now takes an additional parameter that indicates which properties to modify. Here is the old function prototype:

```
Modify_Instance_Status <CLASSNAME>_Provider::modify_instance(
    const <CLASSNAME>* instance);
```

The new prototype adds the `model` parameter as shown below.

```
Modify_Instance_Status <CLASSNAME>_Provider::modify_instance(
    const <CLASSNAME>* model,
    const <CLASSNAME>* instance);
```

The non-null properties of `model` must be modified.

Since this bug fix changes the provider interface, you must patch each provider with `genprov`. As mentioned above, `genprov` can now automatically patch your provider.

4 Migration Notes

This chapter explains how to migrate providers from an earlier version to CIMPLE 1.0.0. This process is automated and relatively easy.

4.1 Class Generation

Regenerate all classes with `genclass`. This is a simple matter of running `genclass` as shown below.

```
$ genclass -r CLASS-1 CLASS-2 ... CLASS-N
```

The class arguments are the names of classes for which there are providers. It is unnecessary to generate all related classes since `genclass` does this automatically.

4.2 Provider Patching

Patch your provider sources with the new `genprov` patching feature. Simply run `genprov` in the directory that contains your provider sources as follows.

```
$ genprov CLASS-1 CLASS-2 ... CLASS-N
```

4.3 Module Generation

Generate `module.cpp` with the new `genmod` utility.

```
$ genmod MODULE-NAME CLASS-1 CLASS-2 ... CLASS-N
```

4.4 Using `genproj`

As mentioned in chapter 1, the new `genproj` tool can run `genclass`, `genprov`, and `genmod` in a single step. The usage is shown below.

```
$ genproj MODULE-NAME CLASS-1 CLASS-2 ... CLASS-N
```

This is a shortcut to the three commands shown in the last three sections.