Open Management Infrastructure (OMI)

Getting Started

Microsoft Corporation
October 5, 2010

Table of Contents

1 Introduction

This manual explains how to get started with OMI. It is by no means a complete reference, but hopefully after reading it you will be able to:

- build from the source distribution
- install the distribution
- start and stop the server
- use the command-line client
- develop and test a simple provider
- develop a simple client application

1.1 What is OMI?

OMI is a software service that runs on managed nodes. It provides the manageability infrastructure for building distributed systems management applications based on DMTF management standards, including:

- CIM Infrastructure Specification (DSP0004)
- CIM Schema (`http://dmtf.org/standards/cim`)
- Generic Operations Specification (DSP0223)
- WS-Management Protocol (DSP0226, DSP0227, DSP0230)

These standards define:

- a **meta-model** that defines rules for forming classes, properties, methods, and instances.
- a **schema** that defines specific classes for various management domains (e.g., storage, networking, operating systems). The schema is expressed using the "Managed Object Format" (DSP0004).
- the **operations** that CIM clients may perform on CIM servers.
- the **protocols** enabling clients and servers to communicate (e.g., WS-Management, CIM-XML).

A software service that implements these standards is a **CIM Server**, also known as a **CIM Object Manager (CIMOM)**. For more information on these standards, visit the DMTF (Desktop Management Task Force) web site: `http://dmtf.org`.

Note: WBEM (Web-Based Enterprise Management) comprises several standards, including CIM (Common Information Model) and WS-Management. **WBEM** refers to the broader set of standards. For this reason, the server is named **OMI**.

1.2 What does a CIM Server do?

In general, a CIM server enables client applications to perform operations on managed resources, such as CPUs, disks, networks, and processes. Typical operations include:

- enumerating resource instances
- invoking a method on a resource

- subscribing to events on a resources

But CIM servers do not perform these operations on resources directly. Instead servers furnish developers with a framework for building pluggable modules called **providers**. Providers are modules that interact directly with one or more resources. For example, a "process provider" interacts with operating system processes. Providers are packaged as shared libraries with a main entry point (used by the server to initialize the provider).

1.3 Operations

OMI enables clients to perform the following operations on providers.

- **GetInstance** - gets a single instance from the server.
- **EnumerateInstances** - enumerates instances of a given CIM class.
- **CreateInstance** - creates an instance of a CIM class.
- **DeleteInstance** - deletes an instance.
- **ModifyInstance** - modifies the properties of an instance.
- **Associators** - finds instances associated with a given instance.
- **References** - finds references that refer to a given instance.
- **Invoke** - invokes a method on a given instance or class.

OMI clients initiate these operations through these protocols:

- The WS-Management protocol
- The local Binary protocol.
- The CIM-XML protocol (not supported yet).

The server accepts client requests and routes them to the appropriate provider. Provider responses are routed back to the requesting client.

1.4 License

Eventually OMI may be licensed as open-source. Until then, OMI is an internal Microsoft project and Microsoft reserves all rights.

1.5 Supported Platforms

OMI supports the following platforms.

- HP-UX 11i v2 and v3 (PA-RISC and IA64)
- Sun Solaris 8 and 9 (SPARC) and Solaris 10 (SPARC and x86)
- Red Hat Enterprise Linux 4 (x86/x64) and 5 (x86/x64) Server
- Novell SUSE Linux Enterprise Server 9 (x86) and 10 SP1 (x86/x64)
- IBM AIX v5.3 and v6.1 (POWER)
- MacOS 10.5 (Intel)

OMI also builds on Windows with a few functional limitations.

1.6 Server Footprint

OMI was expressly designed to work on very small systems. Conventional CIM servers are too large for embedded and mobile operating systems, but OMI will fit easily on these systems. The object size is under 200 kilobytes and the memory consumption is low.

2 Building and Installing

This chapter explains how to build and install OMI. It assumes you have obtained the OMI source distribution (‘`omi-1.0.0.tar`’).

2.1 Prerequisites

OMI depends on the following software. Be sure these are installed on your system before building.

- GNU make
- Native C and C++ compiler
- OpenSSL headers and libraries
- PAM headers and libraries

2.2 Overview

The following commands build and install OMI (these steps are expounded in the sections below).

```
# tar xf omi-1.0.0.tar
# cd omi-1.0.0
# ./configure
# make
# make install
```

The ‘`make install`’ command installs all files under the ‘`/opt/omi-1.0.0`’ directory.

2.3 Unpacking the source distribution

The OMI source distribution is a ‘`tar`’ file. Unpack the distribution with the ‘`tar`’ utility as follows:

```
# tar xf omi-1.0.0.tar
```

This command creates a directory named ‘`omi-1.0.0`’, which contains the source distribution.

2.4 Configuring the build

To configure the build, run the ‘`./configure`’ script from the root of the source distribution. Type the following to print a help message explaining how to use the script.

```
# ./configure --help
```

The options allow you to change where components are installed. For example, to install the programs under ‘`/usr/local/bin`’ and everything else under ‘`/opt/omi`’, configure as follows.

```
# ./configure --prefix=/opt/omi --bindir=/usr/local/bin
```

The default ‘`prefix`’ is ‘`/usr/omi-1.0.0`’. After installing, you will find all OMI programs (with a ‘`omi`’ prefix) under ‘`/usr/local/bin`’.

2.5 Building the distribution

After configuring, build by typing ‘`make`’, where ‘`make`’ refers to GNU make. For example:

```
# make
```

This builds all components.

2.6 Installing the distribution

After building the source distribution, install by typing:

```
# make install
```

You may configure and build as any user. But you must install as root since the install script creates files under root-owned directories. Even if the ‘`--prefix`’ option specifies a non-root owned directory, the PAM authentication file (‘`omi.pam`’) must be copied to a root-owned directory.

2.7 Installation layout

After installing, you will find the installed files in the locations specified by the ‘`./configure`’ options. For example, if you configured with ‘`./configure --prefix=/opt/omi`’, you will find the following files after installing.

```
/opt/omi/bin/omicli
/opt/omi/bin/omigen
/opt/omi/bin/omireg
/opt/omi/bin/omiserver
/opt/omi/bin/omiagent
/opt/omi/etc/ssl/certs/omi.pem
/opt/omi/etc/ssl/certs/omikey.pem
/opt/omi/etc/omicli.conf
/opt/omi/etc/omiregister/root#omi/omiidentify.reg
/opt/omi/etc/omigen.conf
/opt/omi/etc/omiserver.conf
/opt/omi/lib/libmicxx.so
/opt/omi/lib/libomiclient.so
/opt/omi/lib/libomiidentify.so
/opt/omi/share/omischema/CIM_Schema.mof
...
/opt/omi/share/omi.mak
/opt/omi/include/MI.h
/opt/omi/include/omiclient/handler.h
/opt/omi/include/omiclient/linkage.h
/opt/omi/include/omiclient/client.h
/opt/omi/include/micxx/atomic.h
/opt/omi/include/micxx/propertyset.h
/opt/omi/include/micxx/dinstance.h
/opt/omi/include/micxx/instance.h
/opt/omi/include/micxx/field.h
/opt/omi/include/micxx/context.h
```

```
/opt/omi/include/micxx/datetime.h
/opt/omi/include/micxx/micxx.h
/opt/omi/include/micxx/types.h
/opt/omi/include/micxx/arraytraits.h
/opt/omi/include/micxx/array.h
/opt/omi/include/micxx/linkage.h
/opt/omi/include/micxx/string.h
```

In addition to these files, the installer also copies a PAM (Pluggable Authentication Module) file called '`omi.pam`' under the '`/etc/pam`' directory.

The following sections discuss these installed files.

2.7.1 '`bin`'

The '`bin`' directory contains all OMI programs, including:

- `omiserver` – the server program.
- `omiagent` – the provider agent program.
- `omigen` – the provider generation tool.
- `omireg` – the provider registration tool.
- `omicli` – the command-line client tool.

2.7.2 '`etc`'

The '`etc`' directory contains system-wide configuration files used by various programs, including:

- `omicli.conf` – configuration file for omicli program.
- `omigen.conf` – configuration file for omigen program.
- `omiserver.conf` – configuration file for omiserver program.

The `omicli` and `omigen` programs look first for configuration files named `.omiclirc` and `.omigenrc` in the current and home directories (in which case the system-wide configuration file is ignored).

2.7.3 '`lib`'

The '`lib`' directory contains libraries. These include:

- `libmicxx.so` – the C++ provider support library.
- `libomiclient.so` – the C++ binary protocol client library.
- `libomiidentify.so` – the identify provider (OMI_Identify class).

2.7.4 '`include`'

The '`include`' directory contains C and C++ header files required for provider and client application development. These include:

- `MI.h` – C provider header file.
- `micxx/micxx.h` – main C++ provider header file.
- `omiclient/client.h` – main C++ client header file.

2.7.5 ‘omischema’

The ‘omischema’ directory contains MOF files that define the CIM schema. These files are used by the provider generator tool (‘omigen’) while generating provider sources. The directory contains hundreds of MOF files. The main MOF file is called ‘CIM_Schema.mof’ (which includes all others).

2.7.6 ‘etc/ssl/certs’

The ‘etc/ssl/certs’ directory contains PEM-formatted certificates for SSL (private and public). These include:

- omi.pem – the public certificate.
- omikey.pem – the private certificate/key.

2.7.7 ‘etc/omiregister’

The ‘etc/omiregister’ directory contains a *namespace directory* for each CIM namespace. Each namespace directory has the same name as the corresponding CIM namespace, except ‘/’ characters are translated to ‘#’ characters. For example, for the CIM namespace ‘root/cimv2’, there is a directory named ‘root#cimv2’. The server scans the ‘etc/omiregister’ directory during startup to obtain a list of supported namespace.

Each namespace directory contains provider registration files (with a ‘.reg’ extension). Each registration file corresponds to a single provider library. These files are created by the ‘omireg’ utility. The following registration file (named ‘omiidentify.reg’) registers a provider that implements the ‘OMI_Identify’ class.

```
LIBRARY=omiidentify
CLASS=OMI_Identify
```

Placing this file in the ‘etc/omiregister/root#omi’ directory, registers the provider for that namespace. The server scans all namespace directories to discover provider registrations during startup.

2.7.8 ‘share’

The ‘share’ directory contains the ‘omi.mak’ file. This file is included by provider makefiles generated by the ‘omigen’ tool.

3 Using the server

This chapter explains how to use the server program. It explains how to start, validate, and stop the server. It also explains various options and where to find the log files.

3.1 Setting up your path

You may run each program by specifying its fully-qualified path. But for convenience, you may wish to add the ‘`bin`’ directory to your path. The examples below assume you have done so.

3.2 Getting help

To get help with server options, type the following.

```
# omiserver -h
```

This prints a help message that explains the usage, arguments, and options.

3.3 Starting the server

To start the server in the foreground, type this.

```
# omiserver
```

To start the server in the background, use the ‘`-d`’ (daemonize) option.

```
# omiserver -d
```

Multiple instances of the server may run on the same host subject to the following constraints:

- Each server is built with a distinct installation prefix, so that each server has a unique PID file and socket file paths.
- Each server binds to a distinct port. The port may be set with the `--port` command-line option or `port` configuration file option.

If these constraints are not met, attempting to run a second server results in an ‘already running’ message.

3.4 Validating the server

To validate that the server is working correctly, use the ‘`omicli`’ tool to send it a request. Type ‘`omicli -h`’ for help with this tool. When initially installed, the server only has one provider, which provides the ‘`OMI_Identify`’ class. To enumerate all instances of this class, type the following command (‘`id`’ is short for ‘`identify`’).

```
# omicli id
```

If the server is working properly, this command should print single instance to standard output. For example:

```
instance of OMI_Identify
{
    [Key] InstanceID=2FDB5542-5896-45D5-9BE9-DC04430AAABE
    SystemName=linux
    ProductName=OMI 1.0.0
```

```
        ProductVendor=Microsoft
        ProductVersionMajor=1
        ProductVersionMinor=0
        ProductVersionRevision=0
        ProductVersionString=1.0.0
        Platform=LINUX_IX86_GNU
        OperatingSystem=LINUX
        Architecture=IX86
        Compiler=GNU
        ConfigPrefix=/tmp/omi
        ConfigLibDir=/tmp/omi/lib
        ConfigBinDir=/tmp/omi/bin
        ConfigIncludeDir=/tmp/omi/include
        ConfigDataDir=/tmp/omi/share
        ConfigLocalStateDir=/tmp/omi/var
        ConfigSysConfDir=/tmp/omi/etc
        ConfigProviderDir=/tmp/omi/etc
        ConfigLogFile=/tmp/omi/var/log/omiserver.log
        ConfigPIDFile=/tmp/omi/var/run/omiserver.pid
        ConfigRegisterDir=/tmp/omi/etc/omiregister
        ConfigSchemaDir=/tmp/omi/share/omischema
        ConfigNameSpaces={root#omi, interop, root#cimv2}
    }
```

This instance identifies various characteristics of the server and system.

3.5 Stopping the server

To stop the server, type the following.

```
# omiserver -s
```

This stops the server by sending a signal to the process whose process id (pid) is contained in '`var/run/omiserver.pid`'. The server removes this file when it shuts down.

3.6 Server and Agent Logs

Server log messages are directed to '`var/log/omiserver.log`'. The server spawns agent processes ('`omiagent`') in order to run providers as specified users (determined by the provider hosting model). Log messages from agents are written to files whose name has the form:

```
var/log/omiagent.<UID>.<GID>.log
```

'`<UID>`' and '`<GID>`' are the user id and group id of the agent process's owner. To browse logs, look for files under '`var/log`' whose name matches '`omi*`'.

3.7 Server file and directory locations

Sometimes it is helpful to know where the server expects to find various files. Where is the server log file? Where is the provider registration directory? To obtain a list of server and file locations, type the following command.

```
# omiserver -p
```

Running this on a system where OMI was installed under '`/opt/omi`' prints the following.

```
prefix=/opt/omi
libdir=/opt/omi/lib
bindir=/opt/omi/bin
localstatedir=/opt/omi/var
sysconfdir=/opt/omi/etc
providerdir=/opt/omi/lib
certsdir=/opt/omi/etc/ssl/certs
datadir=/opt/omi/share
rundir=/opt/omi/var/run
logdir=/opt/omi/var/log
schemadir=/opt/omi/share/omischema
schemafile=/opt/omi/share/omischema/CIM_Schema.mof
pidfile=/opt/omi/var/run/omiserver.pid
logfile=/opt/omi/var/log/omiserver.log
registerdir=/opt/omi/etc/omiregister
pemfile=/opt/omi/etc/ssl/certs/omi.pem
keyfile=/opt/omi/etc/ssl/certs/omikey.pem
agentprogram=/opt/omi/bin/omiagent
serverprogram=/opt/omi/bin/omiserver
includedir=/opt/omi/include
configfile=/opt/omi/etc/omiserver.conf
socketfile=/opt/omi/var/omiserver.sock
```

4 Using the command-line client (‘omicli’)

This chapter explains how to use the command-line client tool. This tool sends requests to the local CIM server and prints the responses to standard output. For example, ‘omicli ei root/omi OMI_Identify’ sends the ‘ei’ request (‘enumerate-instances’) to the server and then prints the following on standard output.

```
instance of OMI_Identify
{
    [Key] InstanceID=2FDB5542-5896-45D5-9BE9-DC04430AAABE
    SystemName=linux
    ProductName=OMI 1.0.0
    ProductVendor=Microsoft
    ProductVersionMajor=1
    ProductVersionMinor=0
    ProductVersionRevision=0
    ProductVersionString=1.0.0
    Platform=LINUX_IX86_GNU
    OperatingSystem=LINUX
    Architecture=IX86
    Compiler=GNU
    ConfigPrefix=/tmp/omi
    ConfigLibDir=/tmp/omi/lib
    ConfigBinDir=/tmp/omi/bin
    ConfigIncludeDir=/tmp/omi/include
    ConfigDataDir=/tmp/omi/share
    ConfigLocalStateDir=/tmp/omi/var
    ConfigSysConfDir=/tmp/omi/etc
    ConfigProviderDir=/tmp/omi/etc
    ConfigLogFile=/tmp/omi/var/log/omiserver.log
    ConfigPIDFile=/tmp/omi/var/run/omiserver.pid
    ConfigRegisterDir=/tmp/omi/etc/omiregister
    ConfigSchemaDir=/tmp/omi/share/omischema
    ConfigNameSpaces={root#omi, interop, root#cimv2}
}
```

Each ‘instance of { ... }’ construct represents an instance of a CIM class. The braces contain properties and their values. Key properties are annotated with the ‘Key’ qualifier.

The general usage of the tool is:

```
# omicli COMMAND ARGUMENTS
```

The ‘COMMAND’ is one of the following:

- noop – send a no-op request to the server.
- gi – send a get-instance request to the server.
- ci – send a create-instance request to the server.
- mi – send a modify-instance request to the server.
- di – send a delete-instance request to the server.

- `ei` – send an enumerate-instances request to the server.
- `iv` – send an invoke (extrinsic method) request to the server.
- `a` – send an associators request to the server.
- `r` – send an references request to the server.
- `id` – send an identify request to the server.

The ‘`ARGUMENTS`’ are command-specific and are described below.

4.1 Getting help on options

To print a help message, type the following.

```
# omicli -h
```

The message explains the syntax of key commands.

4.2 The socket file

By default, when ‘`omicli`’ and ‘`omiserver`’ are built together (with the same prefix), the ‘`omicli`’ program contains the location of the server’s socket file. But when they are built separately, or if you want to communicate with multiple instances of the server, you must specify the socket file location using the ‘`--socketfile`’ option. The socket file is located here: ‘`*/var/run/omiserver.sock`’, where ‘*’ is the prefix the server was built with.

4.3 The No-Op request

The following command sends a no-op request to the server.

```
# omicli noop
```

This tests whether the server is running and responsive. If so, it prints a message indicating success. If the server is not responsive, the command will timeout.

4.4 Enumerating Instances

The following command enumerates instances of ‘`OMI_Identify`’ within the ‘`root/omi`’ namespace.

```
# omicli ei root/omi OMI_Identify
```

4.5 Getting an Instance

The following command gets a single instance of the ‘`OMI_Identify`’ class from the ‘`root/omi`’ namespace.

```
# omicli gi root/omi { OMI_Identify InstanceID 2FDB5542-5896-45D5-9BE9-DC04430AAABE
```

The expression in braces represents the **instance name** for the given instance. This instance name has a single key, although an instance name may have multiple keys. Consider, for example, the following class.

```
class MyClass
{
    [Key] String Key1;
    [Key] Uint32 Key2;
```

```
        [Key] Boolean Key2;
        ...
    };
```

And now consider the following instance of that class.

```
    instance of MyClass
    {
        Key1=XYZ
        Key2=123
        Key3=false
        ...
    };
```

The instance name for this instance is expressed as follows on the command line.

```
    { MyClass Key1 XYZ Key2 123 Key3 false }
```

In general, instance names are expressed as a class name followed by name-value pairs, all enclosed in braches. Failing to specify values for a complete set of keys results in an error.

4.6 Invoking a method

To invoke an extrinsic method, use the ‘iv’ command, whose usage is:

```
    omicli iv NAMESPACE INSTANCENAME METHODNAME PARAMETERS
```

For example, consider the ‘SetState’ extrinsic method defined by the following CIM class.

```
    class OMI_Frog
    {
        [Key] Uint32 Key;

        Uint32 SetState(
            [In] String NewState,
            [In(false), Out] String OutState);
    };
```

The following command invokes the ‘SetState’ method on the instance of ‘OMI_Frog’ named ‘{ OMI_Frog Key 123 }’.

```
    omicli iv root/omi { OMI_Frog Key 123 } SetState { NewState Hopping }
```

This command prints any output parameters to standard output. For example, the above command might print this:

```
    { OldState Sitting }
```

5 Using the client library '`omiclient`'

The client library defines a C++ API enabling client applications to connect to and send requests to the CIM server. It uses a local binary protocol for communicating with the CIM server; so the client application must reside on the same host as the CIM server. This chapter explains how to get started with this library.

5.1 Client library source examples

The following source files (under the source distribution) illustrate how to use the client library.

```
./omiclient/tests/test_client.cpp
./cli/cli.cpp
```

The first is the unit test for the '`omiclient`' library. The second is the main source file of the '`omicli`' tool discussed above. The examples below show how to do simple things with the client library. For more detail, see these examples.

5.2 The '`omiclient`' library

The base name of client library is '`omiclient`'. The full name is platform dependent. For example, on Linux the full name is '`libomiclient.so`'. This library resides in the '`lib`' directory (selected when the distribution was built). The client application must be linked with this library.

5.3 The '`<omiclient/client.h>`' header

Client applications must include <omiclient/client.h>. This header file defines the full client interface. It resides in the '`include`' directory (selected when the distribution was built).

5.4 Connecting to the local server

The first step is to connect to the local CIM server, illustrated by the following program.

```
#include <omiclient/client.h>

using namespace std;

int main()
{
    const Uint64 timeout = 30000000;

    Client c;
    String locator;
    String username;
    String password;

    if (!c.Connect(locator, username, password, timeout))
    {
        // Error!
17      }
18
29      return 0;
20  }
```

Line 1 includes ‘<omiclient/client.h>’, the main header file for the client interface. This header is located under the installation ‘include’ directory. Consult this file to more details about the interface.

Line 9 instantiates an instance of the ‘Client’ class. This function takes an optional ‘Handler’ instance, which is required when calling the asynchronous member functions. The examples below use the synchronous methods and so no handler is needed.

Line 14 establishes a synchronous connection with the local CIM server. The ‘Client::Connect’ function takes four arguments: ‘locator’, ‘username’, ‘password’, and ‘timeout’. The ‘locator’ is a string that specifies the Unix domain socket file used to connect to the server. If the ‘locator’ is empty, the client uses the default socket file, located under the ‘run’ directory with the name ‘omiserver.sock’.

To connect to a serer whose socket file is not in the default location, set the location parameter to the the full path of that socket file, such as ‘/opt/omi/var/run/omiserver.sock’.

The ‘username’ and ‘password’ parameters are used to authenticate the user with the server. There are two kinds of authentication: explicit and implicit. With **explicit** authentication, the ‘username’ and ‘password’ parameters hold the log on credentials for a given user. With **implicit** authentication, these parameters are empty, in which case the user is authenticated using the identity of the current user (obtained with the ‘getuid’ system call).

The ‘timeout’ parameter specifies how long to wait (in microseconds) before failing. In this example, the timeout is 30 seconds (30,000,000 microseconds).

Developers may call the ‘Client::Disconnect()’ method to explicitly disconnect from the server. Otherwise, the connection is closed implicitly by the ‘Client’ destructor.

5.5 Enumerating instances

The code fragment below enumerates instances of the ‘OMI_Identify"’ class.

```
01      const String nameSpace = "root/omi";
02      const String className = "OMI_Identify";
03      const bool deepInheritance = true;
04      const Uint64 timeout = 2000000;
05      Array<DInstance> instances;
06      MI_Result result;
07
08      if (!c.EnumerateInstances(nameSpace, className, deepInheritance,
09          timeout, instances, result))
10      {
11          // Error!
12      }
13
14      if (result != MI_RESULT_OK)
15      {
```

```
16          // Error!
17      }
18
19      for (Uint32 i = 0; i < instances.GetSize(); i++)
20          instances[i].Print();
```

Line 8 performs an enumeration request. It obtains instances of the given class from the given namespace. The resulting instances are in the ‘`instances`’ parameter upon return.

The ‘`deepInheritance`’ parameter specifies whether to return instances of classes derived from ‘`OMI_Identify`’ (if true) or to return instances of ‘`OMI_Identify`’ only (if false).

Line 19 through 20 print the resulting instances.

Use ‘`EnumerateInstances`’ with regard for memory usage. It places all instances into memory at once, which may exhaust available memory when there are many thousands of instances. To avoid memory exhaustion, use the asynchronous form, called ‘`EnumerateInstancesAsync`’. All asynchronous functions use the ‘`Handler`’ class, which defines virtual functions for delivering instances one at a time. See Appendix B for a fully asynchronous example.

5.6 Getting a single instance

The code fragment below shows how to get a single instance from the server.

```
// Construct an instance name:
const String className = "OMI_Identify";
DInstance instanceName(className, DInstance::CLASS);

// Add a key property:
const String propertyName = "InstanceID";
const String instanceID = "2FDB5542-5896-45D5-9BE9-DC04430AAABE";
const String isNull = false;
const String isKey = true;
instanceName.AddUint32(propertyName, instanceID, isNull, isKey);

// Perform get instance:
const String nameSpace = "root/omi";
const Uint64 TIMEOUT = 2000000;
DInstance instance;
MI_Result result;
if (!c.GetInstance(nameSpace, instanceName, TIMEOUT, instance,
    result))
{
    // Error!
}

if (result != MI_RESULT_OK)
{
    // Error!
}
```

```
27
28  instance.Print();
```

Lines 1 through 10 build an instance name (using the ‘`DInstance`’ class). Lines 2 through 3 construct a ‘`DInstance`’ whose class name is ‘`OMI_Identify`’. Lines 6 through 10 add a key property to the instance name called ‘`InstanceID`’.

Lines 13 through 18 perform the get instance request. Upon return, the ‘`instance`’ parameter contains the result.

Finally, Line 28 prints the resulting instance to standard output.

5.7 Invoking an extrinsic method

This section shows how to invoke an extrinsic method. Recall the definition of the ‘`OMI_Frog`’ class.

```
class OMI_Frog
{
    [Key] Uint32 Key;

    Uint32 SetState(
        [In] String NewState,
        [In(false), Out] String OutState);
};
```

The code fragment shows how to invoke the ‘`OMI_Frog.SetState`’ method.

```
01  // Initialize instance name:
02  DInstance instanceName("OMI_Frog", DInstance::CLASS);
03  instanceName.AddUint32("Key", 123, false, true);
04
05  // Initialize input parameters:
06  DInstance in(T("SetState"), DInstance::METHOD);
07  in.AddString(T("NewState"), "Hopping", false, false);
08
09  // Invoke method:
10  const Uint64 TIMEOUT = 5000000;
11  DInstance out;
12  MI_Result result;
13
14  if (!c.Invoke(
15      "root/omi",
16      instanceName,
17      "SetState",
18      in,
19      TIMEOUT,
20      out,
21      result))
22  {
23      // Error!
24  }
```

```
25
26  if (result != MI_RESULT_OK)
27  {
28      // Error!
29  }
30
31  out.Print();
```

Lines 2 through 3 initialize the instance name, which identifies the instance of ‘`OMI_Frog`’ whose method will be called. This specifies a single key named ‘`Key`’ with the value ‘`123`’.

Lines 6 through 7 initialize the input parameters for the method. In this example, there is a single parameter named ‘`NewState`’ with the value ‘`Hopping`’.

Lines 14 through 21 invoke the ‘`SetState`’ method. Upon return, ‘`out`’ holds any output parameters (the ‘`OldState`’ parameter in this case). The ‘`out`’ parameter always has a parameter named ‘`MIReturn`’, which contains the return value of the function (the return value of ‘`OMI_Frog.SetState`’ in this example). In CIM, all functions are required to return a value.

6 Developing a provider in 5 minutes

This chapter provides a very quick overview of the provider development process. It shows the minimum steps for building a simple instance provider. For a more complete discussion of provider development, see the next chapter.

Appendix A contains a complete listing of all file in this example. These files are also included in the source distribution under 'omi-1.0.0/doc/omi/samples/frog'.

6.1 Defining 'schema.mof'

First we define the class schema shown in the 'schema.mof' file below.

```
class XYZ_Frog
{
    [Key] String Name;
    Uint32 Weight;
    String Color;
};
```

6.2 Generating the provider sources

Next we generate the provider sources and the makefile using the command below.

```
mbrasher@linux:~/gadget> omigen --cpp -m frog schema.mof XYZ_Frog
Creating XYZ_Frog.h
Creating XYZ_Frog_Class_Provider.h
Creating XYZ_Frog_Class_Provider.cpp
Creating schema.c
Creating stubs.cpp
Creating module.cpp
Creating module.h
Creating GNUmakefile
```

6.3 Implementing the 'EnumerateInstances' stub

Next we implement the 'EnumerateInstances' stub. The generated stub looks like this:

```
void XYZ_Frog_Class_Provider::EnumerateInstances(
    Context& context,
    const String& nameSpace,
    const PropertySet& propertySet,
    bool keysOnly,
    const MI_Filter* filter)
{
    context.Post(MI_RESULT_NOT_SUPPORTED);
}
```

The implementation below provides two frogs.

```
void XYZ_Frog_Class_Provider::EnumerateInstances(
    Context& context,
    const String& nameSpace,
    const PropertySet& propertySet,
    bool keysOnly,
    const MI_Filter* filter)
{
    XYZ_Frog_Class frog1;
    frog1.Name_value("Fred");
    frog1.Weight_value(55);
    frog1.Color_value("Green");
    context.Post(frog1);

    XYZ_Frog_Class frog2;
    frog2.Name_value("Sam");
    frog2.Weight_value(65);
    frog2.Color_value("Blue");
    context.Post(frog2);

    context.Post(MI_RESULT_OK);
}
```

Note: In CIM, a property either has a value or is null (has no value). The '`XYZ_Frog.Name_exists`' function returns true if the '`Frog.Name`' property has a value or false if the property is null (has no value). If the property has a value, one may call the '`XYZ_Frog.Name_value`' function to obtain it.

6.4 Registering the provider

Next we register the provider as follows:

```
# make reg
/opt/omi/bin/omireg libfrog.so
Copied provider to /opt/omi/lib/libfrog.so
Created /opt/omi/etc/omiregister/root#cimv2/frog.reg
```

This creates '`frog.reg`' under the registration directory for the default namespace '`root/cimv2`' and it copies the provider to the installed directory.

6.5 Testing the provider

To test the provider, send an enumerate request to the provider as shown below.

```
# omicli ei root/cimv2 XYZ_Frog
instance of XYZ_Frog
{
    [Key] Name=Fred
    Weight=55
    Color=Green
}
instance of XYZ_Frog
{
    [Key] Name=Sam
    Weight=65
    Color=Blue
}
```

6.6 Going further

This chapter provided a brief overview of the provider development process. For a more complete discussion of provider development, please proceed to the next chapter.

7 Developing providers

This chapter shows how to develop providers, a process consisting of 6 stages:

- Defining the MOF schema
- Generating the provider sources
- Implementing the provider operations
- Building the provider
- Registering the provider
- Validating the provider

We discuss each stage, showing how to develop providers that implement the following operations:

- **get-instance**
- **enumerate-instances**
- **associator-names**
- **reference-names**
- **invoke-method**

OMI supports two provider language bindings: C and C++. This chapter only shows how to use the C++ binding. Whether you build C or C++ providers, the development stages are the same although the details of the interface vary. For more information about the C interface, see the '`<MI/MI.h>`' header file and experiment with generating C providers.

7.1 Defining the MOF schema

The first stage is to define the MOF classes comprising your schema. You may extend an existing CIM class like this:

```
class XYZ_MyComputerSystem : CIM_ComputerSystem
{
    ...
};
```

Or you may define a new root class (with no super class):

```
class MyClass
{
    ...
};
```

The MOF language is defined in the **CIM Infrastructure Specification (DSP0004)**, which may be found at (`http://dmtf.org/standards/cim`).

The provider developed below implements the following class definitions (which are placed in a file called '`schema.mof`').

```
// schema.mof

class XYZ_Widget
{
    [Key] Uint32 Key;
    Uint32 ModelNumber;
    String Color;
};

class XYZ_Gadget
{
    [Key] Uint32 Key;
    Uint32 ModelNumber;
    Uint32 Size;

    Uint32 ChangeState(
        [In] Uint32 NewState,
        [In(False), Out] Uint32 OldState);
};

[Association]
class XYZ_Connector
{
    [Key] XYZ_Widget REF Left;
    [Key] XYZ_Gadget REF Right;
};
```

Notice that all classes define above have the '`XYZ_`' prefix. Similarly, all classes in the CIM schema begin have the '`CIM_`' prefix. All classes should have a suitable prefix but for brevity, this prefix is omitted henceforth.

The '`Connector`' class is an association, as indicated by the '`Associator`' qualifier. Each instance of '`Connector`', connects one instance of '`Widget`' with one instance of '`Gadget`'.

7.2 Generating the provider sources

The second stage involves generating the provider sources. The following command generates provider sources from the '`schema.mof`' file defined above.

```
omigen --cpp -m xyzconnector schema.mof \
   XYZ_Gadget=Gadget XYZ_Widget=Widget XYZ_Connector=Connector
Creating Gadget.h
Creating Gadget_Class_Provider.h
Creating Gadget_Class_Provider.cpp
Creating Widget.h
Creating Widget_Class_Provider.h
Creating Widget_Class_Provider.cpp
Creating Connector.h
Creating Connector_Class_Provider.h
Creating Connector_Class_Provider.cpp
Creating schema.c
Creating stubs.cpp
Creating module.cpp
Creating module.h
```

```
    Creating GNUmakefile
```

The '--cpp' option creates 'C++' sources (instead of C sources by default). The '-m xyzconnector' option creates 'GNUmakefile' with rules for building, regenerating, and registering the provider. This makefile creates a library whose base name is given by the '-m' option ('xyzconnector').

Tip: You may regenerate sources by typing 'make gen' or by retyping the command above. The generator will never overwrite editable files. Instead it attempts to patch them. Some files are non-editable and are regenerated completely.

The generator creates sources for classes 'XYZ_Gadget', 'XYZ_Widget', and 'XYZ_Connector'. The command above defines aliases for each class name, allowing shorter names to be used throughout the source code. For example, 'XYZ_Gadget=Gadget' causes 'Gadget.h' to be generated instead of 'XYZ_Gadget.h'. Alias can be used to completely rename a class. For example: 'CIM_ComputerSystem=CompSys'.

To learn more about the 'omigen' options, type 'omigen -h' to print a help message.

The purpose of each generated file is given below.

- Gadget.h – defines C++ class for the CIM Gadget class.
- Widget.h – defines C++ class for the CIM Widget class.
- Connector.h – defines C++ class for the CIM Connector class.
- Gadget_Class_Provider.h – defines Gadget_Class_Provider class.
- Gadget_Class_Provider.cpp – defines Gadget_Class_Provider class.
- Widget_Class_Provider.h – defines Widget_Class_Provider class.
- Widget_Class_Provider.cpp – defines Widget_Class_Provider class.
- Connector_Class_Provider.h – defines Connector_Class_Provider class.
- Connector_Class_Provider.cpp – defines Connector_Class_Provider class.
- schema.c – internal definitions.
- stubs.cpp – internal definitions.
- module.cpp – defines 'MI_Main' library entry point.
- module.h – defines 'Module' class.
- GNUmakefile – defines rules for building the provider library.

Many of these files are not intended to be edited. Developer edits may be made to the following files.

- Gadget_Class_Provider.h
- Gadget_Class_Provider.cpp
- Widget_Class_Provider.h
- Widget_Class_Provider.cpp
- Connector_Class_Provider.h
- Connector_Class_Provider.cpp
- module.h
- module.cpp

For example, to implement the '`get-instances`' operation for the '`Gadget`' class, modify the '`Gadget_Class_Provider.cpp`' file.

7.3 Implementing the provider operations

This section shows how to implement the following provider operations:

- **get-instance**
- **enumerate-instances**
- **associator-names**
- **reference-names**
- **invoke-method**

7.3.1 Implementing enumerate-instances

This section implement the enumerate-instances operation for the `Gadget` class. This implementation provides the following instances (shown in MOF format).

```
instance of XYZ_Gadget
{
    Key = 1003;
    ModelNumber = 3;
    Size = 33;
};

instance of XYZ_Gadget
{
    Key = 1004;
    ModelNumber = 4;
    Size = 43;
};
```

To implement the enumerate-instance operation for the `Gadget` class, start by examining the generated stub (see '`Gadget_Class_Provider.cpp`').

```
void Gadget_Class_Provider::EnumerateInstances(
    Context& context,
    const String& nameSpace,
    const PropertySet& propertySet,
    bool keysOnly,
    const MI_Filter* filter)
{
    context.Post(MI_RESULT_NOT_SUPPORTED);
}
```

This function is invoked by the CIM server. The implementer may respond on the same thread or he may create a new thread if the request is long running. The lifetime of the request is bound to the lifetime of the '`context`' parameter. All parameters remain in scope until the provider calls '`Context::Post`'. The provider may create a new thread to handle the request, in which case the parameters may live beyond the invocation of '`EnumerateInstances`'.

We provide an implementation that provides two instances of the '`Gadget`' class. The following implementation handles the request on the calling thread.

```
void Gadget_Class_Provider::EnumerateInstances(
    Context& context,
    const String& nameSpace,
    const PropertySet& propertySet,
    bool keysOnly,
    const MI_Filter* filter)
{
    // Gadget.Key=1003:
    {
        Gadget_Class g;
        g.Key_value(1003);
        g.ModelNumber_value(3);
        g.Size_value(33);
        context.Post(g);
    }

    // Gadget.Key=1004:
    {
        Gadget_Class g;
        g.Key_value(1004);
        g.ModelNumber_value(4);
        g.Size_value(43);
        context.Post(g);
    }

    context.Post(MI_RESULT_OK);
}
```

This function constructs instances of the '`Gadget`' class and passes them to the '`Context::Post`' function. When all instances have been posted, the provider passes the result status to the '`Context::Post`' function. This finalizes the request.

Tip: By passing the '`--nogi CLASSNAME`' option (no get-instance) to the generator tool, the server uses the '`enumerate-instances`' implementation to satisfy all '`get-instance`' requests. Only use this technique if the number of instances is small, otherwise the provider will be very slow.

7.3.2 Implementing get-instance

Next we show how to implement the `get-instance` operation for the `Gadget` class. Here is the generated stub for the `get-instance` request.

```
void Gadget_Class_Provider::GetInstance(
    Context& context,
    const String& nameSpace,
    const Gadget_Class& instanceName,
    const PropertySet& propertySet)
{
    context.Post(MI_RESULT_NOT_SUPPORTED);
}
```

The `instanceName` parameter holds the keys for the target instance. Here is the full implementation of this function.

```
    void Gadget_Class_Provider::GetInstance(
        Context& context,
        const String& nameSpace,
        const Gadget_Class& instanceName,
        const PropertySet& propertySet)
    {
        if (instanceName.Key_value() == 1003)
        {
            // Gadget.Key=1003:
            Gadget_Class g;
            g.Key_value(1003);
            g.ModelNumber_value(3);
            g.Size_value(33);
            context.Post(g);
            context.Post(MI_RESULT_OK);
        }
        else if (instanceName.Key_value() == 1004)
        {
            // Gadget.Key=1004:
            Gadget_Class g;
            g.Key_value(1004);
            g.ModelNumber_value(4);
            g.Size_value(43);
            context.Post(g);
            context.Post(MI_RESULT_OK);
        }
        else
        {
            context.Post(MI_RESULT_NOT_FOUND);
        }
    }
```

We examine the key and return the matching instance. If neither condition matches, we post the '`MI_RESULT_NOT_FOUND`' result.

7.3.3 Implementing an extrinsic method

This section implements the '`ChangeState`' extrinsic method. The generator produces the following stub.

```
    void Gadget_Class_Provider::Invoke_ChangeState(
        Context& context,
        const String& nameSpace,
        const Gadget_Class& instanceName,
        const Gadget_ChangeState_Class& in)
    {
        context.Post(MI_RESULT_NOT_SUPPORTED);
    }
```

The '`instanceName`' parameter is the instance whose '`ChangeState`' method has been invoked (this parameter is omitted for static methods). The '`in`' parameter contains the input parameters. The implementation should perform the following tasks:

- Read the input parameters.
- Perform the desired action.
- Build the output parameters.
- Set the return value.
- Post the output parameters to the server.
- Return a successful status.

The following implementation performs each of these steps.

```
    void Gadget_Class_Provider::Invoke_ChangeState(
        Context& context,
        const String& nameSpace,
        const Gadget_Class& instanceName,
        const Gadget_ChangeState_Class& in)
    {
        Gadget_ChangeState_Class out;

        // Print the input parameter:
        if (in.NewState_exists())
        {
            printf("NewState=%u\n", in.NewState_value());
        }

        // Perform desired action here:
        ...

        // Set the output parameter:
        out.OldState_value(2);

        // Set the return value:
        out.MIReturn_value(0);

        // Post the 'out' object.
        context.Post(out);

        // Post the result status.
        context.Post(MI_RESULT_OK);
    }
```

7.3.4 Implementing enumerate-instances for an association provider

This section shows how to implement the enumerate-instances operation for the '`Connector`' association class. This operation produces the following instances (shown in MOF format).

```
instance of XYZ_Connector
{
    Left = "XYZ_Widget.Key=1001";
    Right = "XYZ_Gadget.Key=1003";
};

instance of XYZ_Connector
{
    Left = "XYZ_Widget.Key=1002";
    Right = "XYZ_Gadget.Key=1004";
};
```

Here is the implementation.

```
void Connector_Class_Provider::EnumerateInstances(
    Context& context,
    const String& nameSpace,
    const PropertySet& propertySet,
    bool keysOnly,
    const MI_Filter* filter)
{
    // Connector.Left="Gadget.Key=1001","Widget.Key="1003"
    {
        Widget_Class w;
        w.Key_value(1001);
        Gadget_Class g;
        g.Key_value(1003);
        Connector_Class c;
        c.Left_value(w);
        c.Right_value(g);
        context.Post(c);
    }

    // Connector.Left="Gadget.Key=1002","Widget.Key="1004"
    {
        Widget_Class w;
        w.Key_value(1002);
        Gadget_Class g;
        g.Key_value(1004);
        Connector_Class c;
        c.Left_value(w);
        c.Right_value(g);
        context.Post(c);
    }
```

```
        context.Post(MI_RESULT_OK);
    }
```

7.3.5 Implementing get-instances for an association class

The following example implements '`get-instance`' for the '`Connector`' association class.

```
    void Connector_Class_Provider::GetInstance(
        Context& context,
        const String& nameSpace,
        const Connector_Class& instanceName,
        const PropertySet& propertySet)
    {
        if (instanceName.Left_value().Key_value() == 1001 &&
            instanceName.Right_value().Key_value() == 1003)
        {
            // Connector.Left="Gadget.Key=1001","Widget.Key="1003"
            Widget_Class w;
            w.Key_value(1001);

            Gadget_Class g;
            g.Key_value(1003);

            Connector_Class c;
            c.Left_value(w);
            c.Right_value(g);

            context.Post(c);
            context.Post(MI_RESULT_OK);
        }
        else if (instanceName.Left_value().Key_value() == 1002 &&
            instanceName.Right_value().Key_value() == 1004)
        {
            // Connector.Left="Gadget.Key=1002","Widget.Key="1004"
            Widget_Class w;
            w.Key_value(1001);

            Gadget_Class g;
            g.Key_value(1003);

            Connector_Class c;
            c.Left_value(w);
            c.Right_value(g);

            context.Post(c);
            context.Post(MI_RESULT_OK);
        }
        else
        {
            context.Post(MI_RESULT_NOT_FOUND);
        }
    }
```

Note that `instanceName.Left_value()` returns an instance of type `Gadget` (see MOF definitions). And so `instanceName.Left_value().Key_value()` returns the ‘`Key`’ property of the associated `Gadget` instance.

7.3.6 Implementing the associator-instances operation

This section shows how to implement the associator-instances operation. This operation finds the other end of an association. For example, it might find the `Gadget` instances that are associated with a single `Widget` instance (through a `Connector` instance). For example, consider the following MOF definitions.

```
instance of XYZ_Connector
{
    Left = "XYZ_Widget.Key=1001";
    Right = "XYZ_Gadget.Key=1003";
};

instance of XYZ_Connector
{
    Left = "XYZ_Widget.Key=1002";
    Right = "XYZ_Gadget.Key=1004";
};
```

The associator-instances operation starts with the instance name of an instance and finds associated instances. For example, the associator-instances of:

```
XYZ_Widget.Key=1001
```

include:

```
XYZ_Gadget.Key=1003
```

The generator produces two stubs to handle associator-instances requests. The first yields associators in which the `instanceName` parameter matches the first reference property (`Connector.Left`). The second yields associations in which the `instanceName` parameter matches the second reference property (`Connector.Right`). The stubs are defined as follows.

```
void Connector_Class_Provider::AssociatorInstancesLeft(
    Context& context,
    const String& nameSpace,
    const Widget_Class& instanceName,
    const String& resultClass,
    const PropertySet& propertySet,
    bool keysOnly,
    const MI_Filter* filter)
{
    context.Post(MI_RESULT_NOT_SUPPORTED);
}
```

```
void Connector_Class_Provider::AssociatorInstancesRight(
    Context& context,
    const String& nameSpace,
    const Gadget_Class& instanceName,
    const String& resultClass,
    const PropertySet& propertySet,
    bool keysOnly,
    const MI_Filter* filter)
{
    context.Post(MI_RESULT_NOT_SUPPORTED);
}
```

The following implementation yields associators of `Widget` instances.

```
void Connector_Class_Provider::AssociatorInstancesLeft(
    Context& context,
    const String& nameSpace,
    const Widget_Class& instanceName,
    const String& resultClass,
    const PropertySet& propertySet,
    bool keysOnly,
    const MI_Filter* filter)
{
    if (instanceName.Key_value() == 1001)
    {
        // Gadget.Key=1003:
        Gadget_Class g;
        g.Key_value(1003);
        g.ModelNumber_value(3);
        g.Size_value(33);
        context.Post(g);
        context.Post(MI_RESULT_OK);
    }
    else if (instanceName.Key_value() == 1002)
    {
        // Gadget.Key=1004:
        Gadget_Class g;
        g.Key_value(1004);
        g.ModelNumber_value(4);
        g.Size_value(43);
        context.Post(g);
        context.Post(MI_RESULT_OK);
    }
    else
    {
        context.Post(MI_RESULT_NOT_FOUND);
    }
    }
```

@c end cartouche

7.3.7 Implementing the reference-instances operation

The reference-instances operation takes an instance name and finds the reference instances that refer to it. Consider the following MOF definitions.

```
instance of XYZ_Connector
{
    Left = "XYZ_Widget.Key=1001";
    Right = "XYZ_Gadget.Key=1003";
};

instance of XYZ_Connector
{
    Left = "XYZ_Widget.Key=1002";
    Right = "XYZ_Gadget.Key=1004";
};
```

For example, the reference-instance of `XYZ_Widget.Key=1001` includes the first MOF instance shown above. As with associator-instances, the generator produces two stubs:

```
void Connector_Class_Provider::ReferenceInstancesLeft(
    Context& context,
    const String& nameSpace,
    const Widget_Class& instanceName,
    const PropertySet& propertySet,
    bool keysOnly,
    const MI_Filter* filter)
{
    context.Post(MI_RESULT_NOT_SUPPORTED);
}

void Connector_Class_Provider::ReferenceInstancesRight(
    Context& context,
    const String& nameSpace,
    const Gadget_Class& instanceName,
    const PropertySet& propertySet,
    bool keysOnly,
    const MI_Filter* filter)
{
    context.Post(MI_RESULT_NOT_SUPPORTED);
}
```

The implementation of `ReferenceInstancesLeft` is shown below.

```
void Connector_Class_Provider::ReferenceInstancesLeft(
    Context& context,
    const String& nameSpace,
    const Widget_Class& instanceName,
    const PropertySet& propertySet,
    bool keysOnly,
    const MI_Filter* filter)
{
    if (instanceName.Key_value() == 1001)
    {
        // Connector.Left="Gadget.Key=1001","Widget.Key="1003"
        Widget_Class w;
        w.Key_value(1001);
        Gadget_Class g;
        g.Key_value(1003);
        Connector_Class c;
        c.Left_value(w);
        c.Right_value(g);
        context.Post(c);
        context.Post(MI_RESULT_OK);
    }
    else if (instanceName.Key_value() == 1002)
    {
        // Connector.Left="Gadget.Key=1002","Widget.Key="1004"
        Widget_Class w;
        w.Key_value(1002);
        Gadget_Class g;
        g.Key_value(1004);
        Connector_Class c;
        c.Left_value(w);
        c.Right_value(g);
        context.Post(c);
        context.Post(MI_RESULT_OK);
    }
    else
    {
        context.Post(MI_RESULT_NOT_FOUND);
    }
}
```

7.4 Building the provider

To build the provider with the generated `GNUmakefile` just type `make`. This creates a shared library, containing the provider. On Linux, this file will be named '`libxyzconnector.so`'.

7.5 Registering the provider

To register the provider, use the '`omireg`' tool, which creates a registration file ('`xyzconnector.reg`') under the registration directory and copies the provider to the '`lib`' directory. For example:

```
$ omireg libxyzconnector.so
Created /opt/omi/lib/libxyzconnector.so
Created /opt/omi/etc/omiregister/root#cimv2/xyzconnector.reg
```

By default the provider is hosted in the same process as the server. To host the provider in a separate process see the '`--hosting`' option.

Also by default the provider services the '`root/cimv2`' namespace. To change the namespace, see the '`--namespace`' option.

For more help with the '`omireg`' tool, use the '`-h`' option.

The contents of the '`xyzconnector.reg`' file are listed below.

```
# omireg /home/mbrasher/connector/libxyzconnector.so
HOSTING=@inproc@
LIBRARY=xyzconnector
CLASS=XYZ_Connector{XYZ_Widget,XYZ_Gadget}
CLASS=XYZ_Gadget
CLASS=XYZ_Widget
```

It is better to use `omireg` to re-generate these files rather than editing them directly. If you do edit them, you should only need to change the hosting model. The supported hosting models include:

- **@inproc@** – provider runs in same process as server.
- **@requestor@** – provider runs in separate process as the requesters user (the client's authenticated user).
- **USERNAME** – provider runs as this specified user.

7.6 Validating the provider

To validate the provider, use the '`omicli`' tool to send requests to the new providers. The following command enumerates instances of the new '`Widget`' provider.

```
# omicli ei root/cimv2 XYZ_Widget
```

Appendix A Frog Provider Sources

A.1 ‘schema.mof’

```
class XYZ_Frog
{
    [Key] String Name;
    Uint32 Weight;
    String Color;
};
```

A.2 ‘XYZ_Frog.h’

```
/* @migen@ */
/*
**==============================================================================
**
** WARNING: THIS FILE WAS AUTOMATICALLY GENERATED. PLEASE DO NOT EDIT.
**
**==============================================================================
*/
#ifndef _XYZ_Frog_h
#define _XYZ_Frog_h

#include <MI.h>

/*
**==============================================================================
**
** XYZ_Frog [XYZ_Frog]
**
** Keys:
**    Name
**
**==============================================================================
*/

typedef struct _XYZ_Frog
{
    MI_Instance __instance;
    /* XYZ_Frog properties */
    /*KEY*/ MI_ConstStringField Name;
    MI_ConstUint32Field Weight;
    MI_ConstStringField Color;
}
XYZ_Frog;
```

```
typedef struct _XYZ_Frog_Ref
{
    XYZ_Frog* value;
    MI_Boolean exists;
    MI_Uint8 flags;
}
XYZ_Frog_Ref;

typedef struct _XYZ_Frog_ConstRef
{
    MI_CONST XYZ_Frog* value;
    MI_Boolean exists;
    MI_Uint8 flags;
}
XYZ_Frog_ConstRef;

typedef struct _XYZ_Frog_Array
{
    struct _XYZ_Frog** data;
    MI_Uint32 size;
}
XYZ_Frog_Array;

typedef struct _XYZ_Frog_ConstArray
{
    struct _XYZ_Frog MI_CONST* MI_CONST* data;
    MI_Uint32 size;
}
XYZ_Frog_ConstArray;

typedef struct _XYZ_Frog_ArrayRef
{
    XYZ_Frog_Array value;
    MI_Boolean exists;
    MI_Uint8 flags;
}
XYZ_Frog_ArrayRef;

typedef struct _XYZ_Frog_ConstArrayRef
{
    XYZ_Frog_ConstArray value;
    MI_Boolean exists;
    MI_Uint8 flags;
}
XYZ_Frog_ConstArrayRef;

MI_EXTERN_C MI_CONST MI_ClassDecl XYZ_Frog_rtti;
```

```
MI_INLINE MI_Result MI_CALL XYZ_Frog_Construct(
    XYZ_Frog* self,
    MI_Context* context)
{
    return MI_ConstructInstance(context, &XYZ_Frog_rtti,
        (MI_Instance*)&self->__instance);
}

MI_INLINE MI_Result MI_CALL XYZ_Frog_Clone(
    const XYZ_Frog* self,
    XYZ_Frog** newInstance)
{
    return MI_Instance_Clone(
        &self->__instance, (MI_Instance**)newInstance);
}

MI_INLINE MI_Boolean MI_CALL XYZ_Frog_IsA(
    const MI_Instance* self)
{
    MI_Boolean res = MI_FALSE;
    return MI_Instance_IsA(self, &XYZ_Frog_rtti, &res) == MI_RESULT_OK && res;
}

MI_INLINE MI_Result MI_CALL XYZ_Frog_Destruct(XYZ_Frog* self)
{
    return MI_Instance_Destruct(&self->__instance);
}

MI_INLINE MI_Result MI_CALL XYZ_Frog_Delete(XYZ_Frog* self)
{
    return MI_Instance_Delete(&self->__instance);
}

MI_INLINE MI_Result MI_CALL XYZ_Frog_Post(
    const XYZ_Frog* self,
    MI_Context* context)
{
    return MI_PostInstance(context, &self->__instance);
}

MI_INLINE MI_Result MI_CALL XYZ_Frog_Set_Name(
    XYZ_Frog* self,
    const MI_Char* str)
{
    return self->__instance.ft->SetElementAt(
        (MI_Instance*)&self->__instance,
        0,
        (MI_Value*)&str,
        MI_STRING,
        0);
}

MI_INLINE MI_Result MI_CALL XYZ_Frog_SetPtr_Name(
    XYZ_Frog* self,
    const MI_Char* str)
{
    return self->__instance.ft->SetElementAt(
        (MI_Instance*)&self->__instance,
        0,
        (MI_Value*)&str,
        MI_STRING,
        MI_FLAG_BORROW);
}

MI_INLINE MI_Result MI_CALL XYZ_Frog_Clear_Name(
    XYZ_Frog* self)
{
    return self->__instance.ft->ClearElementAt(
        (MI_Instance*)&self->__instance,
        0);
}

MI_INLINE MI_Result MI_CALL XYZ_Frog_Set_Weight(
    XYZ_Frog* self,
    MI_Uint32 x)
{
    ((MI_Uint32Field*)&self->Weight)->value = x;
    ((MI_Uint32Field*)&self->Weight)->exists = 1;
    return MI_RESULT_OK;
}

MI_INLINE MI_Result MI_CALL XYZ_Frog_Clear_Weight(
    XYZ_Frog* self)
{
    memset((void*)&self->Weight, 0, sizeof(self->Weight));
    return MI_RESULT_OK;
}

MI_INLINE MI_Result MI_CALL XYZ_Frog_Set_Color(
    XYZ_Frog* self,
    const MI_Char* str)
{
    return self->__instance.ft->SetElementAt(
        (MI_Instance*)&self->__instance,
        2,
        (MI_Value*)&str,
        MI_STRING,
        0);
}

MI_INLINE MI_Result MI_CALL XYZ_Frog_SetPtr_Color(
    XYZ_Frog* self,
    const MI_Char* str)
{
    return self->__instance.ft->SetElementAt(
        (MI_Instance*)&self->__instance,
        2,
        (MI_Value*)&str,
        MI_STRING,
        MI_FLAG_BORROW);
}

MI_INLINE MI_Result MI_CALL XYZ_Frog_Clear_Color(
    XYZ_Frog* self)
{
    return self->__instance.ft->ClearElementAt(
        (MI_Instance*)&self->__instance,
        2);
}

/*
**==============================================================================
**
** XYZ_Frog provider function prototypes
**
**==============================================================================
*/

/* The developer may optionally define this structure */
typedef struct _XYZ_Frog_Self XYZ_Frog_Self;

MI_EXTERN_C void MI_CALL XYZ_Frog_Load(
    XYZ_Frog_Self** self,
    MI_Module_Self* selfModule,
    MI_Context* context);

MI_EXTERN_C void MI_CALL XYZ_Frog_Unload(
    XYZ_Frog_Self* self,
    MI_Context* context);
```

```
MI_EXTERN_C void MI_CALL XYZ_Frog_EnumerateInstances(
    XYZ_Frog_Self* self,
    MI_Context* context,
    const MI_Char* nameSpace,
    const MI_Char* className,
    const MI_PropertySet* propertySet,
    MI_Boolean keysOnly,
    const MI_Filter* filter);

MI_EXTERN_C void MI_CALL XYZ_Frog_GetInstance(
    XYZ_Frog_Self* self,
    MI_Context* context,
    const MI_Char* nameSpace,
    const MI_Char* className,
    const XYZ_Frog* instanceName,
    const MI_PropertySet* propertySet);

MI_EXTERN_C void MI_CALL XYZ_Frog_CreateInstance(
    XYZ_Frog_Self* self,
    MI_Context* context,
    const MI_Char* nameSpace,
    const MI_Char* className,
    const XYZ_Frog* newInstance);

MI_EXTERN_C void MI_CALL XYZ_Frog_ModifyInstance(
    XYZ_Frog_Self* self,
    MI_Context* context,
    const MI_Char* nameSpace,
    const MI_Char* className,
    const XYZ_Frog* modifiedInstance,
    const MI_PropertySet* propertySet);

MI_EXTERN_C void MI_CALL XYZ_Frog_DeleteInstance(
    XYZ_Frog_Self* self,
    MI_Context* context,
    const MI_Char* nameSpace,
    const MI_Char* className,
    const XYZ_Frog* instanceName);


/*
**==============================================================================
**
** XYZ_Frog_Class
**
**==============================================================================
*/
```

```
#ifdef __cplusplus
# include <micxx/micxx.h>

MI_BEGIN_NAMESPACE

class XYZ_Frog_Class : public Instance
{
public:

    typedef XYZ_Frog Self;

    XYZ_Frog_Class() :
        Instance(&XYZ_Frog_rtti)
    {
    }

    XYZ_Frog_Class(
        const XYZ_Frog* instanceName,
        bool keysOnly) :
        Instance(
            &XYZ_Frog_rtti,
            &instanceName->__instance,
            keysOnly)
    {
    }

    XYZ_Frog_Class(
        const MI_ClassDecl* clDecl,
        const MI_Instance* instance,
        bool keysOnly) :
        Instance(clDecl, instance, keysOnly)
    {
    }

    XYZ_Frog_Class(
        const MI_ClassDecl* clDecl) :
        Instance(clDecl)
    {
    }

    XYZ_Frog_Class& operator=(
        const XYZ_Frog_Class& x)
    {
        CopyRef(x);
        return *this;
    }
```

```
XYZ_Frog_Class(
    const XYZ_Frog_Class& x) :
    Instance(x)
{
}

static const MI_ClassDecl* GetClassDecl()
{
    return &XYZ_Frog_rtti;
}

//
// XYZ_Frog_Class.Name
//

const Field<String>& Name() const
{
    const size_t n = offsetof(Self, Name);
    return GetField<String>(n);
}

void Name(const Field<String>& x)
{
    const size_t n = offsetof(Self, Name);
    GetField<String>(n) = x;
}

const String& Name_value() const
{
    const size_t n = offsetof(Self, Name);
    return GetField<String>(n).value;
}

void Name_value(const String& x)
{
    const size_t n = offsetof(Self, Name);
    return GetField<String>(n).Set(x);
}

bool Name_exists() const
{
    const size_t n = offsetof(Self, Name);
    return GetField<String>(n).exists ? true : false;
}

void Name_clear()
```

```
{
    const size_t n = offsetof(Self, Name);
    GetField<String>(n).Clear();
}

//
// XYZ_Frog_Class.Weight
//

const Field<Uint32>& Weight() const
{
    const size_t n = offsetof(Self, Weight);
    return GetField<Uint32>(n);
}

void Weight(const Field<Uint32>& x)
{
    const size_t n = offsetof(Self, Weight);
    GetField<Uint32>(n) = x;
}

const Uint32& Weight_value() const
{
    const size_t n = offsetof(Self, Weight);
    return GetField<Uint32>(n).value;
}

void Weight_value(const Uint32& x)
{
    const size_t n = offsetof(Self, Weight);
    return GetField<Uint32>(n).Set(x);
}

bool Weight_exists() const
{
    const size_t n = offsetof(Self, Weight);
    return GetField<Uint32>(n).exists ? true : false;
}

void Weight_clear()
{
    const size_t n = offsetof(Self, Weight);
    GetField<Uint32>(n).Clear();
}

//
// XYZ_Frog_Class.Color
```

```
    //

    const Field<String>& Color() const
    {
        const size_t n = offsetof(Self, Color);
        return GetField<String>(n);
    }

    void Color(const Field<String>& x)
    {
        const size_t n = offsetof(Self, Color);
        GetField<String>(n) = x;
    }

    const String& Color_value() const
    {
        const size_t n = offsetof(Self, Color);
        return GetField<String>(n).value;
    }

    void Color_value(const String& x)
    {
        const size_t n = offsetof(Self, Color);
        return GetField<String>(n).Set(x);
    }

    bool Color_exists() const
    {
        const size_t n = offsetof(Self, Color);
        return GetField<String>(n).exists ? true : false;
    }

    void Color_clear()
    {
        const size_t n = offsetof(Self, Color);
        GetField<String>(n).Clear();
    }
};

typedef Array<XYZ_Frog_Class> XYZ_Frog_ClassA;

MI_END_NAMESPACE

#endif /* __cplusplus */

#endif /* _XYZ_Frog_h */
```

A.3 'XYZ_Frog_Class_Provider.h'

```
/* @migen@ */
#ifndef _XYZ_Frog_Class_Provider_h
#define _XYZ_Frog_Class_Provider_h

#include "XYZ_Frog.h"
#ifdef __cplusplus
# include <micxx/micxx.h>
# include "module.h"

MI_BEGIN_NAMESPACE

/*
**==============================================================================
**
** XYZ_Frog provider class declaration
**
**==============================================================================
*/

class XYZ_Frog_Class_Provider
{
/* @MIGEN.BEGIN@ CAUTION: PLEASE DO NOT EDIT OR DELETE THIS LINE. */
private:
    Module* m_Module;

public:
    XYZ_Frog_Class_Provider(
        Module* module);

    ~XYZ_Frog_Class_Provider();

    void Load(
        Context& context);

    void Unload(
        Context& context);

    void EnumerateInstances(
        Context& context,
        const String& nameSpace,
        const PropertySet& propertySet,
        bool keysOnly,
        const MI_Filter* filter);

    void GetInstance(
        Context& context,
        const String& nameSpace,
        const XYZ_Frog_Class& instance,
        const PropertySet& propertySet);

    void CreateInstance(
        Context& context,
        const String& nameSpace,
        const XYZ_Frog_Class& newInstance);

    void ModifyInstance(
        Context& context,
        const String& nameSpace,
        const XYZ_Frog_Class& modifiedInstance,
        const PropertySet& propertySet);

    void DeleteInstance(
        Context& context,
        const String& nameSpace,
        const XYZ_Frog_Class& instance);

/* @MIGEN.END@ CAUTION: PLEASE DO NOT EDIT OR DELETE THIS LINE. */
};

MI_END_NAMESPACE

#endif /* __cplusplus */

#endif /* _XYZ_Frog_Class_Provider_h */
```

A.4 'XYZ_Frog_Class_Provider.cpp'

```
/* @migen@ */
#include <MI.h>
#include "XYZ_Frog_Class_Provider.h"

MI_BEGIN_NAMESPACE

XYZ_Frog_Class_Provider::XYZ_Frog_Class_Provider(
    Module* module) :
    m_Module(module)
{
}

XYZ_Frog_Class_Provider::~XYZ_Frog_Class_Provider()
{
}

void XYZ_Frog_Class_Provider::Load(
        Context& context)
{
    context.Post(MI_RESULT_OK);
}

void XYZ_Frog_Class_Provider::Unload(
        Context& context)
{
    context.Post(MI_RESULT_OK);
}

void XYZ_Frog_Class_Provider::EnumerateInstances(
    Context& context,
    const String& nameSpace,
    const PropertySet& propertySet,
    bool keysOnly,
    const MI_Filter* filter)
{
    XYZ_Frog_Class frog1;
    frog1.Name_value("Fred");
    frog1.Weight_value(55);
    frog1.Color_value("Green");
    context.Post(frog1);

    XYZ_Frog_Class frog2;
    frog2.Name_value("Sam");
    frog2.Weight_value(65);
    frog2.Color_value("Blue");
    context.Post(frog2);

    context.Post(MI_RESULT_OK);
}

void XYZ_Frog_Class_Provider::GetInstance(
    Context& context,
    const String& nameSpace,
    const XYZ_Frog_Class& instanceName,
    const PropertySet& propertySet)
{
    context.Post(MI_RESULT_NOT_SUPPORTED);
}

void XYZ_Frog_Class_Provider::CreateInstance(
    Context& context,
    const String& nameSpace,
    const XYZ_Frog_Class& newInstance)
{
    context.Post(MI_RESULT_NOT_SUPPORTED);
}

void XYZ_Frog_Class_Provider::ModifyInstance(
    Context& context,
    const String& nameSpace,
    const XYZ_Frog_Class& modifiedInstance,
    const PropertySet& propertySet)
{
    context.Post(MI_RESULT_NOT_SUPPORTED);
}

void XYZ_Frog_Class_Provider::DeleteInstance(
    Context& context,
    const String& nameSpace,
    const XYZ_Frog_Class& instanceName)
{
    context.Post(MI_RESULT_NOT_SUPPORTED);
}


MI_END_NAMESPACE
```

A.5 'module.h'

```
#ifndef _Module_t_h
#define _Module_t_h

#include <MI.h>
#include <micxx/micxx.h>

MI_BEGIN_NAMESPACE

/*  instance of this class is automatically created when library is loaded;
    it's a convenient place to store global data associated with the module */
class Module
{
public:
    Module();
    ~Module();

};

MI_END_NAMESPACE
```

```
#endif /* _Module_t_h */
```

A.6 ‘module.cpp’

```
#include <MI.h>
#include "module.h"

MI_BEGIN_NAMESPACE

Module::Module()
{
}

Module::~Module()
{
}

MI_END_NAMESPACE
```

A.7 ‘schema.c’

```
/* @migen@ */
/*
**==============================================================================■
**
** WARNING: THIS FILE WAS AUTOMATICALLY GENERATED. PLEASE DO NOT EDIT.
**
**==============================================================================■
*/
#include <ctype.h>
#include <MI.h>
#include "XYZ_Frog.h"

/*
**==============================================================================■
**
** Schema Declaration
**
**==============================================================================■
*/

extern MI_SchemaDecl schemaDecl;

/*
**==============================================================================■
**
```

```
** Qualifier declarations
**
**==============================================================================
*/

/*
**==============================================================================
**
** XYZ_Frog
**
**==============================================================================
*/

/* property XYZ_Frog.Name */
static MI_CONST MI_PropertyDecl XYZ_Frog_Name_prop =
{
    MI_FLAG_PROPERTY|MI_FLAG_KEY, /* flags */
    0x006E6504, /* code */
    MI_T("Name"), /* name */
    NULL, /* qualifiers */
    0, /* numQualifiers */
    MI_STRING, /* type */
    NULL, /* className */
    0, /* subscript */
    offsetof(XYZ_Frog, Name), /* offset */
    MI_T("XYZ_Frog"), /* origin */
    MI_T("XYZ_Frog"), /* propagator */
    NULL,
};

/* property XYZ_Frog.Weight */
static MI_CONST MI_PropertyDecl XYZ_Frog_Weight_prop =
{
    MI_FLAG_PROPERTY, /* flags */
    0x00777406, /* code */
    MI_T("Weight"), /* name */
    NULL, /* qualifiers */
    0, /* numQualifiers */
    MI_UINT32, /* type */
    NULL, /* className */
    0, /* subscript */
    offsetof(XYZ_Frog, Weight), /* offset */
    MI_T("XYZ_Frog"), /* origin */
    MI_T("XYZ_Frog"), /* propagator */
    NULL,
};
```

```
/* property XYZ_Frog.Color */
static MI_CONST MI_PropertyDecl XYZ_Frog_Color_prop =
{
    MI_FLAG_PROPERTY, /* flags */
    0x00637205, /* code */
    MI_T("Color"), /* name */
    NULL, /* qualifiers */
    0, /* numQualifiers */
    MI_STRING, /* type */
    NULL, /* className */
    0, /* subscript */
    offsetof(XYZ_Frog, Color), /* offset */
    MI_T("XYZ_Frog"), /* origin */
    MI_T("XYZ_Frog"), /* propagator */
    NULL,
};

static MI_PropertyDecl MI_CONST* MI_CONST XYZ_Frog_props[] =
{
    &XYZ_Frog_Name_prop,
    &XYZ_Frog_Weight_prop,
    &XYZ_Frog_Color_prop,
};

static MI_CONST MI_ProviderFT XYZ_Frog_funcs =
{
  (MI_ProviderFT_Load)XYZ_Frog_Load,
  (MI_ProviderFT_Unload)XYZ_Frog_Unload,
  (MI_ProviderFT_GetInstance)XYZ_Frog_GetInstance,
  (MI_ProviderFT_EnumerateInstances)XYZ_Frog_EnumerateInstances,
  (MI_ProviderFT_CreateInstance)XYZ_Frog_CreateInstance,
  (MI_ProviderFT_ModifyInstance)XYZ_Frog_ModifyInstance,
  (MI_ProviderFT_DeleteInstance)XYZ_Frog_DeleteInstance,
  (MI_ProviderFT_AssociatorInstances)NULL,
  (MI_ProviderFT_ReferenceInstances)NULL,
  (MI_ProviderFT_EnableIndications)NULL,
  (MI_ProviderFT_DisableIndications)NULL,
  (MI_ProviderFT_Subscribe)NULL,
  (MI_ProviderFT_Unsubscribe)NULL,
  (MI_ProviderFT_Invoke)NULL,
};

/* class XYZ_Frog */
MI_CONST MI_ClassDecl XYZ_Frog_rtti =
{
    MI_FLAG_CLASS, /* flags */
    0x00786708, /* code */
    MI_T("XYZ_Frog"), /* name */
    NULL, /* qualifiers */
    0, /* numQualifiers */
    XYZ_Frog_props, /* properties */
    MI_COUNT(XYZ_Frog_props), /* numProperties */
    sizeof(XYZ_Frog), /* size */
    NULL, /* superClass */
    NULL, /* superClassDecl */
    NULL, /* methods */
    0, /* numMethods */
    &schemaDecl, /* schema */
    &XYZ_Frog_funcs, /* functions */
};

/*
**==============================================================================
**
** __mi_server
**
**==============================================================================
*/

MI_Server* __mi_server;
/*
**==============================================================================
**
** Schema
**
**==============================================================================
*/

static MI_ClassDecl MI_CONST* MI_CONST classes[] =
{
    &XYZ_Frog_rtti,
};

MI_SchemaDecl schemaDecl =
{
    NULL, /* qualifierDecls */
    0, /* numQualifierDecls */
    classes, /* classDecls */
    MI_COUNT(classes), /* classDecls */
};

/*
**==============================================================================
**
```

```
** MI_Server Methods
**
**==============================================================================
*/

MI_Result MI_CALL MI_Server_GetVersion(
    MI_Uint32* version){
    return __mi_server->serverFT->GetVersion(version);
}

MI_Result MI_CALL MI_Server_GetSystemName(
    const MI_Char** systemName)
{
    return __mi_server->serverFT->GetSystemName(systemName);
}
```

A.8 'stubs.cpp'

```
/* @migen@ */
/*
**==============================================================================
**
** WARNING: THIS FILE WAS AUTOMATICALLY GENERATED. PLEASE DO NOT EDIT.
**
**==============================================================================
*/
#include <MI.h>
#include "module.h"
#include "XYZ_Frog_Class_Provider.h"

using namespace mi;

MI_EXTERN_C void MI_CALL XYZ_Frog_Load(
    XYZ_Frog_Self** self,
    MI_Module_Self* selfModule,
    MI_Context* context)
{
    MI_Result r = MI_RESULT_OK;
    Context ctx(context, &r);
    XYZ_Frog_Class_Provider* prov = new XYZ_Frog_Class_Provider((Module*)selfModule

    prov->Load(ctx);
    if (MI_RESULT_OK != r)
    {
        delete prov;
        MI_PostResult(context, r);
```

```
            return;
        }
        *self = (XYZ_Frog_Self*)prov;
        MI_PostResult(context, MI_RESULT_OK);
    }

    MI_EXTERN_C void MI_CALL XYZ_Frog_Unload(
        XYZ_Frog_Self* self,
        MI_Context* context)
    {
        MI_Result r = MI_RESULT_OK;
        Context ctx(context, &r);
        XYZ_Frog_Class_Provider* prov = (XYZ_Frog_Class_Provider*)self;

        prov->Unload(ctx);
        delete ((XYZ_Frog_Class_Provider*)self);
        MI_PostResult(context, r);
    }

    MI_EXTERN_C void MI_CALL XYZ_Frog_EnumerateInstances(
        XYZ_Frog_Self* self,
        MI_Context* context,
        const MI_Char* nameSpace,
        const MI_Char* className,
        const MI_PropertySet* propertySet,
        MI_Boolean keysOnly,
        const MI_Filter* filter)
    {
        XYZ_Frog_Class_Provider* cxxSelf =((XYZ_Frog_Class_Provider*)self);
        Context  cxxContext(context);

        cxxSelf->EnumerateInstances(
            cxxContext,
            nameSpace,
            __PropertySet(propertySet),
            __bool(keysOnly),
            filter);
    }

    MI_EXTERN_C void MI_CALL XYZ_Frog_GetInstance(
        XYZ_Frog_Self* self,
        MI_Context* context,
        const MI_Char* nameSpace,
        const MI_Char* className,
        const XYZ_Frog* instanceName,
        const MI_PropertySet* propertySet)
    {
```

```
    XYZ_Frog_Class_Provider* cxxSelf =((XYZ_Frog_Class_Provider*)self);
    Context  cxxContext(context);
    XYZ_Frog_Class cxxInstanceName(instanceName, true);

    cxxSelf->GetInstance(
        cxxContext,
        nameSpace,
        cxxInstanceName,
        __PropertySet(propertySet));
}

MI_EXTERN_C void MI_CALL XYZ_Frog_CreateInstance(
    XYZ_Frog_Self* self,
    MI_Context* context,
    const MI_Char* nameSpace,
    const MI_Char* className,
    const XYZ_Frog* newInstance)
{
    XYZ_Frog_Class_Provider* cxxSelf =((XYZ_Frog_Class_Provider*)self);
    Context  cxxContext(context);
    XYZ_Frog_Class cxxNewInstance(newInstance, false);

    cxxSelf->CreateInstance(cxxContext, nameSpace, cxxNewInstance);
}

MI_EXTERN_C void MI_CALL XYZ_Frog_ModifyInstance(
    XYZ_Frog_Self* self,
    MI_Context* context,
    const MI_Char* nameSpace,
    const MI_Char* className,
    const XYZ_Frog* modifiedInstance,
    const MI_PropertySet* propertySet)
{
    XYZ_Frog_Class_Provider* cxxSelf =((XYZ_Frog_Class_Provider*)self);
    Context  cxxContext(context);
    XYZ_Frog_Class cxxModifiedInstance(modifiedInstance, false);

    cxxSelf->ModifyInstance(
        cxxContext,
        nameSpace,
        cxxModifiedInstance,
        __PropertySet(propertySet));
}

MI_EXTERN_C void MI_CALL XYZ_Frog_DeleteInstance(
    XYZ_Frog_Self* self,
    MI_Context* context,
    const MI_Char* nameSpace,
    const MI_Char* className,
    const XYZ_Frog* instanceName)
{
    XYZ_Frog_Class_Provider* cxxSelf =((XYZ_Frog_Class_Provider*)self);
    Context  cxxContext(context);
    XYZ_Frog_Class cxxInstanceName(instanceName, true);

    cxxSelf->DeleteInstance(cxxContext, nameSpace, cxxInstanceName);
}


MI_EXTERN_C MI_SchemaDecl schemaDecl;

void MI_CALL Load(MI_Module_Self** self, struct _MI_Context* context)
{
    *self = (MI_Module_Self*)new Module;
}

void MI_CALL Unload(MI_Module_Self* self, struct _MI_Context* context)
{
    Module* module = (Module*)self;
    delete module;
}

MI_EXTERN_C MI_EXPORT MI_Module* MI_MAIN_CALL MI_Main(MI_Server* server)
{
    /* WARNING: THIS FUNCTION AUTOMATICALLY GENERATED. PLEASE DO NOT EDIT. */
    extern MI_Server* __mi_server;
    static MI_Module module;
    __mi_server = server;
    module.flags |= MI_MODULE_FLAG_STANDARD_QUALIFIERS;
    module.flags |= MI_MODULE_FLAG_CPLUSPLUS;
    module.charSize = sizeof(MI_Char);
    module.version = MI_VERSION;
    module.generatorVersion = MI_MAKE_VERSION(1,0,0);
    module.schemaDecl = &schemaDecl;
    module.Load = Load;
    module.Unload = Unload;
    return &module;
}
```

A.9 ‘GNUmakefile’

```
HOST=$(shell hostname)
include ../../../../output/$(shell hostname)/omi.mak
```

```
PROVIDER = frog
SOURCES = $(wildcard *.c *.cpp)
CLASSES = XYZ_Frog

$(LIBRARY): $(OBJECTS)
        $(CXX) $(CXXSHLIBFLAGS) $(OBJECTS) -o $(LIBRARY) $(CXXLIBS)

%.o: %.c
        $(CC) -c $(CFLAGS) $(INCLUDES) $< $(CLIBS) -o $@

%.o: %.cpp
        $(CXX) -c $(CXXFLAGS) $(INCLUDES) $< -o $@

reg:
        $(BINDIR)/omireg $(CURDIR)/$(LIBRARY)

gen:
        $(BINDIR)/omigen --cpp -m frog schema.mof XYZ_Frog

clean:
        rm -f $(LIBRARY) $(OBJECTS) $(PROVIDER).reg
```

Appendix B Asynchronous Enumerate Instances Client Example

B.1 ‘AsyncEnum.cpp’

```
#include <cstdio>
#include <omiclient/client.h>

#define T MI_T

using namespace mi;

class MyHandler : public Handler
{
public:

    MyHandler() : done(false)
    {
    }

    virtual void HandleConnect()
    {
        printf("==== MyHandler::HandleConnect()\n");
    }

    virtual void HandleNoOp(Uint64 msgID)
    {
        printf("==== MyHandler::HandleNoOp()\n");
    }

    virtual void HandleConnectFailed()
    {
        printf("==== MyHandler::HandleConnectFailed()\n");

        // Handler error!
        done = true;
    }

    virtual void HandleDisconnect()
    {
        printf("==== MyHandler::HandleDisconnect()\n");
        done = true;
    }

    virtual void HandleInstance(Uint64 msgID, const DInstance& instance)
    {
        printf("==== MyHandler::HandleInstance()\n");
        instance.Print();
    }

    virtual void HandleResult(Uint64 msgID, MI_Result result)
    {
        printf("==== MyHandler::HandleResult()\n");
        done = true;
    }

    bool done;
};

int main(int argc, const char* argv[])
{
    int r = 0;

    // Create handler:
    MyHandler* handler = new MyHandler;

    // Construct client:
    Client client(handler);

    String locator;
    String username;
    String password;

    if (!client.ConnectAsync(locator, username, password))
    {
        // Handle error!
    }

    const String nameSpace = "root/omi";
    const String className = "OMI_Identify";
    const bool deep = true;
    Uint64 msgID;

    if (!client.EnumerateInstancesAsync(nameSpace, className, deep, msgID))
    {
        // Handle error!
    }

    // Wait here for 5 seconds for operation to finish.
    while (!handler->done)
    {
        client.Run(1000);
    }
```

```
        return r;
    }
```