Building and Installing:

Prerequisites:

OMI Script Provider 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
- python2.7-dev

OMI and OMIScriptProvider build and install. This will build, link, modify conf files, and install libraries and binaries. In the build-omi-script-provider path:

```
> cd omi/Unix
> ./configure --dev
> make
> make install
> cd ../../scriptprovider
> make
> sudo make install
```

Developing a Python Provider:

This is a very quick overview for how to develop an OMI Provider in Python. It shows the minimum steps for building a simple Python provider. More details will be included in the next section.

Defining the "schema.mof":

First, create a directory for the provider under the lib directory. The location of the file is important and will be discussed later in this document.

```
> cd ../omi/Unix/output/lib
> mkdir XYZ_Frog
> cd XYZ_Frog
```

Then, in that directory, define the class schema shown in the "schema.mof" file below.

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

Generate the Provider Sources:

Next, we generate the provider sources using the command below:

```
> omigen_py schema.mof XYZ_Frog
Creating schema.py
Creating mi_main.py
```

The omigen_py tool will generate the files schema.py and mi_main.py.

schema.py:

```
# @migen@
##==============================================================================
=
##
## WARNING: THIS FILE WAS AUTOMATICALLY GENERATED. PLEASE DO NOT EDIT.
##
##==============================================================================
=
import omi
from omi import *

##==============================================================================
=
##
## Qualifier declarations
##
##==============================================================================
=
qualifierDecls = [
    ]

##==============================================================================
=
##
## XYZ_Frog
##
##==============================================================================
=
XYZ_Frog_quals = [
    ]

XYZ_Frog_Name_quals = [
    ]

XYZ_Frog_Name_prop = MI_PropertyDecl (
    MI_FLAG_PROPERTY | MI_FLAG_KEY, # flags
    'Name', # name
    XYZ_Frog_Name_quals, # qualifiers
    MI_STRING, # type
    None, # className
    'XYZ_Frog', # origin
    'XYZ_Frog', # propagator
    None # value
    )

XYZ_Frog_Weight_quals = [
    ]

XYZ_Frog_Weight_prop = MI_PropertyDecl (
    MI_FLAG_PROPERTY, # flags
    'Weight', # name
```

```
    XYZ_Frog_Weight_quals, # qualifiers
    MI_UINT32, # type
    None, # className
    'XYZ_Frog', # origin
    'XYZ_Frog', # propagator
    None # value
    )

XYZ_Frog_Color_quals = [
    ]

XYZ_Frog_Color_prop = MI_PropertyDecl (
    MI_FLAG_PROPERTY, # flags
    'Color', # name
    XYZ_Frog_Color_quals, # qualifiers
    MI_STRING, # type
    None, # className
    'XYZ_Frog', # origin
    'XYZ_Frog', # propagator
    None # value
    )

XYZ_Frog_properties = [
    XYZ_Frog_Name_prop,
    XYZ_Frog_Weight_prop,
    XYZ_Frog_Color_prop,
    ]

XYZ_Frog_methods = [
    ]

XYZ_Frog_functions = MI_FunctionTable (
    'XYZ_Frog_Load',
    'XYZ_Frog_Unload',
    'XYZ_Frog_GetInstance',
    'XYZ_Frog_EnumerateInstances',
    'XYZ_Frog_CreateInstance',
    'XYZ_Frog_ModifyInstance',
    'XYZ_Frog_DeleteInstance',
    None,
    None,
    None,
    None,
    None,
    None,
    None
    )

XYZ_Frog_class = MI_ClassDecl (
    MI_FLAG_CLASS, # flags
    'XYZ_Frog', # name
    XYZ_Frog_quals, # qualifiers
    XYZ_Frog_properties, # properties
    None, # superclass
    XYZ_Frog_methods, # method
    XYZ_Frog_functions, # FunctionTable
    None # owningclass
    )

classDecls = [
    XYZ_Frog_class,
    ]
```

```
schema = MI_SchemaDecl (
    qualifierDecls,
    classDecls
    )
```

mi_main.py:

```
import omi
from omi import *

import schema


def Load (module, context):
    context.PostResult (MI_RESULT_OK)


def Unload (module, context):
    context.PostResult (MI_RESULT_OK)


def XYZ_Frog_Load (
    module, context):
    context.PostResult (MI_RESULT_OK)


def XYZ_Frog_Unload (
    context):
    context.PostResult (MI_RESULT_OK)


def XYZ_Frog_EnumerateInstances (
    context, nameSpace, className, propertySet, keysOnly):
    context.PostResult (MI_RESULT_NOT_SUPPORTED)


def XYZ_Frog_GetInstance (
    context, nameSpace, className, instanceName, propertySet):
    context.PostResult (MI_RESULT_NOT_SUPPORTED)


def XYZ_Frog_CreateInstance (
    context, nameSpace, className, instance):
    context.PostResult (MI_RESULT_NOT_SUPPORTED)


def XYZ_Frog_ModifyInstance (
    context, nameSpace, className, instance, propertySet):
    context.PostResult (MI_RESULT_NOT_SUPPORTED)


def XYZ_Frog_DeleteInstance (
    context, nameSpace, className, instanceName):
    context.PostResult (MI_RESULT_NOT_SUPPORTED)


def mi_main ():
    r = MI_Module (schema.schema, Load, Unload)
    return r
```

Implementing the "EnumerateInstances" Method:

Make the following changes to the XYZ_Frog_EnumerateInstances method in mi_main.py:

```
def XYZ_Frog_EnumerateInstances (
    context, nameSpace, className, propertySet, keysOnly):
    frog1 = context.NewInstance ('XYZ_Frog')
    frog1.SetValue ('Name', MI_String ('Fred'))
    frog1.SetValue ('Weight', MI_Uint32 (55))
    frog1.SetValue ('Color', MI_String ('Green'))
    context.PostInstance (frog1)
    frog2 = context.NewInstance ('XYZ_Frog')
    frog2.SetValue ('Name', MI_String ('Sam'))
    frog2.SetValue ('Weight', MI_Uint32 (65))
    frog2.SetValue ('Color', MI_String ('Blue'))
    context.PostInstance (frog2)
    context.PostResult (MI_RESULT_NOT_SUPPORTED)
    context.PostResult (MI_RESULT_OK)
```

Registering the Provider:

Next, register the provider as follows:

```
> ../../bin/omireg --Python XYZ_Frog
Created /opt/omi/etc/omiregister/root-cimv2/XYZ_Frog.reg
```

The omiserver needs to reload the registry for this change to take effect. This can be acomplished by restarting the omiserver, or by running the following command:

```
> sudo ../../bin/omiserver -r
../../bin/omiserver: refreshed server
```

Testing the Provider:

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

```
> ../../bin/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
}
```

Going Further:

This provides a brief overview of the provider development process. The next section contains a deeper discussion.

Provider File Location and OMI Registry:

This is a more technical explanation of how the OMI registry works to associate providers to OMI queries. This outlines how the OMI registry works, and how the OMI finds the libraries to execute.

OMI Register:

The OMI Server associates Classes with library files through a collection of registry files. When the OMI Server starts, it reads the registry files and creates links between namespaces and classes, to the library files that handle them. The simplest registry files contain a value for "LIBRARY" and a value for "CLASS". The registry file associates a provider (LIBRARY) with the classes (CLASS) that is has handlers for. The registry files are stored in a predetermined location and have a ".reg" extension. Depending on which options the OMI Server was built with, and particular environment variables, the registry files could be in a number of places. If OMI Server was installed with a package, the registry files are placed in "/etc/opt/omi/conf/omiregister/". If OMI Server was configured with the "--dev" option and built from source, the registry files are placed at "[omi-source-root]/omi/Unix/output/etc/omiregister/". There are a number of directories within the omiregister directory. Each of these subdirectories represents a namespace. And each ".reg" file represents a single provider.

Provider File Location:

The OMI Server has a predesignated root location where it searches for libraries. If OMI server was installed with a package, the root library location is "/opt/omi/lib/". If OMI Server was configured with the "--dev" option and built from source, the root library location is "[omi-source-root]"/omi/Unix/output/lib/".

OMI Reg files:

Each provider is associated to the OMI Server through a ".reg" file. For the XYZ_Frog example above, the call to omireg generates the file "[OMI Register location]/root-cimv2/XYZ_Frog.reg".

```
# omireg XYZ_Frog
INTERPRETER=python2.7
STARTUP=client.py
LIBRARY=XYZ_Frog
CLASS=XYZ_Frog
```

This file, and its location, associates queries with the provider that handles them. For starters, the path "root-cimv2" associates the provider with the "root/cimv2" namespace. The line "INTERPRETER=python2.7" tells the OMI Server that this provider runs in the python2.7 interpreter. The line "STARTUP=client.py" tells the OMI Server that client.py (a system file) is the startup file for this provider. The line "LIBRARY=XYZ_Frog" tells the OMI Server where to find the script files for this

provider. (The LIBRARY parameter will be discussed more thoroughly in the next section.) The line "CLASS=XYZ_Frog" tells the OMI Server which classes are served by this provider.

Python Files for the OMI Script Provider:

The OMI Script Provider uses two particular files, mi_main.py and schema.py, for each provider. The helper application, omigen_py creates boilerplate code for these files from a *.mof file. The schema.py file defines the metadata that is required by the OMI Server to dynamically handle providers. It defines the structures that are used, and details which methods are implemented. Details about the schema.py file are outside of the scope of this document. The mi_main.py file contains some startup and initialization code, and provides the callbacks that the OMI Server uses to execute the provider. The author of the provider may edit the body of these methods to implement the provider. The OMI Server determines where to find these files with the LIBRARY parameter of the ".reg" file. The value of this parameter is the path to the mi_main.py and schema.py files for this provider. This path is relative to the Provider File location (see above). In the XYZ_Frog example, the XYZ_Frog.reg file tells the OMI Server to look for mi_main.py and schema.py at "[omi-source-root]/omi/Unix/output/lib/XYZ_Frog/".