Developing WBEM Clients Using Python

Tim Potter

Hewlett-Packard Company

tpot@hp.com

Quotes

"The best way to learn about something is to teach it."

- *Unknown*

Quotes (2)

"The best way to learn about something is to write a program to do it."

- *Me*

About PyWBEM

PyWBEM was written to answer the following questions:

1. What is this WBEM thing anyway?

2. How can I do useful things with it?

About PyWBEM (2)

- PyWBEM is a pure-Python CIM client
- Runs on Unix/Linux and WIN32
- ~100 downloads since first release
- `http://pywbem.sourceforge.net/`

About PyWBEM (2)

- PyWBEM != PiWBEM

- PiWBEM is another open-source client framework written in Python

- Written by Chris Hobbs from Nortel

About Python

- Python is a popular, open source, object-oriented scripting language

- Clear and elegant syntax

- Good free tutorial on Python at

  `http://diveintopython.org/`

PyWBEM Audience

- Developers
- Testers
- End-users
- “Hackers”

Installing PyWBEM

- For Unix, use standard Python module install process:

```
$ python setup.py build
# python setup.py install
```

- For WIN32, run the executable installer

Example code

- Lots of example code and applications available at the PyWBEM home page

`http://pywbem.sf.net/examples/`

Connections

- Call *`WBEMConnection()` function to make a connection*

```
import pywbem

conn = pywbem.WBEMConnection(
  'https://server.com',
  ('username', 'password'))
```

Instance API

- *Enumeration methods on connection object return a list of CIM objects*

```
result = conn.EnumerateInstances(
    'CIM_OperatingSystem')

result =
  conn.EnumerateInstanceNames(
    'CIM_OperatingSystem')
```

Instance API (2)

- All other instance provider methods available
  - `conn.GetInstance(...)`
  - `conn.DeleteInstance(...)`
  - `conn.ModifyInstance(...)`
  - `conn.CreateInstance(...)`

Instance API (3)

- Parameter passing done using Python keyword arguments

```
conn.EnumerateInstanceNames(
    'CIM_Foo',
    LocalNamespacePath =
      'root/PG_InterOp')
```

Object Model

- Usability *extremely* important

- PyWBEM uses built-in Python data types whenever possible

- Most common use cases should only involve `CIMInstanceName` and `CIMInstance` objects

Object Model (2)

`CIMInstanceName` object

- Access property names through dictionary interface

```
>>> print obj['CreationClassName']
'PG_OperatingSystem'
```

Object Model (3)

`CIMInstance` object

- Same idea as instance name, except for assigning integer property values

```
obj['Status'] = pywbem.Uint16(2)
```

Object Model (4)

- All objects have a `__str__` method

```
>>> print str(obj)
>>>
PG_OperatingSystem.CreationClassName=”CIM_
OperatingSystem”,Name=”Red Hat Enterprise
Linux
AS”,CSCreationClassName=”CIM_UnitaryComput
erSystem”,CSName=”deckchair”
```

Errors and Exceptions

- CIM errors are translated into Python exceptions

```
try:

  conn.DeleteInstance(...)

except pywbem.CIMError, arg:
  print 'CIM error %d: %s' %
    (arg[0], arg[1])
```

Method API

- `InvokeMethod()` function returns a tuple of return value, and output parameters

```
result, output_params =
  conn.InvokeMethod
    ('Foo',
     object,
     param = value, ...)
```

Associations API

- Slightly trickier due to complexity of CIM association API

- Return a list of class names associated with the CIM_ComputerSystem class

```
names = conn.AssociatorNames(
   'CIM_ComputerSystem')
```

Associations API (2)

- Return a list of instance names associated with a `CIM_ComputerSystem` instance

```
cs = conn.EnumerateInstanceNames(
  'CIM_ComputerSystem')[0]

names = conn.AssociatorNames(cs)
```

Associations API (3)

- All association provider methods available
  - `conn.AssociatorNames(...)`
  - `conn.ReferenceNames(...)`
  - `conn.Associators(...)`
  - `conn.References(...)`

Class API

- Enumerations return lists of class names and class objects

```
result = conn.EnumerateClassNames()
result = conn.EnumerateClasses()
```

Class API (2)

- Other class functions available but not well supported in version 0.4
  - `conn.GetClass(...)`
  - `conn.DeleteClass(...)`
  - `conn.CreateClass(...)`
  - `conn.ModifyClass(...)`

Qualifiers API

- Not well supported in PyWBEM 0.4

Indications API

- Nothing implemented yet in PyWBEM 0.4
- Useful tasks would be
  - Framework for receipt of indications
  - Subscription management
  - Parsing of received indications

XML Generation

- WBEM requests requires generation of small bits of XML

- PyWBEM has a library to generate all XML elements in CIM DTD

- API is simple mapping of DTD structure to Python

XML Generation (2)

```
from pywbem import *

mc = METHODCALL('Foo',
  CLASSNAME('CIM_Foo'))

<METHODCALL NAME=”Foo”>
  <CLASSNAME NAME=”CIM_Foo”/>
</METHODCALL>
```

XML Generation (3)

```
CIM(
  MESSAGE(
    SIMPLEREQ(mc), 1001, '1.0'),
  '2.0', '2.0')

<CIM CIMVERSION=”2.0”
     DTDVERSION=”2.0”>
  <MESSAGE ID=”1001” PROTOCOL=”1.0”>
    <SIMPLEREQ>...</SIMPLEREQ>
  </MESSAGE>
</CIM>
```

XML Parsing

- WBEM replies require parsing of large bits of XML

- PyWBEM has library to parse all XML elements in CIM DTD

- API is simple mapping of DTD structure to Python

TODO

- More simplification of APIs
- Indications, classes and qualifiers work
- More documentation

Demo Application #1

- Instance consistency checker

- Checks instances of a class against class definition

`http://pywbem.sf.net/examples/MDC2005/instance-checker.py`

Demo Application #1 (2)

```
#!/usr/bin/python

import pywbem

# Make connection

conn = pywbem.WBEMConnection(
  'https://%s' % server,
  (username, password))
```

Demo Application #1 (3)

```
# Get class definition and
# instances

klass = conn.GetClass(classname)

instances =
conn.EnumerateInstances(classname)

for i in instances:
    # Check instances against definition
```

Demo Application #1 (4)

```
# Check for extra properties

extra =
  set(klass.properties.keys()) -
  set(i.properties.keys())

if len(extra) > 0:
  print 'Extra properties in %s
   instance: %s' %
    (classname, extra)
```

Demo Application #1 (5)

```
# Check for missing properties

missing =
  set(i.properties.keys()) -
  set(klass.properties.keys())

if len(missing) > 0:
  print 'Missing properties in %s
   instance: %s' %
    (classname, missing)
```

Demo Application #1 (6)

```
# Check property types against
# definition

for prop, value in
  i.properties.items():

  if value.type !=
    klass.properties[prop].type:

    print 'Property %s has bad type'
      % klass.properties[key].name
```

Demo Application #2

- Web-based schema browser

- Simple CGI script to browse class definitions

`http://pywbem.sf.net/examples/MDC2005/schema-browser.py`

Demo Application #2 (2)

```
#!/usr/bin/python

import cgi, pywbem

print 'Content-Type: text/html\n'

print '<html><head>'
print '<title>CIM Browser</title>'
print '</head><body>'
```

Demo Application #2 (3)

```
form = cgi.FormContentDict()

if not form.has_key('classname'):

  classnames =
    conn.EnumerateClassNames(DeepInherit
    ance = True)
  classnames.sort()

  print '<h1>Classes</h1>'
```

Demo Application #2 (4)

```
print '<ul>'

[print_html_link(cl) for cl in
classname]

print '</ul>
```

Demo Application #2 (5)

```
if form.has_key('classname'):

  print '<h1>%s</h1>' % classname

  try:
    cl = conn.GetClass(classname,
      LocalOnly = False)
  except pywbem.CIMError, arg:
    print arg[1]
    sys.exit(1)
```

Demo Application #2 (6)

```
# Display qualifiers

[print_qualifier(q.name, q.value)
 for q in cl.qualifiers.values()]

# Display properties

[print_property(key,
  value.qualifiers['Description'].value)
 for key, value in cl.properties.items()]
```

Demo Application #3

- A simple CIMOM leveraging the Twisted Python framework for HTTP transport

- Unfinished with no provider support

`http://pywbem.sf.net/examples/MDC2005/mini-cimom.py`

Questions?

Presentation and examples available at:

`http://pywbem.sf.net/examples/MDC2005`