SDC

STORAGE DEVELOPER CONFERENCE

SNIA SANTA CLARA, 2017

Automation of SMI-S managed storage systems with Pywbem

Karl Schopmeyer

k.schopmeyer@swbell.net

Inova Development, Inc.

V 0.9-6 Sept 17
V 0.91-11 Sept 17

What is Pywbem?

- Python API for communicating with WBEM and SMI servers (implements DMTF WBEM operations and CIM Objects)
- Client platform on which to build SMI client scripts, and applications

Pywbem Overview

- **Python implementation of DMTF CIM/XML client**
  - Python 2.6, 2.7, 3.4 – 3.6
  - Supports DMTF CIM-XML protocol
  - WBEM Client library with a pythonic API for communication with WBEM servers
  - Indication listener
- **Open source and freely available**
- **Maintained**
  - Growing functionality, regular releases, fix issues
- **Complete, tested, compatible with DMTF and SMI specifications**
- **User ready**
  - Download and install with Python **pip**
    - `pip install pywbem`
- **LGPL 2.1 license**
  - **This license causes No problems with pip installed code**
- **Uses:**
  - Writing python based apps for WBEM/SMI clients
  - Writing WBEM/SMI admin scripts
  - Testing WBEM/SMI implementations
- **Core library for a set of python based WBEM Tools**

Pywbem Projects

- **Pywbem** – Primarily API for access WBEM Servers
  - Includes MOF compiler, wbemcli test tool
- **Pywbemtools**
  - Pywbemcli
    - Cmd line tool for script, cmd, and interactive access to wbem servers using pywbem
- **SMIPyping**
  - Server test environment tool for testing in multiserver environments (large groups of servers)

Use Cases

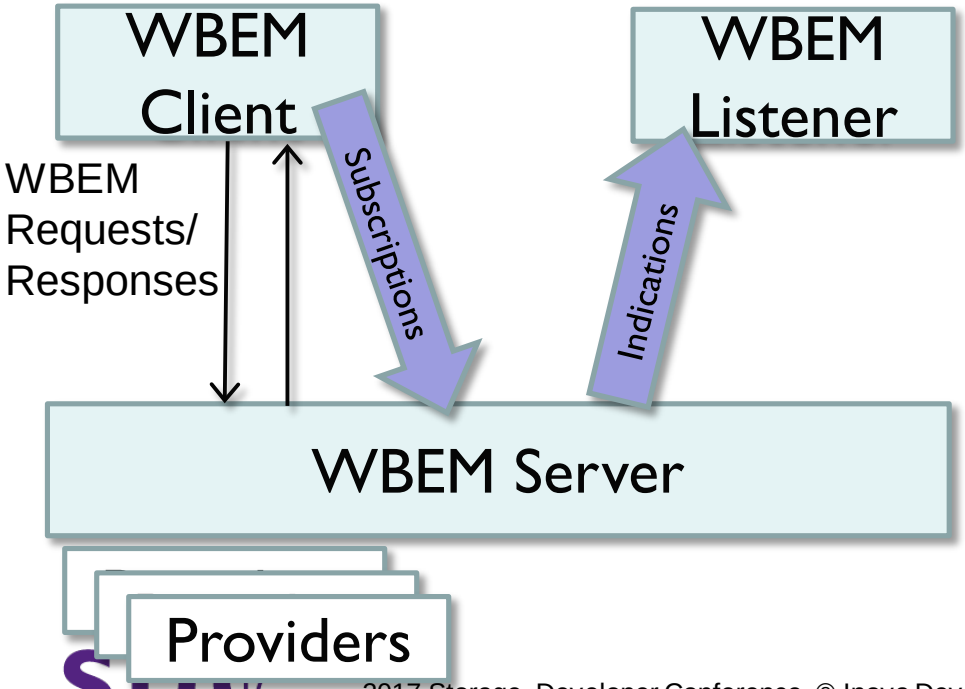
- Developing WBEM/ SMI based client applications that communicate with WBEM servers to manage SMI or other CIM based environments
- Test tool for other client developers because this tool is widely tested, open source
- Test tool for WBEM Server development and testing. It is becoming part of OpenPegasus internal testing.
- Script based apps/tools for WBEM server interface

DMTF CIM and WBEM

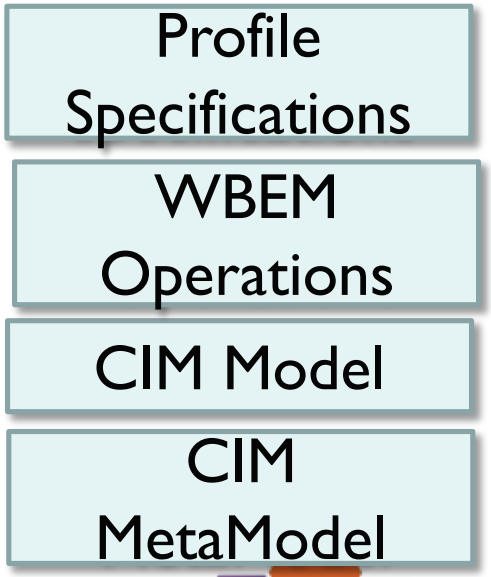
- CIM Is the metamodel and model
  - CIMClass, CIMInstance, CIMQualifier, CIM data types
- WBEM is the operating environment.
  - WBEM generic Operations
    - Operations are not part of the model
    - They define accessing the model (client/server)
    - Generic operations are the model for operations that all protocols should follow
  - The WBEM Protocols
    - Ex. WBEM CIM/XML operations (DSP0200/0201)
  - CIM/WBEM Indications

CIM/WBEM Architecture

Components of the Arcitecture



Specifying Behavior in CIM/WBEM



WBEM Operations

- CIMInstances
  - Read, enumerate*, create, modify, delete, query, association-traversal*
- CIMClasses
  - Read, enumerate, create, modify, delete
- Qualifiers
  - Read, enumerate, set, delete

* Execute on either classes or instances

- Pull Operations
  - Original operations were monolithic
  - Pull operations
    - Open, read, close
  - Mitigate scalability issues
- Classes with common behavior(common profiles)
  - Indications and subscriptions
  - Jobs
  - Profile registration

Profiles, the heart of SMI-S

- A subset of CIM classes from the DMTF schema constrained to manage a specific type of manageable resource
- Documented in a specification
- Give server and client a clear definition of how to manage a particular resource type.
- Advertised as WBEM Server capabilities (registered profiles)

Some profile examples

- SMI-S
  - Array Profile
  - Block Server Profile
  - File System Profile
  - NAS
- DMTF
  - Indication Profile
  - Object Manager Profile
  - Profile Registration Profile

PyWBEM

- PyWBEM implements the DMTF specifications for a WBEM client
- Includes
  - CIM objects as python classes
  - WBEM methods as python methods
  - Extensions for CIM common classes
    - Subscriptions, namespaces, etc.
- Both client and indication listener component

PyWBEM and CIM/WBEM

- CIM data types as Python/Pywbem classes:
  - Boolean (python bool)
  - Char16(python string)
  - string(python string)
  - Uint*/Sint* (8,16,32,64) (pywbem class for each)
  - Real* (32,64)(pywbem classes)
  - CIMDataType(pywbem datetime)
  - Array(Python list)
- **CIMObjects as PyWBEM classes**
  - CIMClass
  - CIMClassName
  - CIMInstance
  - CIMInstanceName
  - CIMProperty
  - CIMParameter
  - CIMQualifier
  - CIMQualifierDeclaration
  - CIMProperty
  - CIMParameter

The Pywbem Operations

- Methods of the class WBEMConnection
- Instances
  - EnumerateInstances
  - EnumerateInstanceNames
  - Associators*
  - AssociatorNames*
  - References*
  - ReferenceNames*
  - GetInstance
  - CreateInstance
  - ModifyInstance
  - DeleteInstance
- InvokeMethod
- ExecQuery

* Execute on either classes or instances

- Classes
  - GetClass
  - EnumerateClasses
  - CreateClass
  - ModifyClass
  - DeleteClass
- QualifierDeclaration operations
  - GetQualifier
  - EnumerateQualifiers
  - SetQualifier
  - DeleteQualifier

- Pull operations, New 2016
- OpenEnumerateInstances
- OpenEnumerateInstanceNames
- OpenAssociators
- OpenReferences
- OpenExecQueyr
- PullInstancesWithPaths
- PullInstancePaths
- PullInstances
- CloseEnumeration

- Iter…Operations
- New 2017
- Merge original and pull
- Pythonic

Pywbem Operations (cont)

- All errors are exceptions.
- Generally server exceptions are pywbem Error or CIMError
- Return data depends on operation type:
  - Enumerates, associators, references
    - List of returned instances, instancenames, classes, qualifierDecls
  - Get
    - Single returned instance of instance, instance name, class, qualifierDecl
  - invokeMethod
    - returnValue, output parameters
  - create/modify
    - Success or failure. If instances, instance path is returned.
  - Pull Operations
    - Instances or paths, end_of_sequence, enumeration_context as a named tuple

Pywbem Iter… methods

- Merge Open/Pull and Enumerate into wrapper methods.
- Move decision on use of pull methods to infrastructure
- Same input parameters as corresponding Open… operation
- User can force pull or non-pull operation usage
  - WBEMConnection(…,use_pull_operations=<None,True,False>
- Use python iterator model for responses
- Iter… for EnumerateInstances, EnumerateInstanceNames, Associators, AssociatorNames, References, ReferenceNames, ExecQuery

Pywbem connecting to a WBEM Server

- **WBEMConnection class defines connection**
- **Lazy execution**
  - Connection not made until request issued
- **Attributes:**
  - url - host name/ip including scheme and port
  - Credentials - if required (name and password)
  - default_namspace – Namespace to use unless overridden by individual namespace on operations
  - X509 – client cert/key if server demands client authentication
  - verify_callback – Callback for optional extra checking of server certs
  - ca_certs – ca authority common with server
  - no_verification, boolean option to inhibit verification of server cert
  - timeout – timeout for server response time

Pywbem; a simple example

```
# Get instances of defined class/subclasses in namespace
CONN = WBEMConnect(url, default_namespace='root/myns',…)
insts = CONN.EnumerateInstances ('CIM_ComputerSystem')
for inst in insts:
        print('%s', inst.tomof())

# Get the names only
names = CONN.EnumerateInstanceNamess (
        'CIM_ComputerSystem')
for name in names:
        print('%s', name')
```

Iter… method example

- Example:

```
conn = WBEMConnection(…)
iter_insts =  conn.IterEnumerateInstances('myclass',
                maxObjectSize=1000)
for instance in iter_insts:
    print(instance.tomof())
```


Indication Subscriptions

CIM_IndicationFilter
Defines the CQL or WQL filter for an indication

CIM_ListenerDestinationCIMXML
Defines the listener destination url

CIM_IndicationSubscription
Association relates the indicationFilter to Destination instances

A single subscription is instances the 3 classes above

- All must exist for a subscription
- They are persistent in the server
- Normally in server interop namespace

```
TEST_CLASS = 'Test_IndicationProviderClass'
TEST_CLASS_NAMESPACE = 'test/TestProvider'
TEST_QUERY = 'SELECT * from %s' % TEST_CLASS
# Create subscription_manager
    subscription_manager = WBEMSubscriptionManager(
        subscription_manager_id='fred')

    # Add server to subscription manager
    server_id = subscription_manager.add_server(server)
    listener_url = '%s://%s:%s' % ('http', 'localhost',
                                   http_listener_port)
    subscription_manager.add_listener_destinations(server_id,
                                                   listener_url)

    # Create owned alert indication filter and subscribe
    filter_path = subscription_manager.add_filter(
        server_id, TEST_CLASS_NAMESPACE,
        TEST_QUERY,
        query_language="DMTF:CQL")
    subscription_paths =
        subscription_manager.add_subscriptions(server_id,
                                               filter_path)
```

PyWBEM Indication Support

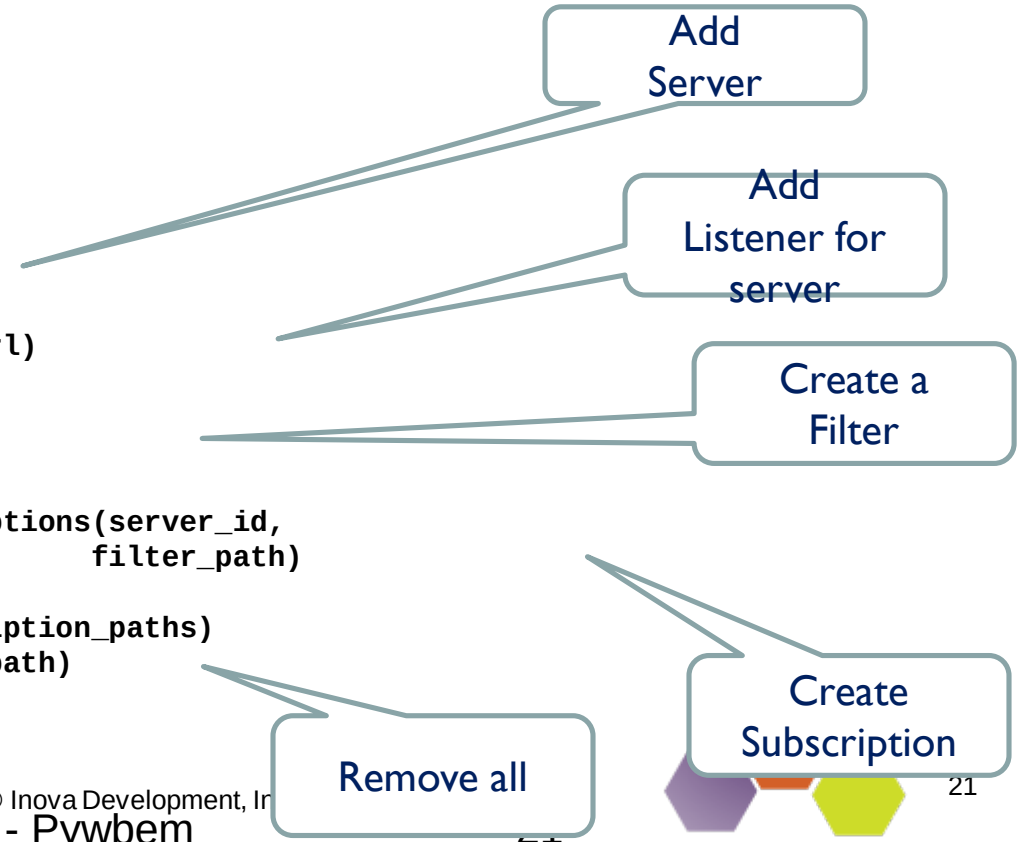
- PyWBEM Subscription Manager to create/ delete indications for multiple servers/listeners
  - Supports persistence differences between server and client
- PyWBEM listener to implement functionality of a WBEM indication listener

Subscription Example

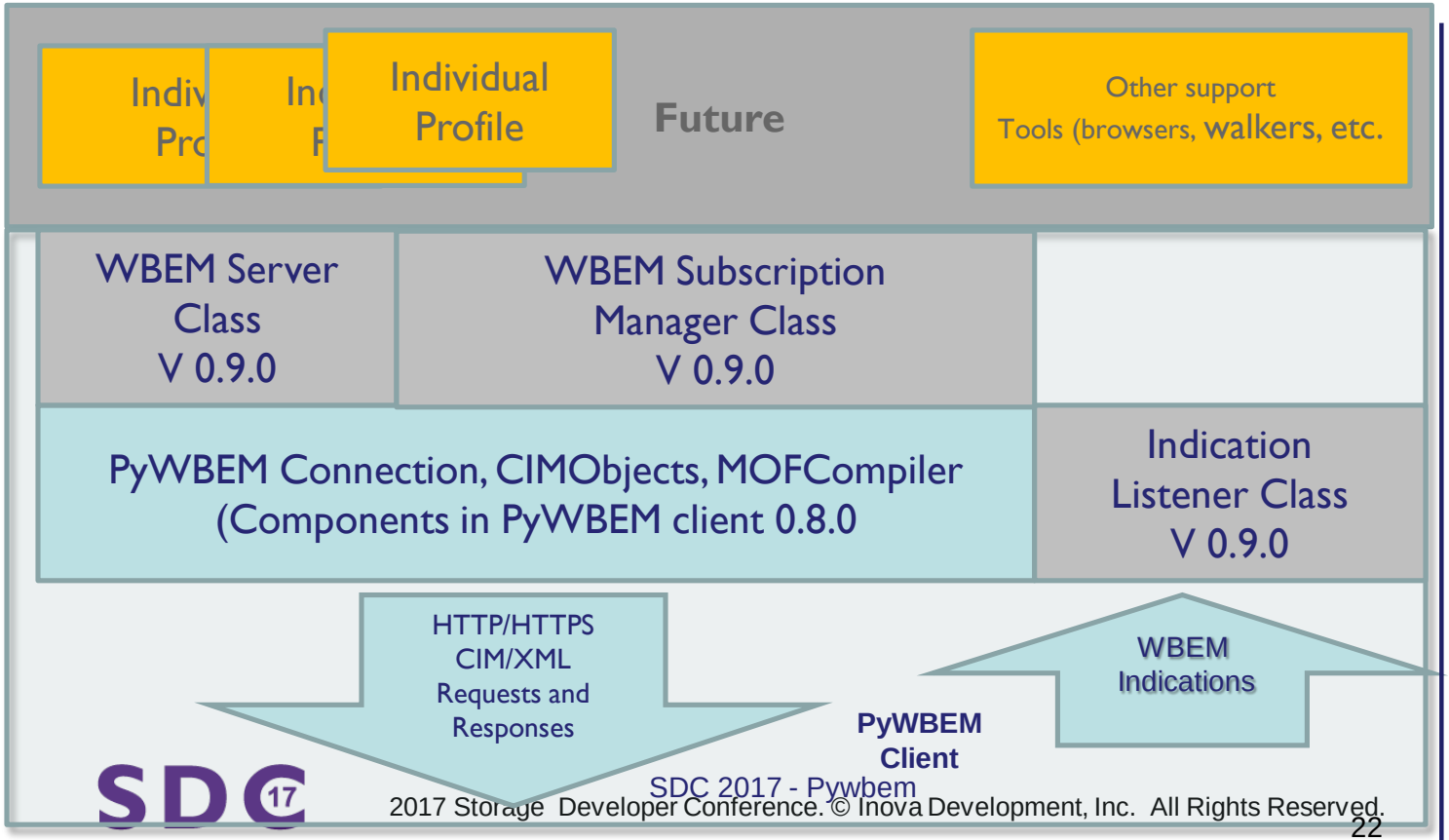
```
TEST_CLASS = 'Test_IndicationProviderClass'
TEST_CLASS_NAMESPACE = 'test/TestProvider'
TEST_QUERY = 'SELECT * from %s' % TEST_CLASS

conn = WBEMConnection(url)
server = WBEMServer(conn)
sub_mgr = SubscriptionManager()
server_id = sub_mgr.add_server(server)
Sub_mgr.add_listener_url(server_id, listener_url)

filter_path = sub_mgr.add_filter(server_id,
    TEST_CLASS_NAMESPACE, TEST_QUERY,
        query_language="DMTF:CQL")
subscription_paths = sub_mgr.add_owned_subscriptions(server_id,
                                                     filter_path)
. . . HERE User may wait
sub_mgr.remove_owned_subscriptions(url, subscription_paths)
sub_mgr.remove_owned_filter(server_id, filter_path)
sub_mgr.remove_server(server_id)
```



PyWBEM Components



Other pywbem project components:
Status: undefined

WBEM Python Providers

Python WBEM Server

22

Pywbem Availability

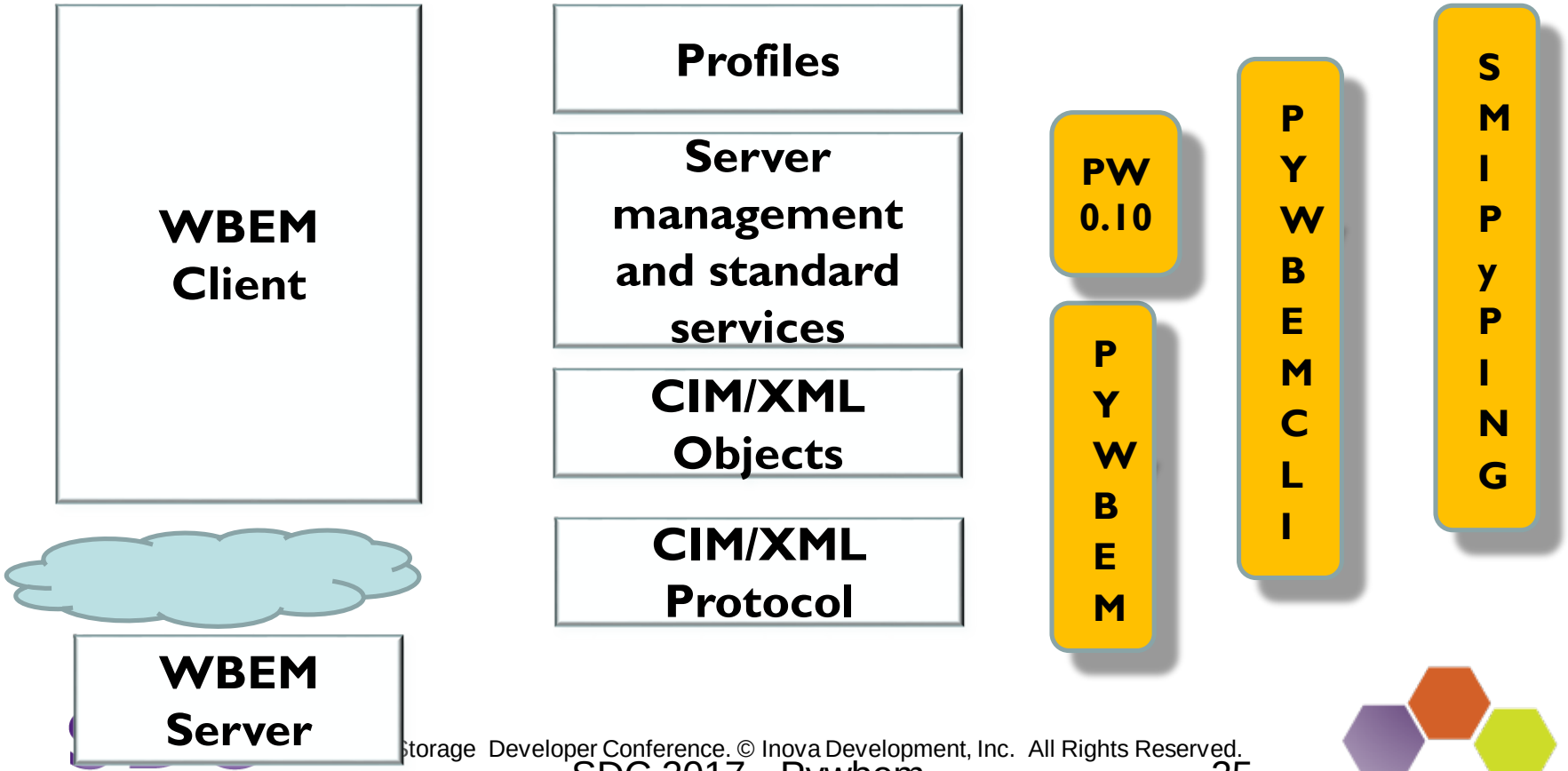
- Client package "pywbem" available in PyPi repository
- Client package available on some Linux distributions
  - Ex. Ubuntu as python-pywbem (v. 0.7.0) BUT OBSOLETE
- Directly available from pywbem project on Github:
  - pywbem is a github group with 4 code repositories (pywbem, cimserver, yawn, pyprov) and a doc repository (pywbem.github.io)
  - Adding new repository now for Pywbemtools
  - Download links on PyWBEM github web site:

    http://pywbem.github.io

PyWBEM Installation

- Standard python packaging
- pip package installation
  - Within your Python environment get from pip
    - pip install pywbem
  - pip from github
    - pip install git+git://github.com/pywbem/pywbem
- Install complete github package
  - Git clone https://github.com/pywbem/pywbem

WBEM/CIM Architecture and Pywbem



Pywbemcli, CIM/ WBEM cmd line browser

- Goals
  - Simplicity
    - Create a clean cli user interface to represent complex set of management tasks
    - View/manage classes, instances, qualifiers, namespace info, WBEM server info, profile info, subscriptions
  - Presentation of rich information
    - CIM Classes, etc. are not flat data but structures and associations are really networks of information
  - Constructing complex operations from simple operations
    - Getting instance information is simple. But to get it in a usable form is not
  - Making the input and results of this tool integratable into larger scripts.
  - Make the tools easily installable and documented.

Pywbemcli functions

- Inspect CIM objects in WBEM Server
- Modify some CIM objects in WBEM Server
- Present overview information on WBEM Server
  - Status, object relationships, etc.
- Get statistics on server
- Log operations*

Implementation overview

- Single application with multilevel commands
  - Interactive or cmd mode
  - Documented with help and separate doc
- Status
  - Near initial release
- Multios (Linux, Windows, Mac)
- Built on pywbem api

Pywbemcli example cmds

- **Command mode**
  - **pywbemcli –s localhost class get ‘CIMManagedElement’**
    - Returns mof for the class
- **Interactive mode**
  - **Pywbemcli repl**
  - **repl: class enumerate ‘CIM_ComputerSystem’ -n**
    - **Returns list of classes**
  - **repl: class get <some class name from list above’**
  - **repl: quit**

Pywbemcli subcmd list (currently)

- **`class`**
  - Get, enumerate, accociators, tree view, invokemethod, find, etc.
- **`instance`**
  - Get enumerate, associators, invokemethod, counts, etc.
- **`qualifier`**
  - Get, enumerate
- **`server`**
  - Namespaces, profiles, overview, etc.
- **`connection`**
  - Manage multiple connections
- subscription
  - Manage, list, create, delete, subscriptions
- Future
  - Profiles
    - Common code for working with all SNIA profiles
  - Further common services
    - Jobs, etc.

SMIPyping

- Parallel to pywbemcli except uses a database to execute operations on multiple servers in parallel in the environment.
- Examples
  - Test existence
  - Test capabilities

Using and working with Pywbem Project

- Pywbem is public repository on github
- Each release is on PyPi (**`pywbem`**)
  - 1 – 2 releases per year
- Documentation is in public repository
- Pywbem uses github issues and pull requests processes.
- Engage with PyWBEM community, for:
  - Reporting issues (pywbem github issues)
  - Feature requests (pywbem github issues)
  - Contributing (for example from github fork)

More Information on PyWBEM

- **Pywbem public project github:**
  - https://github.com/pywbem
- **Pywbem public client project github:**
  - https://github.com/pywbem/pywbem
- **Pywbem client documentation online:**
  - **http://pywbem.github.io/pywbem/**
  - **http://pywbem.readthedocs.io/en/stable/**
    - Includes installation, API documentation, usage
- **Pywbem ipython notebooks online:**
  - **https://pywbem.readthedocs.io/en/latest/tutorial.html#executing-code-in-the-tutorials**
- **SNIA pywbem web page:**
  - **https://www.snia.org/pywbem**

Extra Slides

The following is extra slides for discussion, and more information. It is not part of presentation

Some Relevant DMTF specifications

- DSP0004 – Defines metamodel, model, major characteristics, and MOF
- DSP0201 – Defines WBEM Operations over CIM/XML
- DSP0202 – XML for WBEM Operations over CIM/XML
- DSP0223 – Generic Operations
- Query Language(CQL) – DSP0202
- Operation Query Language (FQL) – DSP0212
- See the DMTF web page:
  - https://www.dmtf.org/standards/published_documents

CIMOperations Iter… methods

- Merge Open/Pull and Enumerate into wrapper methods.
- Moves decision on use of pull methods to infrastructure
- Iter… for EnumerateInstances, EnumerateInstanceNames, Associators, AssociatorNames, References, ReferenceNames, ExecQuery
- Same input parameters as corresponding Open… operation
- User can force pull or non-pull operation usage
- Use python iterator model for responses
- Example:

Iter… Advantages

- Simpler client code
  - Eliminates intermediate variables like end_of_sequence, enum_context
- Matches pythonic pattern of iteration
    - The call returns a generator
- Removes decision making on pull vs. non-pull from users to optimize memory use on servers and clients.
- Returns decisions like enum size, etc. to system level decisions.

New Api Pattern

```
def IterEnumerateInstances(self, ClassName, namespace=None,
                           LocalOnly=None,
                           DeepInheritance=None, IncludeQualifiers=None,
                           IncludeClassOrigin=None, PropertyList=None,
                           FilterQueryLanguage=None, FilterQuery=None,
                           OperationTimeout=None, ContinueOnError=None,
                           MaxObjectCount=DEFAULT_ITER_MAXOBJECTCOUNT, **extra):
```

Enumerate Request Parameters

Open request Extension Parameters

Conn = WBEMConnection(. . . , use_pull_operations=None, …)

- Change for Iter…
- Zero illegal
- Defaults to

- None: Pywbem choses
- True: force pull
- False: use old ops

Returns for each type:

- **EnumerateInstances** : List of instances
- **OpenEnumerateInstances**: Tuple of status and instances
- **IterEnumerateInstances** : Iteration object to be used by for statement or genertor comprehenshion

Iter… functionality

- Iter… method determines if pull can be used by response to first request. If CIM_ERR_NOT_SUPPORTED returned, assumes no pull operations
- Always prefers pull if it exists.
- First call determines if pull exists on server
- Subsequent requests use pull if initial request works.
- WBEMConnection attribute (use_pull_operations) allows caller to override system decisions (force pull or non-pull)
- Allows pull on some request types with non pull on others if the server only supports pull on some.
- Response can be terminated early with iter.close() statement

Operation Comparison

Code that tries pull first

```
If server_has _pull:
    try:
        result = conn.OpenEnumerateInstances(classname,
                                MaxObjectCount=max_open)
        # save instances since we reuse result
        insts = result.instances
        # loop to make pull requests until end_of_sequence
received.
        pull_count = 0
        while not result.eos:
           pull_count += 1
          result = conn.PullInstancesWithPath(result.context,
                                MaxObjectCount=max_pull)
         insts.extend(result.instances)
    except: CIMError as ce:
       if ce.status != ce.status_code ==
CIM_ERR_NOT_SUPPORTED
          raise
else:
  insts =  conn.EnumerateInstances(classname)
For inst in insts
   print(inst.tomof())
```

BECOMES

```
conn = WBEMConnection(…)
iter_obj = conn.IterEnumerateInstances('myclass')
for instance in iter_obj:
    print(instance.tomof())
```

Or to gather all instances with generator expression

```
Instances = (inst for inst in
   conn.IterEnumerateInstances('myclass')
```

Iter… limitations

- Use of queryfilters parameter
  - Since not supported in Enumerate, etc. Iter… oprations fail if fallback to Enumerate with pull operations
- Only do pull to server when local list empty
  - Delays may be visible to client user
- The capability to delay in pull sequence lost
  - Full pull operations allowed request with 0 objects that just reset server timer
  - Pywbem infrastructure does not have enough info to use that concept
- ContinueOnError cannot be used (EnumerateInstances returns all or nothing). Note that almost none of us ever implemented this feature
- Cannot vary size of responses during session nor return zero for OpenEnumerateInstances

CIMInstance PyWBEM Class

- Class Attributes:
  - classname (string)
  - properties(NocaseDict) of CIMProperties
  - qualifiers(NocaseDict) of CIMQualifiers
  - path (CIMInstanceName) optional
  - property_list(list of Strings) -ptional for filter with some operations
- Object Methods for things like
  - Comparison, copy, update, get property info, display
  - See: https://pywbem.readthedocs.io/en/latest/client.html#cim-objects for detailed api documentation

Inspect Instance

- Get path
  - **`path = inst.path`**
- Properties
  - Many ways to access properties (dict, api)
- Access Properties
  - **`if inst.has_key('myPropName'):`**
    `value = inst.get('myPropName')`
  - properties = inst.properties
    - .. Inspect the properties dictionary
  - Etc.

Embedded Instances

- Embedded Instances are the `struct` concept of CIM
- Allow grouping properties within a larger entity
- Normally have no unique identity. They are a component of an instance
- Within PyWBEM.
  - Data type string but with EmbeddedObject flag set.
- Retrieve as value which is converted to CIMInstance
- Create by creating CIMInstance and setting as value in another instance

CIMInstance Methods (examples)

- Create:
  - Required: PropertyName
  - Optional: properties, qualifiers, etc.
  - Inst = CIMInstance('PyWBEM_Foo', properties=<properties
- Copy
  - Inst2 = inst.copy()
- Compare
  - If inst2 == inst1:
- Get a property
  - Property_value = get('p1')
- Test for a property
  - If inst.has_key('p1'):
- Display
  - Inst.tomof(), inst.tocimxmlst(indent=2), repr(inst), str(inst)

CIMProperty PyWBEM Class

- Attributes:
  - name (unicode string) name of property
  - value (CIM data type) Value of property
  - type(unicode string) Name of data type
  - reference_class(unicode string) name of reference class for referenced properties
  - embedded_object indicator if this is embedded instance
  - is_array(bool) indicator if this is array of values
  - array_size(integer) – indicator of fixed size array
  - class_origin(bol)- indicates if property propagated from superclass
  - propagated(bool)
  - qualifiers((NocaseDict)
- Methods for:
  - Copy, display/conversion compare, etc.
  - See: https://pywbem.readthedocs.io/en/latest/client.html#cim-objects

Example: create instances

```
props1 = {
    ‘s1’ : CIMPropertyName(name=‘u1’, type=‘Uint32’
                           value=Uint32(3456)

props2 = {‘UI8’ : True, ‘UI8’ : Uint8(33))}

Inst1 = CIMInstance(‘CIM_foo`, properties=props1)
Inst2 = CIMInstance(‘CIM_foo`, properties=props2)

Inst3 = CIMInstance(‘CIM_Foo`,
                    properties={‘U1` :
                 CIMProperty(‘U1’,
                                    Uint32(42)})
```