SDC 18

September 24-27, 2018
Santa Clara, CA

www.storagedeveloper.org

pywbem
Overview for SMI Client Developers

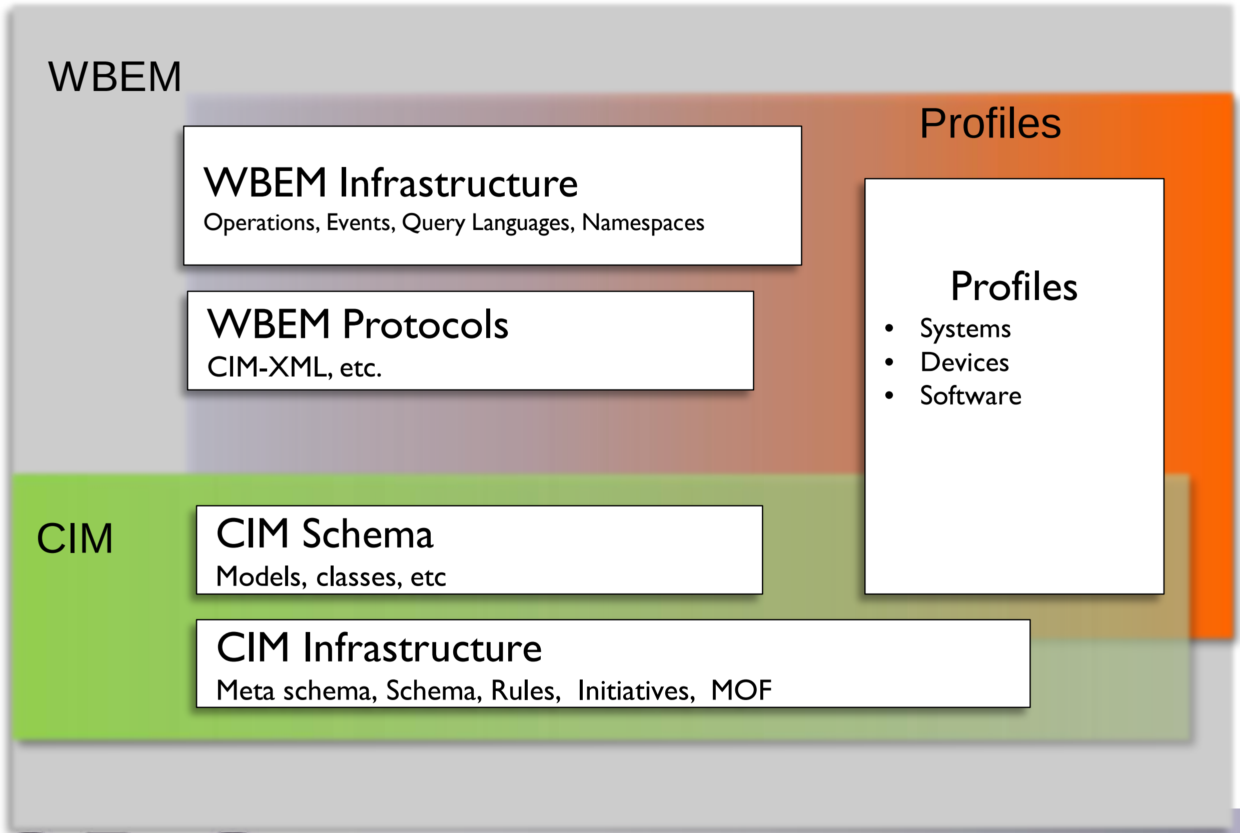
Karl Schopmeyer
Inova Development Inc.

V 0.8.0, Sept 7 2018
V 0.8.1 Sept 14 2018
V0.9.0, Sept 20 2018

Presentation Goals

- Help WBEM and especially SMI client users use pywbem
- How to implement SMI automation with pywbem
- Present real usage examples
- Present overview of technology as required by the client. (Eliminate what is not important to the client)
- Introduce working examples of client usage

What are WBEM and CIM?



A management model/insfrastructure defined by:

- DMTF Specifications
  - CIM Model
  - WBEM Operations
  - Profile Concepts
  - Smash/Dash initiatives
  - Profiles
- SNIA Specifications
  - SMI-S Initiative

What is pywbem?

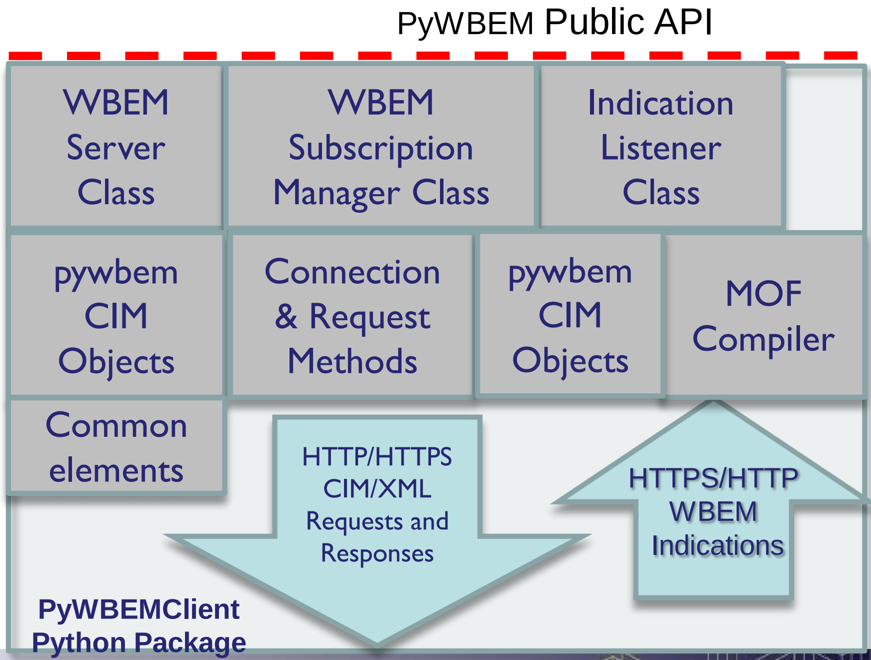
- What is it?
  - Python package for communicating with WBEM and SMI servers (implements DMTF WBEM operations and CIM Objects)
  - Client platform on which to build SMI client scripts, and applications
- Why is it important?
  - Complete, verified implementation of WBEM Client infrastructure
  - Maintained with regular releases
  - Adheres to DMTF/SNIA WBEM/SMIS specifications
  - Well documented

pywbem Overview

- **Python implementation of DMTF CIM/XML client**
  - Python 2.6, 2.7, 3.4 – 3.7
  - Supports DMTF CIM-XML protocol and CIM Model
  - WBEM Client library with a pythonic API for communication with WBEM servers
  - Indication listener
- **Open source and freely available**
- **Multiplatform**
  - **Linux, windows, etc.**
- **Maintained**
  - Growing functionality, regular releases, fix issues
  - Next release: Q4 2018
- **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**
- **Includes diagnostic and support tools**

Pywbem Availabilty

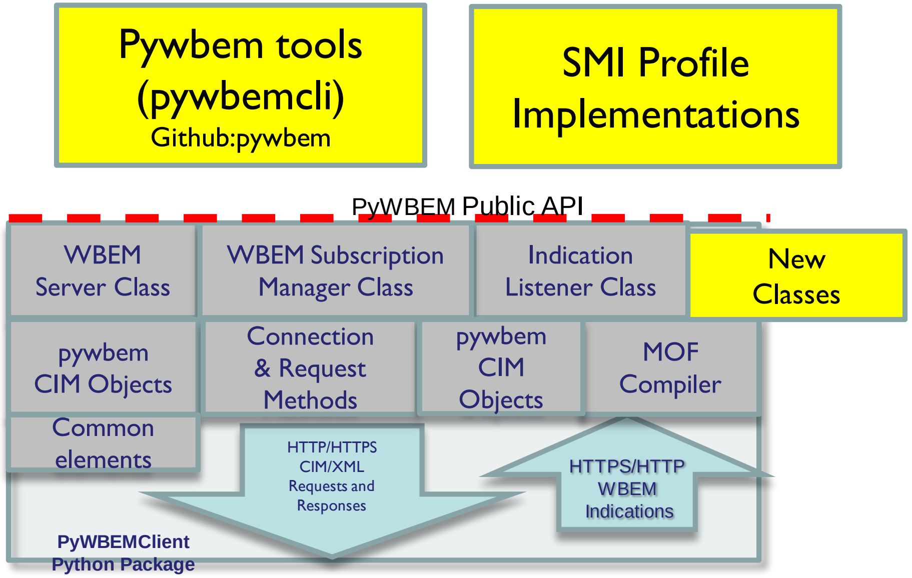
- PyPi package 'pywbem'
- Github project 'pywbem/pywbem'



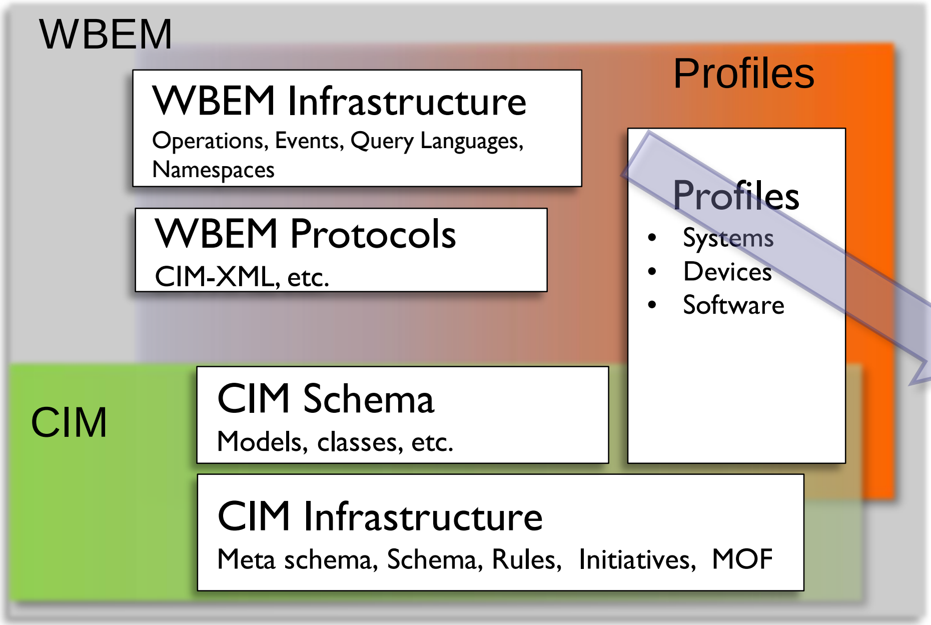
PyWBEM Public API
WBEM Server Class
WBEM Subscription Manager Class
Indication Listener Class
pywbem CIM Objects
Connection & Request Methods
pywbem CIM Objects
MOF Compiler
Common elements
HTTP/HTTPS CIM/XML Requests and Responses
HTTPS/HTTP WBEM Indications
PyWBEMClient Python Package

Pywbem project future directions

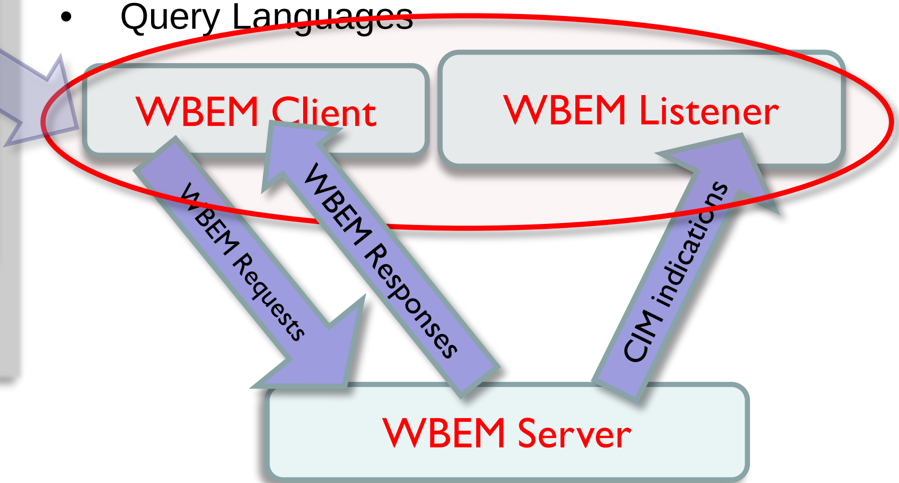
- Release CLI browser (Q4)
- Release pywbem next release (Q4)
- Improve performance and functionality
  - Add general capabilities
- Add and grow tools
- Create project with SMI profile implementations


Pywbem tools (pywbemcli)
Github:pywbem
SMI Profile Implementations
PyWBEM Public API
WBEM Server Class
WBEM Subscription Manager Class
Indication Listener Class
New Classes
pywbem CIM Objects
Connection & Request Methods
pywbem CIM Objects
MOF Compiler
Common elements
HTTP/HTTPS CIM/XML Requests and Responses
HTTPS/HTTP WBEM Indications
PyWBEMClient Python Package

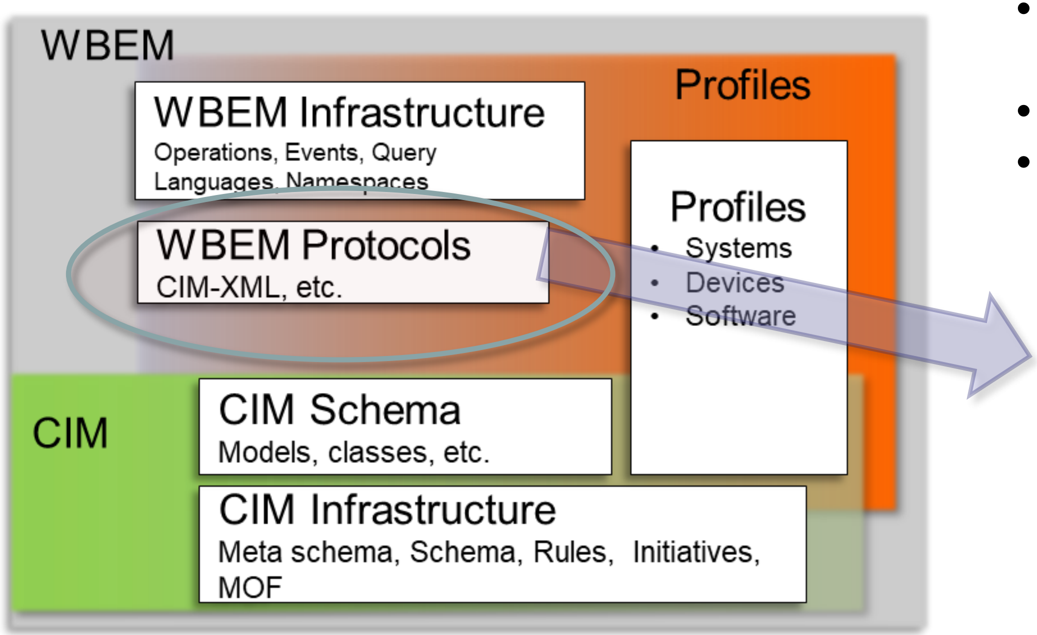
WBEM Infrastructure



- WBEM Architecture Components
  - Server, client, listener
- WBEM Operations definitions
  - Get, Enumerate, Create, Delete, Modify (classes, instances, etc.), InvokeMethod, ExecQuery
- Events(indications)
- Query Languages

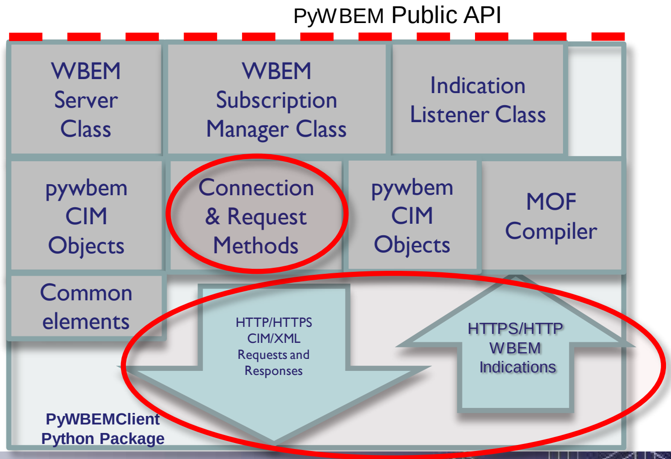


WBEM Protocols



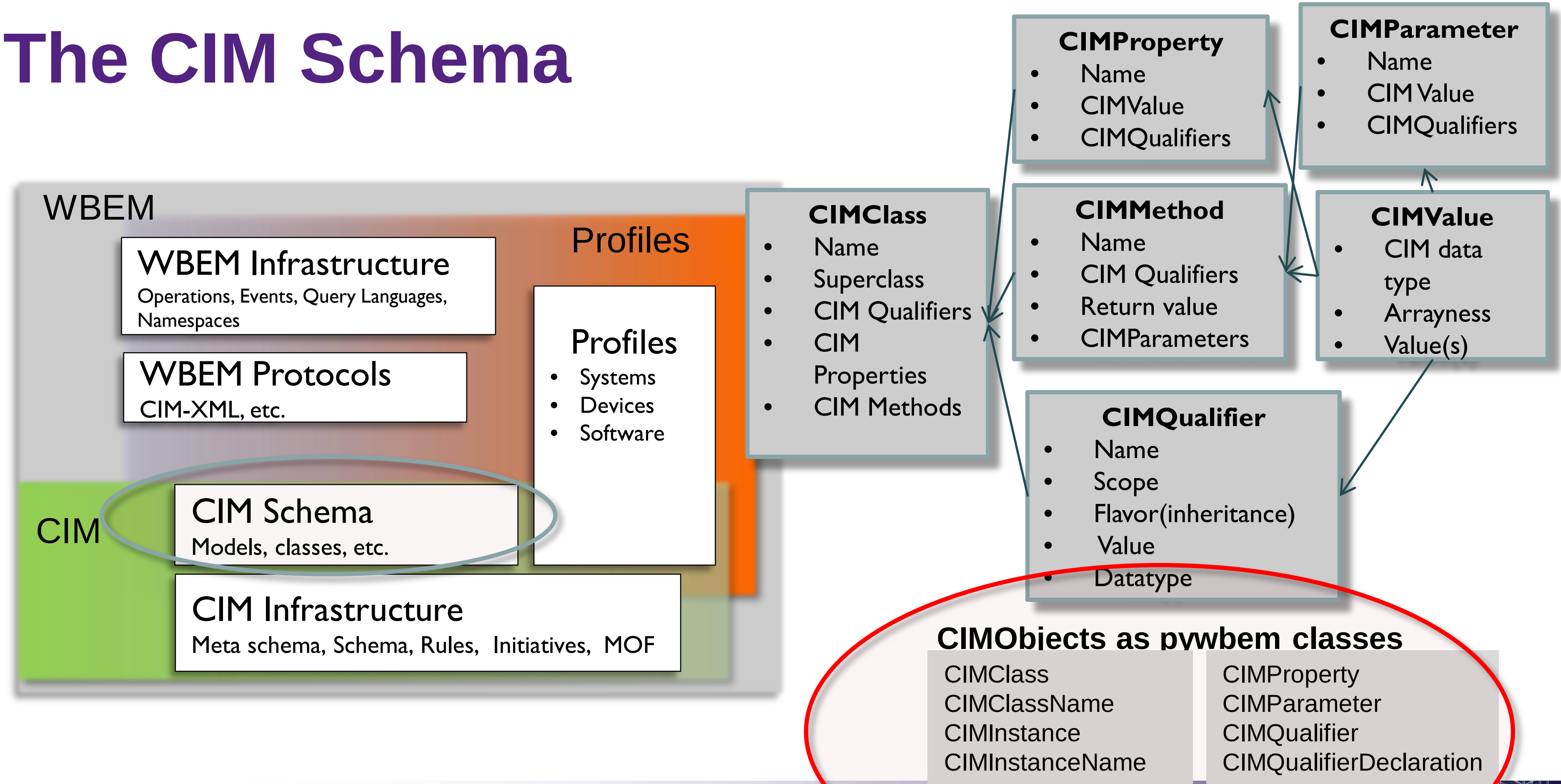
WBEM Protocols

- Communicate between WBEM architecture components
- Define WBEM Message
- Multiple protocols allowed
  - CIM/XML
  - Etc.


PyWBEM Public API
WBEM Server Class
WBEM Subscription Manager Class
Indication Listener Class
pywbem CIM Objects
Connection & Request Methods
pywbem CIM Objects
MOF Compiler
Common elements
HTTP/HTTPS CIM/XML Requests and Responses
HTTPS/HTTP WBEM Indications
PyWBEMClient Python Package

You don't need to know about the protocol, We do it.

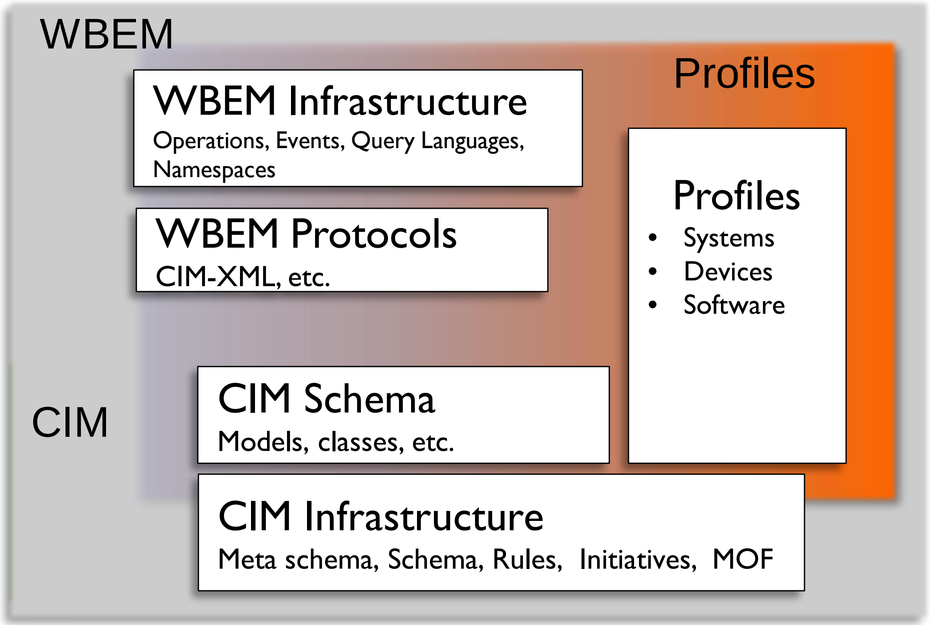
The CIM Schema



CIMObjects as pywbem classes

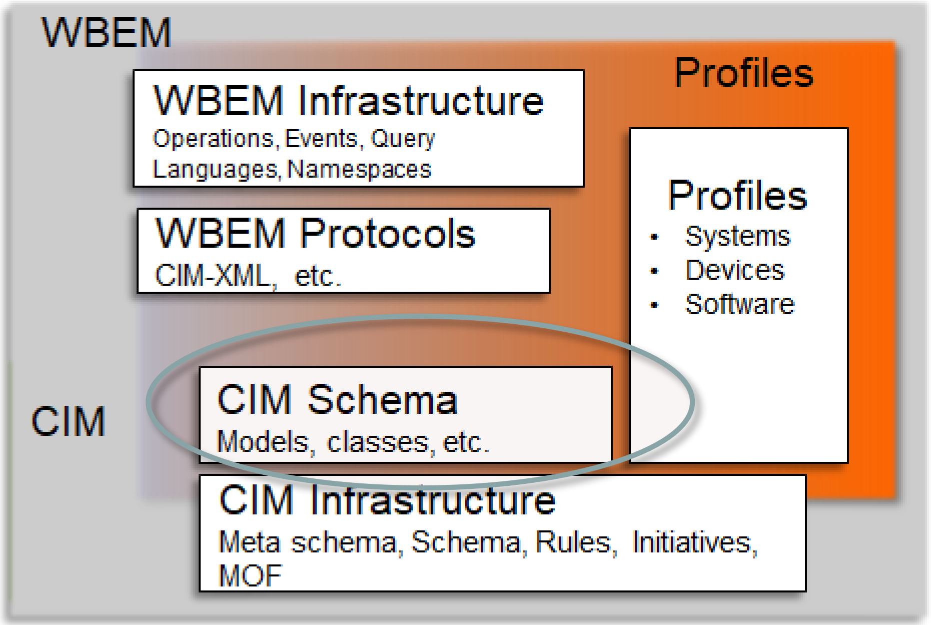
CIMClass	CIMProperty
CIMClassName	CIMParameter
CIMInstance	CIMQualifier
CIMInstanceName	CIMQualifierDeclaration

CIM Schema: Qualifiers



- Qualifiers define characteristics of other CIM model Elements
  - Generally predefined in DMTF Schema
  - Apply to class definitions of class, property, method, parameter
- Major Qualifiers
  - Key (Identify key properties of class)
  - Description
  - Association (identify assoc class)
  - Indication (identify indication class)
  - Required
  - In, Out (Parameter direction)

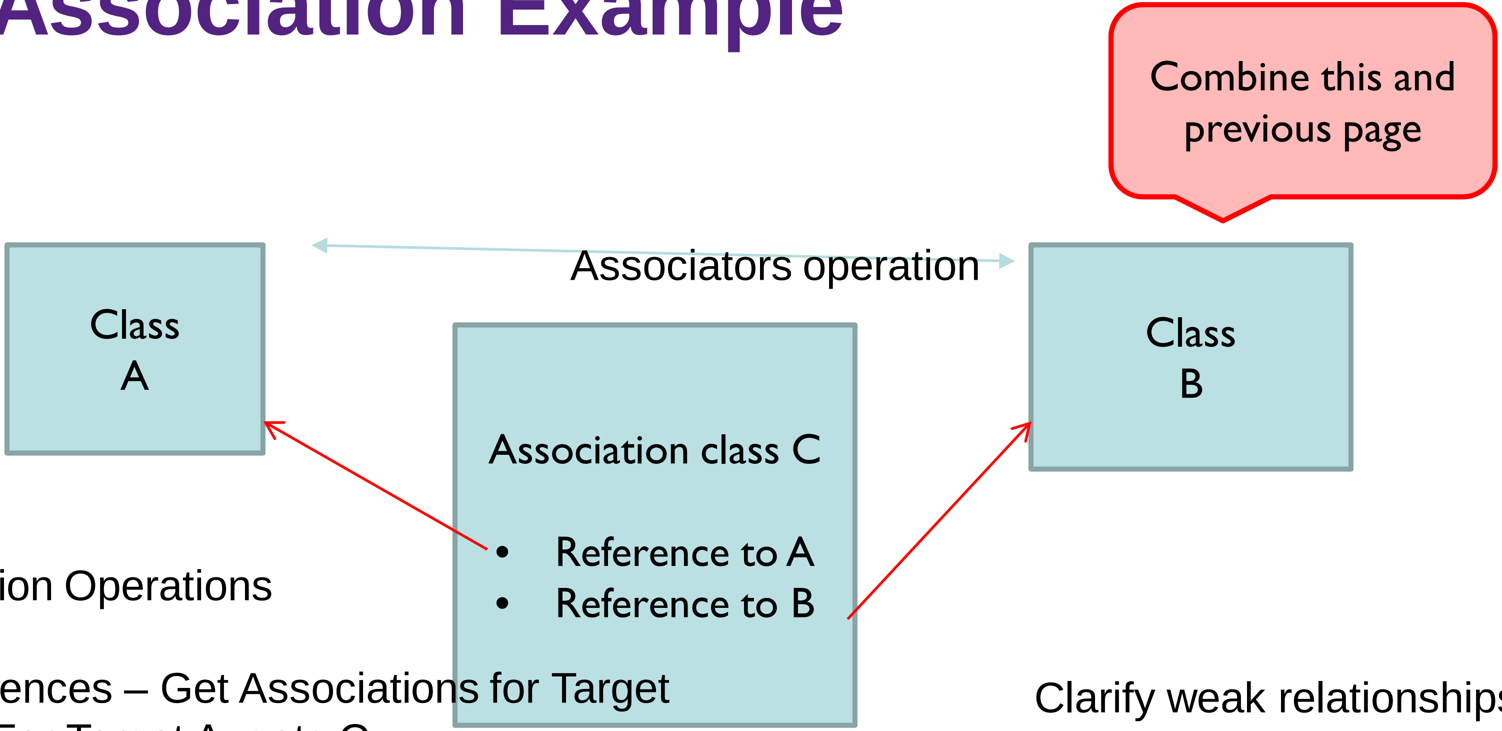
CIM has different types of Classes



- CIM Class
  - Defines manged resources
  - Key properties provide identity
    - CIM Properties defined with key qualifier
- CIM Association
  - Defines relation between classes
  - Reference properties point to other classes/instances
- CIMIndication
  - CIM Class used to pass event information
  - No keys (i.e snapshot of data)

THIS SHOULD HAVE DIAGRAM

Class Association Example

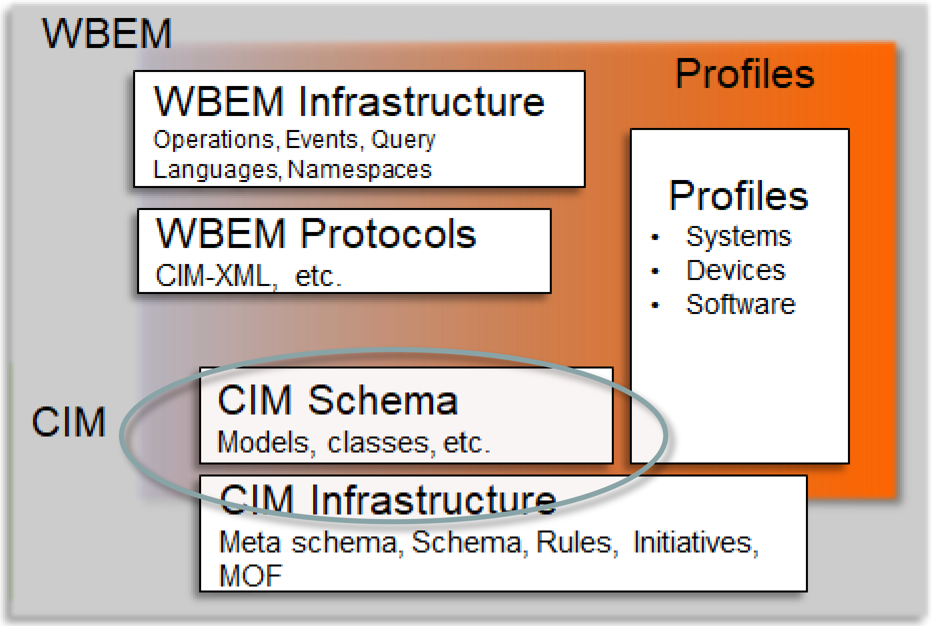


Association Operations

- References – Get Associations for Target
  - For Target A, gets C
- Associators – Get Associated for Target
  - For Target A gets B

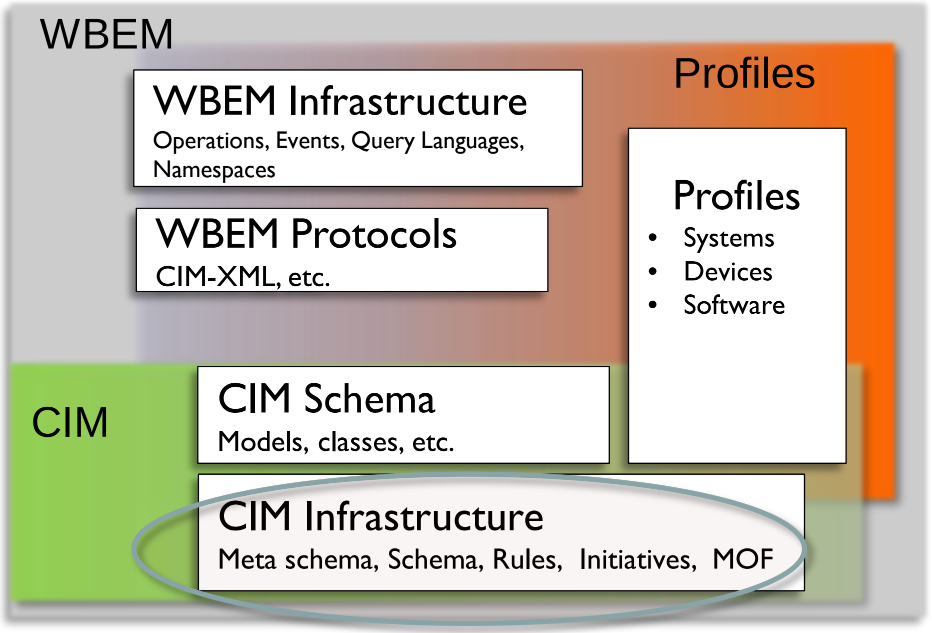
Clarify weak relationships

CIM Schema: Datatypes



CIM Data Type	Pwbem Implementation
Boolean	Python bool
Char16	Python string
string	Python string
Uint/Sint(8,16,32,64)	pywbem class for each
Real (32,64)	pywbem classes
CIMDateTime	pywbem datetime
Array	Python list

CIM Infrastructure



- Meta schema
  - Defines the model
  - Based on UML
- Schema
  - Package of CIM Classes, Qualifiers, etc. representing coherent DMTF release.
- Rules
  - Constraints defined in Specification
- MOF
  - Source code language for CIM Model
  - Defines CIM class, CIM Qualifier Declaration, CIMInstance

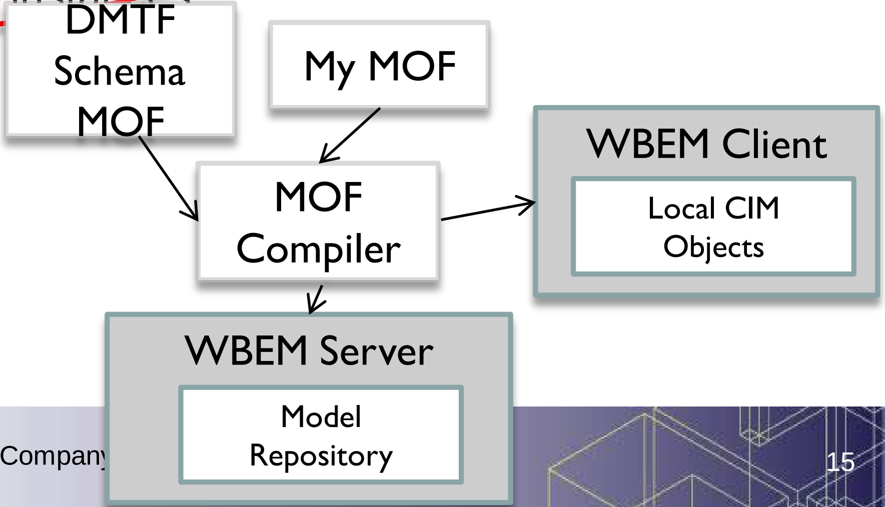
CIM MOF

- Language to define CIM classes, instances, Qualifiers, Methods
- Used by DMTF/SNIA to define Released CIM Schemas
- Can be used to define instances
- Normally compiled by implementation compiler to produce internal representations of classes, instances

Pywbem client includes MOF compiler.

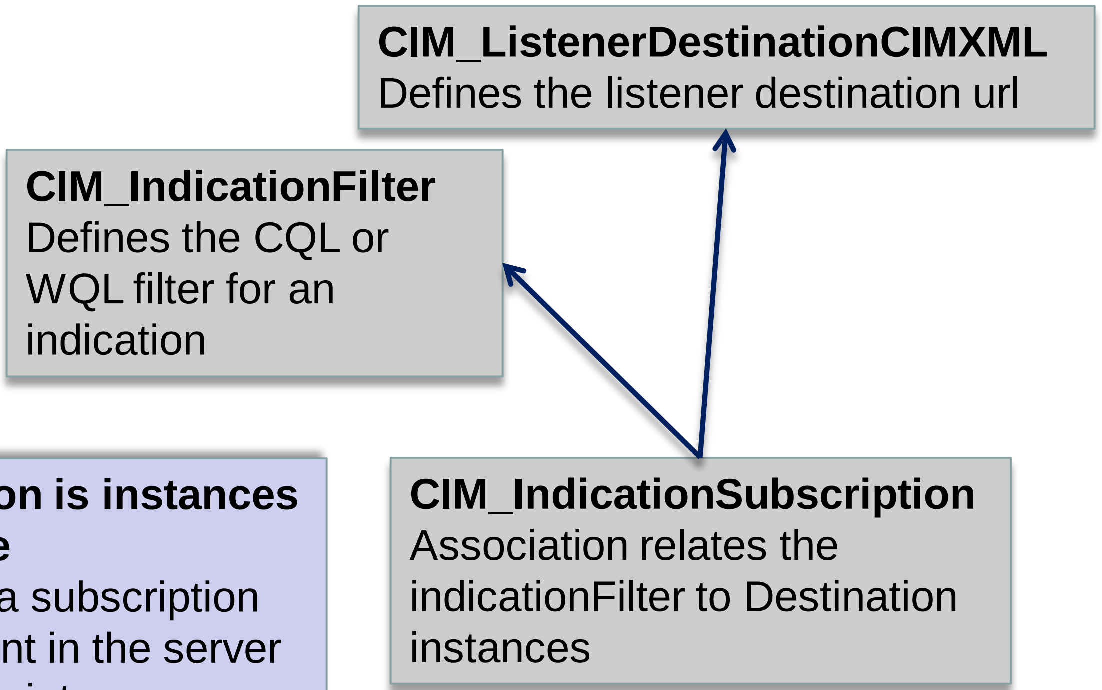
Example MOF:

```
Qualifier Association : boolean = false,
    Scope(association),
    Flavor(DisableOverride, ToSubclass);
Class CIM_Foo {
   [Key, Description(“blah blah”)
  string InstanceID;
  uint32 IntegerProp;
  uint32 ArrayIntProp[];
  uint32 StartService();
```



Indications & Subscriptions

- Indications are representations of specific events sent by server to listener
- Subscriptions define activation and characteristics of specific indications for server



CIM_ListenerDestinationCIMXML
Defines the listener destination url

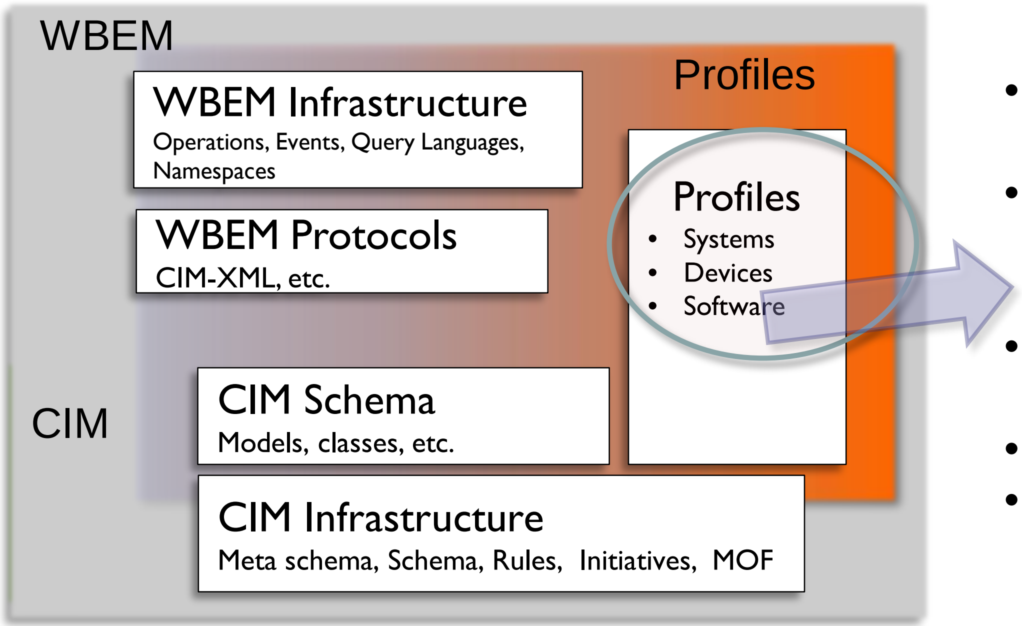
CIM_IndicationFilter
Defines the CQL or WQL filter for an indication

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

Profiles – The Heart of SMI-S



- Subset of schema to represent a management domain
- What is managed by a server and how to access the data in the profile
- Defines resource classes, associations, indications, methods, scripts
- Define constraints not in schema for the domain
- Adaptations of classes
- May be autonomous or components of other profiles

Profiles vs. Schema

- Schema
  - Qualifier declarations
  - Broad set of classes
    - Resources
    - Associations
    - Indications
    - Structures
  - DMTF released schema (~ 3000 classes)

- Profile
  - Subset of classes from the DMTF schema adapted to represent a particular management goal/domain
  - Constraints, usage, scripts
  - Indication definitions

Profile Characteristics

- Structured
  - Autonomous Profiles
    - Standalone. The profile instance not referenced by other profiles.
  - Component Profiles
    - Referenced by other profiles
- Documented in SMI Spec and DMTF specs
- Registered profiles
  - Central Class
  - Scoping Classes

- Array Profile
  - 32 component profiles
    - BlockServices, Health, indications, xxx_targetPorts, xxx_initiator_ports(mandatory)
    - Software, etc. (Optional)
  - 9 classes
    - 2 mandatory

Central and scoping classes

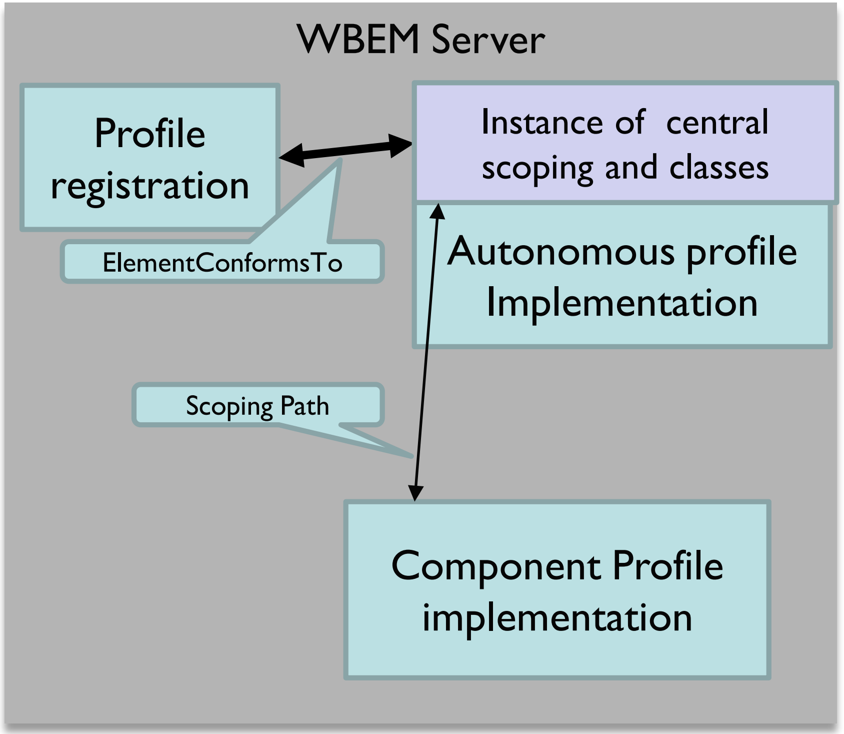
- Central class
  - The focal point for the profile
  - Only required for autonomous profiles
- Scoping class
  - Relate component profile to referenced profile.

- TODO

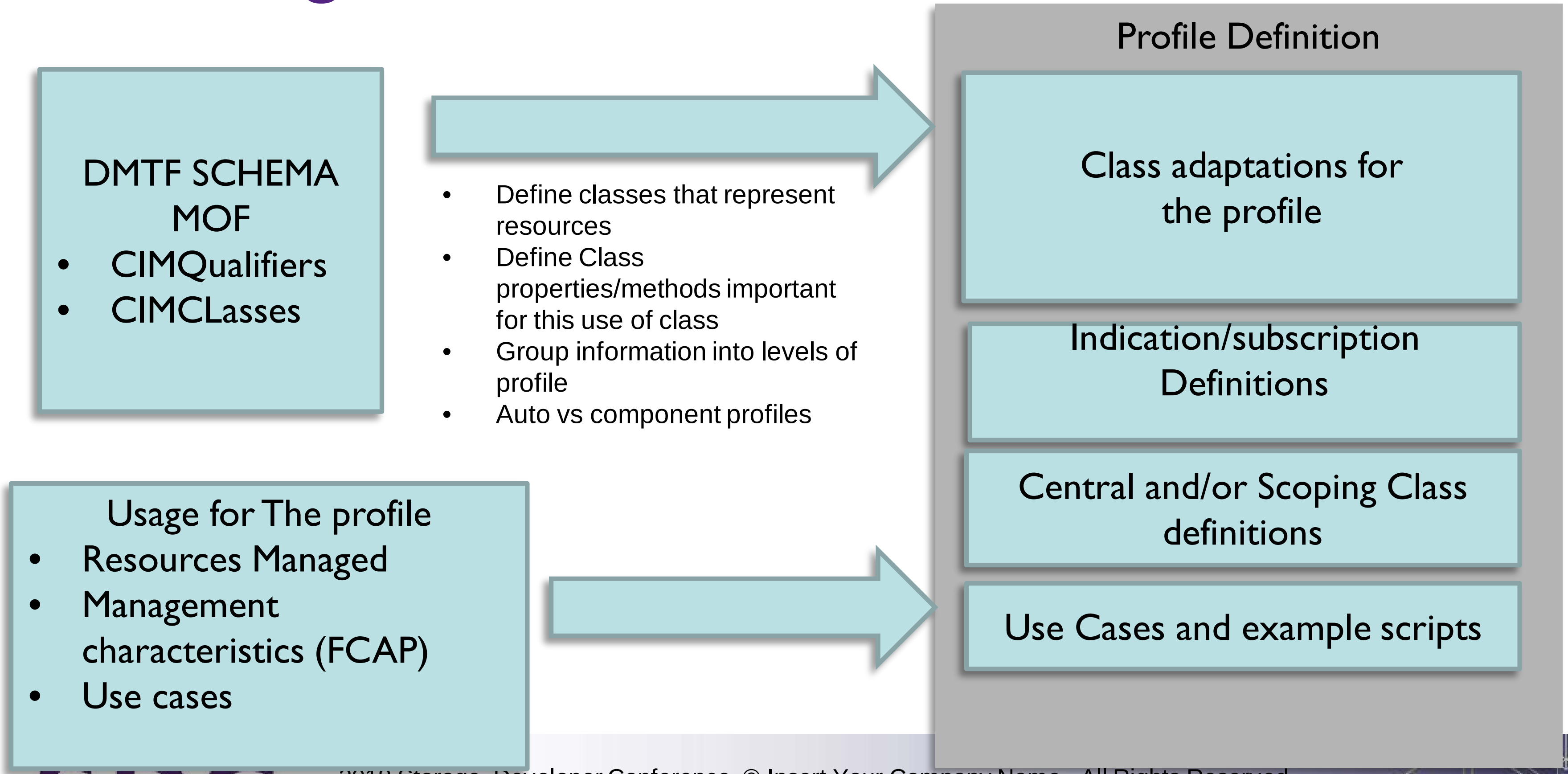
Navigating Profiles from the client

- Start with Registered profiles
  - Server lists profiles it supports
- Association to central class for autonomous profiles
- Component profiles related to autonomous profile by scoping classes

- Example of profiles:
  - Autonomous profiles
    - Server, Array, NAS
  - Component profiles
    - Disk drive
    - Port
    - Software
    - SoftwareIdentity
    - …


WBEM Server
Profile registration
Instance of central scoping and classes
ElementConformsTo
Autonomous profile Implementation
Scoping Path
Component Profile implementation

Defining Profiles



DMTF SCHEMA MOF
- CIMQualifiers
- CIMCLasses

- Define classes that represent resources
- Define Class properties/methods important for this use of class
- Group information into levels of profile
- Auto vs component profiles

Usage for The profile
- Resources Managed
- Management characteristics (FCAP)
- Use cases

Profile Definition

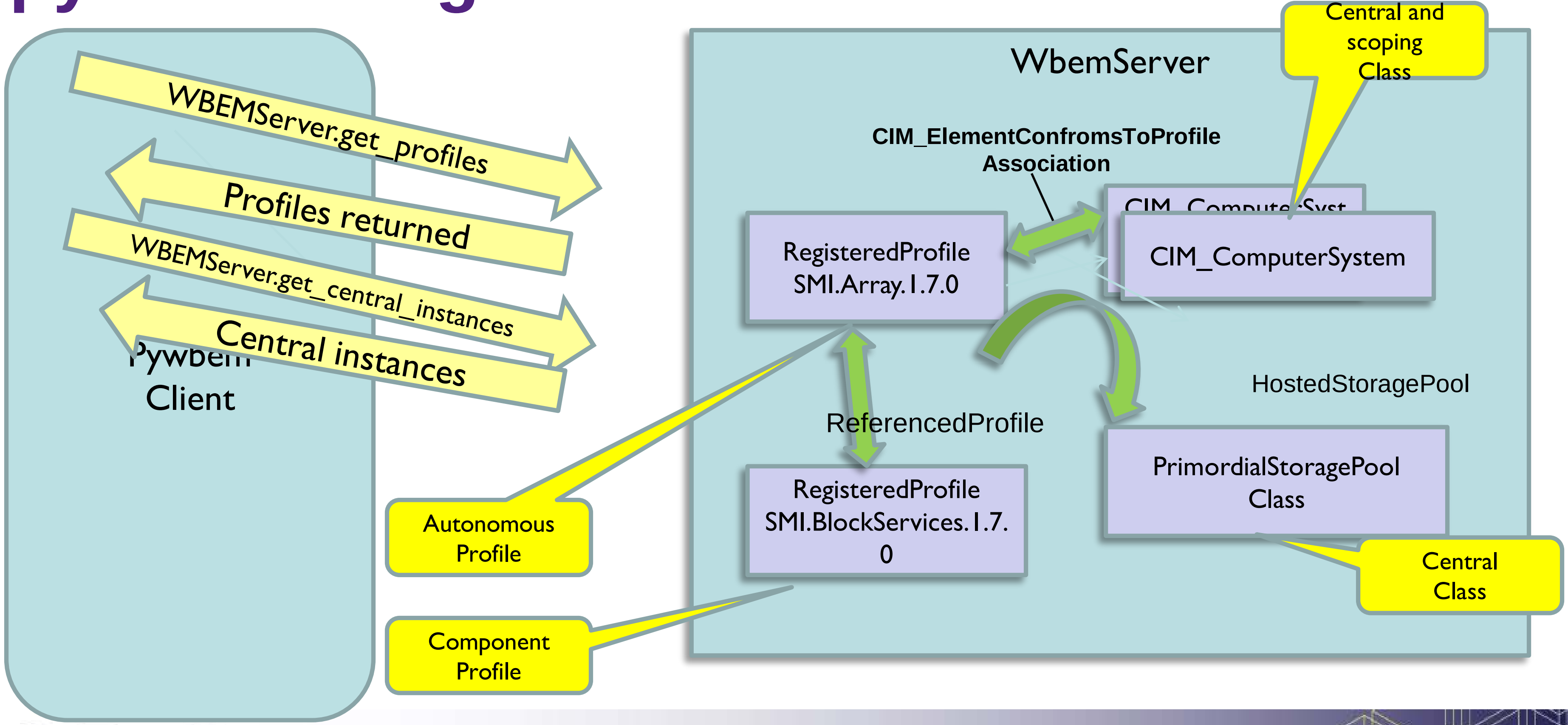
Class adaptations for the profile

Indication/subscription Definitions

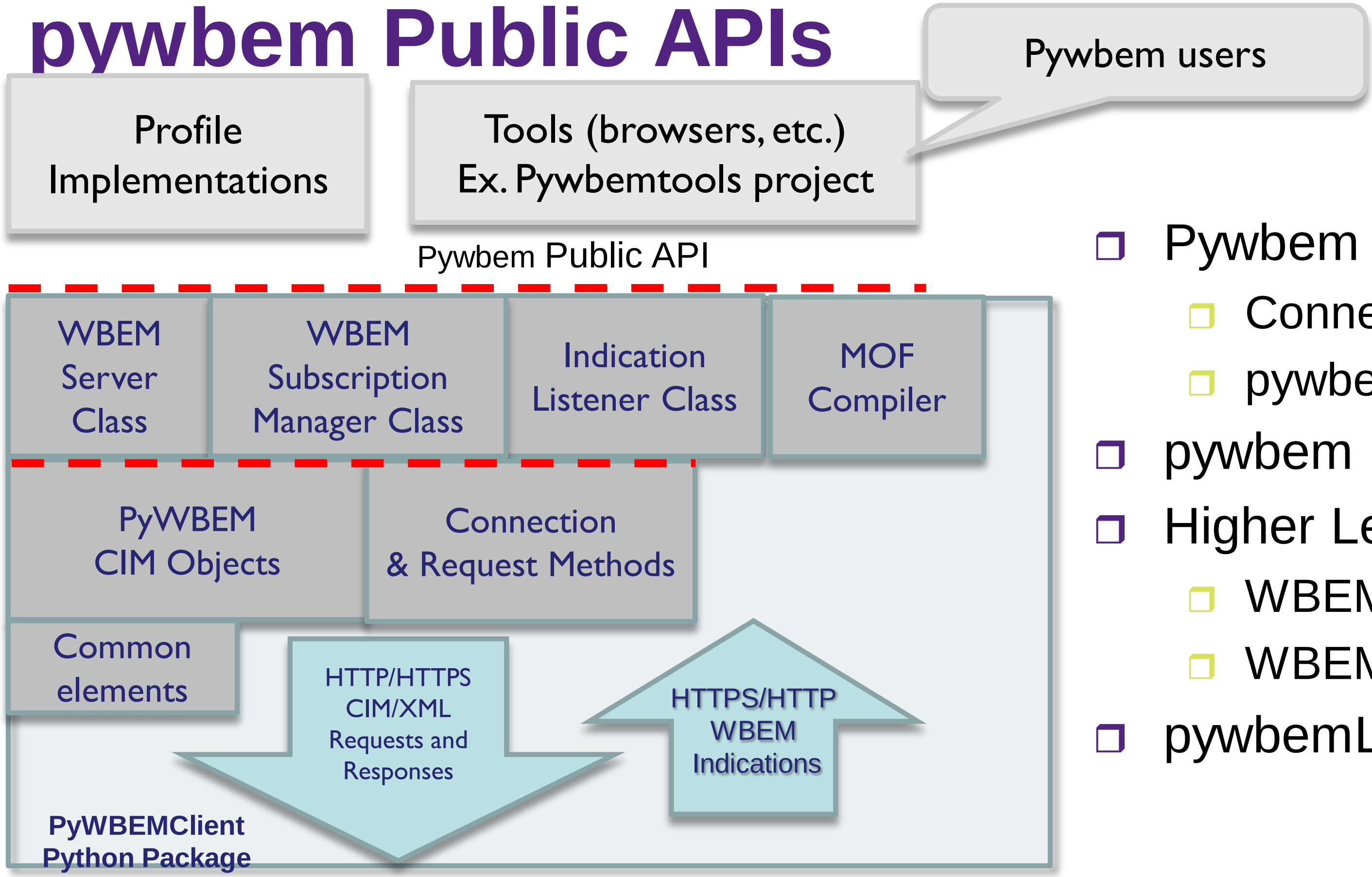
Central and/or Scoping Class definitions

Use Cases and example scripts

pywbem: Registered Profiles and Profiles


WBEMServer.get_profiles
Profiles returned
WBEMServer.get_central_instances
Central instances
Pywbem Client
WbemServer
Central and scoping Class
CIM_ElementConfromsToProfile Association
RegisteredProfile SMI.Array.1.7.0
CIM_ComputerSyst
CIM_ComputerSystem
HostedStoragePool
ReferencedProfile
RegisteredProfile SMI.BlockServices.1.7.0
PrimordialStoragePool Class
Autonomous Profile
Component Profile
Central Class

pywbem Public APIs


Pywbem users
Profile Implementations
Tools (browsers, etc.) Ex. Pywbemtools project
Pywbem Public API
WBEM Server Class
WBEM Subscription Manager Class
Indication Listener Class
MOF Compiler
PyWBEM CIM Objects
Connection & Request Methods
Common elements
HTTP/HTTPS CIM/XML Requests and Responses
HTTPS/HTTP WBEM Indications
PyWBEMClient Python Package

- Pywbem operations
  - Connecting to the Server
  - pywbem WBEM Operations
- pywbem CIM Objects
- Higher Level Classes
  - WBEM Server
  - WBEMSubscriptionManager
- pywbemListener

pywbem and CIM/WBEM

- CIM data types as Python/Pywbem classes:

CIM Data Type	pwbem Implementation
Boolean	Python bool
Char16	Python string
string	Python string
Uint/Sint(8,16,32,64)	pywbem class for each
Real (32,64)	pywbem classes
CIMDateTime	pywbem datetime
Array	python list

- CIMObjects as PyWBEM classes
  - CIMClass
  - CIMClassName
  - CIMInstance
  - CIMInstanceName
  - CIMProperty
  - CIMParameter
  - CIMQualifier
  - CIMQualifierDeclaration
  - CIMProperty
  - CIMParameter

pywbem: WBEM Operations

- Operations for Class, Instance, Qualifier
  - get, enumerate, create, delete, modify
  - InvokeMethod operation executes method on class or instance
  - Operations for references and associators
- Operations support concepts of pull (get partial results) and use of python iterator

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

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

A simple example: Enumerate Instances

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

# Get the names only
inames = CONN.EnumerateInstances (
       ‘CIM_ComputerSystem’)
for iname in inames:
       print(‘%s’% iname)
```

Higher Level objects: WBEM Server

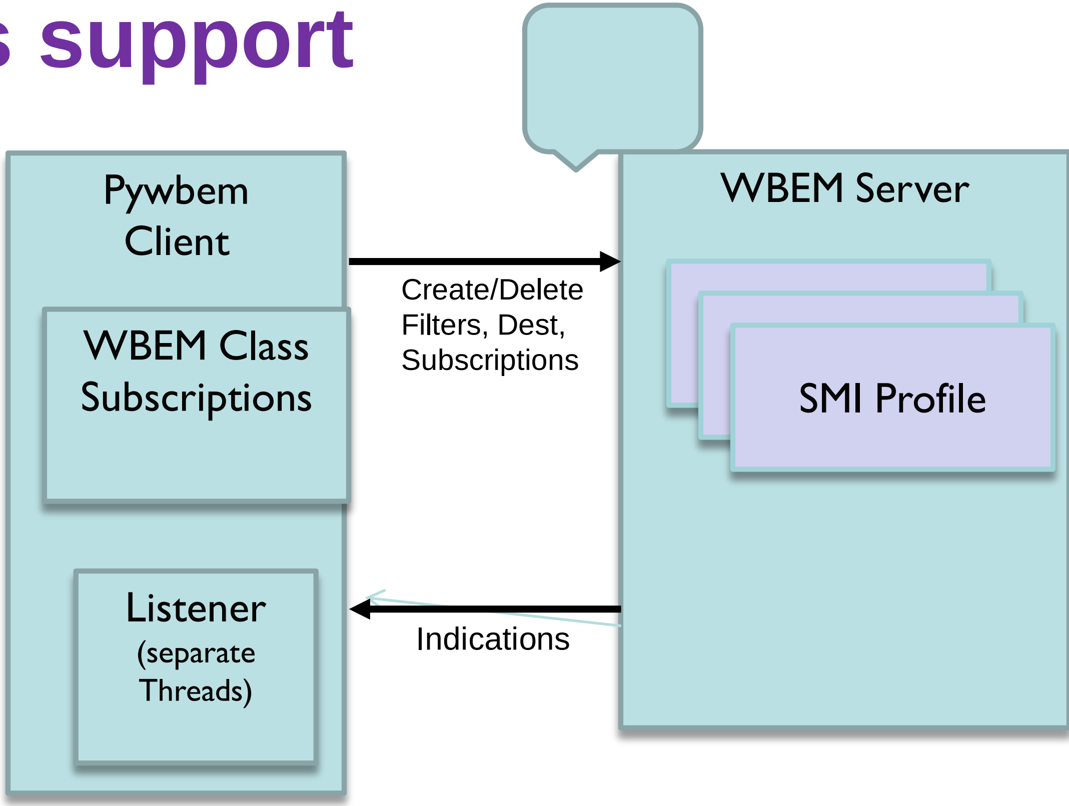
- WBEM Server
  - Interop namespace
  - Namespaces
  - Registered profiles
  - Server brand
  - Server version
  - Central/scoping instances

```
conn = WBEMConnection(…)
svr = WBEMServer(conn)

print('interop %s' svr.interop.ns)
print('ns %s'% svr.namespaces
for inst in svr.profiles:
    org = org_vm.tovalues(…)
    name = inst['RegisteredName'
    vers = inst['RegisteredVersion`]
    print(' %s %s %s' % (org, name,
                         vers))
```

pywbem indications support

- pywbem SubscriptionManager creates/deletes indication subscriptions for multiple servers/listeners
  - Supports persistence differences between server and client
- pywbem listener implements functionality of a WBEM indication listener



Subscription Example

```
From pywbem import WBEMConnection, WBEMServer, SubscriptionManager
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)
```

Add Server

Add Listener for server

Create a Filter

Create Subscription

Remove all

Pywbem and Registered Profiles

- WBEMServer class access profile information
- Get profiles, selected profiles
- Get central and scoping classes

```
conn = WBEMConnection()
svr = WBEMServer(conn)
myprof = svr.get_selected_profiles(“SNIA”,
                                   “Array”
                                   “1.6.1”)
# get central instance of autonomous profile
ci = svr.get_central_instances(myprof[0].path)
```

Using pywbem documentation

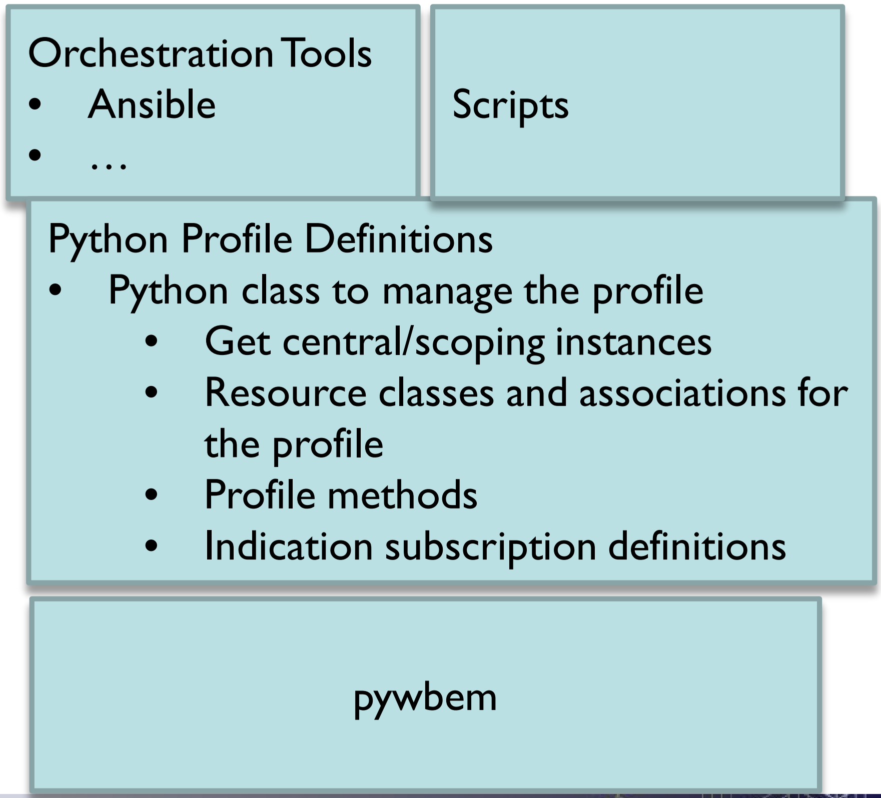
- pywbem well documented
- Documented with sphinx on [readthedocs]
  - APIs
  - Jupyter Notebook examples
  - General usage, installation, developer support
  - Change log

Pywbem Next Steps

- More specific function classes in pywbem
  - Ex. Job Control
- Integrate with specific profile implementations
  - User direct access to profile level required to implement automation, scripts, etc.

Pywbem and automation

- Orchestration tools
  - Ansible, etc.
- Cmd line scripts
  - Python or script languages



Orchestration Tools
- Ansible
- …

Scripts

Python Profile Definitions
- Python class to manage the profile
  - Get central/scoping instances
  - Resource classes and associations for the profile
  - Profile methods
  - Indication subscription definitions

pywbem

How Client Profile Implementation works

- Get the server
- Get the registered profile
- Get the central/scoping classes
- Get the resource class instances
- Execute methods as python methods
- Python methods/functions for more complex actions (use cases), etc. configure, create etc.

pywbem support tools

- pywbem package
  - Operation Logging
  - Last operation capture
  - Operation Recorder
  - MOF Compiler
  - wbemcli
  - WBEM server mock

- pywbem tools package
  - Part of pywbem project
    - pywbemtools repo
  - pywbemcli
    - Cmd line browser
    - Near release now.

pywbem availability and project

- Client package "pywbem" in PyPi
- Client package on some Linux distributions
  - Ex. Ubuntu as python-pywbem
    - **NOTE: Some distros have obsolete version**
- Directly from pywbem project on Github:
  - pywbem is a github project with several code repositories
  - Separate repo for pywbemtools
  - Pywbem.pywbem is client

- Each release is on PyPI (**`pywbem`**)
  - 1 – 2 releases per year
- Documentation in public repository and readthedocs
- 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, new tools, etc.)

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

We are always looking for more ideas, workers, comments.

Any help welcome.

Background Slides

These slides are available for discussions and background.
They are not part of the presentation.

The pywbem WBEM Operations

- Methods of the pywbem class **`WBEMConnection`**
- Instance operations
  - EnumerateInstances
  - EnumerateInstanceNames
  - Associators*
  - AssociatorNames*
  - References*
  - ReferenceNames*
  - GetInstance
  - CreateInstance
  - ModifyInstance
  - DeleteInstance
- InvokeMethod*
- ExecQuery

  * Class and Instance

* Execute on either classes or instances

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

- Pull operations
  - OpenEnumerateInstances
  - OpenEnumerateInstanceNames
  - OpenAssociators
  - OpenReferences
  - OpenExecQuery
  - PullInstancesWithPaths
  - PullInstancePaths
  - PullInstances
  - CloseEnumeration

- Iter…Operations
  - Merge original and pull
  - Pythonic

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

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

- Change for Iter…
- Zero illegal
- Defaults to

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

- 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)})
```