Cisco UCS Python SDK 0.8.3 User Guide

Oct 20, 2014

Table of Contents

1 OVERVIEW 1

2 MANAGEMENT INFORMATION MODEL 3

2.1 MANAGED OBJECTS 3
2.2 REFERENCES TO MANAGED OBJECTS 5
2.3 PROPERTIES OF MANAGED OBJECTS 6
2.4 METHODS 7

3 INSTALLATION 9

3.1 PRE-INSTALL CHECKLIST 9
3.2 INSTALLATION STEPS 9
3.3 GETTING STARTED 9
3.4 WHAT IS NEW IN 0.8.1 10
3.5 WHAT IS NEW IN 0.8.2 10
3.6 WHAT IS NEW IN 0.8.3 10

4 EXAMPLES 12

4.1 GETMANAGEDOBJECT 12
4.2 ADDMANAGEDOBJECT 14
4.3 SETMANAGEDOBJECT 16
4.4 REMOVEMANAGEDOBJECT 17
4.5 COMPAREMANAGEDOBJECT 18
4.6 SYNCMANAGEDOBJECT 20
4.7 CONVERTTOPYTHON 22
4.8 WATCHUCSEVENT 24
4.8.1 AddEventHandler 24
4.8.2 GetEventHandlers 29
4.8.3 RemoveEventHandler 30
4.9 CCO INTEGRATION 31
4.9.1 GetUcsCcoImageList 31
4.9.2 GetUcsCcoImage 31
4.9.3 SendUcsFirmware 33
4.10 GETTECHSUPPORT 35
4.11 BACKUPUCS 38
4.12 IMPORTUCSBACKUP 40
4.13 STARTKVMSESSION 42
4.14 STARTGUISESSION 44
4.15 EXPORTUCSSESSION 45
4.16 IMPORTUCSSESSION 47
4.17 TRANSACTION SUPPORT 49
4.18 METHODS 50
4.18.1 AaaLogin 50
4.18.2 AaaLogout 50
4.18.3 AaaRefresh 50
4.18.4 AaaTokenLogin 50
4.18.5 AaaTokenRefresh 50
4.18.6 ConfigConfMo 50
4.18.7 ConfigConfMoGroup 50
4.18.8 ConfigConfMos 51
4.18.9 ConfigEstimateImpact 52
4.18.10 ConfigFindDependencies 52
4.18.11 ConfigResolveChildren 53
4.18.12 ConfigResolveClass 54
4.18.13 ConfigResolveClasses 54
4.18.14 ConfigResolveDn 54
4.18.15 ConfigResolveDns 55
4.18.16 ConfigResolveParent 55

4.18.17 ConfigScope 55
4.18.18 EventRegisterEventChannel 55
4.18.19 EventRegisterEventChannelResp 55
4.18.20 EventSendEvent 56
4.18.21 EventSendHeartbeat 56
4.18.22 EventSubscribe 56
4.18.23 EventUnRegisterEventChannel 56
4.18.24 FaultAckFault 56
4.18.25 FaultAckFaults 57
4.18.26 FaultResolveFault 57
4.18.27 LsClone 57
4.18.28 LsInstantiateNNamedTemplate 58
4.18.29 LsInstantiateNTemplate 58
4.18.30 LsInstantiateTemplate 58
4.18.31 LsResolveTemplates 58
4.18.32 LsTemplatise 58
4.18.33 StatsClearInterval 59
4.18.34 StatsResolveThresholdPolicy 59

5 SAMPLES 61

REFERENCES 62

1 Overview

Cisco UCS Python SDK is a python module which helps automate all aspects of Cisco UCS management including server, network, storage and hypervisor management.

Bulk of the Cisco UCS Python SDK work on the UCS Manager's Management Information Tree (MIT), performing create, modify or delete actions on the Managed Objects (MO) in the tree. The next chapter provides an overview of the Cisco UCS Management Information Model (MIM).

One of the easiest ways to learn UCS configuration through UCS Python SDK is to automatically generate python script, for configuration actions performed with the UCSM GUI, using ConvertToPython API described in section 4.7.

2 Management Information Model

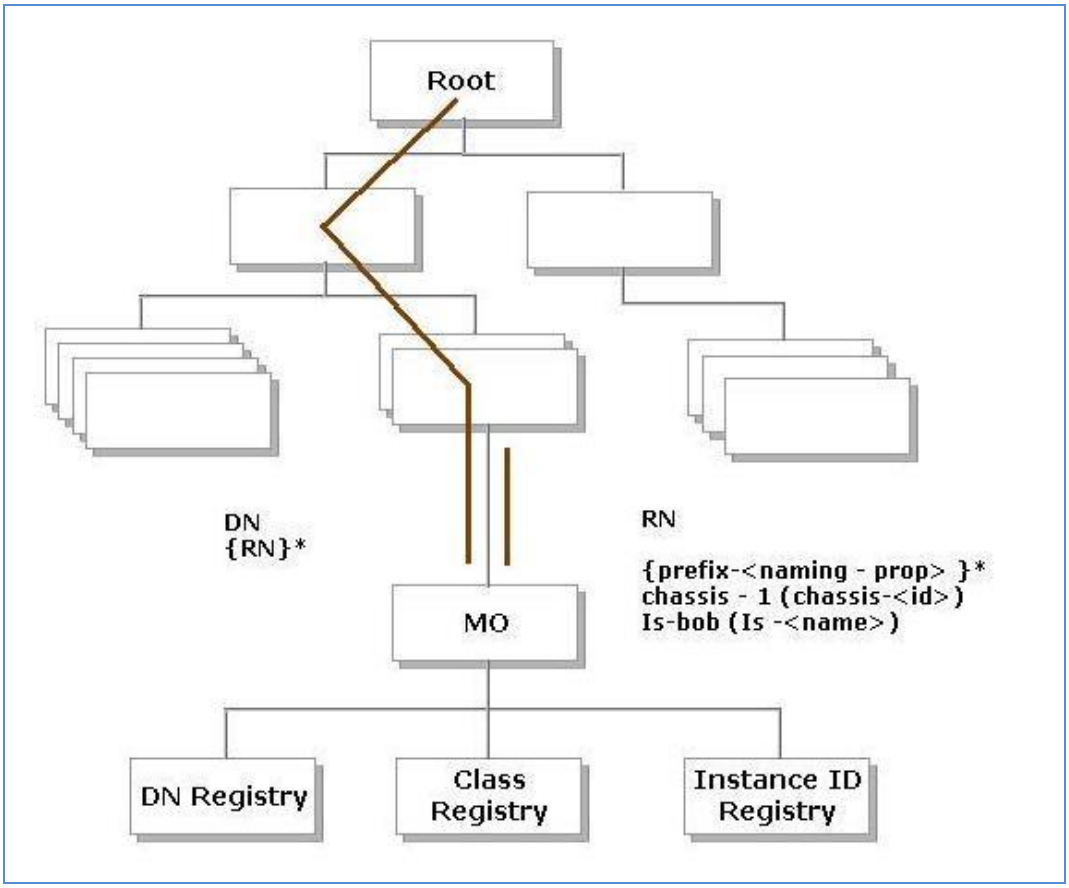
All the physical and logical components that comprise Cisco UCS are represented in a hierarchical Management Information Model (MIM), referred to as the Management Information Tree (MIT). Each node in the tree represents a Managed Object (MO), uniquely identified by its Distinguished Name. (DN)

The figure below illustrates a sample (partial) MIT for three chassis.

```
Tree (topRoot)                Distinguished Name
|— sys                        sys
    |— chassis-1              sys/chassis-1
    |— chassis-2              sys/chassis-2
    |— chassis-3              sys/chassis-3
          |— blade-1          sys/chassis-3/blade-1
          |    |— adaptor-1   sys/chassis-3/blade-1/adaptor-1
          |— blade-2          sys/chassis-3/blade-2
               |— adaptor-1   sys/chassis-3/blade-2/adaptor-1
               |— adaptor-2   sys/chassis-3/blade-2/adaptor-2
```

2.1 Managed Objects

Managed Objects (MO) are abstractions of Cisco UCS resources, such as fabric interconnects, chassis, blades, and rack-mounted servers. Managed Objects represent any physical or logical entity that is configured / managed in the Cisco UCS MIT. For example, physical entities such as Servers, Chassis, I/O cards, Processors and logical entities such as Resource pools, User roles, Service profiles, and Policies are represented as Managed Objects.



Every Managed Object is uniquely identified in the tree with its Distinguished Name (Dn) and can be uniquely identified within the context of its parent with its Relative Name (Rn). The Dn identifies the place of the MO in the MIT. A Dn is a concatenation of all the relative names starting from the root to the MO itself. Essentially, Dn = [Rn]/[Rn]/[Rn]/…/[Rn].

In the example below, Dn provides a fully qualified name for adaptor-1 in the model.

< dn = “sys/chassis-5/blade-2/adaptor-1” />

The above written Dn is composed of the following Rn:

topSystem MO: rn="sys"
equipmentChassis MO: rn="chassis-<id>"
computeBlade MO: rn ="blade-<slotId>"
adaptorUnit MO: rn="adaptor-<id>"

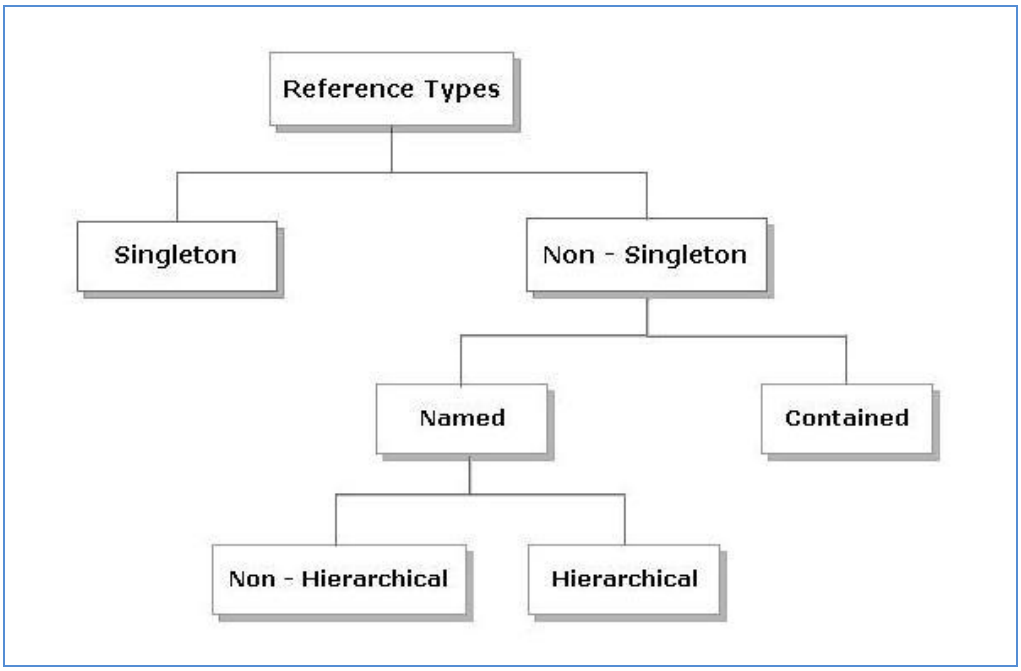
A Relative Name (Rn) ***may*** have the value of one or more of the MO’s properties embedded in it. This allows in differentiating multiple MOs of the same type within the context of the parent. Any properties that form part of the Rn as described earlier are referred to as Naming properties.

For instance, multiple blade MOs reside under a chassis MO. The blade MO contains the blade identifier as part of its Rn (blade-[Id]), thereby uniquely identifying each blade MO in the context of a chassis.

2.2 References to Managed Objects

The contents of the Managed Objects are referred to during the operation of UCS. Some of the MOs are referred to implicitly (PreLoginBanner during login) or as part of deployment of another MO (The Service Profile MO may refer to a template or a VNIC refers to a number of VLAN MOs).

The different types of references can be classified as shown below:



A singleton MO type is found utmost once in the entire MIT and is typically referred to implicitly.

Non-Singleton MO type may be instantiated one or more times in the MIT. In many cases, when an MO refers to another, the reference is made by name. Depending on the type of the referenced MO, the resolution may be hierarchical. For instance, a service profile template is defined under an Org. Since an org may contain sub-orgs, a sub org may have a service profile template defined with the same name. Now, when a service profile instance refers to a service profile template (by name), the name is looked up hierarchically from the org of the service profile instance up until the root org. The first match is used. If no match is found, then the name “default” is looked up in the similar way and the first match is used.

Reference Type	**Example**
Singleton	ChassisDiscoveryPolicy PreLoginBanner
Non-Singleton / Named / Non-Hierarchical	CallHomePolicy
Non-Singleton / Named / Hierarchical	BiosPolicy BootPolicy
Non-Singleton / Contained	BootDefinition under LsServer (ServiceProfile) VnicEtherIf under VnicEther

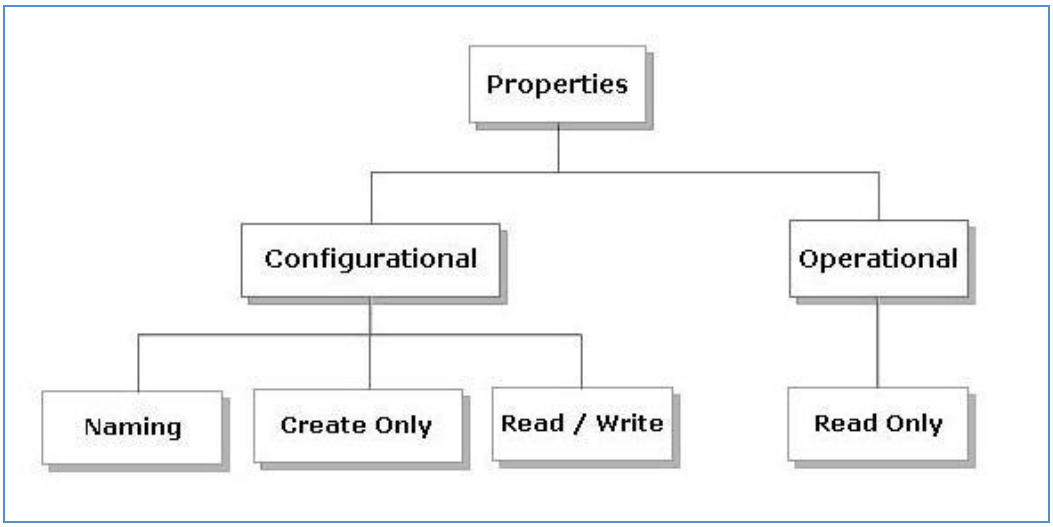
2.3 Properties of Managed Objects

Properties of Managed Objects can be classified as Configuration or Operational.

Configuration properties may be classified as:

- Naming properties: Form part of the Rn. ***Needs*** to be specified only during MO creation and cannot be modified later.
- Create-Only properties: ***May*** be specified only during MO creation and cannot be modified later. If the property is not specified, a default value is assumed.
- Read / Write properties: ***May*** be specified during MO creation and can also be modified subsequently.

Operational properties indicate the current status of the MO / system and are hence read-only.



The table below lists the examples of the various property types.

Property Type	**Example**
Naming	Name in LsServer (Service Profile MO)
Create-Only	Type in LsServer
Read / Write	Description in LsServer
Read-Only	OperState in LsServer

2.4 Methods

Methods are Cisco UCS XML APIs, used to manage and monitor the system. There are methods supported for:

- Authentication
- AaaLogin
- AaaRefresh
- AaaLogout
- Configuration
- ConfigConfMo(s)
- LsClone
- LsInstantiate*
- FaultAckFaults
- Query
- ConfigResolveDn(s)
- ConfigResolveClass(es)
- ConfigResolveChildren
- Event Monitor
- EventSubscribe

The class query methods (ConfigResolveClass(es), ConfigResolveChildren) allow a filter to be specified so that a specific set of MOs are matched and returned by the method.

The supported filters are:
Property Filters:

- allbits – Match if all specified values are present in a multi-valued property
- anybit – Match if any of the specified values are present in a multi-valued property
- bw – Match if the property's value lies between the two values specified
- eq – Match if property's value is the same as the specified value
- ge – Match if property's value is greater than or equal to the specified value
- gt - Match if property's value is greater than the specified value
- le – Match if property's value is lesser than or equal to the specified value
- lt – Match if property's value is lesser than the specified value
- ne – Match if property's value is not equal to the specified value
- wcard – Match if property's value matches the pattern specified

3 Installation

3.1 Pre-Install Checklist

- Ensure you have Python v2.4 or above version of v2.x.
- Supported OS – Cygwin, RHEL 5.x, and other versions of Linux/Unix with Python version v2.4 or above version of v2.x.
- Ensure you have removed the existing Python UCSM Sdk from the installation directory. E.g. in Windows environment, you have installed UcsSdk in Python 2.7. The default installation directory will be "C:\Python27\Lib\site-packages\UcsSdk". So first remove UcsSdk directory and then install the fresh UcsSdk.

3.2 Installation Steps

- Download the source distribution of Python Ucs SDK
- Un-tar the package and execute the following from the package
    - python setup.py build
    - sudo python setup.py install

3.3 Getting Started

1. **Launch python**
2. **Connect to a UCS system.**

```
from UcsSdk import *
handle = UcsHandle()
handle.Login(<ip or hostname>)
handle.Login(<ip or hostname>, username="<username>", password="<password>")
handle.Login(<ip or hostname>, username="<username>", password="<password>",
noSsl=False, port=443, dumpXml=YesOrNo.TRUE)
```

3. **Connect to a multiple UCS system.**

```
from UcsSdk import *
handle1 = UcsHandle()
handle1.Login(<ip or hostname>)
handle2 = UcsHandle()
handle2.Login(<ip or hostname>)
```

4. **Disconnect.**

```
handle.Logout()
```

A sample script to Login and Logout is shown below:

```
from UcsSdk import *

if __name__ == "__main__":
	try:
		handle = UcsHandle()
		handle.Login("0.0.0.0","username","password")
		handle.Logout()

	except Exception, err:
		print "Exception:", str(err)
		import traceback, sys
		print '-'*60
		traceback.print_exc(file=sys.stdout)
		print '-'*60
```

3.4 What is new in 0.8.1

- Fixed the error thrown when using python version 2.7.6
- Errors from communication with UCS are wrapped in `UcsException` and all validation errors are thrown as `UcsValidationException` for custom error handling.

3.5 What is new in 0.8.2

- Refreshed with UCSM Schema 2.2(2c)
- Removed keys() function while iterating dictionary to improve execution speed.
- Fixed "AttributeError", which happens when configuring an unknown MO.

3.6 What is new in 0.8.3

- Embedded dynamic import support to decrease memory utilization.
- XmlParser changed to cElementTree to improve performance.
- Modified URI method to verify correct port.
- Modified Login API to prioritize connection via http or https now depends on both port and noSsl flag rather only noSsl flag.
- Fixed StartGuiSession for jnlpfile variable error and added support to launch GUI in debug mode for java version > 1.7.45
- Fixed GetJavaInstallation method to display JAVA_HOME path in case of an exception.
- Modified WatchUcsGui in order to support multiline query via ConvertToPython API.

3.6.1 Migrating your scripts from previous releases of SDK to release 0.8.3:

If you already have scripts written using earlier versions of UCSM SDK, please make sure you make the following changes to make them work with release 0.8.3.

In previous releases of the SDK, all MO definition were in a single file Mos.py. Now each mo has its own mo definition file. Hence you need to individually import the definition of respective Mos.
This is done to bring down the memory usage and improve the SDK performance.

For example, to use MO definition as "EquipmentChassis.ClassId()", you need to import the respective module as below:

```
from UcsSdk.MoMeta.EquipmentChassis import EquipmentChassis
```

However if you have many MOs in your script and you do not want to write import statements for individual MOs then you can use below initialization method in your script to import all MO definitions.

```
def loadAllMo():
    import os
    import UcsSdk
    exports = []
    globals_, locals_ = globals(), locals()
    package_path = os.path.dirname(UcsSdk.__file__)
    package_name = os.path.basename(package_path)
    mometa_path = os.path.join(package_path,'MoMeta')
    for filename in os.listdir(mometa_path):
        modulename, ext = os.path.splitext(filename)
        if not modulename.endswith('Meta') and ext in ('.py') and modulename
!= '__init__':
            subpackage = '{}.{}'.format(package_name+".MoMeta", modulename)
            module = __import__(subpackage, globals_, locals_, [modulename])
            modict = module.__dict__
            names = [name for name in modict if name[0] != '_']
            exports.extend(names)
            globals_.update((name, modict[name]) for name in names)


if __name__ == "__main__":
    loadAllMo()
    print LsServer.ClassId()
```

4 Examples

4.1 GetManagedObject

This Operation gets a Managed Object from UCS.

```
GetManagedObject(inMo, classId, params=None, inHierarchical=False,)
```

Mandatory Input sets:

(1) ClassId
(or)
(2) "Dn" property in params

Parameter description:

1. inMo:
   - Mandatory parameter:
   - It should be **None** unless a user wants to define a parent scope.
   - It is a list containing single or multiple MOs.
   - If provided, it acts as a parent for the present operation.

2. classId:
   - ClassId of MO/MOs to get

3. params:
   - Optional parameter
   - Semicolon (;) separated list of key/value pairs(key=value), that are used as filters for selecting specific Managed Objects. The key should be a valid property of the Managed Object to be retrieved.

4. inHierarchical:
   - Optional parameter
   - Explores hierarchy if true, else returns Managed Objects at a single level.

5. dumpXml:
   - Optional parameter
   - Outputs the xml requests & responses involved. This is useful for debugging.

 # ClassId Parameter Set

```
from UcsSdk import *
from UcsSdk.MoMeta.OrgOrg import OrgOrg

if __name__ == "__main__":
        try:
                handle = UcsHandle()
                handle.Login("0.0.0.0","username","password")

                # returns a list of all the org Mos at Level 1
                getRsp = handle.GetManagedObject(None, OrgOrg.ClassId(),{OrgOrg.LEVEL:"1"})

                handle.Logout()

        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                print '-'*60
```

 # DN Parameter Set

```
from UcsSdk import *

if __name__ == "__main__":
        try:
                handle = UcsHandle()
                handle.Login("0.0.0.0","username","password")

                getRsp = handle.GetManagedObject(None, None,{OrgOrg.DN:"org-root"})

                handle.Logout()

        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                print '-'*60
```

4.2 AddManagedObject

This Operation adds a Managed Object to UCS.

```
AddManagedObject(inMo, classId, params=None, modifyPresent=False, dumpXml=None)
```

Mandatory Input set:

(1) ClassId

Parameter description:

1. inMo:
   - Mandatory parameter.
   - It should be **None** unless a user wants to define a parent scope.
   - It is a list containing single or multiple MOs.
   - If provided, it acts as a parent for the present operation.

2. classId:
   - ClassId of MO/MOs to get

3. params:
   - Optional parameter
   - Semicolon (;) separated list of key/value pairs(key=value), that are used as filters for selecting specific Managed Objects. The key should be a valid property of the Managed Object to be retrieved.

4. modifyPresent:
   - Optional parameter
   - The ModifyPresent option ensures that the add API modify the MO, if it already exits, instead of returning an error.

5. dumpXml:
   - Optional parameter
   - Outputs the xml requests & responses involved. This is useful for debugging.

 # ClassId Parameter Set

```
from UcsSdk import *
from UcsSdk.MoMeta.OrgOrg import OrgOrg
from UcsSdk.MoMeta.LsServer import LsServer

if __name__ == "__main__":
        try:
                handle = UcsHandle()
                handle.Login("0.0.0.0","username","password")

                # returns a list of all the org Mos at Level 1
                getRsp = handle.GetManagedObject(None, OrgOrg.ClassId(),{OrgOrg.LEVEL:"1"})

                # adds a service profile sp_name with-in every Org returned in the previous operation
                addRsp = handle.AddManagedObject(getRsp, LsServer.ClassId(), { LsServer.NAME :"sp_name"})

                handle.Logout()

        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                print '-'*60
```

4.3 SetManagedObject

This Operation modifies a Managed Object in UCS.

```
SetManagedObject(inMo, classId=None, params=None, dumpXml=None)
```

Parameter description:

1. inMo:
    - Mandatory parameter.
    - It should be **None** unless a user wants to define a parent scope.
    - It is a list containing single or multiple MOs.
    - If provided, it acts as a parent for the present operation.

2. classId:
    - ClassId of MO/MOs to get

3. params:
    - Optional parameter
    - Semicolon (;) separated list of key/value pairs(key=value), that are used as filters for selecting specific Managed Objects. The key should be a valid property of the Managed Object to be retrieved.

4. dumpXml:
    - Optional parameter
    - Outputs the xml requests & responses involved. This is useful for debugging.

```
from UcsSdk import *
from UcsSdk.MoMeta.OrgOrg import OrgOrg
from UcsSdk.MoMeta.LsServer import LsServer

if __name__ == "__main__":
        try:
                handle = UcsHandle()
                handle.Login("0.0.0.0","username","password")

                # returns a list of all the org Mos at Level 1
                getRsp = handle.GetManagedObject(None, OrgOrg.ClassId(),{OrgOrg.LEVEL:"1"})

                # adds a service profile sp_name with-in every Org returned in the previous operation
                addRsp = handle.AddManagedObject(getRsp, LsServer.ClassId(), { LsServer.NAME :"sp_name"})

                # sets the descriptor of every mo returned by AddManagedObject
                setRsp = handle.SetManagedObject(addRsp, LsServer.ClassId(), {LsServer.DESCR:"sp_description"})


                handle.Logout()

        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                print '-'*60
```

4.4 RemoveManagedObject

This Operation removes a Managed Object from UCS.

```
RemoveManagedObject(inMo, classId=None, params=None, dumpXml=None)
```

Mandatory Input set:

(1) inMOs containing MO(s) to be deleted
(or)
(2) classId with "Dn" property in params

Parameter description:

1. inMo:
   - Mandatory parameter.
   - It should be **None** unless a user wants to define a parent scope.
   - It is a list containing single or multiple MOs.
   - If provided, it acts as a parent for the present operation.

2. classId:
   - ClassId of MO/MOs to get

3. params:
   - Optional parameter
   - Semicolon (;) separated list of key/value pairs(key=value), that are used as filters for selecting specific Managed Objects. The key should be a valid property of the Managed Object to be retrieved.

4. dumpXml:
   - Optional parameter
   - Outputs the xml requests & responses involved. This is useful for debugging.

```
from UcsSdk import *
from UcsSdk.MoMeta.OrgOrg import OrgOrg
from UcsSdk.MoMeta.LsServer import LsServer

if __name__ == "__main__":
        try:
                handle = UcsHandle()
                handle.Login("0.0.0.0","username","password")

                # returns a list of all the org Mos at Level 1
                getRsp = handle.GetManagedObject(None, OrgOrg.ClassId(),{OrgOrg.LEVEL:"1"})

                # adds a service profile sp_name with-in every Org returned in the previous operation
                addRsp = handle.AddManagedObject(getRsp, LsServer.ClassId(), { LsServer.NAME :"sp_name"})

                # sets the descriptor of every mo returned by AddManagedObject
                setRsp = handle.SetManagedObject(addRsp, LsServer.ClassId(), {LsServer.DESCR:"sp_description"})

                # removes all the service profiles we had added
                removeRsp = handle.RemoveManagedObject(addRsp)
                handle.Logout()
        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                print '-'*60
```

4.5 CompareManagedObject

This Operation compares managed objects.

```
CompareMangedObject(referenceObject, differenceObject,
excludeDifferent=YesOrNo.FALSE, includeEqual=YesOrNo.FALSE,
noVersionFilter=YesOrNo.FALSE, includeOperational=YesOrNo.FALSE, xlateOrg=None,
xlateMap=None)
```

Mandatory Input set:

(1) referenceObject Objects used as a reference for comparison.

(2) differenceObject Objects that are compared to the reference objects.

Parameter description:

1. referenceObject:
   Mandatory parameter. Objects used as a reference for comparison.

2. differenceObject:
   Mandatory parameter. Objects that are compared to the reference objects.

3. `excludeDifferent`:
   Optional parameter. Displays only the properties of compared objects that are equal.

4. `includeEqual`:
   Optional parameter. Displays properties of compared objects that are equal. By default, only properties that differ between the reference and difference objects are displayed.

5. `noVersionFilter`:
   Optional parameter. Ignore minimum version in properties.

6. `includeOperational`:
   Optional parameter. Specified to include operational properties.

7. `xlateOrg`:
   Translation org to be used.

8. `xlateMap`:
   Translation map with DNs of entities that needs to be translated.

```
from UcsSdk import *
from UcsSdk.MoMeta.LsServer import LsServer

if __name__ == "__main__":
        try:
                handle1 = UcsHandle()
                handle1.Login("1.1.1.1","username2","password2")
                handle2 = UcsHandle()
                handle2.Login("2.2.2.2","username2","password2")

                # returns the service profile Mos according to the provided Dn.
                sp1 = handle1.GetManagedObject(None, LsServer.ClassId(), { LsServer.DN:"org-root/ls-abc" })
                sp2 = handle2.GetManagedObject(None, LsServer.ClassId(), { LsServer.DN:"org-root/ls-def" })

                compareObj = CompareManagedObject(sp1, sp2)
                if (compareObj != None):
                        WriteObject(compareObj)


                handle1.Logout()
                handle2.Logout()

        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                print '-'*60
```

4.6 SyncManagedObject

This Operation syncs Managed Object of type ManagedObject. This operation takes the difference object (output of CompareManagedObject) and applies the differences on reference Managed Object.

```
SyncManagedObject(difference, deleteNotPresent=False, noVersionFilter=False,
dumpXml=None)
```

Mandatory Input set:

(1) `difference` containing difference object to be synced

Parameter description:

1. `difference:`
   Mandatory parameter. Specifies the Difference object (output of ComparesManagedObject) which has differences of the properties of two or more Managed Objects.

2. `deleteNotPresent:`
   Mandatory parameter. If specified, any missing MOs in reference Managed Object set will be deleted.

3. `noVersionFilter:`
   Optional parameter. If specified, minversion for Mos or properties to be added in reference Managed Object will not be checked.

4. `dumpXml:`
   Optional parameter
   Outputs the xml requests & responses involved. This is useful for debugging.

```
from UcsSdk import *
from UcsSdk.MoMeta.OrgOrg import OrgOrg
from UcsSdk.MoMeta.LsServer import LsServer

if __name__ == "__main__":
        try:
                handle1 = UcsHandle()
                handle1.Login("1.1.1.1","username2","password2")
                handle2 = UcsHandle()
                handle2.Login("2.2.2.2","username2","password2")

                # returns the service profile Mos according to the provided Dn.
                sp1 = handle1.GetManagedObject(None, LsServer.ClassId(), { LsServer.DN:"org-root/ls-abc" })
                sp2 = handle2.GetManagedObject(None, LsServer.ClassId(), { LsServer.DN:"org-root/ls-def" })

                #Compares the two service profiles
                compareObj = CompareManagedObject(sp1, sp2)
                if (compareObj != None):
                        WriteObject(compareObj)

                #Sync the difference between the two service profiles
                syncObj = handle1.SyncManagedObject(compareObj, dumpXml=YesOrNo.TRUE)
                if (syncObj != None):
                        WriteObject(syncObj)

                handle1.Logout()
                handle2.Logout()

        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                print '-'*60
```

4.7 ConvertToPython

This Operation generates the respective python-ucs script for the operation performed in UCSM GUI. In addition, ConvertToPython operation generates python-ucs script using stand-alone XML request or a bunch of XML request in a file or from UCSM GUI log.

```
ConvertToPython(xml=False,request=None,guiLog=False,path=None,literalPath=None,dumpXml=False,dumpToFile=False,dumpFilePath=None)
```

Parameter description:

1. xml:
   - Optional parameter (boolean)
   - It should be **False** unless a user wants to generate python-ucs script using xmlrequest or xmlfile.

2. request:
   - Optional parameter (string)
   - It should be **None** unless parameter 'xml' is **True**

3. guiLog:
   - Optional parameter (boolean)
   - It should be **False** unless a user wants to generate python-ucs script using UCSM GUI log file.

   Note: If both parameter 'xml' and 'guilog' are True, preference goes to 'xml' parameter.

4. path:
   - Optional parameter (string)
   - It should be **None** unless a user wants to generate python-ucs script using UCSM GUI logfile or XML logfile.

5. literalPath:
   - Optional parameter (string)
   - It should be **None** unless a user wants to generate python-ucs script using UCSM GUI logfile or XML logfile.

6. dumpXml:
   - Optional parameter (boolean)
   - It should be **False** unless a user wants to display xml used to generate python-ucs script.

7. dumpToFile:
   - Optional parameter (boolean)
   - It should be **False** unless a user wants to generate python-ucs script in a file not on standard output or console.
8. dumpFilePath:
   - Optional parameter (string)
   - It should be **None** unless a user wants to generate python-ucs script in a file not on standard output or console.

 # Different forms of ConvertToPython

```
from UcsSdk import *

if __name__ == "__main__":
        try:

                # variable declaration

                outfilepath = r" C:\Users\test\out.txt"

                xmlrequest = '''
                        <configConfMos inHierarchical="false">
                          <inConfigs>
                           <pair key="sys/chassis-1/blade-2/mgmt/log-SEL-0">
                             <sysdebugMEpLog adminState="backup" dn="sys/chassis-1/blade-2/mgmt/log-SEL-0">
                             </sysdebugMEpLog>
                           </pair>
                          </inConfigs>
                        </configConfMos> '''

                xmlfilepath= r"C:\Users\test\test.xml"

                guilogpath = r" C:\Users\ragupta4\AppData\LocalLow\Sun\Java\Deployment\log\.ucsm\centrale_11.log"


                ## As per the usage, choose the respective ConvertToPython format (As shown below).

                # generate script for the specified actions in UCSM GUI
                ConvertToPython()

                # generate script for the specified actions in UCSM GUI and display xml used in script generation.
                ConvertToPython(dumpXml=True)

                # generate script for the specified actions in UCSM GUI and redirect script output to a file.
                ConvertToPython(dumpToFile=True, dumpFilePath= outfilepath)

                # generate script using XML Request.
                ConvertToPython(xml=True, request= xmlrequest)

                # generate script using  file containing bunch of XML Request.
                ConvertToPython(xml=True, path= xmlfilepath)

                # generate script using  file containing UCSM GUI logs.
                ConvertToPython(guiLog =True, path= guilogpath)


        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                print '-'*60
```

4.8 WatchUcsEvent

Ucs Python SDK provides the capability of subscribing to UCS event streams. This functionality is covered as parts of three APIs that are mentioned below. These APIs also have the capability of filtering out events specific to a user's demand and taking custom actions on them.

4.8.1 AddEventHandler

This method creates an event handle for the user to monitor an event stream generated by UCS Manager. Depending on the parameters, the API can either notify for all events; for events of a selected classId or for events of a specific property change of a selected managed object.
A user can optionally pass a callback method which should be invoked on specific events. If a callback is not passed, a default callback is inserted by the SDK.

```
AddEventHandler(classId = None, managedObject = None, prop = None, successValue =
[], failureValue = [], transientValue = [], pollSec = None, timeoutSec = None,
callback = None)
```

Mandatory Input sets:

(1) `AddEventHandler(timeoutSec<optional>, callback<optional>)`
(or)

(2) `AddEventHandler(classId<mandatory>, timeoutSec<optional>, callback<optional>)`
(or)

(3) `AddEventHandler(managedObject<mandatory>, prop<mandatory>, successValue<mandatory>, failureValue<optional>, transientValue<optional>, pollSec<optional>, timeoutSec<optional>, callback<optional>)`

Parameter description:

1. `classId`:
   - Watch UCS system for events of specified class name.
2. `managedObject`:
   - It should be a single Mo and not list of Mos.
   - Watch UCS system for events of specified Managed Object.
3. `prop`:
   - Watch UCS system for events of particular property of a Managed Object.
4. `successValue`:
   - List of comma (,) separated success values.
   - Watch UCS system for events of particular property to attain specified success value.

5. failureValue:
    - List of comma (,) separated failure values.
    - Watch UCS system for events of particular property to attain specified failure value.

6. transientValue:
    - List of comma (,) separated transient values.
    - Watch UCS system for events of particular property to attain specified transient value.

7. pollSec:
    - Specifies the event polling time in seconds and checks for success value for the property.

8. timeoutSec:
    - Watch UCS system for events for specified time.

9. callback:
    - Callback function that user can pass to do operation.

 # Different forms of AddEventHandler

 # None Parameter Set

```
from UcsSdk import *
import time

if __name__ == "__main__":
        try:

                # variable declaration

                ucsm_ip = '0.0.0.0'
                user = 'username'
                password = 'password'

                def callback_allevents(mce):
                        print 'Received a New Event with ClassId: ' + str(mce.mo.classId)
                        print "ChangeList: ", mce.changeList
                        print "EventId: ", mce.eventId


                handle = UcsHandle()
                handle.Login(ucsm_ip,user, password)


                ## As per the usage, choose the respective AddEventHandler format (As shown below).

                # It will watch UCS system for all events.
                allEventWatcher = handle.AddEventHandler()

                # It will watch UCS system for 100 sec.
                eventWatcherTimeOut = handle.AddEventHandler(timeoutSec = 60)

                # It will trigger callback function “callback_all” for the respective event.
                eventWatcherCB = handle.AddEventHandler( callBack= callback_allevents)


                time.sleep(90)
                handle.RemoveEventHandler(allEventWatcher)
                handle.RemoveEventHandler(eventWatcherTimeOut)
                handle.RemoveEventHandler(eventWatcherCB)

                # loop that keeps the script running for us to get events/callbacks
                while True:
                        time.sleep(5)


                handle.Logout()


        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                print '-'*60
                handle.Logout()
```

 # classId Parameter Set

```
from UcsSdk import *
from UcsSdk.MoMeta.LsServer import LsServer
import time

if __name__ == "__main__":
        try:

                # variable declaration

                ucsm_ip = '0.0.0.0'
                user = 'username'
                password = 'password'

                def callback_lsServerEvents(mce):
                        print 'Received a New Service Profile Event: ' + str(mce.mo.classId)
                        WriteObject(mce.mo)

                # It will watch the UCS system for service profile events.
                eventWatcherCB = handle.AddEventHandler(classId = LsServer.ClassId(), callBack=
                callback_lsServerEvents)

                time.sleep(60)
                handle.RemoveEventHandler(eventWatcherCB)

                # loop that keeps the script running for us to get events/callbacks
                while True:
                        time.sleep(5)


                handle.Logout()


        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                print '-'*60
                handle.Logout()
```

 # managedObject Parameter Set

```
from UcsSdk import *
from UcsSdk.MoMeta.LsServer import LsServer
import time

if __name__ == "__main__":
        try:

                # variable declaration

                ucsm_ip = '0.0.0.0'
                user = 'username'
                password = 'password'

                def calback_mo_successValue(mce):
                        print 'Received Service Profile Event with classId: ' + str(mce.mo.classId)
                        print 'New Value of Property USR_LBL: ' + mce.mo.UsrLbl
                        print "ChangeList: ", mce.changeList
                        print "EventId: ", mce.eventId

                def calback_mo_pollSec(mce):
                        print 'Polling Completed for Managed Object ' + str(mce.mo.classId)
                        print 'New Value of Property USR_LBL: ' + mce.mo.UsrLbl
                        print "ChangeList: ", mce.changeList
                        print "EventId: ", mce.eventId



                sp = handle.GetManagedObject(None, LsServer.ClassId(), {LsServer.DN:"org-root/ls-ServiceProfile"})

                # It will watch the UCS system until the value USR_LBL property of respective SP becomes “Success”
                eventWatcherMO = handle.AddEventHandler(managedObject = sp[0], prop=NamingPropertyId.USR_LBL,
                successValue=["Success"])


                # It will watch the UCS system until the value USR_LBL property of respective SP becomes “Success” and
                call the specific callback function.
                eventWatcherMOCB = handle.AddEventHandler(managedObject = sp[0],
                prop=NamingPropertyId.USR_LBL,  successValue=["Success"], callBack= calback_mo_successValue)

                # It will poll the UCS system every 5sec until the value USR_LBL property of respective SP becomes
                “Success”
                eventWatcherMO_pollSec = handle.AddEventHandler(managedObject = sp[0],
                prop=NamingPropertyId.USR_LBL, successValue=["Success"], callBack= calback_mo_pollSec, pollSec=5)

                # loop that keeps the script running for us to get events/callbacks
                # User needs to manually exit the script here.
                while True:
                        time.sleep(5)


                handle.Logout()


        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                print '-'*60
                handle.Logout()
```

4.8.2 GetEventHandlers

This method returns a list of all the active event handlers.

`GetEventHandlers()`

```
from UcsSdk import *

if __name__ == "__main__":
        try:
                handle = UcsHandle()
                handle.Login("0.0.0.0","username","password")

                allEventWatcher = handle.AddEventHandler()
                ehs = handle.GetEventHandlers()
                for eh in ehs:
                        print eh
                        handle.RemoveEventHandler(eh)

                handle.Logout()

        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                print '-'*60
                handle.Logout()
```

4.8.3 RemoveEventHandler

This method removes the provided event handler.

`RemoveEventHandler(wb)`

Parameter description:

1. `wb`:
    - wb(watchBlock) to remove.

```
from UcsSdk import *

if __name__ == "__main__":
        try:
                handle = UcsHandle()
                handle.Login("0.0.0.0","username","password")

                allEventWatcher = handle.AddEventHandler()
                print handle.GetEventHandlers()
                handle.RemoveEventHandler(allEventWatcher)

                handle.Logout()

        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                print '-'*60
                handle.Logout()
```

4.9 CCO Integration

Ucs Python SDK provides the capability of handling CCO Image. This functionality is covered as parts of three APIs that are mentioned below.

4.9.1 GetUcsCcoImageList

This operation returns a list of CCO Images.

```
GetUcsCcoImageList(username=None, password=None, mdfIdList=[])
```

Mandatory Input sets:

(1) `GetUcsCcoImageList(username=<username>)`
(or)
(2) `GetUcsCcoImageList(username=<username>, mdfIdList=<list of mdfIds>)`

Parameter description:

1. `username`:
   - Mandatory parameter (string).
   - Username of the user who has an access to CCO image server.
2. `password`:
   - Mandatory parameter (string).
   - Password of the user who has an access to CCO image server.
3. `mdfIdList`:
   - Optional parameter (list).
   - List of specific mdfIds.

4.9.2 GetUcsCcoImage

This operation downloads a specific CCO image on local machine.

```
GetUcsCcoImage(image=None, path=None)
```

Parameter description:

1. `image`:
   - Mandatory parameter (Object).
   - UcsCcoImage Object.
2. `path`:
   - Mandatory parameter (string).
   - Specify the path where you want to download the CCO Image.

```
if __name__ == "__main__":
        try:
                from __init__ import *
                CcoImageList = GetUcsCcoImageList('<username>')

                for image in CcoImageList:
                   if image.imageName == "<imagename>":
                      ccoImage = image
                      break

                imageDownloadPath = r"<path>"
                GetUcsCcoImage(ccoImage, imageDownloadPath)

        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                print '-'*60
```

4.9.3 SendUcsFirmware

This operation uploads a specific CCO Image on UCS.

```
SendUcsFirmware(path=None,dumpXml=False)
```

Mandatory Input sets:

(1) `SendUcsFirmware (path=<CCO Image Path>)`

Parameter description:

1. `path`:
   - Mandatory parameter (string).
   - Specify the path of the image to be uploaded.
2. dumpXml:
   - Optional parameter (boolean)
   - It should be **False** unless a user wants to display xml used to generate python-ucs script.

```
from UcsSdk import *
if __name__ == "__main__":
        try:
                handle = UcsHandle()
                handle.Login("0.0.0.0","username","password")

                imagePath = r"C:\Images\ccoimage.bin"
                handle.SendUcsFirmware(path=imagePath)

                handle.Logout()
        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                print '-'*60
                handle.Logout()
```

4.10 GetTechSupport

This operation will create and download the technical support data for the respective UCSM.

GetTechSupport(pathPattern , ucsManager=False, ucsMgmt=False, chassisId=None, cimcId=None, adapterId=None, iomId=None, fexId=None, rackServerId=None, rackAdapterId=None, timeoutSec=600, removeFromUcs=False, dumpXml=None)

Mandatory Input sets:

(1) GetTechSupport(pathPattern , ucsManager=True)
(or)
(2) GetTechSupport(pathPattern , ucsMgmt=True)
(or)
(3) GetTechSupport(pathPattern , chassisId=None, cimcId=None, adapterId=None<Optional>)
(or)
(4) GetTechSupport(pathPattern , chassisId=None, iomId=None)
(or)
(5) GetTechSupport(pathPattern , fexId=None)
(or)
(6) GetTechSupport(pathPattern , rackServerId =None, rackAdapterId=None<Optional>)

Parameter description:

1. pathPattern:
   - Mandatory parameter (string).
   - Path of the Tech Support file.
     Note: Should be a tar file.
2. ucsManager:
   - Optional parameter (boolean)
   - It should be True if user wants technical support data for entire UCSM instance.
3. ucsMgmt:
   - Optional parameter (boolean)
   - It should be True if user wants technical support data for the UCSM management services (excluding fabric interconnects).
4. chassisId:
   - Mandatory parameter (string).
   - Id of a chassis
5. cimcId:
   - Mandatory parameter (string).
   - CIMC ID for a specific chassis. Can be “all” also.
6. adapterId:
   - Optional parameter (number).
   - CIMC Adapter ID for a specific chassis. Can be “all” also.
7. iomId:
   - Mandatory parameter (number).
   - IOM ID for a specific chassis. Can be “all” also.
8. fexId:
   - Mandatory parameter (number).
   - Id of a fabric extender.

9. rackServerId:
    - Mandatory parameter (number).
    - Id of a rack server.
10. rackAdapterId:
    - Optional parameter (number).
    - Adapter Id for a specific rack server. Can be “all’ also.
11. timeoutSec:
    - Optional parameter (string).
    -
12. removeFromUcs:
    - Optional parameter (boolean)
13. dumpXml:
    - Optional parameter (boolean)
    - It should be **False** unless a user wants to display xml used to generate python-ucs script.

```
from UcsSdk import *

if __name__ == "__main__":
        try:

                handle = UcsHandle()
                handle.Login("0.0.0.0","username","password")

                #1 Technical support data for the entire UCSM instance will be created and downloaded to the
                #specified file.
                #Parameter Set 1
                techSuppfilepath_1 = r"C:\techsupp_ucsm.tar"
                GetTechSupport(pathPattern= techSuppfilepath_1 , ucsManager=True)

                #Parameter Set 1
                techSuppfilepath_1 = r"C:\techsupp_ucsm.tar"
                GetTechSupport(pathPattern= techSuppfilepath_1 , ucsManager=True, timeoutSec=300)

                #Parameter Set 1
                techSuppfilepath_1 = r"C:\techsupp_ucsm.tar"
                GetTechSupport(pathPattern= techSuppfilepath_1 , ucsManager=True, timeoutSec=300,
                removeFromUcs=True)

                #2 Technical support data for the UCSM management services(excluding fabric interconnects)
                # will be created and downloaded to the specified file.
                #Parameter Set 2
                techSuppfilepath_2 = r"C:\techsupp_ucsmgmt.tar"
                GetTechSupport(pathPattern= techSuppfilepath_2 , ucsMgmt=True, timeoutSec=300,
                removeFromUcs=True)

                #3 Technical support data for Chassis id 1 and Cimc id 1 will be created and downloaded to
                # specified file.
                #Parameter Set 3
                techSuppfilepath_3 = r"C:\techsupp_chassis1_cimc1.tar"
                GetTechSupport(pathPattern= techSuppfilepath_3 , chassisId=1, cimcId =1, timeoutSec=300,
                removeFromUcs=True)

                #Parameter Set 3
                techSuppfilepath_3 = r"C:\techsupp_chassis1_cimc1_adapterid1.tar"
                GetTechSupport(pathPattern= techSuppfilepath_3 , chassisId=1, cimcId =1,adapterId=1,
                timeoutSec=300, removeFromUcs=True)

                #4 Technical support data for Chassis id 1 and Iom id 1 will be created and downloaded to
                # specified file.
                #Parameter Set 4
                techSuppfilepath_4 = r"C:\techsupp_chassis1_iomid1.tar"
                GetTechSupport(pathPattern= techSuppfilepath_4 , chassisId=1, iomId =1, timeoutSec=300,
                removeFromUcs=True)

                #5 Technical support data for Fex id 1 will be created and downloaded to specified file.
                #Parameter Set 5
                techSuppfilepath_5 = r"C:\techsupp_fexid1.tar"
                GetTechSupport(pathPattern= techSuppfilepath_5 , fexId=1, timeoutSec=300, removeFromUcs=True)

                #6 Technical support data for RackServer id 1 and RackAdapter id 1 will be created and
                # downloaded to specified file.
                #Parameter Set 6
                techSuppfilepath_6 = r"C:\techsupp_ rackserverid 1.tar"
                GetTechSupport(pathPattern= techSuppfilepath_5 , rackServerId =1, timeoutSec=300,
                removeFromUcs=True)

                #Parameter Set 6
                techSuppfilepath_6 = r"C:\techsupp_ rackserverid1_rackadapterid1.tar"
                GetTechSupport(pathPattern= techSuppfilepath_5 , rackServerId =1, rackAdapterId=1,  timeoutSec=300,
                removeFromUcs=True)

                handle.Logout()
        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                print '-'*60
                handle.Logout()
```

4.11 BackupUcs

This operation will create and download the backup of UCS.

BackupUcs(type, pathPattern, timeoutSec = 600, preservePooledValues = False, dumpXml=None)

Mandatory Input sets:

(1) BackupUcs(type , pathPattern, timeoutSec=None<Optional>, preservePooledValues=None<False>)

Parameter description:

1. type:
   - Mandatory parameter (string).
   - This specify the type of backup i.e. fullstate/config-logical/config-system/config-all
2. pathPattern:
   - Mandatory parameter (string).
   - Path of the Backup file.
3. timeoutSec:
   - Optional parameter (string).
   - Wait for specified maximum time in sec for the tech support file to generate else exit.
4. preservePooledValues:
   - Optional parameter (boolean)
5. dumpXml:
   - Optional parameter (boolean)
   - It should be **False** unless a user wants to display xml used to generate python-ucs script.

```
from UcsSdk import *
if __name__ == "__main__":
        try:
                handle = UcsHandle()
                handle.Login("0.0.0.0","username","password")

                #Create and download full-state system backup of UCS. This creates a binary file #that includes a snapshot of the entire system. You can use the file generated from #this backup to restore the system during disaster recovery. This file can restore or #rebuild the configuration on the original fabric interconnect, or recreate the #configuration on a different fabric interconnect. You cannot use this file for an #import.
                BackupFilePath = r"C:\Backups\fullstate.tar.gz"
                handle. BackupUcs(type="full-state", pathPattern=BackupFilePath, timeoutSec=300, preservePooledValues=True)
                #Create and download logical backup of UCS. This creates an XML file that includes #all logical configuration settings such as service profiles, VLANs, VSANs, pools, #and policies. You can use the file generated from this backup to import these #configuration settings to the original fabric interconnect or to a different fabric interconnect. You cannot use this file for a system restore.
                BackupFilePath = r"C:\Backups\configlogical.xml"
                handle. BackupUcs(type="config-logical", pathPattern=BackupFilePath)

                #Create and download system backup of UCS. This creates an XML file that includes  all system configuration settings such as usernames, roles, and locales. You can use the file generated from this backup to import these configuration settings to the original fabric interconnect or to a different fabric interconnect. You cannot use this file for a system restore.
                BackupFilePath = r"C:\Backups\configsystem.xml"
                handle. BackupUcs(type=" config-system", pathPattern=BackupFilePath)

                #Create and download config-all backup of UCS. This creates an XML file that includes all #system and logical configuration settings. You can use the file generated from this backup #to import these configuration settings to the original fabric interconnect or to a different  fabric #interconnect. You cannot use this file for a system restore. This file does not include  passwords #for locally authenticated users.
                BackupFilePath = r"C:\Backups\configall.xml"
                handle. BackupUcs(type=" config-all", pathPattern=BackupFilePath)


                handle.Logout()
        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                print '-'*60
                handle.Logout()
```

4.12 ImportUcsBackup

This operation will upload the UCSM backup taken earlier via GUI or BackupUcs operation for all configuration, system configuration, and logical configuration files. You can perform an import while the system is up and running.

ImportUcsBackup(path=None, merge=False, dumpXml=False)

Mandatory Input sets:

(1) ImportUcsBackup(path, merge=True<Optional>)

Parameter description:

1. path:
    - Mandatory parameter (string).
    - Path of the Backup file.
2. merge:
    - Optional parameter (boolean)
    - It should be **False** unless user wants to merge the backup configuration with the existing UCSM configuration.
3. dumpXml:
    - Optional parameter (boolean)
    - It should be **False** unless a user wants to display xml used to generate python-ucs script.

```
from UcsSdk import *
if __name__ == "__main__":
        try:
                handle = UcsHandle()
                handle.Login("0.0.0.0","username","password")

        # Import all configuration xml (An XML file that includes all system and logical configuration
        #settings. The current configuration information is replaced with the information in the  imported
        #configuration file one object at a time.
        BackupFilePath = r" C:\Backups\config-all.xml"
        handle.ImportUcsBackup(path= BackupFilePath)

        # Import all configuration xml. The information in the imported configuration file is  compared with
        # the existing configuration information. If there are conflicts, the  import operation overwrites the
        # information on the Cisco UCS domain with the information in the import configuration file.
        BackupFilePath = r" C:\Backups\config-all.xml"
        handle.ImportUcsBackup(path= BackupFilePath, merge=True)

        handle.Logout()
except Exception, err:
        print "Exception:", str(err)
        import traceback, sys
        print '-'*60
        traceback.print_exc(file=sys.stdout)
        print '-'*60
        handle.Logout()
```

4.13 StartKvmSession

This operation will launch the KVM session for the specific service profile, blade or rack.

StartKvmSession(serviceProfile = None, blade = None, rackUnit = None, frameTitle = None, dumpXml=None)

Mandatory Input sets:

(1) StartKvmSession(serviceProfile = <LsServer Object>, frameTitle = None<Optional>)
(or)
(2) StartKvmSession(blade = <ComputeBlade Object>, frameTitle = None<Optional>)
(or)
(3) StartKvmSession(rackUnit = <ComputeRackUnit Object>, frameTitle = None<Optional>)

Parameter description:

1. serviceProfile:
    - Mandatory parameter (object).
    - Object of type LsServer.
2. blade:
    - Mandatory parameter (object).
    - Object of type ComputeBlade.
3. rackUnit:
    - Mandatory parameter (object).
    - Object of type ComputeRackUnit.
4. frameTitle:
    - Optional parameter (string)

```
from UcsSdk import *
from UcsSdk.MoMeta.LsServer import LsServer
if __name__ == "__main__":
        try:
                handle = UcsHandle()
                handle.Login("0.0.0.0","username","password")

                # Start a KVM session for service profile and add a customized title for the KVM window.
                sp = handle.GetManagedObject(None, LsServer.ClassId(), { LsServer.NAME :"sp_name"})
                if (sp[0] != None):
                        handle.StartKvmSession(blade=sp[0], frameTitle="ServiceProfile", dumpXml=True)

                # Start a KVM session for blade 1.
                blade1 = handle.GetManagedObject(None, ComputeBlade.ClassId(),
                {ComputeBlade.SLOT_ID:"1"})
                if (blade1[0] != None):
                        handle.StartKvmSession(blade=blade1[0], frameTitle="blade1", dumpXml=True)

                # Start a KVM session for rack unit 1.
                rack1 = handle.GetManagedObject(None, ComputeRackUnit.ClassId(),  {
                ComputeRackUnit.ID:"1"})
                if (rack1[0] != None):
                        handle.StartKvmSession(blade=rack1[0], frameTitle="rack1", dumpXml=True)


                handle.Logout()
        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                print '-'*60
                handle.Logout()
```

4.14 StartGuiSession

This operation will launch the UCSM GUI via specific UCS handle with logging enabled.

StartGuiSession()

```
from UcsSdk import *
if __name__ == "__main__":
        try:
                handle = UcsHandle()
                handle.Login("0.0.0.0","username","password")

                handle.StartGuiSession()


                handle.Logout()

        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                print '-'*60
                handle.Logout()
```

4.15 ExportUcsSession

This operation will store the credential of currently logged in UCS in current session to a file. Password will be stored in encrypted format using key.

ExportUcsSession(filePath, key, merge=YesOrNo.FALSE)

Mandatory Input sets:

(1) ExportUcsSession(filePath, key, merge=True<Optional>)

Parameter description:

1. filePath:
    - Mandatory parameter (string).
    - Path of the credential file.
2. key:
    - Mandatory parameter (string).
    - Any string used for secure encryption.
3. merge:
    - Optional parameter (boolean)
    - It should be **False** unless user wants to append the existing credential file with new credential.

```
from UcsSdk import *
if __name__ == "__main__":
        try:
                handle1 = UcsHandle()
                handle1.Login("0.0.0.0","username","password")

                handle2 = UcsHandle()
                handle2.Login("0.0.0.0","username","password")

                fpath = r"C:\UcsSession\UcsCred.xml"
                ExportUcsSession(filePath = fpath,  key = "passkey" , merge=True)

                handle1.Logout()
                handle2.Logout()

        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                print '-'*60
                handle1.Logout()
                handle2.Logout()
```

4.16 ImportUcsSession

This operation will do a login to each UCS which is present in credential file.

ImportUcsSession(filePath, key)

Mandatory Input sets:

(1) ImportUcsSession(filePath, key)

Parameter description:

1. filePath:
    - Mandatory parameter (string).
    - Path of the credential file.
2. key:
    - Mandatory parameter (string).
    - String used while ExportUcsSession operation.

```
from UcsSdk import *
if __name__ == "__main__":
        try:
                print  “UCS Session: Before”
                print defaultUcs

                fpath = r”C:\UcsSession\UcsCred.xml”
                ImportUcsSession(filePath = fpath,  key = “passkey)

                print  “ UCS Session: After”
                print defaultUcs


        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                print '-'*60
```

4.17 Transaction Support

1. **Start a transaction.**
   `StartTransaction`

2. **Perform an operation.**
   `...`

3. **End a transaction.**
   `CompleteTransaction`

4. **Undo a transaction.**
   `UndoTransaction`

4.18 Methods

4.18.1 AaaLogin

```
AaaLogin(inName, inPassword, dumpXml=None)
```

4.18.2 AaaLogout

```
AaaLogout(dumpXml=None)
```

4.18.3 AaaRefresh

```
AaaRefresh(inName, inPassword, dumpXml=None)
```

4.18.4 AaaTokenLogin

```
AaaTokenLogin(inName, inToken, dumpXml=None)
```

4.18.5 AaaTokenRefresh

```
AaaTokenRefresh(inName, dumpXml=None)
```

4.18.6 ConfigConfMo

```
ConfigConfMo(dn, inConfig, inHierarchical=YesOrNo.FALSE, dumpXml=None)
```

4.18.7 ConfigConfMoGroup

```
ConfigConfMoGroup(inConfig, inDns, inHierarchical=YesOrNo.FALSE, dumpXml=None)
```

4.18.8 ConfigConfMos

The configConfMos method configures managed objects in multiple subtrees using DNs.

```
ConfigConfMos(inConfigs, inHierarchical=YesOrNo.FALSE, dumpXml=None)
```

```
import sys
import os
from UcsSdk import *
from UcsSdk.MoMeta.LsServer import LsServer

ucsm_ip = '0.0.0.0'
user = 'username'
password = 'password'

try:
	handle = UcsHandle()
	handle.Login(ucsm_ip,user, password)

	obj = LsServer()
	obj.setattr(LsServer.DN, "org-root/ls-new_sp")
	obj.setattr(LsServer.NAME, "new_sp")
	obj.setattr(LsServer.STATUS, Status.CREATED)

	pair = Pair()
	pair.Key = obj.Dn
	pair.AddChild(obj)

	configMap = ConfigMap()
	configMap.AddChild(pair)

	ccm = handle.ConfigConfMos(configMap)
	if ccm.errorCode == 0:
		moList = []
		for child in ccm.OutConfigs.GetChild():
			if (isinstance(child, Pair) == True):
				for mo in child.GetChild():
					moList.append(mo)
			elif (isinstance(child, ManagedObject) == True):
				moList.append(child)
		WriteObject(moList)
	else:
		WriteUcsWarning('[Error]: [Code]:' + ccm.errorCode + '
[Description]:' + ccm.errorDescr)

	handle.Logout()

except Exception, err:
	print "Exception:", str(err)
	import traceback, sys
	print '-'*60
	traceback.print_exc(file=sys.stdout)
	print '-'*60
```

4.18.9 ConfigEstimateImpact

The configEstimateImpact method estimates the impact of a set of managed objects modifications in terms of disruption of running services. For example, modifying the UUID pool used by an updating template might require rebooting servers associated to service profiles, instantiated from the template.
User can estimate the impact of a change set by passing the set to the method and inspecting the output parameters. Output parameters are a set of affected service profiles (before and after the changes) and the corresponding ack object for each service profile.
ack objects contain the following information:
• Whether the changes are disruptive (for example, require reboot of the server associated to the service profile).
• Summary of changes.
• When changes are applied (immediately, after user ack, during scheduled occurrence of a maintenance window).
• Date and time at which the changes were made and by whom.

Cisco UCS returns the ack objects before and after the changes are applied. This information helps to determine whether some changes were already pending on the service profile. This condition can occur when maintenance policies are used.
The parameters are defined as:
• configs—Set of changes to be evaluated (add, delete, or modify managed objects).
• affected—Affected service profiles after the changes have been applied (not hierarchical).
• oldAffected—Affected service profiles before applying changes (not hierarchical).
• ackables—Content of the ack object associated to the service profiles, after applying the changes.
• oldAckables—Content of the ack object associated to the service profiles, before applying the changes.

```
ConfigEstimateImpact(inConfigs, dumpXml=None)
```

4.18.10 ConfigFindDependencies

The configFindDependencies method returns the service profile details for a specified policy.

```
ConfigFindDependencies(dn, inReturnConfigs, dumpXml=None)
```

4.18.11 ConfigResolveChildren

The configResolveChildren method retrieves children of managed objects under a specific DN in the managed information tree. A filter can be used to reduce the number of children being returned.

```
ConfigResolveChildren(classId, inDn, inFilter,
                                    inHierarchical=YesOrNo.FALSE, dumpXml=None)
```

```
import sys
import os
from UcsSdk import *

basename = os.path.basename (sys.argv[0])

if (len(sys.argv) != 2):
        print "Usage: %s parentDn" % basename
        sys.exit()

ucsm_ip = '0.0.0.0'
user = 'username'
password = 'password'

parentDn = sys.argv[1]

try:
        handle = UcsHandle()
        handle.Login(ucsm_ip,user, password)

        crc = handle.ConfigResolveChildren(FcpoolInitiators.ClassId(), parentDn, None, YesOrNo.TRUE,
True)
        if (crc.errorCode == 0):
                moList = []
                for child in crc.OutConfigs.GetChild():
                        if (isinstance(child, ManagedObject) == True):
                                moList.append(child)
                WriteObject(moList)

        handle.Logout()

except Exception, err:
        print "Exception:", str(err)
        import traceback, sys
        print '-'*60
        traceback.print_exc(file=sys.stdout)
        print '-'*60
```

4.18.12 ConfigResolveClass

The configResolveClass method returns requested managed object in a given class. If inHierarchical=true,then the results contain children.

```
ConfigResolveClass(classId, inFilter, inHierarchical=YesOrNo.FALSE, dumpXml=None)
```

```
from UcsSdk import *

if __name__ == "__main__":
        try:
                handle = UcsHandle()
                handle.Login("0.0.0.0","username","password")

                crDns = handle.ConfigResolveClass(ComputeBlade.ClassId(), inFilter=None,
inHierarchical=YesOrNo.FALSE, dumpXml=None)
                if (crDns.errorCode == 0):
                        for mo in crDns.OutConfigs.GetChild():
                                print mo.Dn
                else:
                        WriteUcsWarning('[Error]: configResolveDns [Code]:' + crDns.errorCode + '
[Description]:' + crDns.errorDescr)

                handle.Logout()

        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                        print '-'*60
```

4.18.13 ConfigResolveClasses

The configResolveClasses method returns requested managed objects in several classes. If inHierarchical=true, then the results contain children.

```
ConfigResolveClasses(inIds, inHierarchical=YesOrNo.FALSE, dumpXml=None)
```

4.18.14 ConfigResolveDn

For a specified DN, the configResolveDn method retrieves a single managed object.

```
ConfigResolveDn(dn, inHierarchical=YesOrNo.FALSE, dumpXml=None)
```

4.18.15 ConfigResolveDns

For a list of DNs, the configResolveDns method retrieves managed objects.

```
ConfigResolveDns(inDns, inHierarchical=YesOrNo.FALSE, dumpXml=None)
```

```
from UcsSdk import *

if __name__ == "__main__":
        try:
                handle = UcsHandle()
                handle.Login("0.0.0.0","username","password")

                dnSet = DnSet()
                dn = Dn()
                dn.setattr("Value","org-root")
                dnSet.AddChild(dn)
                crDns = handle.ConfigResolveDns(dnSet)
                if (crDns.errorCode == 0):
                        WriteObject(crDns.OutConfigs.GetChild())
                else:
                        WriteUcsWarning('[Error]: configResolveDns [Code]:' + crDns.errorCode + ' [Description]:' +
crDns.errorDescr)

                handle.Logout()

        except Exception, err:
                print "Exception:", str(err)
                import traceback, sys
                print '-'*60
                traceback.print_exc(file=sys.stdout)
                        print '-'*60
```

4.18.16 ConfigResolveParent

For a specified DN, the configResolveParent method retrieves the parent of the managed object.

```
ConfigResolveParent(dn, inHierarchical=YesOrNo.FALSE, dumpXml=None)
```

4.18.17 ConfigScope

The configScope method returns managed objects and details about their configuration.

```
ConfigScope(dn, inClass, inFilter, inRecursive, inHierarchical=YesOrNo.FALSE,
dumpXml=None)
```

4.18.18 EventRegisterEventChannel

```
EventRegisterEventChannel(inDn, dumpXml=None)
```

4.18.19 EventRegisterEventChannelResp

```
EventRegisterEventChannelResp(inCtx, inDn, inReqID, dumpXml=None)
```

4.18.20 EventSendEvent

```
EventSendEvent(inDn, inEvent, inReqId, dumpXml=None)
```

4.18.21 EventSendHeartbeat

The eventSendHeartbeat method allows clients to retrieve any missed event. Each event has an unique event ID. These event IDs operate as counters and are included in all method responses. Each time an event is generated, the events ID counter increases and the new event is assigned a new event ID. This enables the subscriber to track the events. If an event is missed by the client, the client can use the **eventSendEvent** method to retrieve the missed event.

```
EventSendHeartbeat(dumpXml=None)
```

4.18.22 EventSubscribe

The eventSubscribe method allows a client to subscribe to asynchronous events generated by Cisco UCS, including all object changes in the system (created, changed, or deleted).
Event subscription allows a client application to register for event notification from Cisco UCS. When an event occurs, Cisco UCS informs the client application of the event and its type. Only the actual change information is sent. The object's unaffected attributes are not included.

```
EventSubscribe(inFilter, dumpXml=None)
```

4.18.23 EventUnRegisterEventChannel

```
EventUnRegisterEventChannel(inDn, inReqID, dumpXml=None)
```

4.18.24 FaultAckFault

The faultAckFault method acknowledges a fault. The acknowledgement response marks the fault severity as cleared. Faults categorized as auto-cleared do not require acknowledgment.

```
FaultAckFault(inId, dumpXml=None)
```

4.18.25 FaultAckFaults

The faultAckFaults method acknowledges multiple faults. The acknowledgement response marks the fault severity as cleared. Faults categorized as auto-cleared do not require acknowledgment.

```
FaultAckFaults(inIds, dumpXml=None)
```

```
import sys
import os
from UcsSdk import *

ucsm_ip = '0.0.0.0'
user = 'username'
password = 'password'

try:
        handle = UcsHandle()
        handle.Login(ucsm_ip,user, password)

        idSet = IdSet()

        getRsp = handle.GetManagedObject(None, FaultInst.ClassId())
        for mo in getRsp:
                id = Id()
                id.Value = mo.Id
                idSet.AddChild(id)

        handle.FaultAckFaults(idSet)
        handle.Logout()

except Exception, err:
        print "Exception:", str(err)
        import traceback, sys
        print '-'*60
        traceback.print_exc(file=sys.stdout)
        print '-'*60
```

4.18.26 FaultResolveFault

The faultResolveFault method sends a response when a fault has been resolved.

```
FaultResolveFault(inId, dumpXml=None)
```

4.18.27 LsClone

The lsClone method clones a service profile. The new service profile has the same values as the specified service profile.

```
LsClone(dn, inServerName, inTargetOrg, inHierarchical=YesOrNo.FALSE,
dumpXml=None)
```

4.18.28 LsInstantiateNNamedTemplate

For a specified service template, the lsInstantiateNNamedTemplate method instantiates as many service profiles as are specified in the namedSet parameter.
• dn—Specifies the service template used for instantiating.
• nameSet—Contains the names of the service profiles to be instantiated.
• targetOrg—Specifies the organization in which these service profiles are instantiated.

```
LsInstantiateNNamedTemplate(dn, inNameSet, inTargetOrg,
inHierarchical=YesOrNo.FALSE, dumpXml=None)
```

4.18.29 LsInstantiateNTemplate

The lsInstantiateNTemplate method creates a number (N) of service profiles from a template.

```
LsInstantiateNTemplate(dn, inNumberOf, inServerNamePrefixOrEmpty, inTargetOrg,
inHierarchical=YesOrNo.FALSE, dumpXml=None)
```

4.18.30 LsInstantiateTemplate

The lsInstantiateTemplate method creates one service profile from a specified template.

```
LsInstantiateTemplate(dn, inServerName, inTargetOrg,
inHierarchical=YesOrNo.FALSE, dumpXml=None)
```

4.18.31 LsResolveTemplates

The lsResolveTemplates method retrieves the service profile templates from the specified organization, which is matched hierarchically. The search can be further refined by providing standard querying filters in addition to querying by template-type and a flag to exclude-if-bounded.
Template type can be “initial-template” or “updating-template”.

```
LsResolveTemplates(dn, inExcludeIfBound, inFilter, inType,
inHierarchical=YesOrNo.FALSE, dumpXml=None)
```

4.18.32 LsTemplatise

The lsTemplatise method creates a template from a specified service profile.

```
LsTemplatise(dn, inTargetOrg, inTemplateName, inTemplateType,
inHierarchical=YesOrNo.FALSE, dumpXml=None)
```

4.18.33 StatsClearInterval

The statsClearInterval method resets the collection interval timer for the statsClass. All of the statistics' implicit properties (for example, min, max, and avg calculations) are reset, and the corresponding history properties are updated. The interval updates restart from 1, and the stats collection is reset.

```
StatsClearInterval(inDns, dumpXml=None)
```

4.18.34 StatsResolveThresholdPolicy

The statsResolveThresholdPolicy method resolves threshold policy based on the container class ID. The container class is objects with policies (for example, server domain, lan cloud, san cloud, nas cloud, etc.). The Cisco UCS uses the hierarchy of an organization to resolve the names of policies.

```
StatsResolveThresholdPolicy(dn, dumpXml=None)
```

5 Samples

A couple of samples are packaged with the installation.

References

The following documents were referred to while creating this material:

1. Cisco UCS Manager API Management Information Model.
   http://developer.cisco.com/web/unifiedcomputing/docs

2. Cisco UCS Manager CLI Configuration Guide
   http://www.cisco.com/en/US/docs/unified_computing/ucs/sw/cli/config/guide/2.0/b_UCSM_CLI_Configuration_Guide_2_0.pdf