Cisco UCS Python Command Reference for Cisco UCS Manager

Connection Management

Connect to Cisco UCS Manager with Cisco UCS Python SDK

The Python software development kit can connect and communicate with multiple Cisco Unified Computing System™ (Cisco UCS®) domains in parallel.

Connect to a single Cisco UCS domain, get the current session status, and disconnect by using the commands shown here.

Connect to a Cisco UCS domain:
```
from ucsmsdk.ucshandle import UcsHandle
handle = UcsHandle("192.168.0.1","username","password")
handle.login()
```

Disconnect from the Cisco UCS domain:
```
handle.logout()
```

Connect to multiple Cisco UCS domains and then disconnect:
```
from ucsmsdk.ucshandle import UcsHandle
handle1=UcsHandle("192.168.0.1","username","password")
handle2=UcsHandle("192.168.0.2", "username","password")
handle1.login()
handle2.login()
handle1.logout()
handle2.logout()
```

Launch Cisco UCS Manager GUI Sessions

Launch a Cisco UCS Manager GUI from a current session:
```
from ucsmsdk.utils import ucsguilaunch
ucsguilaunch.ucs_gui_launch(handle)
```

Launch Cisco UCS Server Keyboard Video Mouse (KVM) GUI Sessions

Launch a KVM GUI session for a specific Cisco UCS B-Series Blade Server:
```
from ucsmsdk.utils import ucskvmlaunch
ucskvmlaunch.ucs_kvm_launch(handle,blade=<blade object>)
```

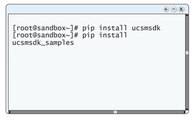
Launch a KVM GUI session for blade-1 in chassis-1
```
mo=handle.query_dn("sys/chassis-1/blade-1")
ucskvmlaunch.ucs_kvm_launch(handle,blade=mo)
```

Launch a KVM GUI session for a specific Cisco UCS C-Series Rack Server:
```
from ucsmsdk.utils import ucskvmlaunch
ucskvmlaunch.ucs_kvm_launch(handle, rack_unit=<rack unit object>)
```

Launch a KVM GUI session for a rack-unit-1
```
mo=handle.query_dn("sys/rack-unit-1")
ucskvmlaunch.ucs_kvm_launch(handle, rack_unit=mo)
```

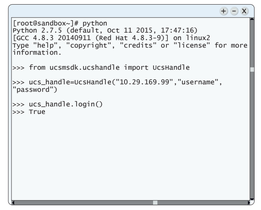
How to Use Convert_to_UCS_Python

Perform an action in the Cisco UCS Manager GUI and then capture a code sequence using `convert_to_ucs_python` that will implement the same step using Python.



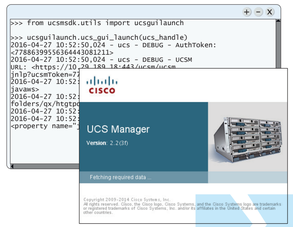
Install the Cisco UCS Python SDK package on your Linux, MacOS X, or Microsoft Windows system using `pip install ucsmsdk`. You can also install the `ucsmsdk_samples` package.

STEP 1



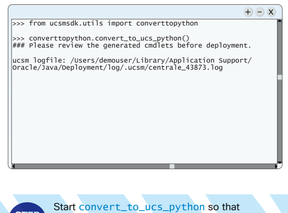
Connect to your Cisco UCS domain using any Python interactive shell (such as `python`, `iPython`, or `idle`) Enter the commands shown using your user ID and password.

STEP 2



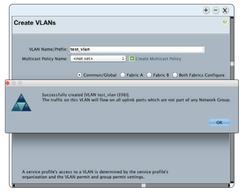
Start the Cisco UCS GUI.

STEP 3



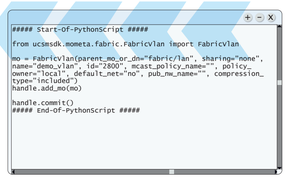
Start `convert_to_ucs_python` so that the Python binding will capture your actions in the Cisco UCS Manager GUI.

STEP 4



Perform an operation in the Cisco UCS Manager GUI (for example, create a VLAN).

STEP 5



Capture the Python script that `convert_to_ucs_python` emits.

STEP 6

The following examples demonstrate `convert_to_ucs_python` use.

You can perform an operation in the Cisco UCS Manager GUI and capture the Python code that will re-create the operation through the scripting language.

Get the Python script corresponding to an operation performed in the GUI:
```
from ucsmsdk.utils import converttopython
converttopython.convert_to_ucs_python()
```

Get the XML requests along with the generated script:
```
converttopython.convert_to_ucs_python(dump_xml=True)
```

Get the XML requests in a file along with the generated script:
```
file_path="/users/ucsm_xml/configrequest.xml"
converttopython.convert_to_ucs_python(dump_to_file=True,
      dump_file_path=file_path,dump_xml=True)
```

Generate the Python script for the action specified in the log:
```
file_path = "/users/ucsm_xml/ucsmlog/centrale_14804.log"
converttopython.convert_to_ucs_python(gui_log=True, path=file_path)
```

Generate the Python script for the specified XML request:
```
xml_str='<configConfMos inHierarchical="false"><inConfigs><pair
key="fabric/lan/net-VDI_7"> <fabricVlan compressionType="included"
defaultNet="no" dn="fabric/lan/net-VDI_7" id="7" mcastPolicyName=""
name="VDI_7" policyOwner="local" pubNwName="" sharing="none"
status="created">  </fabricVlan></pair></inConfigs></configConfMos>'
converttopython.convert_to_ucs_python(xml=True,request=xml_str)
```

Generate the Python script for the specified XML request in a file:
```
file_path="/Users/ucsm_xml/configrequest.xml"
converttopython.convert_to_ucs_python(xml=True,path=file_path)
```

Compare and Synchronize Cisco UCS Managed Objects

The `compare_ucs_mo` function compares any similar managed objects to each other. Output is provided in the form of diff objects, which contain the items that are different between the two objects. The `diff objects` include indicators that show the presence of an item in one object and not in the other, or the presence of the same item in both objects. The indicators are => ,<= , and ==.

The `write_mo_diff` function displays the `diff objects` in a readable format.

The `sync_ucs_mo` function takes the `diff object` output from `compare_ucs_mo` as input to synchronize the differences between two objects.

Sample Uses

Compare a collection of objects created across two Cisco UCS domains:
Use `compare_ucs_mo` to see the VLAN differences between two Cisco UCS domains. The `diff object` output then can be provided as input to `sync_ucs_mo` to synchronize the VLANS between two domains.
```
from ucsmsdk.ucshandle import UcsHandle
from ucsmsdk.utils import comparesyncmo
handle1=UcsHandle("192.168.0.1", "username", "password")
handle2=UcsHandle("192.168.0.2", "username", "password")
handle1.login()
handle2.login()
handle1_vlans=handle1.query_classid("fabricVlan")
handle2_vlans=handle2.query_classid("fabricVlan")
difference_vlans=comparesyncmo.compare_ucs_mo(handle1_vlans, handle2_vlans)
comparesyncmo.write_mo_diff(difference_vlans) # will print the difference to the screen
comparesyncmo.sync_ucs_mo(handle1, difference_vlans, delete_not_present=False)
```
delete_not_present, when set to True, will delete the vlans in handle1 managed object if it is not present in the handle2 managed object

Compare two different objects of the same type in a single Cisco UCS domain or two Cisco UCS domains:
Use `xlate_map` along with `compare_ucs_mo` to compare two different objects of the same type in the same or two different Cisco UCS domains.
```
src_sp=[handle.query_dn("org-root/org-src/ls-R_SP")]
dest_sp=[handle.query_dn("org-root/org-dst/ls-V_SP")]
xlate_map = {"org-root/org-src/ls-R_SP":"org-root/org-dst/ls-V_SP"}
diff_mo=comparesyncmo.compare_ucs_mo(dest_sp, src_sp, xlate_map=xlate_map)
comparesyncmo.write_mo_diff(diff_mo)
```

Compare two different objects of the same type and same name in a single Cisco UCS domain or two Cisco UCS domains:
Use `xlate_org` along with `compare_ucs_mo` to compare two objects of the same type and the same name in the same or two different Cisco UCS domains.
```
src_sp=[handle.query_dn("org-root/org-src/ls-v_SP")]
dest_sp=[handle.query_dn("org-root/org-dst/ls-v_SP")]
xlate_org="org-root/org-dst"
diff_mo=comparesyncmo.compare_ucs_mo(dest_sp, src_sp, xlate_org=xlate_org)
comparesyncmo.write_mo_diff(diff_mo)
```

Extract Information from Cisco UCS Manager Using Python SDK

You can retrieve data from the Cisco UCS Manger by querying the distinguished name (DN), multiple distinguished names (DNs), class ID, and multiple class IDs:

Querying by distinguished name returns the specific object. Querying by class ID returns the list of objects belonging to the class. For example, querying against the class ID **fabricVlan** returns a list of all the Cisco UCS VLAN objects. Querying against the distinguished name of a VLAN returns only that VLAN object.

Querying by distinguished names will return the objects belonging to the specified DNs. Querying by class IDs returns all objects belonging to the specified class IDs.
```
mo_list=handle.query_class_id("<class id>")
mo_dict=handle.query_class_ids("<class id1>", "<class id2>")
mo_list=handle.query_dn("<dn>")
mo_dict=handle.query_dns("<dn1>","<dn2>")
```

This example:
```
mo_list=handle.query_classid("fabricVlan")
```
lists of all VLANs configured in the system.
```
mo_dn=handle.query_dn("fabric/lan/net-OS-1-data")
print mo_dn
```
lists of all the properties associated with the VLAN managed object.

You can make query results more specific by adding filter strings to the function. The filter strings use Python regular expressions:
```
filter_exp='(name,"hx")'
mo_list=handle.query_classid("fabricVlan", filter_str=filter_exp)
for vlan in mo_list:
    print vlan.name
```
lists the VLANs that have the string "hx" in their names.

You can also add regular expression flags to the filter strings:
```
filter_exp='(name,"hx", type="re", flag="i")
```

Transaction Support

You can optimize performance and increase efficiency with the Cisco UCS Python SDK transaction capabilities. You can buffer multiple operations and then commit them, optimizing the request and sending them to Cisco UCS Manager as a single operation. Transaction objects are then processed as atomic operations. If any problem is detected in a particular object, the transaction is discarded with no partial commits applied. Transaction support is implicit whether a single-object or multiple-object operation is in the commit buffer:

The `handle.add(mo)` function adds the managed object to the Cisco UCS Manager database.

The `handle.set(mo)` function updates the managed object in the Cisco UCS Manager database.

The `handle.remove(mo)` function removes the managed object from the Cisco UCS Manager database.

Commit the transactions:
```
handle.commit()
```

Discard the transactions:
```
handle.commit_buffer_discard()
```

Create a list of Cisco UCS service profiles, add them to the handle, and commit:
```
sps = []
from ucsmsdk.mometa.ls.LsServer import LsServer
sps.append(LsServer(parent_mo_or_dn="org-root", name="demo_1"))
sps.append(LsServer(parent_mo_or_dn="org-root", name="demo_1_1"))
sps.append(LsServer(parent_mo_or_dn="org-root", name="demo_1_2"))
sps.append(LsServer(parent_mo_or_dn="org-root", name="demo_2_1"))
sps.append(LsServer(parent_mo_or_dn="org-root", name="demo_2_2"))
sps.append(LsServer(parent_mo_or_dn="org-root", name="DEMO"))
for sp in sps:
    handle.add_mo(sp)
handle.commit()
```

Retrieve a single Cisco UCS service profile, change the description, and commit:
```
sp = handle.query_dn("org-root/ls-demo_1")
sp.descr = "demo_descr"
handle.set_mo(sp)
handle.commit()
```

Retrieve a single Cisco UCS service profile and delete it:
```
sp=handle.query_dn("org-root/ls-demo_1")
handle.remove_mo(sp)
handle.commit()
```

Python SDK Resources

All components of Cisco UCS Manager are considered to be managed objects.

Display the metadata of a managed object (mometa):
```
from ucsmsdk.mometa.compute.ComputeRackUnit import ComputeRackUnit
mo=ComputeRackUnit.mo_meta
vars(mo)
print mo.rn
print mo.children
```

Map Cisco UCS Manager labels to managed object labels:
```
vars(ComputeRackUnit)["prop_map"]
```

Find the Python class of the managed object:
- In the Cisco UCS Manager GUI, right-click any object and select the Copy XML option.
- Paste the XML in a text editor. The first word following < is the class identifier (class ID) of the object.

In the following example, **fabricVlan** is the class ID of the VLAN.
```
<fabricVlan assocPrimaryVlanState="ok"
assocPrimaryVlanSwitchId="NONE" childAction="deleteNonPresent"
cloud="ethlan" compressionType="included" configIssues=""
defaultNet="no" dn="fabric/lan/net-hx-inband-mgmt" epDn=""
fltAggr="0" global="0" id="3091" ifRole="network" ifType="virtual"
local="0" locale="external" mcastPolicyName="Hyperflex" name="hx-
inband-mgmt" operMcastPolicyName="org-root/mc-policy-Hyperflex"
operState="ok" overlapStateForAc="active" overlapStateForS="active"
peerDn="" policyOwner="local" pubNwDn="" pubNwId="1" pubNwName=""
sharing="none" switchId="dual" transport="ether" type="lan"/>
```

Display the hierarchy tree and the metadata for a particular class ID:
```
from ucsmsdk.ucscoreutils import get_meta_info
meta = get_meta_info(class_id="MemoryArray")
```

You can turn off the tree display and metadata display by using the **include_prop=False** and **show_tree=False** parameters.

You can also use `get_meta_info` to list class IDs that match a particular word.

List all the class IDs that have the string "Mem" in them:
```
meta=get_meta_info(class_id="Mem")
```

For more information:

For more information on the Cisco UCS Python SDK see https://github.com/CiscoUcs/stash/blob/master/ucsmsdk_slides/slides.md.

For more information on programmability and management see http://blogs.cisco.com/datacenter/programmability-and-ucs-management.

Backup, Restore, and Install Firmware

Backup

Full-state system backup creates a snapshot of the entire system and places it in a binary file. In the event of a disaster, the file generated from this backup can be used to perform a full restoration of the system using the same or a different fabric interconnect. You can restore from this file, but you cannot import the file using the Cisco UCS Manager GUI.
```
from ucsmsdk.utils import ucsbackup
ucsbackup.backup_ucs(handle, backup_type="full-state", file_dir="/Users/Documents/UCS", file_name="UCS-backup-full-state.tar.gz")
```

Config-all backup creates an XML file that includes all the system and logical configuration settings. You can use these to import the configuration settings to the original or different Cisco UCS domain. This backup cannot be used for a full-state system restoration operation, and it does not include passwords for locally authenticated users.
```
from ucsmsdk.utils import ucsbackup
ucsbackup.backup_ucs(handle, backup_type="config-all", file_dir="/Users/Documents/UCS", file_name="UCS-backup-config-all.xml")
```

Logical backup creates an XML file that includes all logical configuration settings such as Cisco UCS service profiles, VLANs, VSANs, pools, and policies.
```
from ucsmsdk.utils import ucsbackup
ucsbackup.backup_ucs(handle, backup_type="config-logical", file_dir="/Users/Documents/UCS", file_name="UCS-backup-config-logical.xml")
```

System backup creates an XML file that includes all system configuration settings such as user names, roles, and locales.
```
from ucsmsdk.utils import ucsbackup
ucsbackup.backup_ucs(handle, backup_type="config-system", file_dir="/Users/Documents/UCS", file_name="UCS-backup-config-system.xml")
```

Both a logical and a system backup can be imported to the original or a different Cisco UCS domain. Neither can be used for full system restoration.

Restore

Import is available through the `import-ucs_backup` function. You can use this function with `config-all`, `config-logical`, and `config-system` XML configuration files, but not with full-state system backup. You can perform an import operation while the system is running. When an import operation is performed, current configuration information is either merged or replaced with the information in the backup file, one object at a time.

Replace all the configuration information from a configure-all backup:
```
from ucsmsdk.utils import ucsbackup
ucsbackup.import_ucs_backup(handle, file_dir="/Users/Documents/UCS", file_name="UCS-backup-config-all.xml")
```

Merge the data in the config-all.xml backup with the current Cisco UCS Manager database:
```
from ucsmsdk.utils import ucsbackup
ucsbackup.import_ucs_backup(handle, file_dir="/Users/Documents/UCS", file_name="UCS-backup-config-all.xml", merge=True)
```

Firmware

Download, upload, and upgrade the infrastructure and server firmware:
```
from ucsmsdk_samples.firmware import ucsfirmware
```

Download the images in the local/remote file system using your Cisco account credentials
```
ucsfirmware.firmware_download("ucs-k9-bundle-c-series.2.2.6e.C.bin", "username","password","/UCS/Images/Download")
ucsfirmware.firmware_download("ucs-k9-bundle-c-series.2.2.6e.A.bin", "username","password","/UCS/Images/Download")
ucsfirmware.firmware_download("ucs-k9-bundle-c-series.2.2.6e.B.bin", "username","password","/UCS/Images/Download")
```

Install the firmware. This will upgrade infrastructure, blade servers, and rack servers.
```
ucsfirmware.firmware_auto_install(handle, "2.2(6e)", "/UCS/Images/Download")
```


LE-45204-00 05/16

Cisco UCS Python Command Reference