Script Infrastructure and Sample Templates

Table 1: Feature History Table

Feature Name	Release Information	Description
Contextual Script Infrastructure	Release 7.3.2	When you create and run Python scripts on the router, this feature enables a contextual interaction between the scripts, the IOS XR software, and the external servers. This context, programmed in the script, uses Cisco IOS XR Python packages, modules, and libraries to: • obtain operational data from the router • set configurations and conditions • detect events in the network and trigger an appropriate action

You can create Python scripts and execute the scripts on routers running Cisco IOS XR software. The software supports the Python packages, libraries and dictionaries in the software image. For more informtion about the script types and to run the scripts using CLI commands To run the same actions using NETCONF RPCs,

Cisco IOS XR, Release 7.3.2 supports creating scripts using Python version 3.5.

Cisco IOS XR Python Packages

With on-box Python scripting, automation scripts that was run from an external controller is now run on the router. To achieve this functionality, Cisco IOS XR software provides contextual support using SDK libraries and standard protocols.

The following Python third party application packages are supported by the scripting infrastructure and can be used to create automation scripts.

Package	Description	Support Introduced in Release
appdirs	Chooses the appropriate platform-specific directories for user data.	Release 7.3.2
array	Defines an object type that can compactly represent an array of basic values: characters, integers, floating point numbers.	Release 7.3.2
asn1crypto	Parses and serializes Abstract Syntax Notation One (ASN.1) data structures.	Release 7.3.2
chardet	Universal character encoding auto-detector.	Release 7.3.2
concurrent.futures	Provides a high-level interface for asynchronously executing callables.	Release 7.3.2
ecdsa	Implements Elliptic Curve Digital Signature Algorithm (ECDSA) cryptography library to create keypairs (signing key and verifying key), sign messages, and verify the signatures.	Release 7.3.2
enum	Enumerates symbolic names (members) bound to unique, constant values.	Release 7.3.2
email	Manages email messages.	Release 7.3.2
google.protobuf	Supports language-neutral, platform-neutral, extensible mechanism for serializing structured data.	Release 7.3.2
ipaddress	Provides capability to create, manipulate and operate on IPv4 and IPv6 addresses and networks.	Release 7.3.2
jinja2	Supports adding functionality useful for templating environments.	Release 7.3.2
json	Provides a lightweight data interchange format.	Release 7.3.2

Package	Description	Support Introduced in Release
markupsafe	Implements a text object that escapes characters so it is safe to use in HTML and XML.	Release 7.3.2
netaddr	Enables system-independent network address manipulation and processing of Layer 3 network addresses.	Release 7.3.2
pdb	Defines an interactive source code debugger for Python programs.	Release 7.3.2
pkg_resources	Provides runtime facilities for finding, introspecting, activating and using installed distributions.	Release 7.3.2
psutil	Provides library to retrieve information on running processes and system utilization such as CPU, memory, disks, sensors and processes.	Release 7.3.2
pyasn1	Provides a collection of ASN.1 modules expressed in form of pyasn1 classes. Includes protocols PDUs definition (SNMP, LDAP etc.) and various data structures (X.509, PKCS).	Release 7.3.2
requests	Allows sending HTTP/1.1 requests using Python.	Release 7.3.2
shellescape	Defines the function that returns a shell-escaped version of a Python string.	Release 7.3.2
subprocess	Spawns new processes, connects to input/output/error pipes, and obtain return codes.	Release 7.3.2
urllib3	HTTP client for Python.	Release 7.3.2
xmltodict	Makes working with XML feel like you are working with JSON.	Release 7.3.2

Cisco IOS XR Python Libraries

Cisco IOS XR software provides support for the following SDK libraries and standard protocols.

Library	Syntax

xrlog	`# To generate syslogs #`
	`from cisco.script_mgmt import xrlog`
	
	`syslog = xrlog.getSysLogger('template_exec')`

netconf	`#To connect to netconf client #`
	`from iosxr.netconf.netconf_lib import NetconfClient`
	
	`nc = NetconfClient(debug=True)`

xrclihelper	`# To run native xr cli and config commands`
	`from iosxr.xrcli.xrcli_helper import *`
	
	`helper = XrcliHelper(debug = True)`

config_validation	`# To validate configuration #` `import cisco.config_validation as xr`
eem	`# For EEM operations #` `from iosxr import eem`
precommit	`# For Precommit script operations #` `from cisco.script_mgmt import precommit`

Sample Script Templates

Use these sample script templates based on script type to build your custom script.

Follow these instructions to download the sample scripts from the Github repository to your router, and run the scripts:

1. Clone the Github repository.

   ```
   $git clone https://github.com/CiscoDevNet/iosxr-ops.git
   ```

2. Copy the Python files to the router's harddisk or a remote repository.

Config Script

The following example shows a code snippet for config script. Use this snippet in your script to import the libraries required to validate configuration and also generate syslogs.

```
#Needed for config validation
import cisco.config_validation as xr

#Used for generating syslogs
from cisco.script_mgmt import xrlog
syslog = xrlog.getSysLogger('Add script name here')

def check_config(root):
    #Add config validations
    pass

xr.register_validate_callback([<Add config path here>],check_config)
```

Exec Script

Use this sample code snippet in your exec script to import Python libraries to connect to NETCONF client and also to generate syslogs.

```
#To connect to netconf client
from iosxr.netconf.netconf_lib import NetconfClient

#To generate syslogs
syslog = xrlog.getSysLogger('template_exec')

def test_exec():
    """
    Testcase for exec script
    """
    nc = NetconfClient(debug=True)
    nc.connect()
    #Netconf or processing operations
    nc.close()


if __name__ == '__main__':
    test_exec()
```

Process Script

Use the following sample code snippet to trigger a process script and perform various actions on the script. You can leverage this snippet to create your own custom process script. Any exec script can be used as a process script.

```
To trigger script
Step 1: Add and configure script as shown in README.MD

Step 2: Register the application with Appmgr

Configuraton:
appmgr process-script my-process-app
executable test_process.py
run args --threshold <threshold-value>

Step 3: Activate the registered application
appmgr process-script activate name my-process-app

Step 4: Check script status
show appmgr process-script-table

Router#show appmgr process-script-table
Name             Executable         Activated    Status     Restart Policy   Config Pending
---------------  ------------------ --------- ------------- ---------------- --------------
my-process-app   test_process.py      Yes       Running    On Failure            No

Step 5: More operations
Router#appmgr process-script ?
  activate    Activate process script
  deactivate  Deactivate process script
  kill        Kill process script
  restart     Restart process script
  start       Start process script
  stop        Stop process script
"""

#To connect to netconf client
from iosxr.netconf.netconf_lib import NetconfClient
```

```
#To generate syslogs
syslog = xrlog.getSysLogger('template_exec')

def test_process():
    """
    Testcase for process script
    """
    nc = NetconfClient(debug=True)
    nc.connect()
    #Netconf or any other operations
    nc.close()


if __name__ == '__main__':
    test_process()
```

EEM Script

You can leverage the following sample code to import Python libraries to create your custom eem script and also generate syslogs.

```
Required configuration:
User and AAA configuration

event manager event-trigger <trigger-name>
type syslog pattern "PROC_RESTART_NAME"

event manager action <action-name>
username <user>
type script script-name <script-name> checksum sha256 <checksum>

event manager policy-map policy1
trigger event <trigger-name>
action <action-name>

To verify:
Check for syslog EVENT SCRIPT EXECUTED: User restarted <process-name>

"""
#Needed for eem operations
from iosxr import eem

#Used to generate syslogs
from cisco.script_mgmt import xrlog
syslog = xrlog.getSysLogger(<add your script name here>)

# event_dict consists of details of the event
rc, event_dict = eem.event_reqinfo()

#You can process the information as needed and take action for example: generate a syslog.
#Syslog type can be emergency, alert, critical, error, exception, warning, notification,
info, debug

syslog.info(<Add you syslog here>)
```