



Workflow Developer Guide

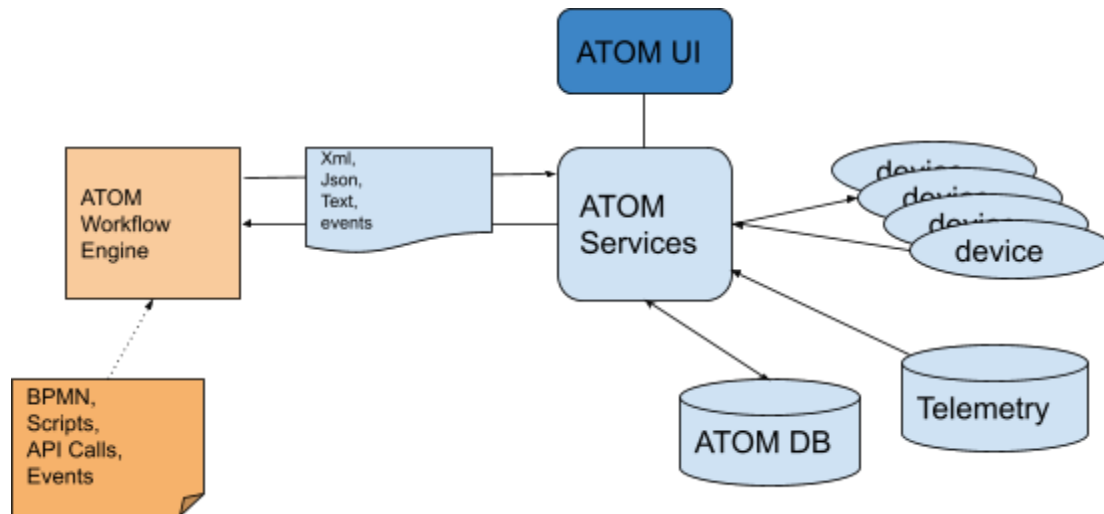
version 11.0

Table of Contents

ATOM Workflow Development - High Level View	4
Outline of the document	4
Network Automation & MOPs	5
Network Services	5
Network MOPs	5
ATOM Workflow	6
Technical requirements for developing Workflow in ATOM	6
Workflow Package Development	7
Create a Workflow package	7
Update the Dependencies & Version in build.gradle	8
Archive the Workflow Package	10
Developing Workflows - BPMN Modelling	11
Use Case 1 - Software Upgrade	11
Figure 1 - BPMN Diagram for SMU with Ping check	13
Figure 2 - BPMN Diagram for SMU with Async Notification	13
Use Case 2 - Config Provisioning	15
User Inputs Form	15
Service Task to configure the Device	17
Use Case 3 - CLA Remediation	21
Deploying & Operating on Workflows	25
Appendix	26
ATOM Workflow FAQs & Examples	26
ATOM Workflow Activities	26
Timer	26
Sub-process	29
Signal End Event & Boundary Event	30
Decision box	34
Multi-instance parallel execution	36
New user input to existing inputs	38
Add new XML tag to the existing payload	39
Restconf & RPC call to ATOM	40
How Workflow Can Program Against Various Events in ATOM	46

NAAS-EVENT	46
TSDDB Alerts	47
Delegate Classes	47
AtomRpcDelegate	48
AtomRestconfDelegate	48
AtomEventsSubscriptionDelegate	49
http-connector	49
Scripting support in ATOM workflow	50
Device Connection Timeout	50
Handling larger responses from device	50
Commenting code	52
Custom form fieldTypes in ATOM workflow	52
Examples for custom fieldTypes	53
Validations/Constraints for custom form fields	62
ATOM SDK	65
Introduction	65
ATOM SDK folder hierarchy	65
Setting up the environment for ATOM Package Plugin	66
Prerequisites	66
Setting up the environment in Ubuntu	66
Setting up the environment in Windows	66
Setting up the repository for developing packages	67
Migration of Workflows	71
ATOM API Development and Testing Reference	71
References	71
YANG	71
RESTCONF	71
Gradle	71
BPMN	71

ATOM Workflow Development - High Level View



Outline of the document

ATOM Platform provides users to develop with various extensions to out-of-the box capabilities.

- 1) Device Drivers - Device Drivers allow ATOM to work with devices to Collect configuration, Provision Configuration, Collect Performance & Other Operational Data, Execute Show and Diagnostic Commands.
 - a) Configuration Discovery & Provisioning
 - b) Performance & Inventory Collection (SNMP, SNMP Trap, Syslog, Telemetry)
- 2) Network Automation
 - a) Stateful Services like Application Delivery, L3 VPN, etc.,
 - b) MOP Automation like Software Upgrade, Password Rotation etc.,**

The document covers Network MOP Automation Development Flows. Following is a high level breakdown of the content:

1. Workflow Package Development
2. Developing Workflows - BPMN Modelling
3. Deploying Workflow Package

In the [Appendix](#), additional examples, library utils, ATOM SDK and FAQs are mentioned in detail.

Network Automation & MOPs

Network automation scenarios fall into following categories:

1. Network Services

a. **Stateful Services** - Networking provisioning use case with following life cycle:

- i. Discovery - Discovery of existing Services
- ii. Create - Create a green field service
- iii. Update - Update Service. This may be repeated multiple times
- iv. Delete - Retire the service

Examples:

- i. Application Deployment in Data Center
- ii. Layer-3 VPN
- iii. Layer-2 VPN
- iv. Private Cloud to Public Cloud Interconnect

What's involved in developing Stateful services ? - Refer to ATOM Network Services Development SDK

2. Network MOPs

a. **One-Time or Task Oriented Provisioning Activities** - These activities may be repeated from time-to-time but do not need information on prior state.

Examples below:

- i. Password Rotation
- ii. SNMP Community String, SNMP Trap, Syslog config rotation
- iii. Configuration Migration projects like - IPV4 to IPV6 Migration

b. **One-Time or Task Oriented Operational Activities** - Network MOPs typically involve Network maintenance activities like the following:

- i. Device Software Image Upgrade
- ii. Device RMA
- iii. Device Configuration Rollback to a prior state

Typical activities in a MOP are as follows:

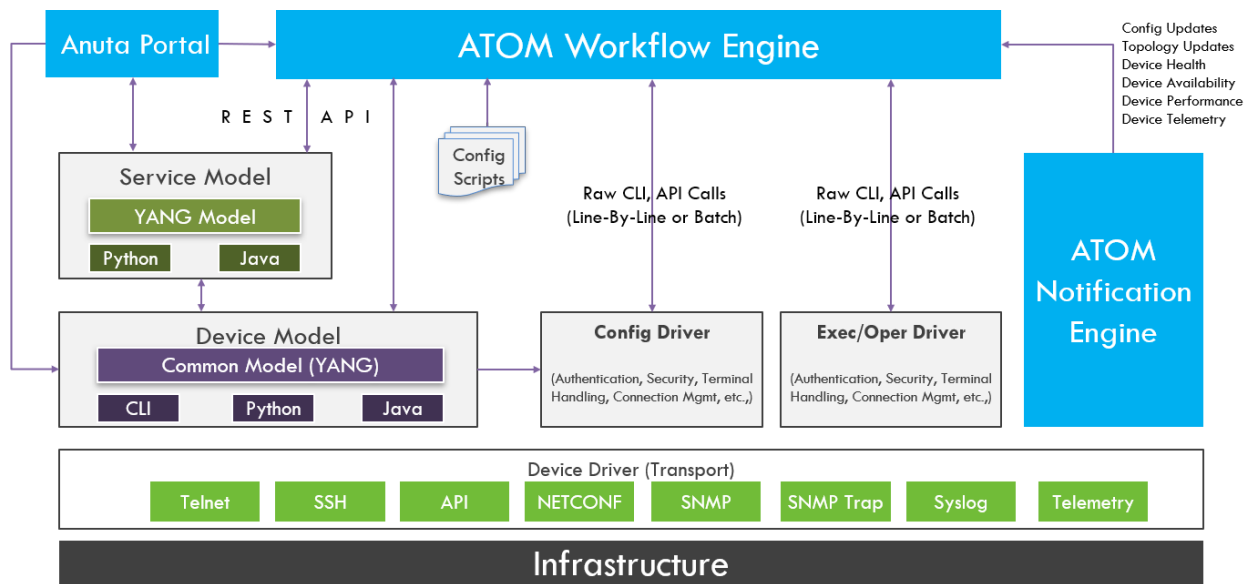
- Performing Actions on Device - Show commands, Exec Commands, Config Commands
- Handling/Parsing Device Response
- Checkpointing various states for comparison
- Conditions on checkpoints

ATOM Workflow provides a mechanism to define various Activities required to build Network MOP.

ATOM Workflow

ATOM Workflow applications work on top of services/apis enabled by various ATOM APIs, Device APIs, Direct CLI invocations, Events etc.,. These applications involve a set of activities or sub-tasks that include fetching data from devices, pushing configurations to devices, executing show commands, executing exec operations like ping, install, acting on device events, timers, end user actions etc.,. Applications usually need to express logic in terms of steps (sequential, parallel, conditional etc) and ATOM Workflow is a natural fit for those requirements.

ATOM workflow engine implements BPMN standard processes. ATOM being a YANG model driven platform, it exposes APIs and models expressed in YANG schema, although ATOM also lets application developers to by-pass device yang models and use CLIs or device native apis directly.



Workflow Engine Communication

Technical requirements for developing Workflow in ATOM

Workflow Automation can be a combination of Stateless and Stateful actions. In such scenarios MOP will contain stateless actions like pre-checks, while performing API invocations against Device or Service Models to perform stateful transactional action.

Overall Developers would need some familiarity with the following

1. [BPMN](#)
2. ATOM Workflow APIs
3. Device configurations (CLI)

4. A Scripting language (Python, Groovy, JavaScript)
5. ATOM SDK and Tooling - ATOM uses a package structure to ingest application models and programming. Workflow is also ingested using the same package structure. ATOM provides SDK and [GRADLE](#) based Tooling to help develop these packages. In essence, workflow development starts with package development. In the following sections we will elaborate the ATOM SDK and tooling in relation to workflow development. ATOM SDK uses the Gradle build system.
6. [YANG](#) (Especially, when Device yang models are used).
7. [RESTCONF](#)

Workflow Package Development

Create a Workflow package

After the successful one time setup of the ATOM SDK environment (Refer Appendix section [ATOM SDK](#)), you can create the required workflow package as below.

1. Run below command to create the package:

```
python sdk.py -c
```

create.py: This script helps you create different types of package: service package, device package or device driver package.

```
root@User:/home/supritha/Desktop/AtomSDK/atom-package-plugin# python sdk.py -c
Running create script
This script creates a package

Select from the following options
1.SERVICE_MODEL
2.DEVICE
3.DEVICE_DRIVER
choose any one from above:█
```

2. Select the type SERVICE_MODEL package type as shown below

```
This script creates a package

Select from the following options
1.SERVICE_MODEL
2.DEVICE
3.DEVICE_DRIVER
choose any one from above:1
enter the name of the package:█
```

3. Enter the name of the package and other inputs as shown below

```

This script creates a package
Select from the following options
1.SERVICE_MODEL
2.DEVICE
3.DEVICE_DRIVER
choose any one from above:1
enter the name of the package:serverportautomation
enter the description (optional):
enter the atom version (optional):
enter the absolute directory path to create the package (optional):
creating the current directory
destination is: /home/anuta/Music/atomsdk/serverportautomation
initializing the package
:wrapper
:init
BUILD SUCCESSFUL
Total time: 3.866 secs

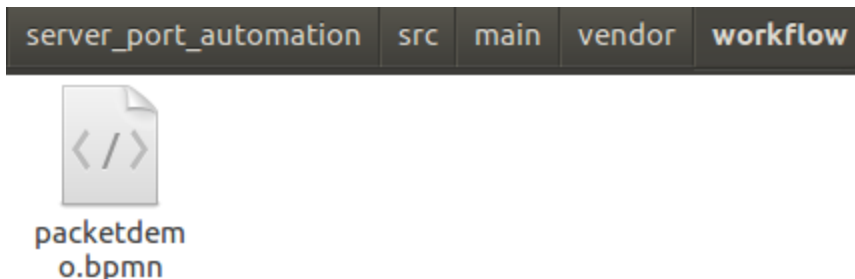
This build could be faster, please consider using the Gradle Daemon: https://docs.gradle.org/2.10/userguide/gradle_daemon.html
Enter the dependency dictionary (optional) :
creating the folder structure
:init
The build file 'build.gradle' already exists. Skipping build initialization.
:init SKIPPED
:initPackage
{} created. src
{} created. src/main
{} created. src/main/scripts
{} created. src/main/vendor
{} created. src/main/model
{} created. src/main/resources
BUILD SUCCESSFUL
Total time: 6.449 secs

This build could be faster, please consider using the Gradle Daemon: https://docs.gradle.org/2.10/userguide/gradle_daemon.html

```

After the successful run of the above build, the service package folder structure for workflow purpose is created.

In the vendor folder create a new folder named *workflow*. After development of workflow bpmn as described in section [Developing Workflows - BPMN Modelling](#) place the bpmn in this folder as shown below.



Update the Dependencies & Version in build.gradle

After a successful creation of a workflow package, there could be some additional package(s) required as 'dependencies'. Accordingly modify the default dependencies listed in the build.gradle file, which is located in the root level of the created package.


```

group 'com.anuta.ncx.packages'
version '8.0.0.0'
apply plugin: 'ear'
apply plugin: 'java'
apply plugin: 'ncx-package-plugin'

repositories {
    mavenCentral()
    flatDir(dirs: "/home/anutauser/Desktop/codegen/atomsdk/packages")
}

dependencies {
    earlib group: 'com.anuta.ncx.packages', name: 'servicemodel', version: '7.0.4.0', ext: 'zip'
    earlib group: 'com.anuta.ncx.packages', name: 'Anuta', version: '7.0.2.0', ext: 'zip'
    earlib group: 'com.anuta.ncx.packages', name: 'bitarray', version: '7.0.0.0', ext: 'zip'
    earlib group: 'com.anuta.ncx.packages', name: 'abstractdevicemodels', version: '7.0.4.0', ext: 'zip'
    earlib group: 'com.anuta.ncx.packages', name: 'devicemodel', version: '7.5.0.0', ext: 'zip'
    earlib group: 'com.anuta.ncx.packages', name: 'pyangbind', version: '7.0.0.0', ext: 'zip'
}

packageXml {
    name 'server_port_automation'
    type 'SERVICE_MODEL'
    description 'server-port-automation Base Package'
    moduleName 'server_port_automation'
    ncxVersion '[8.0.0.0,)'
    deployOnAgent false
    autoStart false
}

buildscript {
    repositories {
        mavenCentral()
        flatDir(dirs: "/home/anutauser/Desktop/codegen/atomsdk/packages")
    }
    dependencies {
        classpath "com.anuta.ncx.packages:ncx-package-plugin:7.0.0.0"
        classpath "org.apache.httpcomponents:httpmime:4.5.3"
        classpath "org.apache.cleressa.ext:org.json.simple:0.4"
    }
}

```

In scenarios where MOP performs stateless actions on the device directly either via CLI/Native APIs make sure dependency of the servicemodel package is there. This package has the required library utilities for connecting to the device and executing CLIs/APIs.

In scenarios where MOP performs API invocations against Device or Service Models, make sure dependency of that respective model package is there.

Let's consider a MOP which performs both stateless actions and API invocations against Juniper Device Models, then make sure dependency of *servicemodel-7.0.4.0*, *juniper-8.0.0.1*, *juniper_cli-8.0.0.1* and *workflowlib-7.5.1.0* are mentioned as below. (The package names are a combination of the name of the package and the version number separated by a hyphen)

```

dependencies {
    earlib group: 'com.anuta.ncx.packages', name: 'servicemodel', version: '7.0.4.0', ext: 'zip'
    earlib group: 'com.anuta.ncx.packages', name: 'juniper', version: '8.0.0.1', ext: 'zip'
    earlib group: 'com.anuta.ncx.packages', name: 'juniper_cli', version: '8.0.0.1', ext: 'zip'
    earlib group: 'com.anuta.ncx.packages', name: 'workflowlib', version: '8.1.0.0', ext: 'zip'
}

```

Resolve the dependencies

Run the command : **gradle build --refresh-dependencies**

Successful execution of this command ensures that the dependencies mentioned in the *build.gradle* file are mapped fine.

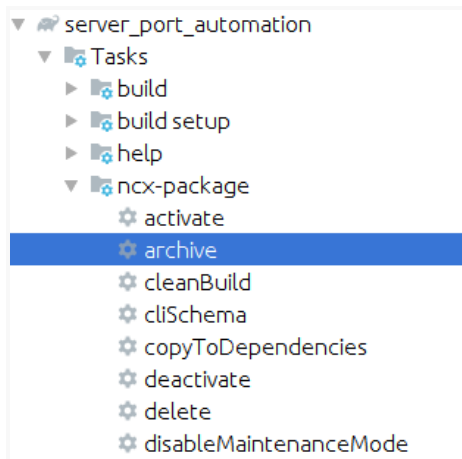
Update the version

In the “*build.gradle*” file, metadata about the package is present in the version & packageXml object. Update the version based on the revision of the service package you are working on.

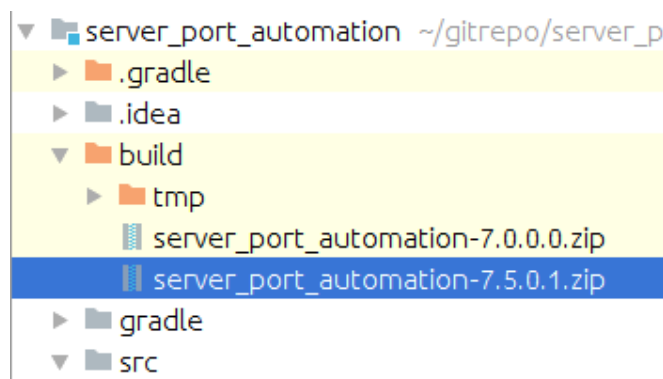
```
group 'com.anuta.ncx.packages'
version '8.0.0.0'
apply plugin: 'ear'
apply plugin: 'java'
apply plugin: 'ncx-package-plugin'
```

Archive the Workflow Package

Once the Workflow bpmn is defined and placed in the vendor/workflow folder of the package structure, use the gradle task “gradle archive” for creating the uploadable zip with its dependencies.



The zip will be stored into the build folder like below



Now the workflow package zip is ready for Upload to ATOM.

Developing Workflows - BPMN Modelling

Workflow can be used in multiple scenarios such as Config provisioning, Software Upgrade, Protocol migration, Closed Loop automation etc..

Use Case 1 - Software Upgrade

Frequently organisations are confronted with the challenge of upgrading their network devices to the latest patch version. In Large scale Enterprise environments, we will be running a similar version across the network based on their role.

These situations are very critical in the lifecycle of a device, and need to execute them carefully with predefined MOP approved by Network Architects.

Typical Software Upgrade will involve the following basic constructs:

1. User Inputs to begin the change like Device, Image Version, Image Repo Details (FTP, SCP etc..), Software Image
2. Pre Checks - Set of pre checks before proceeding for the actual migration such as check current running software version, Hardware details, Config backup, Interface and Protocol level checks etc..
 - a. If any of the prechecks failed then raise a ticket and stop the migration activity for that device.
3. Copy the Image from Remote repository to the device (eg: Disk, NVRAM etc..) and set the boot options as required.
 - a. Raise an incident if the image copy fails for some reason.
4. Request Network Admin for device reboot and proceed on approval.
5. Check for devices reachability whichever way is feasible as specified below
 - a. Continuous Ping check within stipulated time. [[Fig 1](#)]
 - b. Listening for any asynchronous notifications like SNMP trap. [[Fig 2](#)]
6. Perform Post checks
 - a. Compare Pre and post check results & if the validation fails then call for a rollback of that device upgrade activity.

The above sequence of steps are readily understood by networking professionals. Now, let us see the thought process to translate that logic into ATOM Workflow. Not all the logic is explained here (A full explanation with diagrams will follow this table).

#	Use case Requirements	Relevant Concept in ATOM Workflow	Notes	Which Artifact in package this goes into ?
1	User Inputs to begin the change like Device, Image Version, Image Repo Details (FTP, SCP etc.), Software Image	User Inputs are captured via 'User Tasks' in workflow.	User Task is a BPMN concept, hence it goes into the bpmn model.	bpmn
2	Pre Checks - Set of pre checks before proceeding for the actual migration such as <i>check current running software version, Hardware details, Config backup, Interface and Protocol level checks etc..</i>	<p>This involves fetching relevant data from devices. When fetching data two scenarios are possible in ATOM. May be the information is already available in the ATOM database (this happens if device yang models are being used). Or, maybe you want to fetch it by running a command on the device. Either way, fetching is an ATOM api call. You would use the relevant api and interpret the response.</p> <p>Making an api call is done by using relevant "Delegate" class and filling in API inputs. At the same time, interpreting the API response may involve extracting from the xml/json result or parsing the command text.</p> <p>There may be utilities available to help with these programming tasks.</p>	<p>API invocation, extracting/using the response etc are part of the bpmn model itself.</p> <p>Different Delegate classes (aka the API) are explained here.</p>	API call, sending the input, processing output all go into the bpmn
	a. If any of the prechecks failed then raise a ticket and stop the migration activity for that device.	Raising a ticket translates to calling an external service (Such as ServiceNow) via its API.	Refer to 3rd party integrations	API call, sending the input, processing output all go into the bpmn
4	<p>Copy the Image from Remote repository to the device (eg: Disk, NVRAM etc..) and set the boot options as required.</p> <p>a. Raise an incident if the image copy is fails for some reason</p>	<p>Most of the time business functions are made available as YANG rpcs. You can use ATOM UI developer tools to browse through available rpcs. The same list will also be available in the workflow designer.</p> <p>Calling an RPC is acheived by using ATOMRPCDelegate Class.</p>		
5	Request Network Admin for device reboot and proceed on approval .	Approval maps to a UserTask		

The following diagram depicts a fully expressed bpmn process for the above requirement.

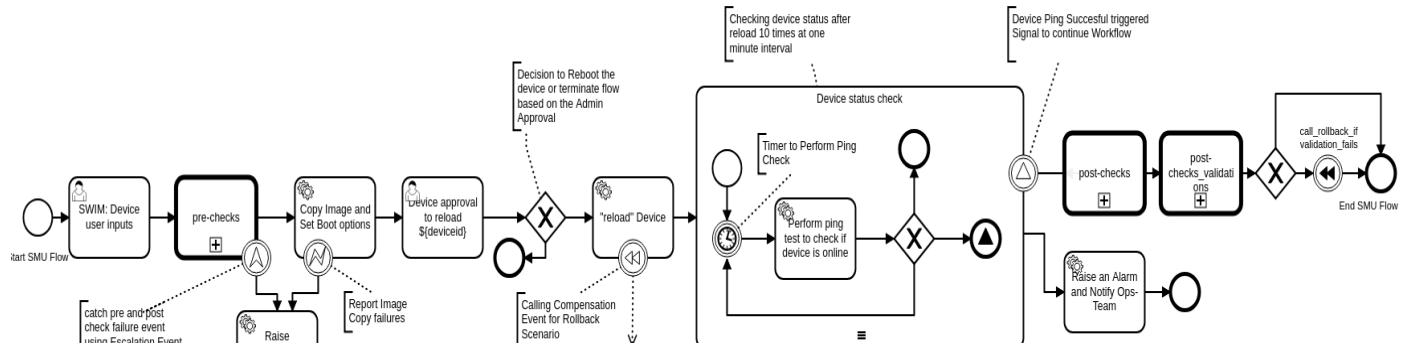


Figure 1 - BPMN Diagram for SMU with Ping check

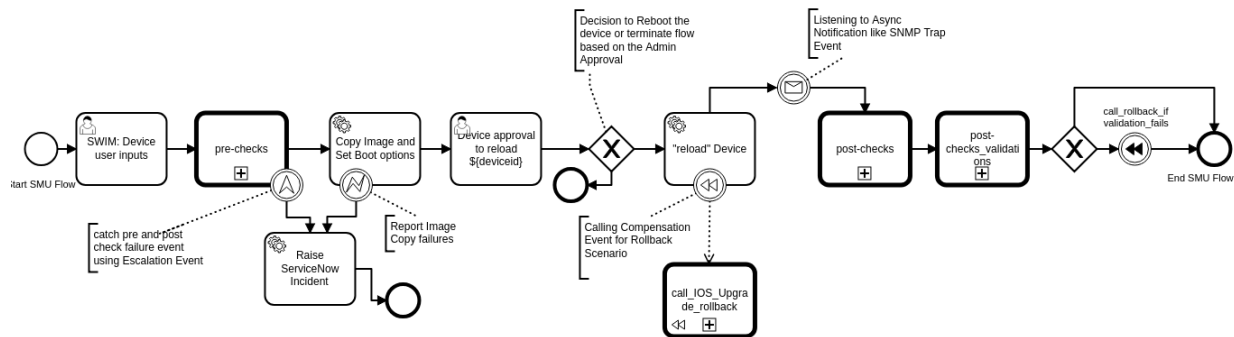
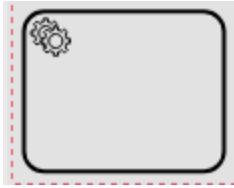

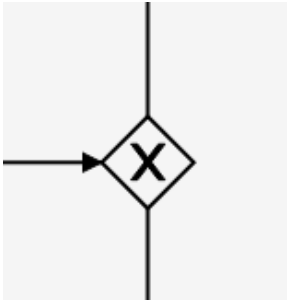


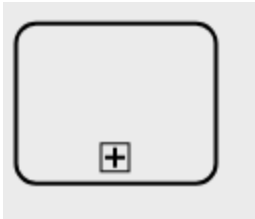

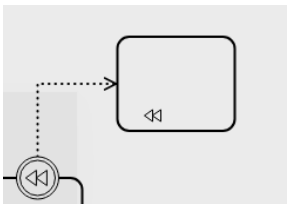


Figure 2 - BPMN Diagram for SMU with Async Notification

BPMN is a fairly large specification. But, the set of constructs we need on a regular basis are small. The following table explains some of the regularly used BPMN constructs.

Workflow Task/ Event type	Interaction/ Description	Task Name	Symbol
User Task	User interacts with ATOM	SWIM: Device user inputs, Device approval to reload	

Service task	ATOM to Device Interaction (CLI/API)	Copy Image and Set Boot options,"reload" Device,Perform ping test to check if device is online	
Service task	ATOM to External API's	Raise an Alarm and Notify Ops-Team,Raise ServiceNow Incident	
Gateway/Decision box	Control flow within the process	NA	
Timers	Synchronous wait events within the process	NA	
SubProcess MultiInstance	Similar to Looping construct in programming	Device status check	
Call Activity	Similar to DRY principle such as Function calls	pre-checks,post-checks,post-checks_validations	
Error Boundary Event	Raise an alarm or incident for any service task failures	NA	
Compensation Activity (Rollback on Failures)	Handle the task failures such as Config Rollback etc..	call_IOS_Upgrade_rollback	

Refer to the [Appendix](#) section for more details.

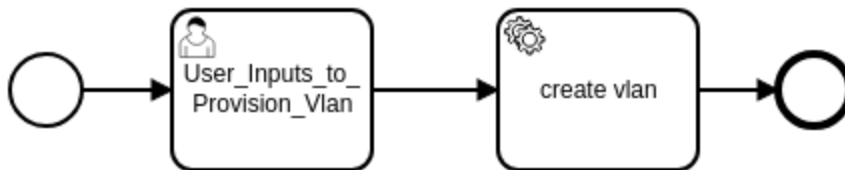
Python Reference	Workflow Construct
For loop	Multi-instance
For loop with range	Multi instance with loop cardinality
If condition	Conditional sequences
Sleep	Timer Duration
Throw and catch exception	Error Handling Event
continue	Non-interrupting Event
break	Interrupting Event
functions	Call activity
Modules	sub-process

Use Case 2 - Config Provisioning

Provision the vlan on the user specified device.

So below are high level tasks to be performed in the workflow.

1. Get inputs from the user such as Device IP, Vlan Number, Vlan description.
2. Form the payload and configure the device



BPMN Diagram for Config Provisioning

User Inputs Form

- First name the whole usecase appropriately like below, so this name can be seen in ATOM UI.

Provision_Vlan

General Listeners Extensions

General

Id
Provision_Vlan x
This maps to the process definition key.

Name
Provision_Vlan

Version Tag
7.5.0.0 x

☒ Executable

- First action is to get inputs from the user, so let's name the task "User_Inputs_to_Provision_Vlan" and "Id" field will be auto-generated, but can be changed if needed.

User_Inputs_to_Provision_Vlan

General Forms Listeners Input/Output Extensions

General

Id
User_Inputs_to_Provision_Vlan x

Name
User_Inputs_to_Provision_Vlan

- To take the inputs from the user, Click on the "Forms" tab next to "General" and give input field names like below.

Forms

Form Key
custom

Form Fields
deviceid
vlan-id
vlan-name

Form Field

ID
vlan-name

Type
string

Label
vlan-name

Default Value

These form inputs given by the user can be accessed anywhere in this workflow task.

Service Task to configure the Device

- Form the payload with these inputs and send that as an output. So we create one “output parameter” with a name called “payload” and “Type” as “script” where script format is “groovy”. The payload formation is done as xml with user inputs. The last line in the script will be considered as output of this first workflow task and stored in the output parameter name “payload” in the above case.
- To perform the POST operation, create a service task and name it as “Create Vlan”
To provision the device with provided CLI commands use the following Java class: “com.anuta.atom.workflow.delegate.AtomRpcDelegate” as shown below.

ServiceTask_1ticbfm

General Listeners Input/Output Field Injections Extensions

General

Id
ServiceTask_1ticbfm x

Name
create vlan /

Details

Implementation
Java Class ▼

Java Class
com.anuta.atom.workflow.delegate.AtomRpcDelegate x

- Provide Input Parameters for the POST operation that includes atom_url, atom_action, atom_payload and then the Output Parameter like below.

ServiceTask_1ticbfm

General Listeners **Input/Output** Field Injections Extensions

Parameters

Input Parameters x +
atom_url : Text
atom_action : Text
atom_payload : Script

Output Parameters x +
vlanoutput : Script

ATOM_URL:

ServiceTask_0nxuauo

General

Listeners

Input/Output

Field Injections

Extensions

Parameters

Input Parameters

atom_url : Text

atom_action : Text

atom_payload : Script

Output Parameters

vlanoutput : Script

Input Parameter

Name

atom_url

Type

Text

Value

/workflowlib:execute-command

ATOM_ACTION:

ServiceTask_0nxuauo

General

Listeners

Input/Output

Field Injections

Extensions

Parameters

Input Parameters

atom_url : Text

atom_action : Text

atom_payload : Script

Output Parameters

vlanoutput : Script

Input Parameter

Name

atom_action

Type

Text

Value

POST

ATOM_PAYLOAD:

Parameters

Input Parameters

atom_url : Text
atom_action : Text
atom_payload : Script

Output Parameters

vlanoutput : Script

Input Parameter

Name
atom_payload

Type
Script

Script Format
Groovy

Script Type
Inline Script

Script

```
def deviceId=execution.getVariable("deviceId");
def vlan-id=execution.getVariable("vlan-id");
def vlan-name = execution.getVariable("vlan-name");

def cmds = ""
vlan ${vlan-id}
name ${vlan-name}
""

"<input><device-id>"+deviceId+"</device-id><command>"+cmds+"</command><splitter>\n</splitter></input>";
```

ATOM_OUTPUT:

Parameters

Input Parameters

atom_url : Text
atom_action : Text
atom_payload : Script

Output Parameters

vlanoutput : Script

Output Parameter

Name
vlanoutput

Type
Script

Script Format
Javascript

Script Type
Inline Script

Script

```
var vlanoutput = execution.getVariable("atom_rpc_output");
vlanoutput;
```

- Place this developed workflow bpmn file in the vendor/workflow folder of the package structure generated initially in section [Creating workflow package using ATOM SDK](#)

Use Case 3 - CLA Remediation

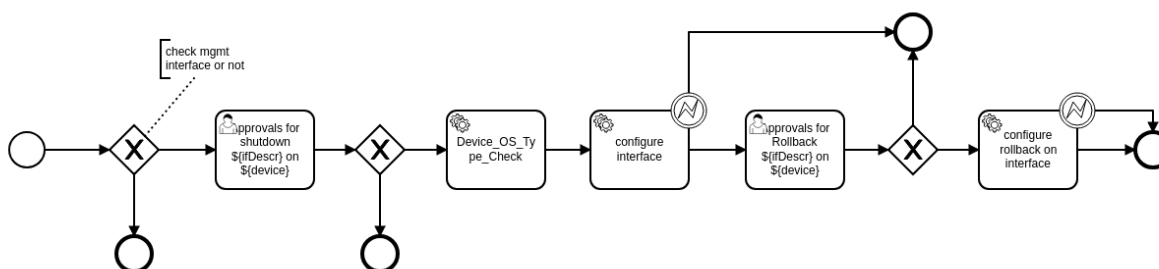
Workflows can be used as action items against NaaS Alerts/TSDB alerts. Atom has various methods to subscribe and listen to alerts in an async manner which can be found in the appendix section [How Workflow Can Program Against Various Events in ATOM](#) of workflow guide.

Following use case requires us to shutdown the interface which has flapped more than 10 times in the last 15 mins. To achieve this use case we require four things :

- Create a SNMP collection.
- Create an alert definition.
- Create Workflow.
- Map the workflow as an action item for the alert definition.

For steps 1,2 and 4 refer to the main Atom Guide.

Below is a schematic view of the workflow we will develop :



First Step to create the workflow would be to break the use-case into small portions:

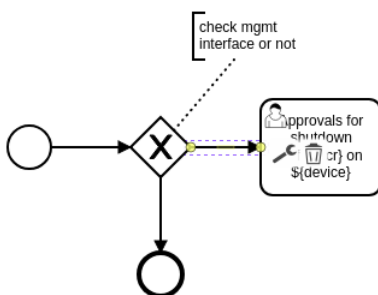
- Check if the flapping interface is the management interface of the device. If yes, then terminate the workflow without any action.
- Take an approval from the Network Admin to shutdown the interface.
- Once the Approval is received, check the device os type so that we generate corresponding payload to be pushed to the device.
- Shutdown the interface
- We can then wait for the Network Admin approval to unshut the interface after they troubleshoot the issue.
- Once approved we rollback the shut commands and terminate the flow.

This same flow can be augmented with various steps like :

- Opening a Ticket in the ITSM tool and getting approvals on the ITSM tool. [Refer API Integration]
- Performing certain pre or post checks.[Troubleshooting logs can be collected and appending these logs to ITSM tool.]
- Adding Error Handling for all the tasks and covering negative scenarios.
- Config retrievals and Correlation for alert enrichment and impact analysis.

Before we begin building the workflow, ATOM sends all the relevant alert details [severity,device_id,ifDescr,acknowledged/resolved status,alertname,message,entity affected,alert record id] as seed data whenever the workflow is triggered.These can be used as process variables in the workflow and need not be user inputs.

Step-1: Check if the flapping interface is the management interface of the device. If yes, then terminate the workflow without any action. We use a Decision gateway from our palette and check if the interface name is among the standard management interfaces for devices.



Details

Condition Type

Expression

Expression

\$ifDescr != "GigabitEthernet0" && ifDescr != "GigabitEthernet1" && ifDescr != "mgmt0" && ifDescr != "fxp0" && ifDescr != "fxp0.0" && ifDescr != "GigabitEthern

Step-2: Take an approval from the Network Admin to shutdown the interface. Create a User form with a boolean input that will be used as a decision control to shut down the interface.

UserTask_1e96nnr

General Forms Listeners Input/Output Extensions

Forms

Form Key
custom

Form Fields
approval_to_shut_Interface

Form Field

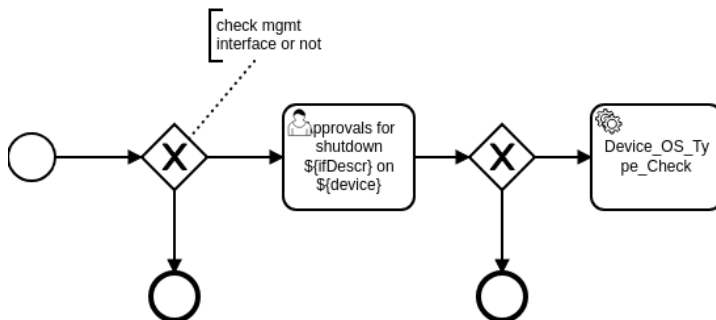
ID
approval_to_shut_Interface

Type
boolean

Label
Value can be true(interactive) or false(for non-interactive)

Default Value

Step-3: Once the Approval is received, check the device os type so that we generate corresponding payload to be pushed to the device. For this step we use the already parsed basic inventory content of the device stored in our yang engine via a restconf Operation.Details about the java class and required params can be found in the appendix.Refer below screenshots for any queries.



ServiceTask_07sxfdo

General

Listeners

Input/Output

Field Injections

Extensions

Parameters

Input Parameters

atom_url : Script
atom_action : Text
atom_task_prefix : Text

Output Parameters

os_type : Script

Output Parameter

Name

os_type

Type

Script

Script Format

Groovy

Script Type

Inline Script

Script

def resp = execution.getVariable("atom_restconf_output");
def resp2 = new XmlParser().parseText(resp);

def finalRes = "\${resp2.text()}" as String
println finalRes;

ServiceTask_07sxfdo

General

Listeners

Input/Output

Field Injections

Extensions

Parameters

Input Parameters

atom_url : Script
atom_action : Text
atom_task_prefix : Text

Output Parameters

os_type : Script

Input Parameter

Name

atom_url

Type

Script

Script Format

Groovy

Script Type

Inline Script

Script

def deviceid = execution.getVariable("device");
'/controller:devices/device='+deviceid+'/ostype-string';

Step-4: Shutdown the interface. For this step we use the existing credentials in the atom database , login to the device via any transport [SSH/API] and execute the desired action. We use a custom RPC which is available out of the box [workflow_utils:execute-command] for implementing this step.Note that the same RPC can be used for executing any commands on the device.[Pre post checks show /configuration commands]. XML parsing can be done via groovy script as shown below.

ServiceTask_0wo974v

General | Listeners | **Input/Output** | Field Injections | Extensions

Parameters

Input Parameters

atom_uri : Text	x	+
atom_action : Text		
atom_payload : Script		
atom_task_prefix : Text		

Output Parameters

	x	+
--	---	---

Input Parameter

Name

atom_payload

Type

Script

Script Format

Groovy

Script Type

Inline Script

Script

```
def deviceId=execution.getVariable("device");
def interface_name = execution.getVariable("ifDescr");
def os_type = execution.getVariable("os_type");

def cmds = "";
if (os_type == "IOS" || os_type == "IOSXE" || os_type == "IOSXR" || os_type == "NXOS") {
    cmds = "interface "+interface_name+"!\nshutdow\n!\n";
}
else if (os_type == "JUNOS") {
    cmds = "set interfaces "+interface_name+" disable!\n";
}

"<input><device-id>"+deviceId+"</device-id><command>"+cmds+"</command></input>";
```

Step 5 & 6 : We can then wait for the Network Admin approval to unshut the interface after they troubleshoot the issue. Once approved we rollback the shut commands and terminate the flow. These steps are similar to approval and shutting down in implementation and can be copy pasted and edited wherever necessary.

Place this developed workflow bpmn file in the vendor/workflow folder of the package structure generated initially in section [Creating workflow package using ATOM SDK](#)

Deploying & Operating on Workflows

Please refer to **ATOM User/Admin Guide** for details on Uploading, Deploying and Inspecting Workflows.

Appendix

ATOM Workflow FAQs & Examples

Below we cover many FAQs and Examples of workflow for quick understanding of support and usage.

ATOM Workflow Activities

Timer

Timer can be configured in any of the following ways,

- 1) Time Date
- 2) Time Duration
- 3) Time Cycle

All the configurations are based on [ISO 8601](#)

We can use the following symbol and configuration in the properties panel.



1. Date

If we want to add wait between the tasks for a fixed time and date or start workflow after a fixed time and date, then date can be used. The configuration would look like below, where we can specify how long the timer should run before it is fired. In the example below, the timer will run till 1st July 2019, 12:13:14 UTC timezone, after which it is fired & the next task is triggered.

Details

Timer Definition Type
Date ▼

Timer Definition
2019-07-01T12:13:14Z x

Initiator

Asynchronous Continuations
☐ Asynchronous Before
☐ Asynchronous After

Job Configuration

2. Time Duration

If we want to add wait time between the tasks for a fixed time or start workflow after a fixed time, then duration can be used. The configuration would look like below, where we can specify how long the timer should run before it is fired. In the example below, the timer will run till 5 minutes, after which it is fired & the next task is triggered.

IntermediateCatchEvent_08weno0

General | Listeners | Input/Output | Extensions

General

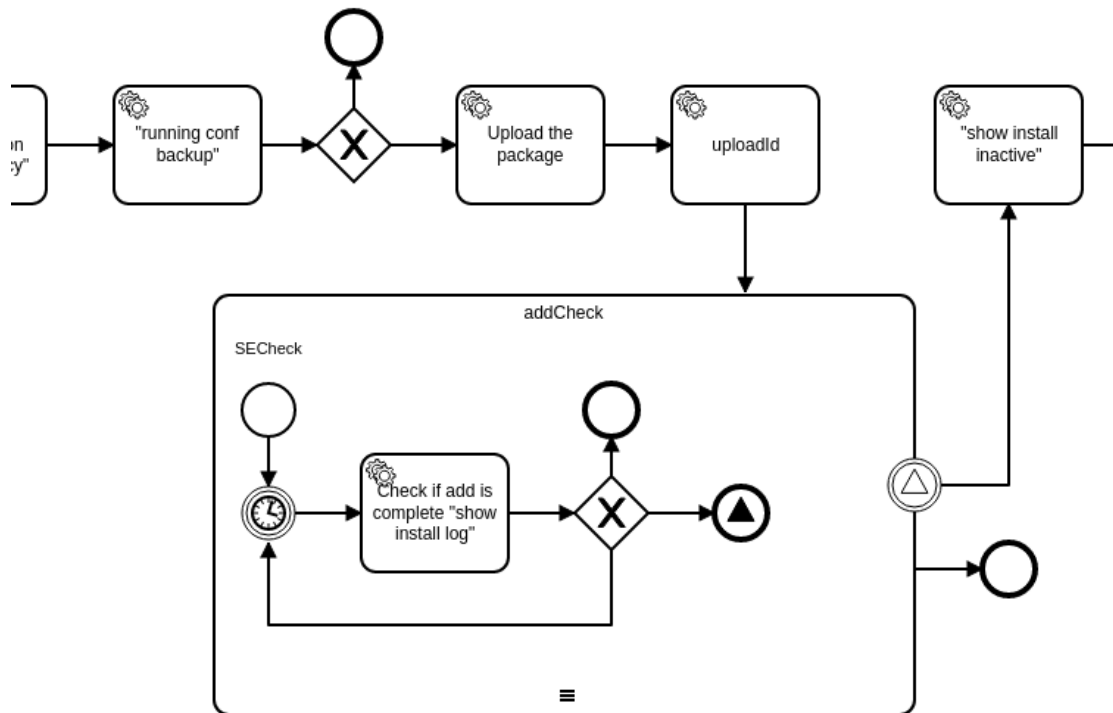
Id
IntermediateCatchEvent_08weno0 x

Name

Details

Timer Definition Type
Duration ▼

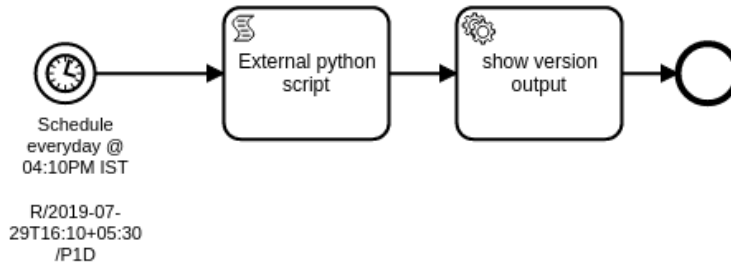
Timer Definition
PT5M x



3. Time Cycle

If we want to start a workflow periodically, then a cycle can be used. The configuration would look like below, where we can specify repeating intervals. In the example below, workflow will run every day starting from 29th July 2019, 04:10 PM IST timezone, without any end since R does not have any value.

Details	
Timer Definition Type	Cycle
Timer Definition	R/2019-07-29T16:10+05:30/P1D
Initiator	
Asynchronous Continuations	
<input type="checkbox"/>	Asynchronous Before
<input type="checkbox"/>	Asynchronous After



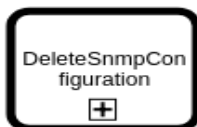
Sub-process

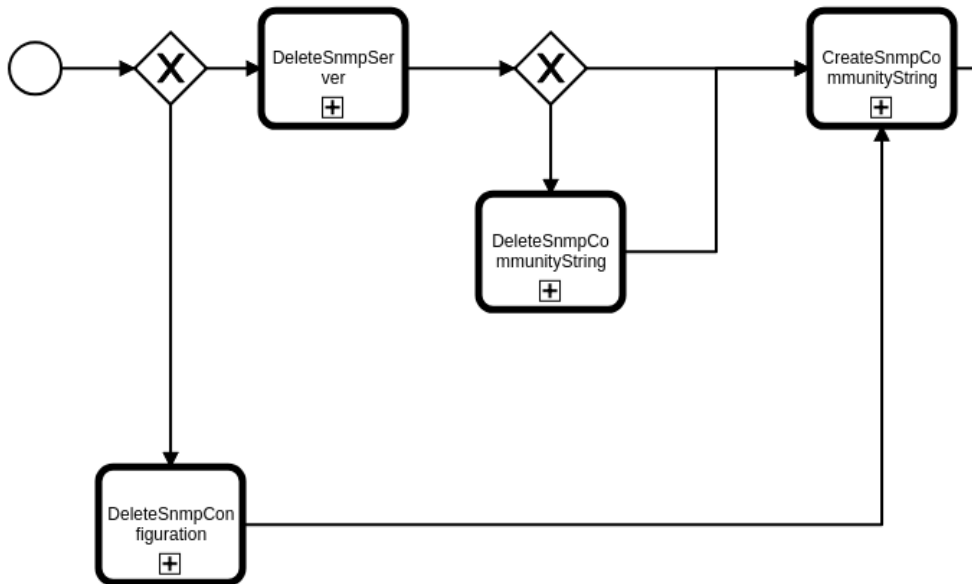
In ATOM Workflow Modeler we can include a bpmn file which can be treated as a generic library into another bpmn file.

For example:

There is a bpmn file which will add a vlan or delete vlan. It can be added into a complex workflow where we have a vlan addition requirement serving as a reusability.

In the below example “**DeleteSnmpConfiguration**” is a subprocess which is a bpmn file where icon representation will be as follows.





How we map other bpmn file:

In the properties panel, we have to provide “call Activity type” as “BPMN” by providing the bpmn file name as input for field “Called Element” and “binding” as “latest” from the dropdown as shown below.

deletesnmpconfiguration
General Variables Listeners Input/Output Extensions

General

Id
deletesnmpconfiguration

Name
DeleteSnmpConfiguration

Details

CallActivity Type
BPMN

Called Element
deleteconfiguration

Binding
latest

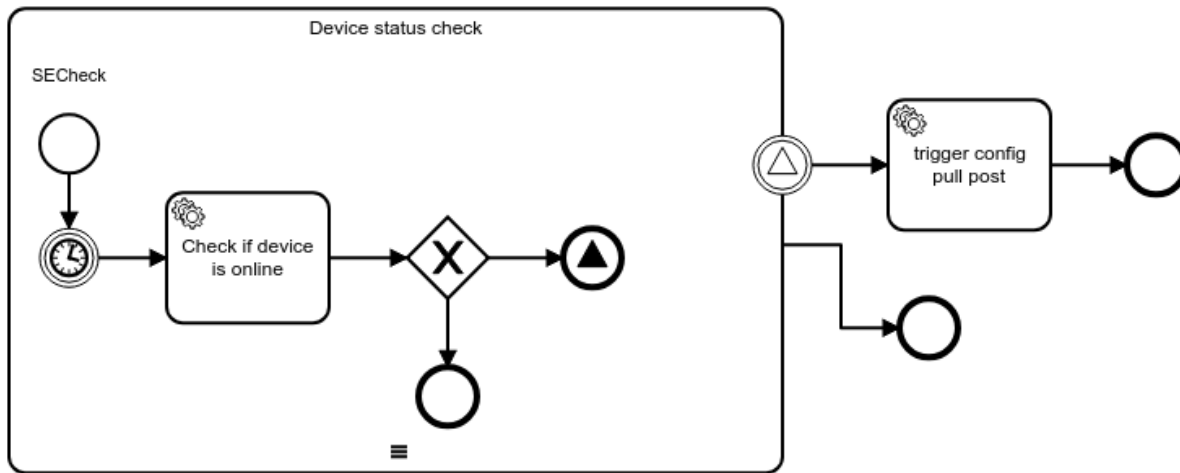
Tenant Id

☐ Business Key

Signal End Event & Boundary Event

Below we consider an example where Signal End Event & Boundary Events are used.

Example:



Here we are checking 'Device Inventory' of the device in a loop after device reboot operation.

- 1) If the 'Device Inventory' is successful either in one or two or more(up to 10) iterations, then it will exit from the signal end event.
- 2) To execute another step first it will verify the signal boundary event. If both matches then it will redirect to the next step else it will stop subprocess.

SubProcess_0cyi6yn

General | Listeners | Input/Output | Extensions

General

Id
SubProcess_0cyi6yn

Name
Device Status check

Multi Instance

Loop Cardinality
10

Collection

Element Variable

Completion Condition

☐ Multi Instance Asynchronous Before

☐ Multi Instance Asynchronous After

A signal end event can be used to end a process instance using a named signal.

When deploying a process definition with one or more signal end events, the following considerations apply:

The name of the signal end event must be unique across a given process definition, i.e., a process definition must not have multiple signal end events with the same name. So first we

need to get the current process id and append to the signal end event name then it will be unique.

ServiceTask_0ep8iow

General Listeners **Input/Output** Field Injections Extensions

Parameters

Input Parameters

- atom_url : Text
- atom_action : Text
- atom_payload : Script
- continue_on_remote_error : Text
- atom_task_prefix : Text

Output Parameters

- inventoryStatus : Script
- get_current_process : Script**

Output Parameter

Name
get_current_process

Type
Script

Script Format
Groovy

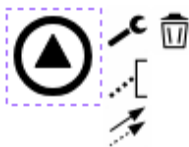
Script Type
Inline Script

Script

```
def currentprocessid = execution.getProcessInstanceId();
execution.setVariable("currentprocessid", currentprocessid);
```

Same way we need to apply Boundary events also.

Signal End Event Symbol and definition in above example



EndEvent_1myb5ft

General

Variables

Listeners

Input/Output

Extensions

General

Id

EndEvent_1myb5ft

Name

Details

Signal

UpgradeCheck_\${currentprocessid} (id=Signal_0kdvnhl)

Signal Name

UpgradeCheck_\${currentprocessid}

Asynchronous Continuations

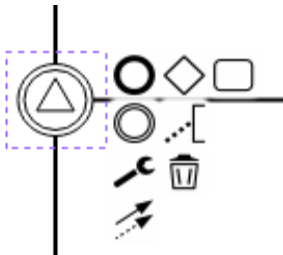
☐ Asynchronous Before

☐ Asynchronous After

Documentation

Element Documentation

Signal Boundary Event symbol and definition in above example



BoundaryEvent_0sf4o3e

General
Listeners
Extensions

General

Id

BoundaryEvent_0sf4o3e

Name

Details

Signal

+

UpgradeCheck_\${currentprocessid} (id=Signal_0kdvnhl)

▼

Signal Name

UpgradeCheck_\${currentprocessid}

x

Asynchronous Continuations

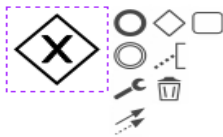
☐ Asynchronous Before
☐ Asynchronous After

Documentation

Element Documentation

Decision box

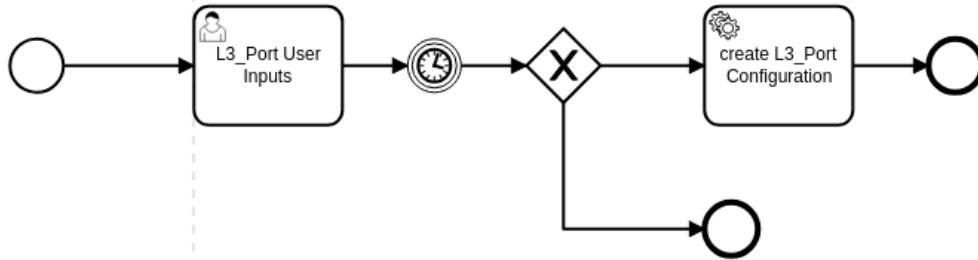
The following symbol represents the decision making based on the conditional statements :



The conditions will be specified on the connectors from the decision box

For example:

If we want to check whether the “payload” variable from User Inputs is not empty and take decision accordingly like below



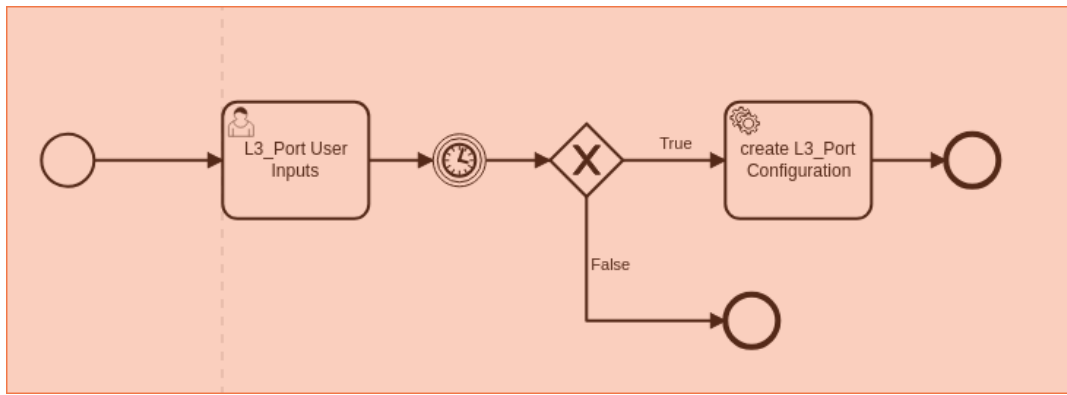
We will write condition on the connectors using condition-type as “expression” for both connectors with respective conditions.

True case:

Details	
Condition Type	Expression
Expression	<code>\${payload != None}</code>

False case:

Details	
Condition Type	Expression
Expression	<code>\${payload == None}</code>



If “payload” is not “None”, control will direct to “Create L3_port Configuration” task otherwise control will exit to the following exit symbol given.



Multi-instance parallel execution

Create a new task and then change type to ‘Call Activity’ and select ‘Parallel Multi Instance’ after that do call activity for any BPMN file.

The activity with the plus sign is called a collapsed subprocess.

The plus(+) sign suggests that you could click on it and make the subprocess expand.

The Parallel(|||) sign acts as multi-instance execution with parallelly and each instance stored in a separate process ids in the ATOM workflow instance.

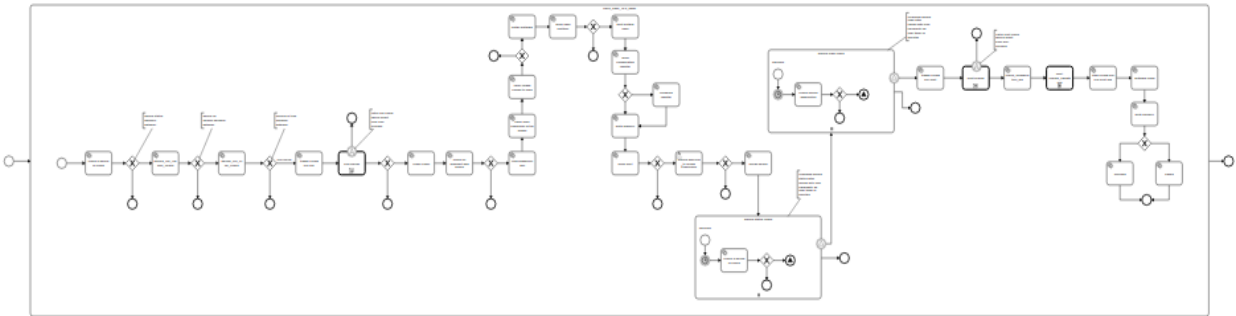
Append the current process id wherever we are using signal end and boundary events.

If we run call activity with parallel then every subprocess is a unique automatically

Parent Process



Sub Process



Refer below snapshot for multi-instance settings.

General	Variables	Listeners	Input/Output	Extensions
General				
Id Task_10gjpax				
Name SWIM_Cisco_3850_16.x				
Details				
CallActivity Type BPMN				
Called Element cisco_3850_16.x_swim				
Binding latest				
Tenant Id 				
<input type="checkbox"/> Business Key				
Delegate Variable Mapping 				
Multi Instance				
Loop Cardinality 				
Collection deviceIdList				
Element Variable deviceId				
Completion Condition 				
<input checked="" type="checkbox"/> Multi Instance Asynchronous Before				
<input type="checkbox"/> Multi Instance Asynchronous After				
<input checked="" type="checkbox"/> Multi Instance Exclusive				
Multi Instance Retry Time Cycle 				
Asynchronous Continuations				
<input checked="" type="checkbox"/> Asynchronous Before				
<input type="checkbox"/> Asynchronous After				
<input type="checkbox"/> Exclusive				

New user input to existing inputs

1. Select “Create Task” and change type as “User Input”.
2. Click on “User Input” task.
3. In the “properties panel” on the right, click on the tab called “forms”.
4. Add new input from the user by clicking on the “+” sign in the form.
5. “Form Key” should always be selected as custom.
6. Provide the user input name you want in the ‘ID’ field, which will be displayed on ATOM UI.
7. Select the “Type” for the user input from the dropdown.
8. Provide the field description you want in the ‘Label’ field, which will be displayed on the ATOM UI.
9. Provide the default value if any.

UserTask_1mswi2k

General Forms Listeners Input/Output Extensions

Forms

Form Key
custom

Form Fields

- static_route_dest_ip_address
- static_route_dest_community
- static_route_next_hop_ip
- ae_interface_name__unit_number__description__mtu
- ae_interface_ipv4_address
- ae_interface_ipv6_address
- Member_1_Name__description
- Member_2_Name__description
- deviceid
- New_user_input**

Form Field

ID
New_user_input

Type
string

Label
New_user_input

Default Value

10. This input value given for “ID” parameter can be referred/used all through the workflow in Groovy/Javascript coding like following example snippet:

Groovy:

```
def nhips =
execution.getVariable("ae_interface_name___unit_number___description___mtu");
```

Javascript:

```
var operation = execution.getVariable("community-string-to-be-deleted");
```

Add new XML tag to the existing payload

To add a new tag to the payload, we need to add the respective xml payload to the existing payload in the task “User Inputs” .

Example:

```
<vlans>
  <vlan>
    <name>VXLAN-1005</name>
    <vlan-id>1005</vlan-id>
    <vxlan>
      <vni>1005</vni>
      <ingress-node-replication/>
    </vxlan>
  </vlan>
</vlans>
```

Here vlan-id can be input from the user .We will add user input field as explained in “[Adding a new user input:](#)”

Note: When we are adding a new tag in payload, it should be supported in device models.

In the “input/output” section of “User Input” task, get the data from the input variable given by user and add it in the xml like follow:

Getting value:

```
def userVlan = execution.getVariable("vlan-id");
```

Form payload:

```
<vlans>
  <vlan>
    <name>' +userVlan+ '</name>
    <vlan-id>' + userVlan+ '</vlan-id>
    <vxlan>
      <vni>' + userVlan+ '</vni>
      <ingress-node-replication/>
    </vxlan>
  </vlan>
</vlans>
```

Now include the above modified payload and the code in the output parameter section in the Input/Output tab of “User Inputs”.

UserTask_1mswi2k

General Forms Listeners **Input/Output** Extensions

Parameters

Input Parameters x +

Output Parameters x +

payload : Script

Output Parameter

Name
payload x

Type
Script

Script Format
Groovy x

Script Type
Inline Script

Script

```
def nhiparr = nhips.split("__");
def nhipList = nhiparr.toList();

println "fine till here";
def atm = '<configuration><interfaces> <interface><name>' + destipList[0] + '</name><description>' + destipList[1] +
'</description><ether-options><ieee-802.3ad><lacp><force-up>true</force-up></lacp><bundle>' + nhipList[0] +
'</bundle></ieee-802.3ad></ether-options></interface><interface><name>' + destmaskList[0] + '</name>
<description>' + destmaskList[1] + '</description><ether-options><ieee-802.3ad><bundle>' + nhipList[0] +
'</bundle></ieee-802.3ad></ether-options></interface><interface><name>' + nhipList[0] + '</name><description>' +
nhipList[2] + '</description><mtu>' + nhipList[3] + '</mtu><aggregated-ether-options><lacp><active>true</active>
</lacp></aggregated-ether-options><unit><name>' + nhipList[1] + '</name><family><inet><address><name>' +
aeinterfaceip4address + '</name></address></inet><inet6><address><name>' + aeinterfaceip6address +
'</name></address></inet6></family></unit></interface> </interfaces><routing-options><static><route>
<community>' + destcommunity + '</community><name>' + destipaddress + '</name><next-hop>' + nexthopip +
'</next-hop></route></static></routing-options></configuration>';
println "payload";
println atm;
atm;
```

Restconf & RPC call to ATOM

Any Restconf call to ATOM from Workflow is done by using the java class
“com.anuta.atom.workflow.delegate.AtomRestconfDelegate”

ServiceTask_1wd6m9q

General | Listeners | Input/Output | Field Injections | Extensions

General

Id
ServiceTask_1wd6m9q

Name
create L3_Port Configuration

Details

Implementation
Java Class

Java Class
com.anuta.atom.workflow.delegate.AtomRestconfDelegate

Asynchronous Continuations

☐ Asynchronous Before

☐ Asynchronous After

On the contrary, if an RPC call is made to ATOM then we need to use the below java class:
“com.anuta.atom.workflow.delegate.AtomRpcDelegate”

Task_1ykgo6g

General | Listeners | Input/Output | Field Injections | Extensions

General

Id
Task_1ykgo6g

Name
Create_RPCcncx_exec:ncx_script_execute

Details

Implementation
Java Class

Java Class
com.anuta.atom.workflow.delegate.AtomRpcDelegate

Asynchronous Continuations

☐ Asynchronous Before

☐ Asynchronous After

Documentation

Element Documentation

POST needs the three parameters basically:

- > URL - used for POST or GET or UPDATE operations
- > ACTION - Can be one of POST/GET/UPDATE

-> PAYLOAD (In case of GET this would be not required)

The screenshot shows the 'ServiceTask_1wd6m9q' configuration window with the 'Input/Output' tab selected. The 'Parameters' section is divided into 'Input Parameters' and 'Output Parameters'. The 'Input Parameters' list includes 'atom_url : Script', 'atom_action : Text', and 'atom_payload : Script'. The 'Output Parameters' list includes 'L3_Port_output : Script'. Each list has a small 'x' icon to remove and a '+' icon to add parameters.

RESTCONF Representation

URL:

This screenshot shows the 'ServiceTask_1wd6m9q' configuration window with the 'Input/Output' tab selected. The 'Parameters' section is divided into 'Input Parameters' and 'Output Parameters'. The 'Input Parameters' list includes 'atom_url : Script', 'atom_action : Text', and 'atom_payload : Script'. The 'Output Parameters' list includes 'L3_Port_output : Script'. Below the 'Input Parameters' list, the 'Input Parameter' details for 'atom_url' are shown. The details include: Name: 'atom_url', Type: 'Script', Script Format: 'Javascript', Script Type: 'Inline Script', and a Script code block containing the following JavaScript code:

```
var deviceid=execution.getVariable("deviceid");
if (deviceid == null) {
    deviceid = "172.16.5.106";
}

'/controller:devices/device='+deviceid+'/configuration';
```

ACTION:

ServiceTask_1wd6m9q

General Listeners **Input/Output** Field Injections Extensions

Parameters

Input Parameters x +

- atom_url : Script
- atom_action : Text**
- atom_payload : Script

Output Parameters x +

- L3_Port_output : Script

Input Parameter

Name

atom_action x

Type

Text ▼

Value

UPDATE

PAYLOAD:

ServiceTask_1wd6m9q

General Listeners **Input/Output** Field Injections Extensions

Parameters

Input Parameters x +

- atom_url : Script
- atom_action : Text
- atom_payload : Script**

Output Parameters x +

- L3_Port_output : Script

Input Parameter

Name

atom_payload x

Type

Script ▼

Script Format

Groovy x

Script Type

Inline Script ▼

Script

```
def destipList=execution.getVariable("payload");
destipList;
```

Here we are forming a payload from the previous task “User Inputs” which is stored in output variable “payload”. So we used “execution.getVariable(“payload”);” to get the content and send it.

RPC Representation

URL:

ServiceTask_0kri8kg

General Listeners **Input/Output** Field Injections Extensions

Parameters

Input Parameters x +

- atom_url : Text
- atom_action : Text
- atom_payload : Script
- continue_on_remote_error : Text

Output Parameters x +

- inventoryStatus : Script

Input Parameter

Name

atom_url x

Type

Text ▼

Value

/controller:run-device-inventory

ACTION:

ServiceTask_0kri8kg

General Listeners **Input/Output** Field Injections Extensions

Parameters

Input Parameters x +

- atom_url : Text
- atom_action : Text
- atom_payload : Script
- continue_on_remote_error : Text

Output Parameters x +

- inventoryStatus : Script

Input Parameter

Name

atom_action x

Type

Text ▼

Value

POST

PAYLOAD:

The screenshot shows the configuration interface for a ServiceTask named 'ServiceTask_0kri8kg'. The 'Input/Output' tab is selected. Under the 'Parameters' section, there are two lists: 'Input Parameters' and 'Output Parameters'. The 'Input Parameters' list contains four items: 'atom_url : Text', 'atom_action : Text', 'atom_payload : Script' (which is highlighted in blue), and 'continue_on_remote_error : Text'. The 'Output Parameters' list contains one item: 'inventoryStatus : Script'. Below these lists is a detailed view for the 'atom_payload' input parameter. It shows the 'Name' as 'atom_payload', the 'Type' as 'Script', the 'Script Format' as 'Groovy', and the 'Script Type' as 'Inline Script'. The 'Script' field contains the following Groovy code:

```
def deviceid=execution.getVariable("device_id");  
"<input><device-id>" + deviceid + "</device-id></input>";
```

ServiceTask_0kri8kg

General Listeners **Input/Output** Field Injections Extensions

Parameters

Input Parameters x +

- atom_url : Text
- atom_action : Text
- atom_payload : Script**
- continue_on_remote_error : Text

Output Parameters x +

- inventoryStatus : Script

Input Parameter

Name: atom_payload x

Type: Script ▼

Script Format: Groovy x

Script Type: Inline Script ▼

Script:

```
def deviceid=execution.getVariable("device_id");  
"<input><device-id>" + deviceid + "</device-id></input>";
```

Output representation:

After the RESTCONF call the output will be stored in the variable **"atom_restconf_output"**.
On the contrary, for RPC calls the output will be stored in the variable **"atom_rpc_output"**.

ServiceTask_1wd6m9q

General Listeners **Input/Output** Field Injections Extensions

Parameters

Input Parameters

atom_url : Script
atom_action : Text
atom_payload : Script

Output Parameters

L3_Port_output : Script

Output Parameter

Name
L3_Port_output

Type
Script

Script Format
Javascript

Script Type
Inline Script

Script

```
var L3_Port_output = execution.getVariable("atom_rpc_output");
L3_Port_output;
```

The variable which is at the end of the code block will be taken as the output variable of this task to the next task. In the above example we got the content and it will be stored in the output parameter “L3_Port_output”.

How Workflow Can Program Against Various Events in ATOM

ATOM platform publishes the following types of events

#	Type	Description	How to Subscribe
1	NAAS-EVENT	ATOM will generate an event based on the Rules defined such as License expiry or User login attempts and also Device events such as Syslog, SNMP traps are converted into a slightly enhanced format called NAAS-EVENT. Refer Administration → System → Rule Engine section in ATOM User Guide for Event generation.	A Delegate called AtomRpcDelegate is available to subscribe for these alerts by using /rule-engine:execute-rule RPC.

2	TSDB Alerts	Alert rules are submitted beforehand to the time series database(TSDB). When the conditions satisfy TSDB will publish an alert onto a workflow engine.	<p>A Delegate called ATOMEventsSubscriptionDelegate is available to subscribe for these alerts.</p> <p>This serves cases where a workflow needs to wait for a specific telemetry alert.</p> <p>A variation of this is possible. It is called CLA (Closed Loop Automation) where you can designate a process to be executed upon the occurrence of an alert.</p>
---	-------------	--	--

Delegate Classes

Following Table summarizes various Network Automation activities and the target ATOM Delegate Classes to be used.

Type of Workflow Activity	Description	Delegate To Use
Execute a Direct CLI Command to Device	Direct CLI execution will bypass ATOM data model validations.	AtomRpcDelegate
Execute a Direct API Command to Device	Direct API execution will bypass ATOM data model validations.	AtomRpcDelegate
Execute an ATOM RPC	<p>ATOM RPCs are Actions available in ATOM.</p> <p>Example - Run a Diagnostic on the device</p>	AtomRpcDelegate
Execute a RESTCONF against ATOM Data Model based APIs	<p>Activities that execute RESTCONF Operations against YANG Data model driven features like Device Model (Common Model, Native Device Model, or OpenConfig), Service Model or any other ATOM Features.</p> <p>Example - Create-VRF, Create-VLAN, etc.,</p>	AtomRestconfDelegate
Activity to Wait/Act on an Event	<p>Activities involve waiting on a device event or any other asynchronous notification.</p> <p>Example - Waiting on SNMP Trap like Device Reboot, Wait on Interface Utilization Alert etc.,</p>	AtomEventsSubscriptionDelegate
Activity to integrate into a 3rd Party connector	Activities that bypass ATOM Device Management Layer and communicate with	http-connector

	an end-point directly with 3rd Party provider APIs..	
--	--	--

AtomRpcDelegate

This Delegate helps in creating Workflow Activities that execute CLI/API operations against the device or to invoke any ATOM RPCs. This bypasses ATOM Data Device/Other YANG Data model infrastructure like validations etc.,

Parameter	Sample-Value	Description
Java Class	com.anuta.atom.workflow.delegate.AtomRpcDelegate	Used to call Custom RPCs written by user
atom_action	POST	Sets the method of the Custom RPCs request
atom_url	/config-provision:execute-command	To execute any command on the device
	/config-provision:append-task-details	To append commands or any output to Task details for viewing in the specific task being executed.
	/rule-engine:execute-rule	To execute a rule and wait for an event.
	/developerutils:invoke-rest-driver-rpc	To interact with any API device such as RESTCONF/SOAP/NETCONF
atom_payload	<Valid XML Payload>	Sets the corresponding payload of the RPC call (check ATOM API Development and Testing Reference for the expected payload based on RPC)

AtomRestconfDelegate

This Delegate helps in creating Workflow Activities that executes RESTCONF Operations against YANG Data model driven features like Device Model (Common Model, Native Device Model, or OpenConfig), Service Model or any other ATOM Features.

Parameter	Sample-Value	Description
Java Class	com.anuta.atom.workflow.delegate.AtomRestconfDelegate	Used to execute atom defined Restconf calls
atom_action	GET/POST/UPDATE	Set the method of the Restconf request
atom_url	/controller:services/poc-service	Sets the URL of the Restconf call (check ATOM API Development and Testing Reference for the expected URL based on YANG)
atom_payload	<Valid XML Payload>	Sets the payload of the Restconf Call (check ATOM API Development and Testing Reference for the expected payload based on YANG)

NOTE: As the YANG models support is limited based on the vendor, we should make sure that Model or package is available in ATOM to perform any CRUD operations using ATOMRestConfDelegate.

AtomEventsSubscriptionDelegate

Parameter	Sample-Value	Description
Java Class	com.anuta.atom.workflow.delegate.ATOMEventsSubscriptionDelegate	Used to wait for any event such as SNMP trap, Telemetry alert or any custom alert defined in ATOM
atom_notification_payload	<Valid XML Payload>	Sets the corresponding payload of the RPC call
atom_task_name	<any string>	Task name which resembles the alert and shown as part of ATOM tasks

http-connector

Parameter	Sample-Value	Description
Connector ID	http-connector	Used to execute REST calls from ATOM workflow to external/third-party APIs

method	GET/POST/PUT/PATCH	Set the method of the REST request
url	/controller:services/poc-service	Sets the URL of the REST call
payload	<Valid XML Payload> {Valid JSON Payload}	Sets the payload of the REST Call

Other Headers as required by REST call can be added as input parameters like Authentication-tokens, Accept and Content-type Headers.

Scripting support in ATOM workflow

ATOM workflow supports Groovy & Javascript as inline scripts and Python as external scripts or RPCs. Sample groovy code written as part of workflow development can be tested in the following online links.

https://www.tutorialspoint.com/execute_groovy_online.php

Or

<https://groovyconsole.appspot.com/>

Device Connection Timeout

Consider a scenario of a workflow task that has to copy the new software image from tftp-server to the device.

If the workflow task times out while still downloading the image, where the command did execute and complete on the device but the workflow timed out with below error.

Error in device command execution no response from the device 10.92.33.64 in 60 seconds.last response from device : archive download-sw /imageonly /leave-old-sw /no-set-boot tftp://153.6.140.225/c2960s-universalk9-tar.152-2.E8.tar

In the above, the limit of 60 sec is coming from the default value taken in the ATOM credential-set attached to the device 10.92.33.64. So increase the value of parameter **CLI Configure Command TimeOut** from 60 to 210 or so based on copy time taken in device.

Handling larger responses from device

Consider a scenario of verifying the MD5 of the new software image on the device.

Lets say you execute the command which computes the MD5 hash and capture that response. Then if you try executing a `.contains()` function on the response to check whether the response contains the expected MD5 hash or not, you may see it to be not working sometimes.

Output Parameter

Name

✕

Type

▼

Script Format

✕

Script Type

▼

Script

```
def resp = execution.getVariable("atom_rpc_output");
def md5 = execution.getVariable("md5_of_image");
def out = resp.contains(md5);
out;
```

28/01/2020, 16:31:01 - 28/01/2020, 16:31:31

Time Taken : 29 seconds

TASKID : EDw6C-Xx1ITyam8Y23TdSuUA

2020/01/28 09:31:01 PM: POST http://atom-frontend:8890/app/restconf/operations/cisco_swim:execute-command

2020/01/28 09:31:01 PM: **Request**

```
{
  "input": {
    "device-id": "10.92.33.64",
    "command": "verify /md5 flash:c2960s-universalk9-mz.152-2.E8/c2960s-universalk9-mz.152-2.E8.bin"
  }
}
```

2020/01/28 09:31:01 PM: interactive = true, pattern-needed = true, timeout = 2400, custom-response-patterns = false

2020/01/28 09:31:31 PM: {"successful":true,"response":"verify /md5 flash:c2960s-universalk9-mz.152-2.E8/c2960s-universalk9-mz.152-2.E8.bin

```
.....
verify /md5 (flash:c2960s-universalk9-mz.152-2.E8/c2960s-universalk9-mz.152-2.E8.bin) = 262e5da4a7440cc37b2e1d0cc2bad7d5
```

eitlab-anuta-2960-01-sw#}

2020/01/28 09:31:31 PM: OUTPUT is: verify /md5 flash:c2960s-universalk9-mz.152-2.E8/c2960s-universalk9-mz.152-2.E8.bin

```
.....
verify /md5 (flash:c2960s-universalk9-mz.152-2.E8/c2960s-universalk9-mz.152-2.E8.bin) = 262e5da4a7440cc37b2e1d0cc2bad7d5 eitlab-anuta-2960-01-sw#
```

This problem can occur if the response (resp) output is more than 2000 chars and getting auto converted as byte characters which would not match with md5 value which is type text. As a general practice use below code snippet where **Utils** converts that byte chars to text and it would work for matching with .contains()

```
import com.anuta.atom.workflow.scripts.Utils
def resp=Utils.getVariable(execution,"atom_rpc_output")
def md5 = execution.getVariable("md5_of_image");
def out = resp.contains(md5);
out;
```

Commenting code

Groovy & Javascript:

Single line comments can be done using //

Multi-liner comments can be done using /* */

Custom form fieldTypes in ATOM workflow

Below section describes how Workflow end-user Input forms can be enhanced by using various customer fieldTypes during workflow development.

1. An User can define workflow custom field types under the Properties section. Add a property and fill 'Id' as 'fieldType' and 'Value' as one among the custom field types possible for a given Type as described below.

Id	Value
fieldType	cidr

- Below are the workflow custom form field types supported under Type 'string'

Type: 'string' and Property: <Id: fieldType> <Value: below custom field types>

cidr
ipaddress
multiLineString
leafRef
multiSelect
Filter

whenstmt

- Below are the workflow custom form field types supported under type 'long'

Type: 'long' and Property: <Id: fieldType> <Value: below custom field types>

int8
int16
int32
int64
uint8
uint16
uint32
uint64
decimal64
Password
readonly

2. For grouping of fields in the form, you can use 'groupBy' as the property Id and Value as required group name which shows up as Title

Examples for custom fieldTypes

1. Examples for **cidr**, **ipaddress**, **multiLineString**, **int8**, **int64**, **uint8**, **decimal64**, etc.

UserTask_0rkxm5t

General

Forms

Listeners

Input/Output

Extensions

Forms

Form Key

custom

x

Form Fields

cidr

x

+

Form Field

ID

cidr

x

Type

string

v

Label

Valid CIDR (A.B.C.D/E for e.x: 172.16.1.1/24)

x

Default Value

1.1.1.1/24

x

Validation

Add Constraint

+

Properties

Add Property

+

Id

Value

fieldType

cidr

x

2. Examples for leafRef, multiSelect

For these custom types we need to add extra properties named yangPath, bindLabel, bindValue

yangPath → Defines the yang schema path

Eg: /controller:devices/device/interface:interfaces/interface

bindLabel → value (does not change)

bindValue → This is the yang **key/non-key leaf** in that specific yangPath

Note :

- 1) **bindLabel** and **bindValue** are required when we want to show other than key values in the yang
- 2) As defined above **bindValue** will vary based on the key/non-key leaf in the respective **yangPath**.
- 3) Schema-browser can help in understanding what is the yang key leaf for a particular schema **yangPath**.

leafRef

User_Inputs

General

Forms

Listeners

Input/Output

Extensions

Forms

Form Key

custom

x

Form Fields

x

+

cidr

ipaddress

multiLineString

leafRef

multiSelect

Form Field

ID

leafRef

x

Type

string

v

Label

Select device from dropdown

x

Default Value

Validation

Add Constraint

+

Properties

Add Property

+

Id

Value

fieldType

leafRef

x

yangPath

/controller:devices/device

x

multiSelect

User_Inputs

General

Forms

Listeners

Input/Output

Extensions

Forms

Form Key

custom

Form Fields

ipaddress

multiLineString

leafRef

multiSelect

filter

Form Field

ID

multiSelect

Type

string

Label

Select multiple values from dropdown

Default Value

Validation

Add Constraint

Properties

Add Property

Id	Value	
fieldType	multiSelect	x
yangPath	/controller:devices/device	x

3. Examples for properties `dynamicValueStmt`, `whenstmt`, `readonly`, `password`

For these custom types we need to add extra properties `yangPath`, `bindLabel`, `bindValue`

dynamicValueStmt - Fetching Interfaces under a selected device example

This helps in showing a filtered set of drop-down values dynamically at run-time based on other form field values selected .

With bindValue and bindLabel

Properties		
Add Property <input data-bbox="407 373 431 401" type="button" value="+"/>		
Id	Value	
fieldType	leafRef	x
yangPath	/controller:devices/device/interface:interfaces/interface	x
dynamicValueStmt	/controller:devices/device[id=current()/leafRef]/interface:interfaces/interface/long-name	x
bindLabel	value	x
bindValue	long-name	x

fieldType → leafRef

yangPath → /controller:devices/device/interface:interfaces/interface

dynamicValueStmt →

/controller:devices/device[id=current()/device]/interface:interfaces/interface/long-name

bindLabel → value

bindValue → long-name

The usage of current()/device indicates to filter interfaces drop-down values to show only interfaces related to that particular device value given as input in other form fields.

If your other workflow form parameter name is device_id and we need interfaces to be displayed for the previous chosen device_id form parameter, the dynamicValueStmt will be below

dynamicValueStmt →

/controller:devices/device[id=**current()/device_id**]/interface:interfaces/interface/long-name

In this example bindValue(long-name) is the key leaf of the

yangPath(/controller:devices/device/interface:interfaces/interface) and bindLabel(value) is the value of the key element.

Without bindValue & bindLabel - Fetching devices present in a Resource pool example

User_Inputs

General

Forms

Listeners

Input/Output

Extensions

Forms

Form Key

custom

Form Fields

multiLineString

leafRef

multiSelect

filter

resource_pool

Form Field

ID

resource_pool

Type

string

Label

Selectr resource pool from dropdown

Default Value

Validation

Add Constraint

+

Properties

Add Property

+

Id	Value	
fieldType	leafRef	x
yangPath	/resourcepool:resource-pools/resource-pool	x

dynamicValueStmt ->

/resourcepool:resource-pools/resource-pool[name=current()/resource_pool]/device/id

User_Inputs

General Forms Listeners Input/Output Extensions

Forms

Form Key
custom

Form Fields

- leafRef
- multiSelect
- filter
- resource_pool
- resource_pool_devices

Form Field

ID
resource_pool_devices

Type
string

Label
Selecr resource pool devices from dropdown

Default Value

Validation

Add Constraint +

Properties

Add Property +

Id	Value
fieldType	leafRef
yangPath	/resourcepool:resource-pools/resource-pool
dynamicValueStmt	/resourcepool:resource-pools/resource-pool[name=current()/resource_pool]/device/id

whenstmt

Using whenstmt a particular workflow form field can be hidden or displayed based on other form field input given.

Usage:

whenstmt → @<form-field-name> == <id of enum>

We cae use logical operators like ||, &&

Ex : Display form field only when enumeration is not equal to test2 and enumeration1 is equal to Value_3brgg9e

@enumeration != "test2" && @enumeration1 == "Value_3brgg9e"

Values

Add Value

Id	Name	
test1	cisco	x
test2	arista	x
test3	anuta	x
Value_3vfeb04	juniper	x

User_Inputs

Forms

Form Key

Form Fields

enumeration1

cidr

ipaddress

multiLineString

leafRef

multiSelect

Form Field

ID

Type

Label

Default Value

Validation

Add Constraint

Properties

Add Property

Id	Value	
fieldType	cidr	x
whenstmt	@enumeration != "test2" && @enumeration1 == "Value_3brgg9e"	x

readonly

User_Inputs

General

Forms

Listeners

Input/Output

Extensions

Forms

Form Key

custom

Form Fields

date

union

allowedValues

password

readonly

Form Field

ID

readonly

Type

string

Label

Readonly

Default Value

anuta

Validation

Add Constraint

+

Properties

Add Property

+

Id	Value
disabled	true

password

User_Inputs

General

Forms

Listeners

Input/Output

Extensions

Forms

Form Key

custom

x

Form Fields

x

+

date

union

allowedValues

password

readonly

Form Field

ID

password

x

Type

string

v

Label

Provide password

x

Default Value

Validation

Add Constraint

+

Properties

Add Property

+

Id

Value

fieldType

password

x

Validations/Constraints for custom form fields

1. Validation for Type “string” can be added as below

UserTask_0in8jih

General

Forms

Listeners

Input/Output

Extensions

Forms

Form Key

custom

Form Fields

string

Form Field

ID

string

Type

string

Label

Vlaue can be anything

Default Value

test

Validation

Add Constraint +

Name	Config	
required		x
minlength	5	x
maxlength	10	x

Properties

Add Property +

- Validation for Type “long” can be added as below

UserTask_0in8jih

General

Forms

Listeners

Input/Output

Extensions

Forms

Form Key

custom

Form Fields

long

Form Field

ID

long

Type

long

Label

Value can be number

Default Value

Validation

Add Constraint +

Name	Config	
min	5	x
max	10	x
required		x

Properties

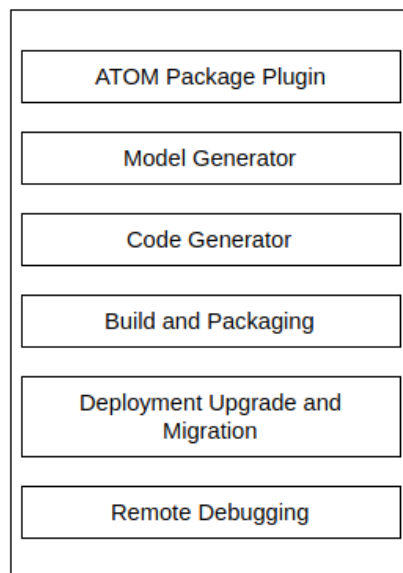
Add Property +

ATOM SDK

Introduction

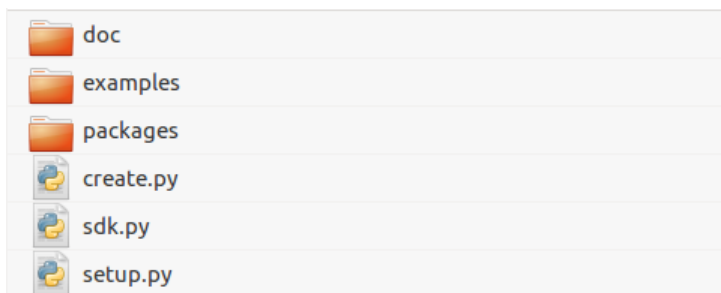
ATOM Software Development Kit (SDK) provides a gradle-based plugin **Package-Plugin jar** that serves as a backbone for any package development in ATOM. ATOM SDK provides CLI and also integrates into IDE like IntelliJ. The plugin enables you to perform the following tasks of Services/Drivers/MOP Development process in ATOM:

- Develop device packages
- Develop service packages (Includes Workflow/MOP)
- Compile, validate, generate device and service packages
- Load Packages to ATOM
- Upgrade of Packages



ATOM SDK folder hierarchy

Unzip the contents of the ATOM SDK zip to view the following folder structure:.



- **doc** - This folder contains README and the plugin documentation.
- **examples** - This folder has package zip files for different types of packages.
- **packages** - The core Package Plugin jar is part of the packages folder, which also has a

few more library and base dependency packages required for development of new devices and service packages.

- *create.py*, *sdk.py*, *setup.py* - These are the python files required for setting up device and service packages environment.

Setting up the environment for ATOM Package Plugin

ATOM Package Plugin supports multiple gradle tasks that help create an environment suited for developing packages. These tasks can be triggered from an **IDE or CLI**.

For the plugin tasks to run, ensure that the prerequisites are met with.

Prerequisites

To setup the environment, you must ensure that the following software requirements are met:

1. Python (2.7.12)
2. Python setup tools
3. Python Pip and Python modules bitarray, cmd2, TAPI, XEGER
4. Pyang(1.7.8). Refer Appendix section for the details of [pyang installation](#).
5. JAVA (java 1.8 or greater)
6. Gradle

For information about installing gradle in your environment, visit <http://gradle.org>.

Setting up the environment in Ubuntu

1. Execute the following commands:

```
sudo apt-get install python python-setuptools
sudo easy_install pip
sudo pip install bitarray
sudo pip install cmd2
sudo pip install tapi
sudo pip install xeger
sudo pip install requests
```

2. Install Oracle JDK for Linux and unzip it.

Set the JAVA_HOME environment variable pointing to jdk directory.

3. Install gradle by executing the following command:

```
sudo apt-get install gradle
```

Setting up the environment in Windows

1. Download get-pip.py from <https://bootstrap.pypa.io/get-pip.py>
2. Execute the following command: `python get-pip.py`
3. Install Visual C++: <https://www.microsoft.com/en-us/download/details.aspx?id=44266>

4. Execute the following commands in the following order:

```
pip install setuptools --upgrade
pip install bitarray
pip install cmd2
pip install tapi
pip install xeger
pip install requests
```

5. Set the JAVA_HOME environment variable pointing to jdk directory.

Example: **C:\Program Files\Java\jdk1.8.0_91**

NOTE: Proper installation of gradle can be verified by using the command *gradle -version*.

- 6 . Gradle Installation in windows

Step 1. <https://gradle.org/releases/> get the latest Gradle distribution

Step 2. Unpack the distribution zip

Step 3. Configure your system environment **Path** variable

For e.x: **C:\Gradle\gradle-4.10.2\bin.**

Step 4. Verify your installation

Open a console (or a Windows command prompt) and run **gradle -v** to run gradle and verify the version, e.g.:

```
$ gradle -v
```

```
-----
Gradle 4.10.2
```

Setting up the repository for developing packages

In ATOM SDK, the **sdk.py** script sets up the SDK plugin environment for creating various packages. To setup the repository of your choice, follow the steps as outlined below:

```

root@User:/home/supritha/Desktop/AtomSDK/atom-package-plugin# python sdk.py -h
Usage: to setup the repo and create new packages
command to run:
python sdk.py [options]

Options:
  -h, --help, --h           displays help command options
  -c, --createpackage, --c   This helps you to create the different types of
                             package like SERVICE package,DEVICE package and DEVICE
                             DRIVER package etc: SHOULD RUN ONLY AFTER SETUP
                             COMMAND FOR THE FIRST TIME commands like python sdk.py
                             [-c] or [--c] or [--createpackage]
  -s, --setup, --s          This Script will help you setup repository for core-
                             dependent packages commands like python sdk.py [-s]
                             or [--s] or [--setup]

```

1. Run the command: `python sdk.py -s`

This command runs the **setup.py** script which setups an environment for packages repository.

setup.py - This script is used to setup repositories for core-dependent packages. The core-dependent packages are present inside the “packages” folder and are necessary for developing new device and service packages.

2. Select the repository of your choice.

You can either set up a local repository or can publish the core-dependent packages to an artifact repository such as Nexus.

```

root@User:/home/supritha/Desktop/AtomSDK/atom-package-plugin# python sdk.py -s
Running setup script
This script sets up repository for core-dependent packages

select the repository of your choice
1> Maven
2> Flat directory
enter your choice: 

```

- **Local Repository (Flat Directory Structure)** : This option enables you to copy the core-dependent packages present in the “packages” folder to a flat directory.
- The absolute path of this particular flat directory, for example, “/home/” as shown below. (verify that this folder is present already)

```

root@User:/home/supritha/Desktop/AtomSDK/atom-package-plugin# python sdk.py -s
Running setup script
This script sets up repository for core-dependent packages

select the repository of your choice
1> Maven
2> Flat directory
enter your choice: 2
enter the absolute directory path to copy the dependent packages (optional) : /h
ome/supritha/Desktop/AtomSDK/atom-package-plugin/dependencies
/home/supritha/Desktop/AtomSDK/atom-package-plugin/dependencies

```

- **Maven Artifact Repository** : This option enables the user to copy the core-dependent packages in the “packages” folder uploaded to the artifact repository, for example Nexus.

```

root@User:/home/supritha/Desktop/AtomSDK/atom-package-plugin# python sdk.py -s
Running setup script
This script sets up repository for core-dependent packages

select the repository of your choice
1> Maven
2> Flat directory
enter your choice: 1
Enter the maven repository URL: █

```

After setting up the repository, the script generates a **config.xml** file. This file contains two tags:

- a) **repo-type** : Maven or Flat Directory
- b) **repo-path** : The absolute path or URL of the directory.

The metadata present in the *config.xml* is important to run the subsequent scripts.

Let us take the example of the selected repository as the Flat Directory(a local repository) and the steps to be followed are illustrated below:

1. Enter the IP address of ATOM

```

root@User:/home/supritha/Desktop/AtomSDK/atom-package-plugin# python sdk.py -s
Running setup script
This script sets up repository for core-dependent packages

select the repository of your choice
1> Maven
2> Flat directory
enter your choice: 2
enter the absolute directory path to copy the dependent packages (optional) :

Proper directory path was not provided. Assuming packages directory as the default dependency directory

Enter the atom host ip of the atom instance to be used for developing packages.
atom instance ip = 127.0.0.1
Enter the username of the atom instance : admin
Enter the password of the atom instance : admin█

```

If port is required for accessing the ATOM application then mention that as well. E.g:
172.16.1.10:30443, 127.0.0.1:8890

2. Enter the credentials to login into ATOM

```

root@User:/home/supritha/Desktop/AtomSDK/atom-package-plugin# python sdk.py -s
Running setup script
This script sets up repository for core-dependent packages

select the repository of your choice
1> Maven
2> Flat directory
enter your choice: 2
enter the absolute directory path to copy the dependent packages (optional) :

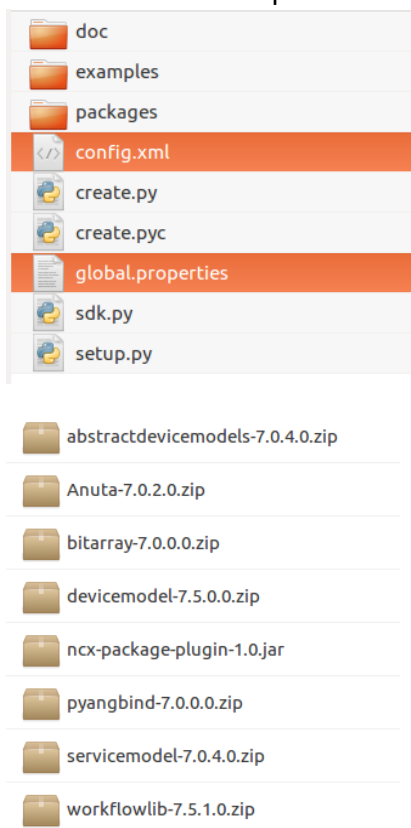
Proper directory path was not provided. Assuming packages directory as the default dependency directory

Enter the atom host ip of the atom instance to be used for developing packages.
atom instance ip = 127.0.0.1
Enter the username of the atom instance : admin
Enter the password of the atom instance : admin
root@User:/home/supritha/Desktop/AtomSDK/atom-package-plugin#

```

After the successful setup process, the following files and folders are generated :

- *global.properties* - contains the username, password and ATOM ip which will be used in the package development process.
- *config.xml* - contains the information of repo-type and path to dependencies.
- **dependencies** - The dependency packages for development of device and service models are copied to the destination folder of your choice.



IMPORTANT: Do not delete these files or folders.

Migration of Workflows

As seen in the above section workflows are deployed in atom by packaging them with the help of sdk. We can upgrade the package by changing the version in package.xml file. Atom automatically deploys the latest workflow version.

Key Points to Remember

- Only the latest workflow deployed can be started from Atom.
- Old running workflow instances continue to run on older versions.
- Atom maintains the history of all old workflow instances in workflow instances tab.

ATOM API Development and Testing Reference

Please refer to section ***Tools for API Development and Testing*** in **ATOM API Guide**

References

Entry	Description	Reference
YANG	YANG is a data modeling language used to model configuration data, state data, Remote Procedure Calls, and notifications for network management protocols.	https://tools.ietf.org/html/rfc7950
RESTCONF	An HTTP-based protocol that provides a programmatic interface for accessing data defined in YANG	https://tools.ietf.org/html/rfc8040
Gradle	Gradle helps teams build, automate and deliver better software, faster.	https://gradle.org/
BPMN	Business Process Model and Notation (BPMN) is the global standard for process modeling and one of the most important components of successful Business-IT-Alignment.	https://www.omg.org/spec/BPMN/2.0/