Managing Tasks

This chapter contains the following sections:

Tasks

Workflow Tasks provide the necessary artifacts to contribute to the Task library maintained by Cisco UCS Director. The task can be used in a Workflow definition.

At a minimum, a task should have the following classes:

- A class that implements the TaskConfigIf interface.
- A class that extends and implements methods in the AbstractTask class.

TaskConfigIf

A class that implements this interface becomes a Task's input. That is, a task that wants to accept inputs for its execution shall depend on a class that implements **TaskConfigIf**. The class that implements this interface should also contain all the input field definitions appropriately annotated for prompting the user. The class should also have JDO annotations to enable the Platform runtime to persist this object in the database.

A sample Config class is shown in the sample code.

AbstractTask

A task implementation must extend the **AbstractTask** abstract class and should provide implementation for all the abstract methods. This is the main class where all the business logic pertaining to the task goes. The most important method in this class, where the business logic implementation will be scripted, is **executeCustomAction( )**. The rest of the methods provide sufficient context to the Platform runtime to enable the task to appear in the Orchestration designer tree and to enable the task to be dragged and dropped in a Workflow.

Developing a TaskConfigIf

To develop a task, you must first implement **TaskConfigIf**. During the process of setting up the task configuration interface, you must determine what data is required to perform your task.

In the following example, **EnableSNMPConfig** exposes details of the process of developing a **TaskConfigIf**. The **Enable SNMP** task is designed to enable SNMP on a Cisco Nexus device.

To proceed, you must have the IP address of the Nexus device, the login, and the password.

You see the annotation at the beginning of **EnableSNMPConfig**.

```
@PersistenceCapable(detachable= "true", table = "foo_enable_snmp_config")
public class EnableSNMPConfig implements TaskConfigIf
{
```

You must provide a `PersistenceCapable` annotation with a table name that is prefixed with your module ID. You must follow this convention; because Cisco UCS Director prevents a task from being registered if you try to use a table name that is not prefixed with your module ID.

Next, see the following fields:

- **handler name**
- **configEntryId**
- **actionId**

```
public static finald String HANDLER_NAME = "Enable SNMP for Nexus";

//configEntryId and actionId are mandatory fields
@Persistent
private long   configEntryId
@Persistent
private long   actionId
```

The handler name is the name of the task. The name should be a unique string; you will create problems if you use the same handler name in multiple tasks.

Each task must have a **configEntryId** and **actionId**, exactly as shown above. You must have corresponding getter and setters for these two fields. These two fields are absolutely mandatory; you must have these fields in your config object.

Next, you see the data actually needed to perform the task:

```
//This is the ip address for the Nexus device on which you want to enable SNMP.
@FormField(label = "Host IP Address", help = "Host AP Address", mandatory = true,
     type = FormFieldDefinition.FIELD_TYPE_EMBEDDED_LOV,
     lovProvider = ModuleConstants.NEXUS_DEVICES_LOV_PROVIDER)
@USerInputField(type = ModuleConstants.NEXUS_DEVICE_LIST)
@Peristent
private String   ipAddress   = "";

@FormField(label = "Login", help = "Login", mandatory = true
```

```
@Persistent
private String   login;

@FormField(label = "Password", help = "Password", mandatory = true
@Persistent
private String   password;
```

As you review the code sample above, note that the developer needs the following:

- The IP address of the device.
  In this example, an LOV is used to get this IP address. See Annotations for more information about annotations and LOVs.
- The login and password, which the user must enter.
  To obtain these, use the form field annotations to mark these fields as data that will be provided by the user.
- Getters and setters for each of these fields.

Once the config object is completed, you must mark it for Java Data Object (JDO) enhancement.

Before You Begin

You must have the Cisco UCS Director Open Automation software development kit (SDK).

Step 1 Include a jdo.files file in the same package as your config objects.
See the jdo.files and packaging in the SDK example. Note that the jdo.files must be named exactly in this way.

Step 2 In the jdo.files, specify all the classes that need to go through JDO enhancement.
The build script supplied with the SDK will complete JDO enhancement for you if you have executed this step properly.

What to Do Next

The handler object is where you actually execute your custom code. A handler object must implement **AbstractTask**. The executeCustomAction method enables you to retrieve the corresponding config object that you developed previously to execute your code.

Developing an Abstract Task

When your config object is ready, you must extendAbstractTask to actually use the new config object. This example shows the EnableSNMPTask.

At this point, you should look at this method: executeCustomAction.

```
public void executeCustomAction(CustomActionTriggerContext context, CustomActionLogger
actionLogger) throws Exception
{
   long configEntryId = context.getConfigEntry().getConfigEntryID();
   //retrieving the corresponding config object for this handler
   EnableSNMPConfig config = (EnableSNMPConfig) context.loadConfigObject();
```

executeCustomAction is where the custom logic takes place. When you call context.loadConfigObject(), you can cast it to the config object that you defined earlier. This process allows you to retrieve all the details that you need to perform your task. This example shows that after getting the config object, the SSH APIs are used to execute the enable SNMP commands.

When a workflow is rolled back, a task must provide a way to undo the changes it has made. This example shows the use of a change tracker:

```
//If the user decides to roll back a workflow containing this task,
//then using the change tracker, we can take care of rolling back this task (i.e.,
//disabling snmp)
context.getChangeTracker().undoableResourceAdded("assetType", "idString",
SNMP enabled", "SNMP enabled on " + config.getIpAddress(),
new DisableSNMPNexusTask().getTaskName(), new DisableSNMPNexusConfig(config));
```

The rollback code informs the system that the undo task of Enable SNMP task is the Disable SNMP task. You provide the undo config object and its name. The rest of the arguments are about logging data, which you might or might not want to provide.

DisableConfig actually takes place in the **EnableConfig**. In this case, the enable config contains the device details, so when the Disable SNMP task is called, you know exactly which device to disable SNMP on.

You must also implement getTaskConfigImplementation. This example instantiates an instance of the config object in returning it:

```
@Override
public TaskConfigIf getTaskConfigImplementation() {
   return new EnbleSNMPConfig();
}
```

Note Make sure that you specify the config object that you intend to use with this task.

What to Do Next: Include this task in your module to make it ready for use in Cisco UCS Director.

About Schedule Tasks

If you need to develop a purge task or aggregation task, or some other kind of repeatable task, you can use the Schedule Task framework, which includes the following components:

- **AbstractScheduleTask**
- **AbstractCloupiaModule**

AbstractScheduleTask

Your task logic should be placed in the execute() method of this class. Provide your module ID and a string that describes this task to get started. You must provide your own module ID, or the module will not be registered properly.

For more information, refer the DummyScheduleTask class in the foo module.

```
public DummyScheduleTask(){
super("foo");
}
```

Adding/Removing Schedule Tasks

AbstractCloupiaModule has an add and remove schedule task API. Typically, in the onStart() implementation of your **AbstractCloupiaModule**, you would instantiate your tasks and register them with the add method by calling the addScheduleTask method in your module class as follows:

```
addScheduleTask(new DummyScheduleTask());
```

For more information, refer the FooModule.java class.

Registering Custom Workflow Inputs

You can develop your own input types in Cisco UCS Director. For more information, refer to *Cisco UCS Director Orchestration Guide, Release 4.1*. However, they must be prefixed with your module ID. See Developing a TaskConfigIf, on page 2, in which an additional annotation is used to specify a custom workflow input.

```
public static final String NEXUS_DEVICE_LIST = "foo_nexus_device_list";
@UserInputField(type = ModuleConstants.NEXUS_DEVICE_LIST)
```

In this example, ModuleConstants.NEXUS_DEVICE_LIST resolves to `foo_nexus_device_list`.

Before You Begin

Develop the required TaskConfigIf and the AbstractTask components for your custom workflow.

What to Do Next

Register a custom workflow output. See Registering Custom Task Output, on page 5.

Registering Custom Task Output

You can enable a task to add an output.

Before You Begin

See the EmailDatacentersTask to see an example of how to create custom task outputs.

SUMMARY STEPS

1. Implement the method getTaskoutputDefinitions() in the task implementation and return the output definitions that the task is supposed to return.
2. Set the output from the task implementation.

DETAILED STEPS

Step 1 Implement the method getTaskoutputDefinitions() in the task implementation and return the output definitions that the task is supposed to return.

```
@Override
public TaskOutputDefinition[] getTaskOutputDefinitions() {
   TaskOutputDefinition[] ops = new TaskOututDefiniton[1];
   ops[0] = FooModule.OP_TEMP_EMAIL_ADDRESS;
   return ops;
]
```

Step 2 Set the output from the task implementation.

```
@Override
public void executeCustomAction(CustomActionTriggerContext context,
CustomerActionLogger action Logger) throws Exception
{

long configEntryId = context.getConfigEntry().getConfigEntryId();
//retrieving the corresponding config object for this handler
EmailDatacentersConfig config = (EmailDatacentersConfig context.loadConfigObject();

if (config == null)
```

```
{
    throw net Exception("No email configuration found for custom Action"
        + context.getAction().getName
        + "entryId" + configEntryId);
}

|.......
 .......

try
{
    context.saveOutputValue(OutPutConstants.OUTPUT_TEMP_EMAIL_ADDRESS, toAddresses);
```

Consuming Custom Output as Input in Other Tasks

This section describes how output can be used as input in another task. This section uses some aspects of the example in the previous section. The output definition is defined as follows:

```
@Override
public TaskOutputDefinition[] get TaskOutputDefinitions() {
    TaskOutputDefinition[] ops = new TaskOutputDefinitions[1};
//NOTE: If you want to use the output of this task as input to another task. Then the second
 argument
//of the output definition MUST MATCH the type of UserInputField in the config of the task
 that will
//be recieving this output. Take a look at the HelloWorldConfig as an example.
ops[0] = new TaskOutputDefinition(
      FooConstants.EMAIL_TASK_OUTPUT_NAME,
      FooConstants.FOO_HELLO_WORLD_NAME,
      "EMAIL IDs");
return ops;
}
,
```

The example defines an output with the FooConstants.EMAIL_TASK_OUTPUT_NAME name, and with the FooConstants.FOO_HELLO_WORLD_NAME type. To configure another task that can consume the output as input, you must make the types match.

So, in the new task that consumes FooConstants.FOO_HELLO_WORLD_NAME as input, you must enter the following in the configuration object:

```
//This field is supposed to consume output from the EmailDatacentersTask.
//You'll see the type in user input field below matches the output type
//in EmailDatacentersTasks's output definition.
@FormField(label = "name", help = "Name passed in from a previous task", mandatory = true)
@UserInputField(type = FooConstants.FOO_HELLO_WORLD_NAME)
@Persistent
private String     login;
```

The type in the `UserInputField` annotation matches the type that is registered in the output definition. With that match in place, when you drag and drop the new task in the Cisco UCS Director **Workflow Designer**, you can map the output from one task as input to the other task while you are developing the workflow.

Consuming Output from Existing Tasks as Input

This section shows how to consume output from built-in workflow tasks as input to your custom task. This process is similar to setting up custom outputs to be consumed as input in one important way: the configuration object of your task must have a field whose type is exactly the same as the type of the output that you want.

SUMMARY STEPS

1. Choose **Policies > Orchestration > Workflows**, and then click **Task Library**.
2. Find the task that you want to add, and then choose it to see the information displayed under the heading: **User and Group Tasks: Add Group**.
3. Pick the appropriate Type value from the Outputs table.
4. Specify the Type value in the UserInputField.
5. Configure the mapping as you develop your workflow, using the **User Input Mapping to Task Input Attributes** window as you add an action to the workflow, or edit related information in the workflow.

DETAILED STEPS

Step 1 Choose **Policies > Orchestration > Workflows**, and then click **Task Library**.

Tip Press **Cntl–Find** to locate tasks in the very long list that appears. For example, entering Group takes you directly to **User and Group Tasks**.

Step 2 Find the task that you want to add, and then choose it to see the information displayed under the heading: **User and Group Tasks: Add Group**.

Tip Press**Cntl–Find** to locate tasks in the very long list that appears. For example, entering Group takes you directly to **User and Group Tasks**.

The crucial type data is provided in the Outputs table, the last table provided under the heading.

Table 1: Add Group - Outputs Table

Output	Description	Type
OUTPUT_GROUP_NAM	Name of the group that was created by admin	gen_text_input
OUTPUT_GROUP_ID	ID of the group that was created by admin	gen_text_input

Step 3 Pick the appropriate Type value from the Outputs table.
The goal is to obtain the Type value that will be matched to the Task. In the example, the task consumes the group ID, so you know that the Type is *gen_text_input*.

Step 4 Specify the Type value in the UserInputField.

Example:

```
@FormField(label = "Name", help = "Name passed in from previous task",
mandatory = true)
@UserInputField(type="gen_text_input")
@Persistent
private String       name;
```

Note You could also use *@UserInputField(type = WorkflowInputFieldTypeDeclaration.GENERIC_TEXT)*. This is equivalent to using *@UserInputField(type="gen_text_input")*. You may find it easier to use *type = WorkflowInputFieldTypeDeclaration.GENERIC_TEXT* which uses the constants defined in the SDK.

The last step is to configure the mapping properly when you are developing your workflow.

Step 5 Configure the mapping as you develop your workflow, using the **User Input Mapping to Task Input Attributes** window as you add an action to the workflow, or edit related information in the workflow.

Verifying the Custom Task Is In Place

Assuming that your module is working properly, you can verify that the custom task is in place by opening the Cisco UCS Director Task Library and verifying that the task appears in it.

SUMMARY STEPS

1. In Cisco UCS Director, choose **Policies** > **Orchestration**, and then choose the **Workflows** tab.
2. In the **Workflows** tree directory, navigate to a workflow in which the task appears, and and then choose that workflow row.
3. With workflow selected, click **Workflow Designer**.
4. Verify that the task of interest appears in the list of available tasks and in the graphic representation of the tasks in the workflow.

DETAILED STEPS

Step 1 In Cisco UCS Director, choose **Policies** > **Orchestration**, and then choose the **Workflows** tab.
The **Workflows** tab displays a table that lists all available workflows.

Step 2 In the **Workflows** tree directory, navigate to a workflow in which the task appears, and and then choose that workflow row.
To facilitate navigation, use the Search option in the upper right-hand corner, above the table, to navigate to the workflow.

Additional workflow-related controls appear above the workflows table.

Step 3 With workflow selected, click **Workflow Designer**.
The **Workflow Designer** screen opens, displaying an **Available Tasks** list and the Workflow Design graphic view.

Step 4 Verify that the task of interest appears in the list of available tasks and in the graphic representation of the tasks in the workflow.