Developing with VMware vCenter Orchestrator

vCenter Orchestrator 5.5.1

This document supports the version of each product listed and supports all subsequent versions until the document is replaced by a new edition. To check for more recent editions of this document, see http://www.vmware.com/support/pubs.

EN-001341-01

vmware®

You can find the most up-to-date technical documentation on the VMware Web site at:

http://www.vmware.com/support/

The VMware Web site also provides the latest product updates.

If you have comments about this documentation, submit your feedback to:

docfeedback@vmware.com

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Contents

Developing with VMware vCenter Orchestrator

Developing with VMware vCenter Orchestrator provides information and instructions for developing custom VMware® vCenter Orchestrator workflows and actions.

In addition, the documentation contains information about the Orchestrator elements that require scripting and provides JavaScript examples. *Developing with VMware vCenter Orchestrator* also provides instructions about how to create resources and packages.

Intended Audience

This information is intended for developers who want to create custom Orchestrator workflows and actions, as well as custom building blocks.

Updated Information

Developing with VMware vCenter Orchestrator is updated with each release of the product or when necessary.

This table provides the update history of *Developing with VMware vCenter Orchestrator*.

Revision	Description
EN-001341-01	Added "Maven Archetypes," on page 158 and "Plug-In Development Best Practices," on page 158.
EN-001341-00	Initial release.

Developing Workflows 1

You develop workflows in the Orchestrator client interface. Workflow development involves using the workflow editor, the built-in Mozilla Rhino JavaScript scripting engine, and the Orchestrator and vCenter Server APIs.

- Key Concepts of Workflows on page 13

  Workflows consist of a schema, attributes, and parameters. The workflow schema is the main component of a workflow as it defines all the workflow elements and the logical connections between them. The workflow attributes and parameters are the variables that workflows use to transfer data. Orchestrator saves a workflow token every time a workflow runs, recording the details of that specific run of the workflow.

- Phases in the Workflow Development Process on page 14

  The process for developing a workflow involves a series of phases. You can follow a different sequence of phases or skip a phase, depending on the type of workflow that you are developing. For example, you can create a workflow without custom scripting.

- Access Rights for the Orchestrator Client on page 14

  By default, only members of the Orchestrator administrator LDAP group can access the Orchestrator client.

- Testing Workflows During Development on page 15

  You can test workflows at any point during the development process, even if you have not completed the workflow or included an end element.

- Creating and Editing a Workflow on page 15

  You create workflows in the Orchestrator client and edit them in the workflow editor. The workflow editor is the IDE of the Orchestrator client for developing workflows.

- Provide General Workflow Information on page 18

  You provide a workflow name and desription, define attributes and certain aspects of workflow behavior, set the version number, check the signature, and set user permissions in the **General** tab in the workflow editor.

- Defining Attributes and Parameters on page 18

  After you create a workflow, you must define the global attributes, input parameters, and output parameters of the workflow.

- Workflow Schema on page 21

  A workflow schema is a graphical representation of a workflow that shows the workflow as a flow diagram of interconnected workflow elements. The workflow schema defines the logical flow of a workflow.

- Restore Deleted Workflows on page 81

  You can restore workflows that have been deleted from the workflow library.

- Develop a Simple Example Workflow on page 81

  Developing a simple example workflow demonstrates the most common steps in the workflow development process.

- Develop a Complex Workflow on page 104

  Developing a complex example workflow demonstrates the most common steps in the workflow development process and more advanced scenarios, such as creating custom decisions and loops.

Key Concepts of Workflows

Workflows consist of a schema, attributes, and parameters. The workflow schema is the main component of a workflow as it defines all the workflow elements and the logical connections between them. The workflow attributes and parameters are the variables that workflows use to transfer data. Orchestrator saves a workflow token every time a workflow runs, recording the details of that specific run of the workflow.

Workflow Parameters

Workflows receive input parameters and generate output parameters when they run.

Input Parameters

Most workflows require a certain set of input parameters to run. An input parameter is an argument that the workflow processes when it starts. The user, an application, another workflow, or an action passes input parameters to a workflow for the workflow to process when it starts.

For example, if a workflow resets a virtual machine, the workflow requires as an input parameter the name of the virtual machine.

Output Parameters

A workflow's output parameters represent the result from the workflow run. Output parameters can change when a workflow or a workflow element runs. While workflows run, they can receive the output parameters of other workflows as input parameters.

For example, if a workflow creates a snapshot of a virtual machine, the output parameter for the workflow is the resulting snapshot.

Workflow Attributes

Workflow elements process data that they receive as input parameters, and set the resulting data as workflow attributes or output parameters.

Read-only workflow attributes act as global constants for a workflow. Writable attributes act as a workflow’s global variables.

You can use attributes to transfer data between the elements of a workflow. You can obtain attributes in the following ways:

- Define attributes when you create a workflow
- Set the output parameter of a workflow element as a workflow attribute
- Inherit attributes from a configuration element

Workflow Schema

A workflow schema is a graphical representation that shows the workflow as a flow diagram of interconnected workflow elements. The workflow schema is the most important element of a workflow as it determines its logic.

Workflow Presentation

When users run a workflow, they provide the values for the input parameters of the workflow in the workflow presentation. When you organize the workflow presentation, consider the type and number of input parameters of the workflow.

Workflow Tokens

A workflow token represents a workflow that is running or has run.

A workflow is an abstract description of a process that defines a generic sequence of steps and a generic set of required input parameters. When you run a workflow with a set of real input parameters, you receive an instance of this abstract workflow that behaves according to the specific input parameters you give it. This specific instance of a completed or a running workflow is called a workflow token.

Workflow Token Attributes

Workflow token attributes are the specific parameters with which a workflow token runs. The workflow token attributes are an aggregation of the workflow's global attributes and the specific input and output parameters with which you run the workflow token.

Phases in the Workflow Development Process

The process for developing a workflow involves a series of phases. You can follow a different sequence of phases or skip a phase, depending on the type of workflow that you are developing. For example, you can create a workflow without custom scripting.

Generally, you develop a workflow through the following phases.

1 Create a new workflow or create a duplicate of an existing workflow from the standard library.

2 Provide general information about the workflow.

3 Define the input parameters of the workflow.

4 Lay out and link the workflow schema to define the logical flow of the workflow.

5 Bind the input and output parameters of each schema element to workflow attributes.

6 Write the necessary scripts for scriptable task elements or custom decision elements.

7 Create the workflow presentation to define the layout of the input parameters dialog box that the users see when they run the workflow.

8 Validate the workflow.

Access Rights for the Orchestrator Client

By default, only members of the Orchestrator administrator LDAP group can access the Orchestrator client.

The Orchestrator administrator can grant access to the Orchestrator client to other user groups by setting at least the **View** permission.

To allow you to access the Orchestrator client, the administrator must either add you to the Orchestrator administrator LDAP group, or set **View**, **Inspect**, **Edit**, **Execute**, or **Admin** permissions to a group that you are a member of.

Testing Workflows During Development

You can test workflows at any point during the development process, even if you have not completed the workflow or included an end element.

By default, Orchestrator checks that a workflow is valid before you can run it. You can deactivate automatic validation during workflow development, to run partial workflows for testing purposes.

NOTE Do not forget to reactivate automatic validation when you finish developing the workflow.

Procedure

1 In the Orchestrator client menu, click **Tools > User preferences**.

2 Click the **Workflows** tab.

3 Deselect the **Validate workflow before running it** check box.

You deactivated automatic workflow validation.

Creating and Editing a Workflow

You create workflows in the Orchestrator client and edit them in the workflow editor. The workflow editor is the IDE of the Orchestrator client for developing workflows.

You open the workflow editor by editing an existing workflow.

- Create a Workflow on page 15

  You can create workflows in the workflows hierarchical list of the Orchestrator client.

- Edit a Workflow on page 16

  You edit a workflow to make changes to an existing workflow or to develop a new empty workflow.

- Edit a Workflow from the Standard Library on page 16

  Orchestrator provides a standard library of workflows that you can use to automate operations in the virtual infrastructure. The workflows in the standard library are locked in the read-only state.

- Workflow Editor Tabs on page 17

  The workflow editor consists of tabs on which you edit the components of the workflows.

Create a Workflow

You can create workflows in the workflows hierarchical list of the Orchestrator client.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Design**.

2 Click the **Workflows** view.

3 (Optional) Right-click the root of the workflows hierarchical list, or a folder in the list, and select **Add folder** to create a new workflow folder.

4 (Optional) Type the name of the new folder.

5 Right-click the new folder or an existing folder and select **New workflow**.

6 Name the new workflow and click **OK**.

A new empty workflow is created in the folder that you chose.

What to do next

You can edit the workflow.

Edit a Workflow

You edit a workflow to make changes to an existing workflow or to develop a new empty workflow.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Design**.

2 Click the **Workflows** view.

3 Expand the workflows hierarchical list to navigate to the workflow that you want to edit.

4 To open the workflow for editing, right-click the workflow and select **Edit**.

The workflow editor opens the workflow for editing.

Edit a Workflow from the Standard Library

Orchestrator provides a standard library of workflows that you can use to automate operations in the virtual infrastructure. The workflows in the standard library are locked in the read-only state.

To edit a workflow from the standard library, you must create a duplicate of that workflow. You can edit duplicate workflows or custom workflows.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Design**.

2 Click the **Workflows** view.

3 (Optional) Right-click the root of the hierarchical list of workflow folders and select **New folder** to create a folder to contain the workflow to edit.

4 Expand the **Library** hierarchical list of standard workflows to navigate to the workflow to edit.

5 Right-click the workflow to edit.

The **Edit** option is dimmed. The workflow is read-only.

6 Right-click the workflow and select **Duplicate workflow**.

7 Provide a name for the duplicate workflow.

By default, Orchestrator names the duplicate workflow `Copy of` *`workflow_name`*.

8 Click the **Workflow folder** value to search for a folder in which to save the duplicate workflow.

Select the folder you created in Step 3. If you did not create a folder, select a folder that is not in the library of standard workflows.

9 Click **Yes** or **No** to copy the workflow version history to the duplicate.

Option	Description
Yes	The version history of the original workflow is replicated in the duplicate.
No	The version of the duplicate reverts to 0.0.0.

10 Click **Duplicate** to duplicate the workflow.

11 Right-click the duplicate workflow and select **Edit**.

The workflow editor opens. You can edit the duplicate workflow.

You duplicated a workflow from the standard library. You can edit the duplicate workflow.

Workflow Editor Tabs

The workflow editor consists of tabs on which you edit the components of the workflows.

Table 1-1. Workflow Editor Tabs

Tab	Description
General	Edit the workflow name, provide a description of what the workflow does, set the version number, see the user permissions, define the behavior of the workflow if the Orchestrator server restarts, and define the workflow's global attributes.
Inputs	Define the parameters that the workflow requires when it runs. These input parameters are the data that the workflow processes. The workflow's behavior changes according to these parameters.
Outputs	Define the values that the workflow generates when it completes its run. Other workflows or actions can use these values when they run.
Schema	Build the workflow. You build the workflow by dragging workflow schema elements from the workflow palette on the left side of the **Schema** tab. Clicking an element in the schema diagram allows you to define and edit the element's behavior in the bottom half of the **Schema** tab.
Presentation	Define the layout of the user input dialog box that appears when users run a workflow. You arrange the parameters and attributes into presentation steps and groups to ease identification of parameters in the input parameters dialog box. You define the constraints on the input parameters that users can provide in the presentation by setting the parameter properties.
Parameters References	View which workflow elements consume the attributes and parameters in the logical flow of the workflow. This tab also shows the constraints on these parameters and attributes that you define in the **Presentation** tab.
Workflow Tokens	View details about each workflow run. This information includes the workflow's status, the user who ran it, the business status of the current element, and the time and date when the workflow started and ended.
Events	View information about each individual event that occurs when the workflow runs. This information includes a description of the event, the user who triggered it, the type and origin of the event, and the time and date when it occurred.
Permissions	Set the permissions to interact with the workflow for users or groups of users.

Provide General Workflow Information

You provide a workflow name and desription, define attributes and certain aspects of workflow behavior, set the version number, check the signature, and set user permissions in the **General** tab in the workflow editor.

Prerequisites

Open a workflow for editing in the workflow editor.

Procedure

1 Click the **General** tab in the workflow editor.

2 Click the **Version** digits to set a version number for the workflow.

The **Version Comment** dialog box opens.

3 Type a comment for this version of the workflow and click **OK**.

For example, type `Initial creation` if you just created the workflow.

A new version of the workflow is created. You can later revert the state of the workflow to this version.

4 Define how the workflow behaves if the Orchestrator server restarts by setting the **Server restart behavior** value.

- Leave the default value of **Resume workflow run** to make the workflow resume at the point at which its run was interrupted when the server stopped.
- Click **Resume workflow run** and select **Do not resume workflow run (set as FAILED)** to prevent the workflow from restarting if the Orchestrator server restarts.

Prevent the workflow from restarting if the workflow depends on the environment in which it runs. For example, if a workflow requires a specific vCenter Server and you reconfigure Orchestrator to connect to a different vCenter Server, restarting the workflow after you restart the Orchestrator server causes the workflow to fail.

5 Type a detailed description of the workflow in the **Description** text box.

6 Click **Save** at the bottom of the workflow editor.

A green message at the bottom left of the workflow editor confirms that you saved your changes.

You defined aspects of the workflow behavior, set the version, and defined the operations that users can perform on the workflow.

What to do next

You must define the workflow attributes and parameters.

Defining Attributes and Parameters

After you create a workflow, you must define the global attributes, input parameters, and output parameters of the workflow.

Workflow attributes store data that workflows process internally. Workflow input parameters are data provided by an outside source, such as a user or another workflow. Workflow output parameters are data that the workflow delivers when it finishes its run.

- Define Workflow Parameters on page 19

  You can use input and output parameters to pass data into and out of the workflow.

- Define Workflow Attributes on page 19

  Workflow attributes are the data that workflows process.

- Attribute and Parameter Naming Restrictions on page 20

  You can use OGNL expressions to determine input parameters dynamically when a workflow runs. The Orchestrator OGNL parser uses certain keywords during OGNL processing that you cannot use in workflow attribute or parameter names.

Define Workflow Parameters

You can use input and output parameters to pass data into and out of the workflow.

You can define the parameters of a workflow in the workflow editor. The input parameters are the initial data that the workflow requires to run. Users provide the values for the input parameters when they run the workflow. The output parameters are the data the workflow returns when it completes its run.

Prerequisites

Open a workflow for editing in the workflow editor.

Procedure

1 Click the appropriate tab in the workflow editor.

- Click **Inputs** to create input parameters.
- Click **Outputs** to create output parameters.

2 Right-click inside the parameters tab and select **Add parameter**.

3 Click the parameter name to change it.

The default name is `arg_in_`*X* for input parameters and `arg_out_`*X* for output parameters, where *X* is a number.

4 (Optional) To change the value of the parameter type, click the value and select one from the list of available values.

The value for the parameter type is String by default.

5 Add a description for the parameter in the **Description** text box.

6 (Optional) If you decide that the parameter should be an attribute rather than a parameter, right-click the parameter and select **Move as attribute** to change the parameter into an attribute.

You have defined an input or output parameter for the workflow.

What to do next

After you define the workflow's parameters, build the workflow schema.

Define Workflow Attributes

Workflow attributes are the data that workflows process.

NOTE You can also define workflow attributes in the workflow schema elements when you create the workflow schema. It is often easier to define an attribute when you create the workflow schema element that processes it.

Prerequisites

Open a workflow for editing in the workflow editor.

Procedure

1 Click the **General** tab in the workflow editor.

The attributes pane appears in the bottom half of the **General** tab.

2 Right-click in the attributes pane and select **Add Attribute**.

A new attribute appears in the attributes list, with String as its default type.

3 Click the attribute name to change it.

The default name is `attX`, where *X* is a number.

NOTE Workflow attributes must not have the same name as any of the workflow's parameters.

4 Click the attribute type to select a new type from a list of possible values.

The default attribute type is String.

5 Click the attribute value to set or select a value according to the attribute type.

6 Add a description of the attribute in the **Description** text box.

7 If the attribute is a constant rather than a variable, click the check box to the left of the attribute name to make its value read-only.

The lock icon identifies the column of read-only check boxes.

8 (Optional) If you decide that the attribute should be an input or output parameter rather than an attribute, right-click the attribute and select **Move as INPUT/OUTPUT parameter** to change the attribute into a parameter.

You defined an attribute for the workflow.

What to do next

You can define the workflow's input and output parameters.

Attribute and Parameter Naming Restrictions

You can use OGNL expressions to determine input parameters dynamically when a workflow runs. The Orchestrator OGNL parser uses certain keywords during OGNL processing that you cannot use in workflow attribute or parameter names.

Using a reserved OGNL keyword as a prefix to an attribute name does not break OGNL processing. For example, you can name a parameter `trueParameter`. Reserved keywords are not case-sensitive.

IMPORTANT The use of OGNL expressions in workflow presentations is deprecated as of Orchestrator 4.1. Using OGNL expressions in workflow presentations is not supported in releases of Orchestrator later than 4.1.

You cannot use the following keywords in workflow attribute and parameter names.

Table 1-2. Forbidden Keywords in Attribute and Parameter Names

Forbidden Keyword	Forbidden Keyword	Forbidden Keyword
abstract	eof	_memberAccess
back_char_esc	esc	native
back_char_literal	exponent	package
boolean	export	private
byte	extends	public
char	false	root
char_literal	final	short
class	flt_literal	static
_classResolver	flt_suff	string_esc
const	ident	string_literal
context	implements	synchronized
debugger	import	this
dec_digits	in	_traceEvaluations
dec_flt	int	true
default	int_literal	_typeConverter
delete	interface	volatil
digit	_keepLastEvaluation	with
double	_lastEvaluation	WithinBackCharLiteral
dynamic_subscript	letter	WithinCharLiteral
enum	long	WithinStringLiteral

Workflow Schema

A workflow schema is a graphical representation of a workflow that shows the workflow as a flow diagram of interconnected workflow elements. The workflow schema defines the logical flow of a workflow.

- View Workflow Schema on page 22

  You view the schema of a workflow in the **Schema** tab for that workflow in the Orchestrator client.

- Building a Workflow in the Workflow Schema on page 22

  Workflow schemas consist of a sequence of schema elements. Workflow schema elements are the building blocks of the workflow, and can represent decisions, scripted tasks, actions, exception handlers, or even other workflows.

- Schema Elements on page 25

  The workflow editor presents the workflow schema elements in menus in the **Schema** tab.

- Schema Element Properties on page 28

  Schema elements have properties that you can define and edit in the **Schema** tab of the workflow palette.

- Links and Bindings on page 31

  Links between elements determine the logical flow of the workflow. Bindings populate elements with data from other elements by binding input and output parameters to workflow attributes.

- Decisions on page 36

  Workflows can implement decision functions that define different courses of action according to a Boolean `true` or `false` statement.

- Exception Handling on page 39

  Exception handling catches any errors that occur when a schema element runs. Exception handling defines how the schema element behaves when the error occurs.

- Foreach Elements and Composite Types on page 40

  You can insert a Foreach element in the workflow that you develop to run a subworkflow that iterates over arrays of parameters or attributes. To improve the understanding and readability of the workflow, you can group several workflow parameters of different types that are logically connected in a single type that is called a composite type.

View Workflow Schema

You view the schema of a workflow in the **Schema** tab for that workflow in the Orchestrator client.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Design**.

2 Navigate to a workflow in the workflow hierarchical list.

3 Click the workflow.

  Information about that workflow appears in the right pane.

4 Select the **Schema** tab in the right pane.

You see the graphical representation of the workflow.

Building a Workflow in the Workflow Schema

Workflow schemas consist of a sequence of schema elements. Workflow schema elements are the building blocks of the workflow, and can represent decisions, scripted tasks, actions, exception handlers, or even other workflows.

You build workflows in the workflow editor by dragging schema elements from the workflow palette on the left of the workflow editor into the workflow schema diagram.

Edit a Workflow Schema

You build a workflow by creating a sequence of schema elements that define the logical flow of the workflow.

By default, all elements in the workflow schema are linked. Links between the elements are represented as arrows. When you add a new element to the workflow schema, you must drag it onto an arrow or an existing workflow element that is not linked to a next element. After you add workflow elements to the schema, you can delete existing links and create new links to define the logical flow of the workflow.

You can copy an element or a selection of elements from the schema of an existing workflow to the schema of the workflow that you are editing. See "Copy Workflow Schema Elements," on page 23.

A workflow schema must have at least one **End workflow** element, but it can have several.

Prerequisites

Open a workflow for editing in the workflow editor.

Procedure

1 Click the **Schema** tab in the workflow editor.

2 Drag a schema element from the **Generic** menu in the left pane, to the workflow schema.

3 Double-click the element you dragged to the workflow schema, type an appropriate name, and press Enter.

  You must provide elements with unique names in the context of the workflow.

  You cannot rename **Waiting timer**, **Waiting event**, **End workflow**, or **Throw exception** elements.

4 (Optional) Right-click an element in the schema and select **Copy**.

5 (Optional) Right-click at an appropriate position in the schema and select **Paste**.

Copying and pasting existing schema elements is a quick way of adding similar elements to the schema. All of the settings of the copied element appear in the pasted element, except for the business state. Adjust the pasted element settings accordingly.

6 Drag schema elements from the **Basic**, **Log**, or **Network** menus to the workflow schema.

You can edit the names of the elements in the **Basic**, **Log**, or **Network** menus. You cannot edit their scripting.

7 Drag schema elements from the **Generic** menu to the workflow schema.

When you drag actions or workflows to the workflow schema, a dialog box in which you can search for the action or workflow to insert appears.

8 In the **Filter** text box, type the name or part of the name of the workflow or action to insert in the workflow.

The workflows or actions that match the search appear in the dialog box.

9 Double-click a workflow or action to select it.

You inserted the workflow or action in the workflow schema.

10 Repeat this procedure until you have added all of the required schema elements to the workflow schema.

What to do next

Define the properties of the elements you added to the workflow schema and link and bind them all together.

Copy Workflow Schema Elements

You can copy an element or a selection of elements from the schema of an existing workflow to the schema of the workflow that you are editing.

Prerequisites

Open a workflow for editing in the workflow editor.

Procedure

1 Click the **Schema** tab in the workflow editor.

2 From the left pane, select the workflow from which you want to copy schema elements.

- Click **All Workflows** and select the workflow from the hierarchical list of workflows.
- Type the name of the workflow in the search text box and press Enter.

3 Right-click the selected workflow and select **Open**.

A window displaying the workflow's properties appears.

4 In the workflow's window, click the **Schema** tab.

5 Select one or more workflow schema elements, right-click the selection, and select **Copy**.

6 In the **Schema** tab of the workflow that you are editing, right-click and select **Paste**.

You copied workflow schema elements from one workflow to another.

What to do next

You must link and bind the copied schema elements to the existing workflow schema.

Promote Input and Output Parameters

You can promote the input and output parameters of a child element to the parent workflow.

You can promote a custom attribute that you have defined on the **General** tab of the workflow editor. You can promote predefined attributes only by replacing an input parameter with an attribute of matching type.

NOTE If you promote a predefined attribute and assign a custom value to it, a duplicate attribute is created to avoid overwriting the value of the original attribute. The duplicate attribute retains the name of the original attribute and increments the numerical value at the end of the attribute's name.

Prerequisites

Open a workflow for editing in the workflow editor.

Procedure

1 Click the **Schema** tab in the workflow editor.

2 Add a workflow or an action element to the workflow schema.

The following notification appears at the top of the schema pane.

```
Do you want to add the activity's parameters as input/output to the current workflow?
```

3 On the notification, click **Setup**.

A pop-up window with the available options appears.

4 Select the mapping type for each input parameter.

Option	Description
Input	The argument is mapped to an input workflow parameter.
Skip	The argument is mapped to a `NULL` value.
Value	The argument is mapped to an attribute with a value that you can set from the Value column.

5 Select the mapping type for each output parameter.

Option	Description
Output	The argument is mapped to an output workflow parameter.
Skip	The argument is mapped to a `NULL` value.
Local variable	The argument is mapped to an attribute.

6 Click **Promote**.

You promoted parameters to the parent workflow.

Modify Search Results

You use the **Search** text box to find elements such as workflows or actions. If a search returns a partial result, you can modify the number of results that the search returns.

When you use the search for an element, a green message box indicates that the search lists all the results. A yellow message box indicates that the search lists only partial results.

Procedure

1 (Optional) If you are editing a workflow in the workflow editor, click **Save and Close** to exit the editor.

2 From the Orchestrator client menu, select **Tools > User preferences**.

3 Click the **General** tab.

4 Type the number of results for searches to return in the **Finder Maximum Size** text box.

5 Click **Save and Close** in the User Preferences dialog box.

You modified the number of results that searches return.

Schema Elements

The workflow editor presents the workflow schema elements in menus in the **Schema** tab.

You can use the schema elements available in the **Schema** tab to build a workflow.

Table 1-3. Schema Elements and Icons

Schema Element Name	Description	Icon	Location in Workflow Editor
Start Workflow	The starting point of the workflow. All workflows contain this element and it cannot be removed from the workflow schema. A workflow can have only one start element. Start elements have one output and no input.		Always present in the **Schema** tab
Scriptable task	General purpose tasks you define. You write JavaScript functions in this element.		**Generic** workflow palette
Decision	Boolean function. Decision elements take one input parameter and return either `true` or `false`. The type of decision that the element makes, depends on the type of the input parameter. Decision elements allow the workflow to branch into different directions, depending on the input parameter the decision element receives. If the received input parameter corresponds to an expected value, the workflow continues along a certain route. If the input is not the expected value, the workflow continues on an alternative path.		**Generic** workflow palette
Custom decision	Boolean function. Custom decisions can take several input parameters and process them according to custom scripts. Returns either `true` or `false`.		**Generic** workflow palette
Decision activity	Boolean function. A decision activity runs a workflow and binds its output parameters to a `true` or a `false` path.		**Generic** workflow palette

Table 1-3. Schema Elements and Icons (Continued)

Schema Element Name	Description	Icon	Location in Workflow Editor
User interaction	Allows users to pass new input parameters into the workflow. You can design how the user interaction element presents the request for input parameters and place constraints on the parameters that users can provide. You can set permissions to determine which users can provide the input parameters. When a running workflow arrives at a user interaction element, it enters a passive state and prompts the user for input. You can set a timeout period within which the users can answer. The workflow resumes according to the data the user passes to it, or returns an exception if the timeout period expires. While it is waiting for the user to respond, the workflow token is in the `waiting` state.		**Generic** workflow palette
Waiting timer	Used by long-running workflows. When a running workflow arrives at a Waiting Timer element it enters a passive state. You set an absolute date at which the workflow resumes running. While it is waiting for the date, the workflow token is in the `waiting-signal` state.		**Generic** workflow palette
Waiting event	Used in long-running workflows. When a running workflow arrives at a Waiting Event element it enters a passive state. You define a trigger event that the workflow awaits before it resumes running. While it is waiting for the event, the workflow token is in the `waiting-signal` state.		**Generic** workflow palette
End workflow	The end point of the workflow. You can have multiple end elements in a schema, to represent the different possible outcomes of the workflow. End elements have one input with no output. When a workflow reaches an End Workflow element, the workflow token enters the `completed` state.		**Generic** workflow palette
Thrown exception	Creates an exception and stops the workflow. Multiple occurrences of this element can be present in the workflow schema. Exception elements have one input parameter, which can only be of the String type, and have no output parameter. When a workflow reaches an Exception element, the workflow token enters the `failed` state.		**Generic** workflow palette
Workflow note	Allows you to annotate sections of the workflow. You can stretch notes to delineate sections of the workflow. You can change the background color of the notes to differentiate between different workflow zones. Workflow notes provide visual information only, to help you understand the schema.		**Generic** workflow palette

Table 1-3. Schema Elements and Icons (Continued)

Schema Element Name	Description	Icon	Location in Workflow Editor
Action element	Calls on an action from the Orchestrator libraries of actions. When a workflow reaches an action element, it calls and runs that action.		**Generic** workflow palette
Workflow element	Starts another workflow synchronously. As soon as a workflow reaches a workflow element in its schema, it runs that workflow as part of its own process. The original workflow does not continue until the called workflow completes its run.		**Generic** workflow palette
Foreach element	Runs a workflow on every element from an array. For example, you can run the Raname Virtual Machine workflow on all virtual machines from a folder.		**Generic** workflow palette
Asynchronous workflow	Starts a workflow asynchronously. When a workflow reaches an asynchronous workflow element, it starts that workflow and continues its own run. The original workflow does not wait for the called workflow to finish before continuing.		**Generic** workflow palette
Schedule workflow	Creates a task to run the workflow at a set time, then the workflow continues its run.		**Generic** workflow palette

Table 1-3. Schema Elements and Icons (Continued)

Schema Element Name	Description	Icon	Location in Workflow Editor
Nested workflows	Starts several workflows simultaneously. You can choose to nest local workflows and remote workflows that are in a different Orchestrator server. You can also run workflows with different credentials. The workflow waits until all the nested workflows complete before it continues its run.		**Generic** workflow palette
Pre-Defined Task	Noneditable scripted elements that perform standard tasks that workflows commonly use. The following tasks are predefined: **Basic** ■ Sleep ■ Change credential ■ Wait until date ■ Wait for custom event ■ Increase counter ■ Decrease counter **Log** ■ System log ■ System warning ■ System error ■ Server log ■ Server warning ■ Server error ■ System+server log ■ System+server warning ■ System+server error **Network** ■ HTTP post ■ HTTP get ■ Send custom event		**Basic**, **Log**, and **Network** workflow palette

Schema Element Properties

Schema elements have properties that you can define and edit in the **Schema** tab of the workflow palette.

Edit the Global Properties of a Schema Element

You define the global properties of a schema element in the element's Info tab.

Prerequisites

Verify that the **Schema** tab of the workflow editor contains elements.

Procedure

1 Click the **Schema** tab in the workflow editor.

2 Select an element to edit by clicking the **Edit** icon ( ).

A dialog box that lists the properties of the element appears.

3 Click the **Info** tab.

4 Provide a name for the schema element in the **Name** text box.

This is the name that appears in the schema element in the workflow schema diagram.

5 From the **Interaction** drop-down menu, select a description.

The **Interaction** property allows you to select between standard descriptions of how this element interacts with objects outside of the workflow. This property is for information only.

6 (Optional) Provide a business status description in the **Business Status** text box.

The **Business Status** property is a brief description of what this element does. When a workflow is running, the workflow token shows the Business Status of each element as it runs. This feature is useful for tracking workflow status.

7 (Optional) In the **Description** text box, type a description of the schema element.

Schema Element Properties Tabs

You access the properties of a schema element by clicking on an element that you have dragged into the workflow schema. The properties of the element appear in tabs at the bottom of the workflow editor.

Different schema elements have different properties tabs.

Table 1-4. Properties Tabs per Schema Element

Schema Element Property Tab	Description	Applies to Schema Element Type
Attributes	Attributes that elements require from an external source, such as the user, an event, or a timer. The attributes can be a timeout limit, a time and date, a trigger, or user credentials.	■ User Interaction ■ Waiting Event ■ Waiting Timer
Decision	Defines the decision statement. The input parameter that the decision element receives either matches or does not match the decision statement, resulting two possible courses of action.	Decision
End Workflow	Stops the workflow, either because the workflow completed successfully, or because it encountered an error and returned an exception.	■ End ■ Exception
Exception	How this schema element behaves in the event of an exception.	■ Action ■ Asynchronous Workflow ■ Exception ■ Nested Workflows ■ Predefined Task ■ Schedule Workflow ■ Scriptable Task ■ User Interaction ■ Waiting Event ■ Waiting Timer ■ Workflow
External Inputs	Input parameters that the user must provide at a certain moment while the workflow runs.	User Interaction

Table 1-4. Properties Tabs per Schema Element (Continued)

Schema Element Property Tab	Description	Applies to Schema Element Type
IN	The IN binding for this element. The IN binding defines the way in which the schema element receives input from the element that precedes it in the workflow.	■ Action ■ Asynchronous Workflow ■ Custom Decision ■ Predefined Task ■ Schedule Workflow ■ Scriptable Task ■ Workflow
Info	The schema element's general properties and description. The information the **Info** tab displays depends on the type of schema element.	■ Action ■ Asynchronous Workflow ■ Custom Decision ■ Decision ■ Nested Workflows ■ Note ■ Predefined Task ■ Schedule Workflow ■ Scriptable Task ■ User Interaction ■ Waiting Event ■ Waiting Timer ■ Workflow
OUT	The OUT binding for this element. The OUT binding defines the way in which the schema element binds output parameters to the workflow attributes or to the workflow output parameters.	■ Action ■ Asynchronous Workflow ■ Predefined Task ■ Schedule Workflow ■ Scriptable Task ■ Workflow
Presentation	Defines the layout of the input parameters dialog box the user sees if the workflow needs user input while it is running.	User Interaction
Scripting	Shows the JavaScript function that defines the behavior of this schema element. For Asynchronous Workflow, Schedule Workflow, and Action elements this scripting is read-only. For scriptable task and custom decision elements, you edit the JavaScript in this tab.	■ Action ■ Asynchronous Workflow ■ Custom Decision ■ Predefined Task ■ Schedule Workflow ■ Scriptable Task
Visual Binding	Shows a graphical representation of how the parameters and attributes of this schema element bind to the parameters and attributes of the elements that come before and after it in the workflow. This is another representation of the element's IN and OUT bindings.	■ Action ■ Asynchronous Workflow ■ Predefined Task ■ Schedule Workflow ■ Scriptable Task ■ Workflow
Workflows	Selects the workflows to nest.	Nested Workflows

Links and Bindings

Links between elements determine the logical flow of the workflow. Bindings populate elements with data from other elements by binding input and output parameters to workflow attributes.

To understand links and bindings, you must understand the difference between the logical flow of a workflow and the data flow of a workflow.

Logical Flow of a Workflow

The logical flow of a workflow is the progression of the workflow from one element to the next in the schema as the workflow runs. You define the logical flow of the workflow by linking elements in the schema.

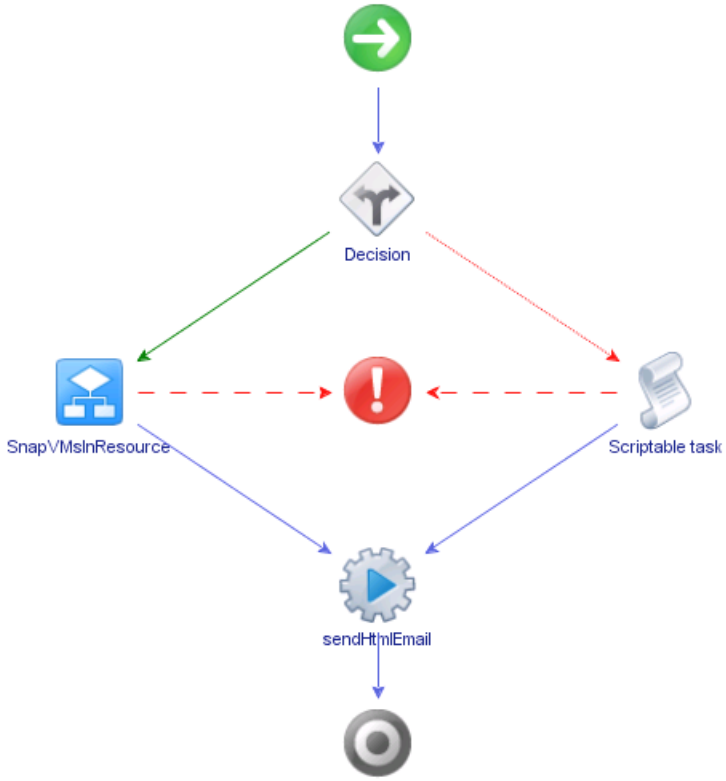
The standard path is the path that the workflow takes through the logical flow if all elements run as expected. The exception path is the path that the workflow takes through the logical flow if an element does not run as expected.

Different styles of arrows in the workflow schema denote the different paths that the workflow can take through its logical flow.

- A blue arrow denotes the standard path that the workflow takes from one element to the next.
- A green arrow denotes the path that the workflow takes if a Boolean decision element returns `true`.
- A red dotted arrow denotes the path that the workflow takes if a Boolean decision element returns `false`.
- A red dashed arrow denotes the exception path that the workflow takes if a workflow element does not run correctly.

The following figure shows an example workflow schema that demonstrates the different paths that workflows can take.

Figure 1-1. Different Workflow Paths Through the Logical Flow of the Workflow



This example workflow can take the following paths through its logical flow.

- Standard path, `true` decision result, no exceptions.
  a The decision element returns `true`.
  b The `SnapVMsInResourcePool` workflow runs successfully.
  c The `sendHtmlEmail` action runs successfully.
  d The workflow ends successfully in the `completed` state.
- Standard path, `false` decision result, no exceptions.
  a The decision element returns `false`.
  b The operation the scriptable task element defines runs successfully.
  c The `sendHtmlEmail` action runs successfully.
  d The workflow ends successfully in the `completed` state.
- `true` decision result, exception.
  a The decision element returns `true`.
  b The `SnapVMsInResourcePool` workflow encounters an error.
  c The workflow returns an exception and stops in the `failed` state.
- `false` decision result, exception.
  a The decision element returns `false`.
  b The operation the Scriptable task element defines encounters an error.
  c The workflow returns an exception and stops in the `failed` state.

Element Links

Links connect schema elements and define the logical flow of the workflow from one element to the next.

Elements can usually set only one outgoing link to another element in the workflow and one exception link to an element that defines its exception behavior. The outgoing link defines the standard path of the workflow. The exception link defines the exception path of the workflow. In most cases, a single schema element can receive incoming standard path links from multiple elements.

The following elements are exceptions to the preceding statements.

- The Start Workflow element cannot receive incoming links and has no exception link.
- Exception elements can receive multiple incoming exception links, and have no outgoing or exception links.
- Decision elements have two outgoing links that define the paths the workflow takes depending on the decision's `true` or `false` result. Decisions have no exception link.
- End Workflow elements cannot have outgoing links or exception links.

Create Standard Path Links

Standard path links determine the normal run of the workflow.

When you link one element to another, you always link the elements in the order in which they run in the workflow. You always start from the element that runs first to create a link between two elements.

Prerequisites

- Open a workflow for editing in the workflow editor.

- Verify that the **Schema** tab of the workflow editor contains elements.

Procedure

1 Place the pointer on the element that you want to connect to another element.

A blue and a red arrow appear on the element's right.

2 Place the pointer on the blue arrow.

The blue arrow enlarges.

3 Left-click the blue arrow, hold down the left mouse button, and move the pointer to the target element.

A blue arrow appears between the two elements and a green rectangle appears around the target element.

4 Release the left mouse button.

The blue arrow remains between the two elements.

A standard path now links the elements.

What to do next

The elements are joined, but you have not defined the data flow. You must define the IN and OUT bindings to bind incoming and outgoing data to workflow attributes.

Data Flow of a Workflow

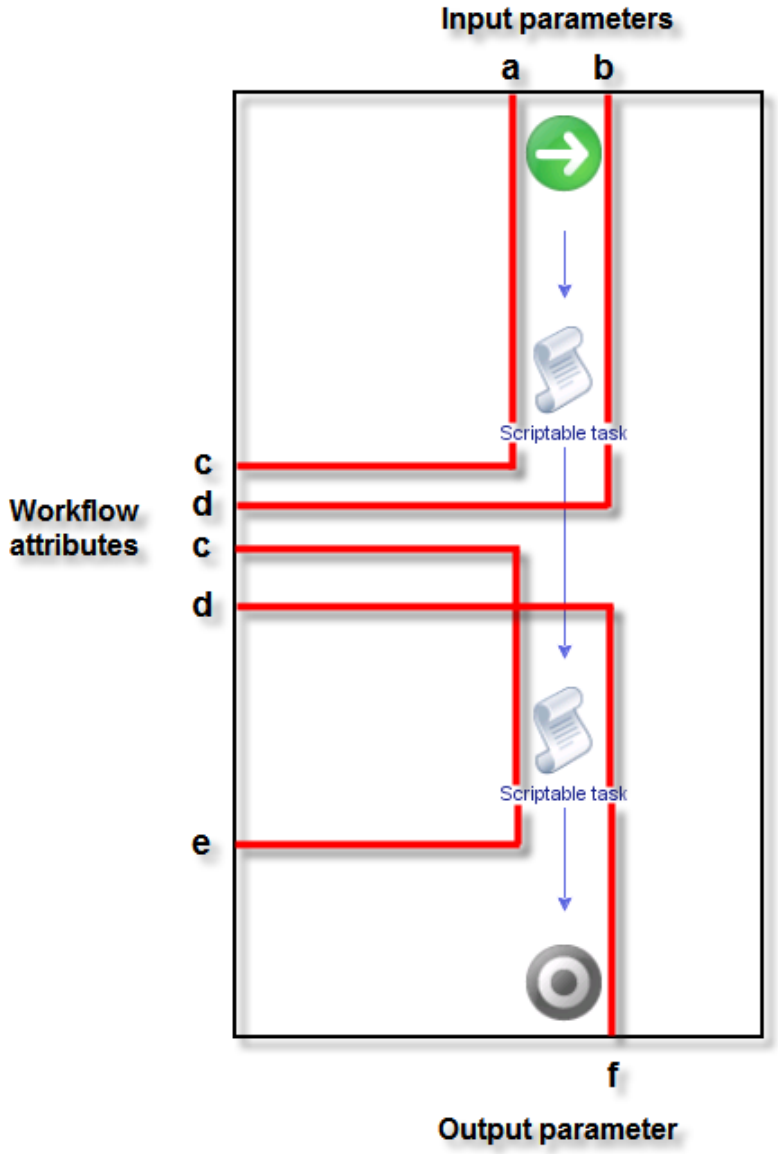
The data flow of a workflow is the manner in which workflow element input and output parameters bind to workflow attributes as each element of the workflow runs. You define the data flow of a workflow by using schema element bindings.

When an element in the workflow schema runs, it requires data in the form of input parameters. It takes the data for its input parameters by binding to a workflow attribute that you set when you create the workflow, or by binding to an attribute that a preceding element in the workflow set when it ran.

The element processes the data, possibly transforms it, and generates the results of its run in the form of output parameters. The element binds its resulting output parameters to new workflow attributes that it creates. Other elements in the schema can bind to these new workflow attributes as their input parameters. The workflow can generate the attributes as its output parameters at the end of its run.

The following figure shows a very simple workflow. The blue arrows represent the element linking and the logical flow of the workflow. The red lines show the data flow of the workflow.

Figure 1-2. Example of Workflow Data Flow



The data flows through the workflow as follows.

1 The workflow starts with input parameters a and b.

2 The first element processes parameter a and binds the result of the processing to workflow attribute c.

3 The first element processes parameter b and binds the result of the processing to workflow attribute d.

4 The second element takes workflow attribute c as an input parameter, processes it, and binds the resulting output parameter to workflow attribute e.

5 The second element takes workflow attribute d as an input parameter, processes it, and generates output parameter f.

6 The workflow ends and generates workflow attribute f as its output parameter, the result of its run.

Element Bindings

You must bind all workflow element input and output parameters to workflow attributes. Bindings set data in the elements, and define the output and exception behavior of the elements. Links define the logical flow of the workflow, whereas bindings define the data flow.

To set data in an element, generate output parameters from the element after processing, and handle any errors that might occur when the element runs, you must set the element binding.

IN bindings	Set a schema element's incoming data. You bind the element's local input parameters to source workflow attributes. The **IN** tab lists the element's input parameters in the Local Parameter column. The **IN** tab lists the workflow attributes to which the local parameter binds in the Source Parameter column. The tab also displays the parameter type and a description of the parameter.
OUT bindings	Change workflow attributes and generate output parameters when an element finishes its run. The **OUT** tab lists the element's output parameters in the Local Parameter column. The **OUT** tab lists the workflow attributes to which the local parameter binds in the Source Parameter column. The tab also displays the parameter type and a description of the parameter.
Exception bindings	Link to exception handlers if the element encounters an exception when it runs.

IN bindings read values from the bound source parameter. OUT bindings write values into the bound source parameter.

You must use IN bindings to bind every attribute or input parameter you use in a schema element to a workflow attribute. If the element changes the values of the input parameters that it receives when it runs, you must bind them to a workflow attribute by using an OUT binding. Binding the element's output parameters to workflow elements lets other elements that follow it in the workflow schema to take those output parameters as their input parameters.

A common mistake when creating workflows is to not bind output parameter values to reflect the changes that the element makes to the workflow attributes.

IMPORTANT When you add an element that requires input and output parameters of a type that you have already defined in the workflow, Orchestrator sets the bindings to these parameters. You must verify that the parameters that Orchestrator binds are correct, in case the workflow defines different parameters of the same type to which the element can bind.

Define Element Bindings

After you link elements to create the logical flow of the workflow, you define element bindings to define how each element processes the data it receives and generates.

Prerequisites

Verify that you have a workflow schema in the **Schema** tab of the workflow editor, and that you have created links between the elements.

Procedure

1 Click the **Edit** icon () of the element on which to set the bindings.

A dialog box that lists the properties of the element appears.

2 Click the **IN** tab.

The contents of the **IN** tab depend on the type of element you selected.

- If you selected a predefined task, workflow, or action element, the **IN** tab lists the possible local input parameters for that type of element, but the binding is not set.
- If you selected another type of element, you can select from a list of input parameters and attributes you already defined for the workflow by right-clicking in the **IN** tab and selecting **Bind to workflow parameter/attribute**.
- If the required attribute does not exist yet, you can create it by right-clicking in the **IN** tab and selecting **Bind to workflow parameter/attribute > Create parameter/attribute in workflow**.

3 If an appropriate parameter exists, choose an input parameter to bind, and click the **Not set** button in the **Source Parameter** text box.

A list of possible source parameters and attributes to bind to appears.

4 Choose a source parameter to bind to the local input parameter from the list proposed.

5 (Optional) If you have not defined the source parameter to which to bind, you can create it by clicking the **Create parameter/attribute in workflow** link in the parameter selection dialog box.

6 Click the **OUT** tab.

The contents of the **OUT** tab depend on the type of element you selected.

- If you selected a predefined task, workflow, or action element, the **OUT** tab lists the possible local output parameters for that type of element, but the binding is not set.
- If you selected another type of element, you can select from a list of output parameters and attributes you defined for the workflow by right-clicking in the **OUT** tab and selecting **Bind to workflow parameter/attribute**.
- If the required attribute does not exist, you can create it by right-clicking in the **IN** tab and selecting **Bind to workflow parameter/attribute > Create parameter/attribute in workflow**.

7 Choose a parameter to bind.

8 Click the **Source Parameter > Not set** button.

9 Choose a source parameter to bind to the input parameter.

10 (Optional) If you did not define the parameter to which to bind, you can create it by clicking the **Create parameter/attribute in workflow** button in the parameter selection dialog box.

You defined the input parameters that the element receives and the output parameters that it generates, and bound them to workflow attributes and parameters.

What to do next

You can create forks in the path of the workflow by defining decisions.

Decisions

Workflows can implement decision functions that define different courses of action according to a Boolean `true` or `false` statement.

Decisions are forks in the workflow. Workflow decisions are made according to inputs provided by you, by other workflows, by applications, or by the environment in which the workflow is running. The value of the input parameter that the decision element receives determines which branch of the fork the workflow takes. For example, a workflow decision might receive the power status of a given virtual machine as its input. If the virtual machine is powered on, the workflow takes a certain path through its logical flow. If the virtual machine is powered off, the workflow takes a different path.

Decisions are always Boolean functions. The only possible outcomes for each decision are `true` or `false`.

Custom Decisions

Custom decisions differ from standard decisions in that you define the decision statement in a script. Custom decisions return `true` or `false` according to the statement you define, as the following example shows.

```
if (decision_statement){
    return true;
}else{
    return false;
}
```

Create Decision Element Links

Decision elements differ from other elements in a workflow. They have only `true` or `false` output parameters. Decision elements have no exception linking.

Prerequisites

Verify that the **Schema** tab of the workflow editor contains elements, including at least one decision element that is not linked to other elements.

Procedure

1 Place the mouse pointer on a decision element to link it to two other elements that define two possible branches in the workflow.

A blue arrow and a red arrow appear on the element's right.

2 Place the pointer on the blue arrow, and while keeping the left mouse button pressed, move the pointer to the target element.

A green arrow appears between the two elements and the target element turns green. The green arrow represents the `true` path the workflow takes if the input parameter or attribute received by the decision element matches the decision statement.

3 Release the left mouse button.

The green arrow remains between the two elements. You have defined the path the workflow takes when the decision element receives the expected value.

4 Place the pointer on the decision element, hold down the left mouse button, and move the pointer to the target element.

A dotted red arrow appears between the two elements and the target element turns green. The red arrow represents the `false` path that the workflow takes if the input parameter or attribute received by the decision element does not match the decision statement.

5 Release the left mouse button.

The dotted red arrow remains between the two elements. You have defined the path the workflow takes when the decision element receives unexpected input.

You have defined the possible `true` or `false` paths that the workflow takes depending on the input parameter or attribute the decision element receives.

What to do next

Define the decision statement. See "Create Workflow Branches Using Decisions," on page 38.

Delete a Linked Decision Element

When you delete a linked decision element from a workflow schema, you must specify which workflow paths to delete.

Prerequisites

Verify that the **Schema** tab of the workflow editor contains elements, including at least one decision element with `true` and `false` paths.

Procedure

1 Select the decision element and press Delete.

A dialog box with available options appears.

2 Select which decision branch to delete.

Option	Description
Success branch	The decision element and all elements that follow the `true` decision path are deleted from the workflow schema.
Failure branch	The decision element and all elements that follow the `false` decision path are deleted from the workflow schema.
Both branches	The decision element and all elements that follow both decision paths are deleted from the workflow schema.
None	Only the decision element and its links are deleted from the workflow schema. All elements that follow both decision paths remain in the workflow schema.

3 Click **OK**.

Create Workflow Branches Using Decisions

Decision elements are simple Boolean functions that you use to create branches in workflows. Decision elements determine whether the input received matches the decision statement you set. As a function of this decision, the workflow continues its course along one of two possible paths.

Prerequisites

Verify that you have a decision element linked to two other elements in the schema in the workflow editor before you define the decision.

Procedure

1 Click the **Edit** icon ( ) of the decision element.

A dialog box that lists the properties of the decision element appears.

2 Click the **Decision** tab in the dialog box.

3 Click the **Not Set (NULL)** link to select the source input parameter for this decision.

A dialog box that lists all the attributes and input parameters defined in this workflow appears.

4 Select an input parameter from the list by double-clicking it.

5 If you did not define the source parameter to which to bind, create it by clicking the **Create attribute/parameter in workflow** link in the parameter selection dialog box.

6 Select a decision statement from the drop-down menu.

The statements that the menu proposes are contextual, and differ according to the type of input parameter selected.

7 Add a value that you want the decision statement to match.

Depending on the input type and the statement you select, you might see a **Not Set (NULL)** link in the value text box. Clicking this link gives you a predefined choice of values. Otherwise, for example for Strings, this is a text box in which you provide a value.

You defined a statement for the decision element. When the decision element receives the input parameter, it compares the value of the input parameter to the value in the statement and determines whether the statement is true or false.

What to do next

You must set how the workflow handles exceptions.

Exception Handling

Exception handling catches any errors that occur when a schema element runs. Exception handling defines how the schema element behaves when the error occurs.

All elements in a workflow, except for decisions and start and end elements, contain a specific output parameter type that serves only for handling exceptions. If an element encounters an error during its run, it can send an error signal to an exception handler. Exception handlers catch the error and react according to the errors they receive. If the exception handlers you define cannot handle a certain error, you can bind an element's exception output parameter to an Exception element, which ends the workflow run in the `failed` state.

Exceptions act as a `try` and `catch` sequence within a workflow element. If you do not need to handle a given exception in an element, you do not have to bind that element's exception output parameter.

The output parameter type for exceptions is always an `errorCode` object.

Create Exception Bindings

Elements can set bindings that define how the workflow behaves if it encounters an error in that element.

Prerequisites

Verify that the **Schema** tab of the workflow editor contains elements.

Procedure

1 Place the pointer on the element for which you want to define exception binding.

A blue and a red arrow appear on the element's right.

2 Place the pointer on the red arrow until it enlarges, hold down the left mouse button, and drag the red arrow to the target element.

A red dashed arrow links the two elements. The target element defines the behavior of the workflow if the element that links to it encounters an error.

3 Click the element that links to the exception handling element.

4 Click the **Exceptions** tab in the schema element properties tabs at the bottom of the **Schema** tab.

5 Click the **Not set** button to set the **Output Exception Binding** value.

- Select a parameter to bind to the exception output parameter from the exception attribute binding dialog box.
- Click **Create parameter/attribute in workflow** to create an exception output parameter.

6 Click the target element that defines the exception handling behavior.

7 Click the **IN** tab in the schema element properties tabs at the bottom of the **Schema** tab.

8 Right-click in the **IN** tab and select **Bind to workflow parameter/attribute**.

9 Select the exception output parameter and click **Select**.

10 Click the **OUT** tab for the exception handling element in the schema element properties tabs at the bottom of the **Schema** tab

11 Define the behavior of the exception handling element.

- Right-click in the **OUT** tab and select **Bind to workflow parameter/attribute** to select an output parameter for the exception handling element to generate.
- Click the **Scripting** tab and use JavaScript to define the behavior of the exception handling element.

You defined how the element handles exceptions.

What to do next

You must define how to obtain input parameters from users when they run the workflow.

Foreach Elements and Composite Types

You can insert a Foreach element in the workflow that you develop to run a subworkflow that iterates over arrays of parameters or attributes. To improve the understanding and readability of the workflow, you can group several workflow parameters of different types that are logically connected in a single type that is called a composite type.

Using Foreach Elements

A Foreach element runs a subworkflow iteratively over an array of input parameters or attributes. You can select the arrays over which the subworkflow is run, and can pass the values for the elements of such an array when you run the workflow. The subworkflow runs as many times as the number of elements that you have defined in the array.

If you have a configuration element that contains an array of attributes, you can run a workflow that iterates over these attributes in a Foreach element.

For example, suppose that you have 10 virtual machines in a folder that you want to rename. To do this, you must insert a Foreach element in a workflow and define the Rename virtual machine workflow as a subworkflow in the element. The Rename virtual machine workflow takes two input parameters, a virtual machine and its new name. You can promote these parameters as input to the current workflow, and as a result, they become arrays over which the Rename virtual machine workflow will iterate. When you run your workflow, you can specify the 10 virtual machines in the folder and their new names. Every time the workflow runs, it takes an element from the array of the virtual machines and an element from the array of the new names for the virtual machines.

Using Composite Types

A composite type is a group of more than one input parameter or attribute that are connected logically but are of different types. In a Foreach element, you can bind a group of parameters as a composite value. In this way, the Foreach element takes the values for the grouped parameters at once in every subsequent run of the workflow.

For example, suppose that you are about to rename a virtual machine. You need the virtual machine object and its new name. If you have to rename multiple virtual machines, you need two arrays, one for the virtual machines and one for their names. These two arrays are not explicitly connected. A composite type lets you have one array where each element contains both the virtual machine and its new name. In this way, the connection between those two parameters in case of multiple values is specified explicitly and not implied by the workflow schema.

NOTE You cannot run a workflow that contains composite types from an Orchestrator Web view or from the vSphere Web Client.

Define a Foreach Element

If you want to run a subworkflow multiple times by passing different values for its parameters or attributes in every subsequent run, you can insert a Foreach element in the parent workflow.

When you insert a Foreach element, you must select at least one array over which the Foreach element iterates. An array element can have different values for each subsequent workflow run.

If the subworkflow has output parameters, you should select the output parameters of the Foreach element in which to accumulate workflow outputs , so that the subworkflow can iterate over them as well.

Prerequisites

Open a workflow for editing in the workflow editor.

Procedure

1 In the workflow editor, select the **Schema** tab.

2 From the **Generic** menu, drag a Foreach element in the workflow schema.

3 Select a workflow from the Chooser dialog box.

The following notification appears at the top of the schema pane.

```
Do you want to add the activity's parameters as input/output to the current workflow?
```

4 On the notification, click **Setup**.

A pop-up window with the available options appears.

5 Select the mapping type for each input parameter.

Option	Description
Input	The argument is mapped to an input workflow parameter.
Skip	The argument is mapped to a `NULL` value.
Value	The argument is mapped to an attribute with a value that you can set from the Value column.

6 Select the mapping type for each output parameter.

Option	Description
Output	The argument is mapped to an output workflow parameter.
Skip	The argument is mapped to a `NULL` value.
Local variable	The argument is mapped to an attribute.

7 Click **Promote**.

8 Right-click the Foreach element and select **Synchronize > Synchronize presentation**.

A confirmation dialog box appears.

9 Click **Ok** to propagate the presentation of the Foreach element to the current workflow.

A dialog box displays information about the outcome of the operation.

10 On the **Inputs** tab, verify that the subworkflow's parameters are added as elements of type array.

11 On the **Outputs** tab, verify that the subworkflow's parameters are added as elements of type array.

You defined a Foreach element in your workflow. The Foreach element runs a workflow that takes as parameters every element from the array of parameters or attributes that you have defined.

For parameters or attributes that are not defined as arrays, the workflow takes the same value in every subsequent run.

Example: Rename Virtual Machines by Using a Foreach Element

You can use a Foreach element to rename several virtual machines at once. You have to insert a Foreach element in a workflow and promote the `vm` and the `newName` parameters as input to the current workflow. In this way, when you run the workflow, you specify the virtual machines to rename and the new names for the virtual machines. The virtual machines are included as elements in the array that you created for the `vm` parameter. The new names for the virtual machines are included in the array that you created for the `newName` parameter.

Define a Composite Type in a Foreach Element

You can group multiple workflow parameters that are connected logically in a new type that is called a composite type. You can use a Foreach element to bind a group of parameters as a composite value to connect several arrays of parameters in a single array.

Prerequisites

- Open a workflow for editing in the workflow editor.
- Verify that you have a Foreach element in your workflow.

Procedure

1 Select the **IN** or the **OUT** tab of the Foreach element.

2 Select a local parameter that you want to group with other local parameters in a composite type.

3 Click **Bind a group of parameters as composite value** at the top of the **IN** or the **OUT** tab.

4 In the Bindings pane, select the parameters that you want to group as a composite type.

5 Select **Bind as iterator**.

You have set the Foreach element to iterate over an array of the composite type.

6 Click **Accept**.

You defined a composite type and made sure that the workflow will iterate over an array of this composite type. Parameters that are grouped as a composite type are named *composite_type_name.parameter_name*. For example, if you create a `snapshots` composite type, the parameters that are group in the type can be `snapshots.vm[in-parameter]` or `snapshots.name[in-parameter]`. Every element from the array of the composite type contains a single instance of every parameter that you grouped in the composite type.

Example: Rename Virtual Machines

Suppose that you want to rename 10 virtual machines at a time. For this, you insert a Foreach element in a workflow and select the Rename virtual machine workflow in the element. You create a composite type to connect the `vm` and the `newName` parameters explicitly. You bind the composite type as an iterator, thus creating a single array that contains both the `vm` and the `newName` parameter.

Obtaining Input Parameters from Users When a Workflow Starts

If a workflow requires input parameters, it opens a dialog box in which users enter the required input parameter values when it runs. You can organize the content and layout, or presentation, of this dialog box in **Presentation** tab in the workflow editor.

The way you organize parameters in the **Presentation** tab translates into the input parameters dialog box when the workflow runs, and in the dialog box that opens when you run a workflow from a Web view.

The **Presentation** tab also allows you to add descriptions of the input parameters to help users when they provide input parameters. You can also set properties and constraints on parameters in the **Presentation** tab to limit the parameters that users provide. If the parameters the user provides do not meet the constraints you set in the **Presentation** tab, the workflow will not run.

IMPORTANT The use of OGNL expressions in workflow presentations is deprecated as of Orchestrator 4.1. Using OGNL expressions in workflow presentations is not supported in releases of Orchestrator later than 4.1.

- Creating the Input Parameters Dialog Box In the Presentation Tab on page 43

  You define the layout of the dialog box in which users provide input parameters when they run a workflow in the **Presentation** tab of the workflow editor.

- Setting Parameter Properties on page 45

  Orchestrator allows you to define properties to qualify the input parameter values that users provide when they run workflows. The parameter properties you define impose limits on the types and values of the input parameters the users provide.

Creating the Input Parameters Dialog Box In the Presentation Tab

You define the layout of the dialog box in which users provide input parameters when they run a workflow in the **Presentation** tab of the workflow editor.

The **Presentation** tab allows you to group input parameters into categories and to define the order in which these categories appear in the input parameters dialog box.

Presentation Descriptions

You can add an associated description for each parameter or group of parameters, which appears in the input parameters dialog box. The descriptions provide information to the users to help them provide the correct input parameters. You can enhance the layout of the description text by using HTML formatting.

Defining Presentation Input Steps

By default, the input parameters dialog box lists all the required input parameters in a single list. To help users enter input parameters, you can define nodes, called input steps, in the presentation tab. Input steps group input parameters of a similar nature. The input parameters under an input step appear in a distinct section in the input parameters dialog box when the workflow runs.

Defining Presentation Display Groups

Each input step can have nodes of its own called display groups. The display groups define the order in which parameter input text boxes appear within their section of the input parameters dialog box. You can define display groups independently of input steps.

Create the Presentation of the Input Parameters Dialog Box

You create the presentation of the dialog box in which users provide input parameters when they run a workflow in the **Presentation** tab in the workflow editor.

Prerequisites

- Open a workflow for editing in the workflow editor.
- Verify that the workflow has a defined list of input parameters.

Procedure

1 In the workflow editor, click the **Presentation** tab.

By default, all of the workflow's parameters appear under the main **Presentation** node in the order in which you create them.

2 Right-click the **Presentation** node and select **Create new step**.

A **New Step** node appears under the **Presentation** node.

3 Provide an appropriate name for the step and press Enter.

This name appears as a section header in the input parameters dialog box when the workflow runs.

4 Click the input step and add a description in the **General** tab in the bottom half of the **Presentation** tab.

This description appears in the input parameters dialog box to provide information to the users to help them provide the correct input parameters. You can enhance the layout of the description text by using HTML formatting.

5 Right-click the input step you created and select **Create display group**.

A **New Group** node appears under the input step node.

6 Provide an appropriate name for the display group and press Enter.

This name appears as a subsection header in the input parameters dialog box when the workflow runs.

7 Click the display group and add a description in the **General** tab in the bottom half of the **Presentation** tab.

This description appears in the input parameters dialog box. You can enhance the layout of the description text by using HTML formatting. You can add a parameter value to a group description by using an OGNL statement, such as `${#param}`.

8 Repeat the preceding steps until you have created all the input steps and display groups to appear in the input parameters dialog box when the workflow runs.

9 Drag parameters from under the **Presentation** node to the steps and groups of your choice.

You created the layout of the input parameters dialog box through which users provide input parameter values when the workflow runs.

What to do next

You must set the parameter properties.

Setting Parameter Properties

Orchestrator allows you to define properties to qualify the input parameter values that users provide when they run workflows. The parameter properties you define impose limits on the types and values of the input parameters the users provide.

Every parameter can have several properties. You define an input parameter's properties in the **Properties** tab for a given parameter in the **Presentation** tab.

Parameter properties validate the input parameters and modify the way that text boxes appear in the input parameters dialog box. Some parameter properties can create dependencies between parameters.

Static and Dynamic Parameter Property Values

A parameter property value can be either static or dynamic. Static property values remain constant. If you set a property value to static, you set or select the property's value from a list that the workflow editor generates according to the parameter type.

Dynamic property values depend on the value of another parameter or attribute. You define the functions by which dynamic properties obtain values by using an object graph navigation language (OGNL) expression. If a dynamic parameter property value depends on the value of another parameter property value and the other parameter property value changes, the OGNL expression recalculates and changes the dynamic property value.

Important The use of OGNL expressions in workflow presentations is deprecated as of Orchestrator 4.1. Using OGNL expressions in workflow presentations is not supported in releases of Orchestrator later than 4.1.

Set Parameter Properties

When a workflow starts, it validates input parameter values from users against any parameter properties that you set.

Prerequisites

- Open a workflow for editing in the workflow editor.
- Verify that the workflow has a defined list of input parameters.

Procedure

1. In the workflow editor, click the **Presentation** tab.
2. Click a parameter in the **Presentation** tab.

   The parameter's **General** and **Properties** tabs appear at the bottom of the **Presentation** tab.
3. Click the parameter's **Properties** tab.
4. Right-click in the **Properties** tab and select **Add property**.

   A dialog box opens, presenting a list of the possible properties for a parameter of the type selected.
5. Select a property from the list presented in the dialog box and click **OK**.

   The property appears in the **Properties** tab.

6 Under **Value**, make the property value either static or dynamic by selecting the corresponding symbol from the drop-down menu.

Option	Description
	Static property
	Dynamic property

7 If you set the property value to static, you select a property value according to the type of parameter for which you are setting the properties.

8 If you set the property value to dynamic, you define the function to obtain the parameter property value by using an OGNL expression.

The workflow editor provides help writing the OGNL expression.

a Click the  icon to obtain a list of all the attributes and parameters defined by the workflow that this expression can call upon.

b Click the  icon to obtain a list of all the actions in the Orchestrator API that return an output parameter of the type for which you are defining the properties.

Clicking items in the proposed lists of parameters and actions adds them to the OGNL expression.

9 Click **Save** at the bottom of the workflow editor.

You defined the properties of the workflow's input parameters.

What to do next

Validate and debug the workflow.

Workflow Input Parameter Properties

You can set parameter properties to constrain the input parameters that users provide when they run workflows.

Different parameter types can have different properties.

Table 1-5. Workflow Input Parameter Properties

Parameter Property	Parameter Type	Description
Maximum string length	String	Sets a maximum length for the parameter.
Minimum string length	String	Sets a minimum length for the parameter.
Matching regular expression	String	Validates the input using a regular expression.
Maximum number value	Number	Sets a maximum value for the parameter.
Minimum number value	Number	Sets a minimum value for the parameter.
Number format	Number	Formats the input for the parameter.
Mandatory input	All simple types	Makes the parameter mandatory.

Table 1-5. Workflow Input Parameter Properties (Continued)

Parameter Property	Parameter Type	Description
Predefined answers	All simple types	Pre-defines a list of possible values for the property as an array of simple types. You either define the array manually or the property calls an action that returns an array of objects of the appropriate type.
Predefined list of elements	Any simple or complex types	Pre-defines a list of possible values for the property as an array of simple or complex types. Calls an action that returns an array of objects of the appropriate type.
Show parameter input	Any simple or complex types	Shows or hides a parameter text box in the presentation dialog box, depending on the value of a preceding Boolean parameter.
Hide parameter input	Any simple or complex types	Similar to **Show parameter input**, but takes the negative value of a previous Boolean parameter.
Matching expression	Any parameter type obtained from a plug-in	The input parameter matches a given expression.
Show in inventory	Any parameter type obtained from a plug-in	If set, you can run the present workflow on any object of this type by right-clicking it in the inventory view and selecting **Run workflow**.
Specify a root object to be shown in the chooser. Root object is provided from a parameter or attribute.	Any parameter type obtained from a plug-in	Specifies the root object if the selector for this parameter is a hierarchical list selector.
Select as	Any parameter type obtained from a plug-in	Use a list or hierarchical list selector to select the parameter.
Default value	Any simple or complex types	Default value for this parameter.
Custom validation	OGNL scriptable validation	If the OGNL expression returns a string, the validation shows this string as the text of the error result.
Data binding	Any simple or complex types	Binds to a property that you have already defined in another parameter.
Authorized only	Any parameter type obtained from a plug-in	Only authorized users can access this parameter.
Multi-lines text input	Any simple or complex types	Allows users to enter multiple lines of text in the input parameters dialog box.

Predefined Constant Values for OGNL Expressions

You can use predefined constants when you create OGNL expressions to obtain dynamic parameter property values.

Orchestrator defines the following constants for use in OGNL expressions.

Table 1-6. Predefined OGNL Constant Values

Constant Value	Description
`${#__current}`	Current value of the custom validation property or matching expression property
`${#__username}`	User name of the user who started the workflow
`${#__userdisplayname}`	Display name of the user who started the workflow
`${#__serverurl}`	URL containing the IP address of the server from which the user starts the workflow. The URL consists of the server IP address and a lookup port: *{ServerIP}:{lookupPort}*
`${#__datetime}`	Current date and time
`${#__date}`	Current date, with time set to 00:00:00
`${#__timezone}`	Current timezone

(Optional) Requesting User Interactions While a Workflow Runs

A workflow can sometimes require additional input parameters from an outside source while it runs. These input parameters can come from another application or workflow, or the user can provide them directly.

For example, if a certain event occurs while a workflow runs, the workflow can request human interaction to decide what course of action to take. The workflow waits before continuing, either until the user responds to the request for information, or until the waiting time exceeds a possible timeout period. If the waiting time exceeds the timeout period, the workflow returns an exception.

The default attributes for user interactions are `security.group` and `timeout.date`. When you set the `security.group` attribute to a given LDAP user group, you limit the permission to respond to the user interaction request to members of that user group.

When you set the `timeout.date` attribute, you set a time and date until which the workflow waits for the information from the user. You can set an absolute date, or you can create a scripted workflow element to calculate a time relative to the current time.

Procedure

1 Add a User Interaction to a Workflow on page 49

You request input parameters from users during a workflow run by adding a **User Interaction** schema element to the workflow. When a workflow encounters a **User Interaction** element, it suspends its run and waits for the user to provide the data that it requires.

2 Set the User Interaction security.group Attribute on page 49

The `security.group` attribute of a user interaction element sets which users or groups of users have permission to respond to the user interaction.

3 Set the timeout.date Attribute to an Absolute Date on page 50

You set the `timeout.date` attribute for a user interaction to set how long the workflow waits for a user to respond to a user interaction.

4 Calculate a Relative Timeout for User Interactions on page 51

You can calculate in a `Date` object a relative time and date at which a user interaction times out.

5 Set the timeout.date Attribute to a Relative Date on page 52

You can set the `timeout.date` attribute of a **User Interaction** element to a relative time and date by binding it to a `Date` object. You define the object in a scripted function.

6 Define the External Inputs for a User Interaction on page 53

You specify the information that users must provide during a workflow run as the input parameters of a user interaction.

7 Define User Interaction Exception Behavior on page 54

If a user does not provide the input parameters within the timeout period, the user interaction returns an exception. You can define the exception behavior in a scripted function.

8 Create the Input Parameters Dialog Box for the User Interaction on page 55

Users provide input parameters during a workflow run in an input parameters dialog box, in the same way that they provide input parameters when a workflow first starts.

9 Respond to a Request for a User Interaction on page 56

Workflows that require interactions from users during their run suspend their run either until the user provides the required information or until the workflow times out.

Add a User Interaction to a Workflow

You request input parameters from users during a workflow run by adding a **User Interaction** schema element to the workflow. When a workflow encounters a **User Interaction** element, it suspends its run and waits for the user to provide the data that it requires.

Prerequisites

- Create a workflow.
- Open the workflow for editing in the workflow editor.
- Add some elements to the workflow schema.

Procedure

1 Drag a **User Interaction** element to the appropriate position in the workflow schema.

2 Click the **User Interaction** element to display its properties tabs in the bottom half of the **Schema** tab.

3 Click the **Edit** icon () of the **User Interaction** element.

4 Provide a name and a description for the user interaction in the **Info** tab and click **Close**.

5 Click **Save**.

You added a user interaction element to a workflow. When the workflow reaches this element, it waits for information from the user before continuing its run.

What to do next

Set the `security.group` attribute of the user interaction to limit permission to respond to the user interaction to a user or user group. See "Set the User Interaction security.group Attribute," on page 49.

Set the User Interaction security.group Attribute

The `security.group` attribute of a user interaction element sets which users or groups of users have permission to respond to the user interaction.

Prerequisites

- Create a workflow.
- Open the workflow for editing in the workflow editor.
- Add some elements and a user interaction to the workflow schema.

- Identify an LDAP user group to respond to the user interaction request.

Procedure

1. Click the **Edit** icon () of the **User Interaction** element in the workflow schema.
2. Click the **Attributes** tab for the user interaction.
3. Click **Not set** for the `security.group` source parameter to set which users can respond to the user interaction.
4. (Optional) Select **NULL** to allow all users to respond to the request for user interaction.
5. To limit the permission to respond to a specific user or user group, click **Create parameter/attribute in workflow**.

   The **Parameter information** dialog box opens.
6. Name the parameter.
7. Select **Create workflow ATTRIBUTE with the same name** to create the `LdapGroup` attribute in the workflow.
8. Click **Not set** for the parameter value to open the **LdapGroup** selection box.
9. Type the name of the LDAP user group in the **Filter** text box.
10. Select the LDAP user group from the list and click **Select**.

    For example, selecting the **Administrators** group means that only members of that group can respond to this request for user interaction.

    You limited the permission to respond to the user interaction request.
11. Click **OK** to close the **Parameter information** dialog box.

You set the `security.group` attribute for the user interaction.

What to do next

Set the `timer.date` attribute to set the timeout period for the user interaction.

- To set the timeout to an absolute date and time, see "Set the timeout.date Attribute to an Absolute Date," on page 50.
- To create a function to calculate a timeout that is relative to the current date and time, see "Calculate a Relative Timeout for User Interactions," on page 51.

Set the timeout.date Attribute to an Absolute Date

You set the `timeout.date` attribute for a user interaction to set how long the workflow waits for a user to respond to a user interaction.

You set an absolute time and date in the `Date` object. When the time on the given date arrives, the workflow that is waiting for a user interaction times out and ends in the `Failed` state. For example, you can set the user interaction to timeout at midday on February 12th. To calculate a timeout that is relative to the current time and date, see "Calculate a Relative Timeout for User Interactions," on page 51.

Prerequisites

- Open a workflow for editing in the workflow editor.
- Add a user interaction element to the workflow schema.
- Set the `security.group` attribute for the user interaction.

Procedure

1 Click the **Edit** icon () of the **User Interaction** element in the workflow schema.

2 Click the **Attributes** tab for the user interaction.

3 Click **Not set** for the `timeout.date` source parameter to set the timeout parameter value.

4 (Optional) Select **NULL** to allow the user interaction to set the workflow to wait indefinitely for the user to respond to the user interaction.

5 Click **Create parameter/attribute in workflow** to set the workflow to fail after a timeout period.

The **Parameter information** dialog box opens.

6 Name the parameter.

7 Select **Create workflow ATTRIBUTE with the same name** to create a `Date` attribute in the workflow.

8 Click **Not set** for the parameter **Value**.

9 Use the calendar to select an absolute date and time until which the workflow waits for the user to respond.

10 Click **OK** to close the calendar.

11 Click **OK** to close the **Parameter information** dialog box.

You set the `timeout.date` attribute to an absolute date. The workflow times out if the user does not respond to the user interaction before this time and date.

What to do next

Define the external input parameters that the user interaction requires from the user. See "Define the External Inputs for a User Interaction," on page 53.

Calculate a Relative Timeout for User Interactions

You can calculate in a `Date` object a relative time and date at which a user interaction times out.

You can set an absolute time and date in a `Date` object. When the time on the given date arrives, the request for a user interaction times out. Alternatively, you can create a workflow element that calculates and generates a relative `Date` object according to a function that you define. For example, you can create a relative `Date` object that adds 24 hours to the current time.

Prerequisites

- Open a workflow for editing in the workflow editor.
- Add a user interaction element to the workflow schema.
- Set the `security.group` attribute for the user interaction.

Procedure

1 Drag a **Scriptable task** element from the **Generic** menu to the schema of a workflow, before the element that requires the relative `Date` object for its `timeout.date` attribute.

2 Click the **Edit** icon () of the **Scriptable task** element in the workflow schema.

3 Provide a name and description for the scripted workflow element in the **Info** properties tab.

4 Click the **OUT** properties tab, and click the **Bind to workflow parameter/attribute** icon ().

5 Click **Create parameter/attribute in workflow** to create a workflow attribute.

   a Name the attribute `timerDate`.

   b Select `Date` from the list of attribute types.

   c Select **Create workflow ATTRIBUTE with the same name**.

   d Leave the attribute value set to **Not set**, because a scripted function will provide this value.

   e Click **OK**.

6 Click the **Scripting** tab for the scripted workflow element.

7 Define a function to calculate and generate a `Date` object named `timerDate` in the scripting pad in the **Scripting** tab.

   For example, you can create a `Date` object by implementing the following JavaScript function, in which the timeout period is a relative delay in milliseconds.

```
timerDate = new Date();
System.log( "Current date : '" + timerDate + "'" );
timerDate.setTime( timerDate.getTime() + (86400 * 1000) );
System.log( "Timer will expire at '" + timerDate + "'" );
```

   The preceding example JavaScript function defines a `Date` object that obtains the current date and time by using the `getTime` method and adds 86,400,000 milliseconds, or 24 hours. The **Scriptable Task** element generates this value as its output parameter.

8 Click **Close**.

9 Click **Save**.

You created a function that calculates a time and date relative to the current time and date and generates a `Date` object. A **User Interaction** element can receive this `Date` object as an input parameter to set the timeout period until which it waits for input from the user. When the workflow arrives at the **User Interaction** element, it suspends its run and waits either until the user provides the required information, or for 24 hours before it times out.

What to do next

You must bind the `Date` object to the **User Interaction** element's `timeout.date` parameter. See "Set the timeout.date Attribute to a Relative Date," on page 52.

Set the timeout.date Attribute to a Relative Date

You can set the `timeout.date` attribute of a **User Interaction** element to a relative time and date by binding it to a `Date` object. You define the object in a scripted function.

If you create a relative `Date` object in a scripted function, you can bind the `timeout.date` attribute of a user interaction to this `Date` object. For example, if you bind the `timeout.date` attribute to a `Date` object that adds 24 hours to the current time, the user interaction times out after waiting for 24 hours.

Prerequisites

- Add a user interaction element to the workflow schema.
- Set the `security.group` attribute for the user interaction.
- Create a scripted function that calculates a relative time and date and encapsulates it in a `Date` object in the workflow. See "Calculate a Relative Timeout for User Interactions," on page 51.

Procedure

1 Click the **Edit** icon ( ) of the **User Interaction** element in the workflow schema.

2 Click the **Attributes** tab for the user interaction.

3 Click **Not set** for the `timeout.date` source parameter to set the timeout parameter value.

4 Select the `Date` object that encapsulates a relative time and date that you defined in a scripted function and click **Select**.

You set the `timeout.date` attribute to a relative date and time that a scripted function calculates.

What to do next

Define the external input parameters that the user interaction requires from the user. See "Define the External Inputs for a User Interaction," on page 53.

Define the External Inputs for a User Interaction

You specify the information that users must provide during a workflow run as the input parameters of a user interaction.

When a workflow reaches a user interaction element, it waits until a user provides the information that the user interaction requires as its input parameters.

Prerequisites

- Add a user interaction element to the workflow schema.
- Set the `security.group` attribute for the user interaction.
- Set the `timer.date` attribute for the user interaction

Procedure

1 Click the **Edit** icon ( ) of the **User Interaction** element in the workflow schema.

2 Click the **External inputs** tab.

3 Click the **Bind to workflow parameter/attribute** icon ( ) to define the parameters that the user must provide in the user interaction.

4 (Optional) If you already defined the input parameters in the workflow, select the parameters from the proposed list.

5 Click **Create parameter/attribute in workflow** to create a workflow attribute to bind to the input parameter that the user provides.

6 Give the parameter an appropriate name.

7 Select the input parameter type from the list of types by searching for an object type in the **Filter** box.

For example, if the user interaction requires the user to provide a virtual machine as an input parameter, select `VC:VirtualMachine`.

8 Select **Create workflow ATTRIBUTE with the same name** to bind the input parameter that the user provides to a new attribute in the workflow.

9 Leave the input parameter value set to **Not set**.

The user provides this value when they respond to the user interaction during the workflow run.

10 Click **OK** to close the **Parameter information** dialog box.

You defined the input parameters that the user provides during a user interaction.

What to do next

Define the exception behavior if the user interaction encounters an error. See "Define User Interaction Exception Behavior," on page 54.

Define User Interaction Exception Behavior

If a user does not provide the input parameters within the timeout period, the user interaction returns an exception. You can define the exception behavior in a scripted function.

If you do not define the action for the workflow to take if the user interaction times out, the workflow ends in the `Failed` state. Defining the exception behavior is a good workflow development practice.

Prerequisites

- Add a user interaction element to the workflow schema.
- Set the `security.group` and `timer.date` attributes for the user interaction.
- Define the external input parameters of the user interaction.

Procedure

1. Click the **Edit** icon () of the **User Interaction** element in the workflow schema.
2. Click the **Exception** tab.
3. Click **Not set** for the output exception binding.
4. Click **Create parameter/attribute in workflow** to create an exception attribute to which to bind the user interaction.

   The **Parameter information** dialog box opens.
5. Create an `errorCode` attribute.

   An `errorCode` attribute has the following default properties:

   - Name: **errorCode**
   - Type: string
   - Create: **Create workflow ATTRIBUTE with the same name**
   - Value: Type an appropriate error message.
6. Click **OK** to close the **Parameter information** dialog box.
7. Drag a scriptable task element over the user interaction element in the workflow schema.

   A red dashed arrow, which represents the exception link, appears between the two elements. The scriptable task element binds automatically to the `errorCode` attribute from the user interaction.
8. Double-click the scriptable task element and provide an appropriate name.

   For example, **`Log timeout`**.
9. In the **Scripting** tab of the scriptable task element, write a JavaScript function to handle the exception.

   For example, to record the timeout in the Orchestrator log, write the following function:

   ```
   System.log("No response from user. Timed out.");
   ```

10 Link and bind the scriptable task element that handles exceptions to the element that follows it in the workflow.

For example, link and bind the scriptable task element to a **Throw exception** element to end the workflow with an error.

You defined the exception behavior if the user interaction times out.

What to do next

Create the dialog box in which users provide input parameters. See "Create the Input Parameters Dialog Box for the User Interaction," on page 55.

Create the Input Parameters Dialog Box for the User Interaction

Users provide input parameters during a workflow run in an input parameters dialog box, in the same way that they provide input parameters when a workflow first starts.

You create the layout of the dialog box in the **Presentation** tab of the user interaction element, not in the **Presentation** tab for the whole workflow. The **Presentation** tab of the whole workflow creates the layout of the input parameters dialog box that appears when you start a workflow. The **Presentation** tab of the user interaction element creates the layout of the input parameters dialog box that opens when a workflow arrives at a user interaction element during its run.

Prerequisites

- Add a user interaction element to the workflow schema.
- Set the `security.group` and `timer.date` attributes for the user interaction.
- Define the external input parameters of the user interaction.
- Define the exception behavior.

Procedure

1 Click the **Edit** icon () of the **User Interaction** element in the workflow schema.

2 Click the **Presentation** tab of the user interaction element.

The **Presentation** tab shows the external input parameters that you created for the user interaction.

3 (Optional) Right-click the **Presentation** node in the **Presentation** tab and select **Create new step**.

Steps allow you to create sections in the dialog box, with descriptions and headings under which you can organize the input parameters.

4 (Optional) Right-click the **Presentation** node in the **Presentation** tab and select **Create display group**.

Display groups allow you to sort the order in which input parameters appear in the steps, and allow you to add sub-headers and instructions to the dialog box.

5 Click an input parameter in the list and add a description of the input parameter in the **General** tab for that parameter.

The description text that you type appears as a label in the input parameters dialog box to inform the user of the information they must provide when they respond to the user interaction.

6 Define input parameter properties.

Input parameter properties allow you to qualify the input parameter values that users can provide, and to determine parameter values dynamically by using OGNL expressions.

7 Click **Save and close** to close the workflow editor.

You created the input parameters dialog box in which users provide input parameters to respond to a user interaction during a workflow run.

What to do next

For information about creating the presentation steps and groups and setting input parameter properties, see "Creating the Input Parameters Dialog Box In the Presentation Tab," on page 43.

Respond to a Request for a User Interaction

Workflows that require interactions from users during their run suspend their run either until the user provides the required information or until the workflow times out.

Workflows that require user interactions define which users can provide the required information and direct the requests for interaction.

Prerequisites

Verify that at least one workflow is in the Waiting for User Interaction state.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Run**.

2 Click the **My Orchestrator** view in the Orchestrator client.

3 Click the **Waiting for Input** tab.

The **Waiting for Input** tab lists the workflows that are waiting for user inputs from you or from members of your user group that have permission.

4 Double-click a workflow that is waiting for input.

The workflow token that is waiting for input appears in the **Workflows** hierarchical list with the following symbol:  .

5 Right-click the workflow token and select **Answer**.

6 Follow the instructions in the input parameters dialog box to provide the information that the workflow requires.

You provided information to a workflow that was waiting for user input during its run.

Calling Workflows Within Workflows

Workflows can call on other workflows during their run. A workflow can start another workflow either because it requires the result of the other workflow as an input parameter for its own run, or it can start a workflow and let it continue its own run independently. Workflows can also start a workflow at a given time in the future, or start multiple workflows simultaneously.

- Workflow Elements that Call Workflows on page 57

  There are four ways to call other workflows from within a workflow. Each way of calling a workflow or workflows is represented by a different workflow schema element.

- Call a Workflow Synchronously on page 59

  Calling a workflow synchronously runs the called workflow as a part of the run of the calling workflow. The calling workflow can use the called workflow's output parameters as input parameters when it runs its subsequent schema elements.

- Call a Workflow Asynchronously on page 60

  Calling a workflow asynchronously runs the called workflow independently of the calling workflow. The calling workflow continues its run without waiting for the called workflow to complete.

- Schedule a Workflow on page 61

  You can call a workflow from a workflow and schedule it to start at a later time and date.

- Prerequisites for Calling a Remote Workflow from Within Another Workflow on page 61

  If the workflow that you develop calls another workflow that resides on a remote Orchestrator server, certain prerequisites must be fulfilled so that the remote workflow can run successfully.

- Call Several Workflows Simultaneously on page 62

  Calling several workflows simultaneously runs the called workflows synchronously as part of the run of the calling workflow. The calling workflow waits for all of the called workflows to complete before it continues. The calling workflow can use the results of the called workflows as input parameters when it runs its subsequent schema elements.

Workflow Elements that Call Workflows

There are four ways to call other workflows from within a workflow. Each way of calling a workflow or workflows is represented by a different workflow schema element.

Synchronous Workflows

A workflow can start another workflow synchronously. The called workflow runs as an integral part of the calling workflow's run, and runs in the same memory space as the calling workflow. The calling workflow starts another workflow, then waits until the end of the called workflow's run before it starts running the next element in its schema. Usually, you call a workflow synchronously because the calling workflow requires the output of the called workflow as an input parameter for a subsequent schema element. For example, a workflow can call the Start virtual machine and wait workflow to start a virtual machine, and then obtain the IP address of this virtual machine to pass to another element or to a user by email.

Asynchronous Workflows

A workflow can start a workflow asynchronously. The calling workflow starts another workflow, but the calling workflow immediately continues running the next element in its schema, without waiting for the result of the called workflow. The called workflows run with input parameters that the calling workflow defines, but the lifecycle of the called workflow is independent from the lifecycle of the calling workflow. Asynchronous workflows allow you to create chains of workflows that pass input parameters from one workflow to the next. For example, a workflow can create various objects during its run. The workflow can then start asynchronous workflows that use these objects as input parameters in their own runs. When the original workflow has started all the required workflows and run its remaining elements, it ends. However, the asynchronous workflows it started continue their runs independently of the workflow that started them.

To make the calling workflow wait for the result of the called workflow, either use a nested workflow or create a scriptable task that retrieves the state of the workflow token of the called workflow and then retrieves the result of the workflow when it completes.

Scheduled Workflows

A workflow can call a workflow but defer starting that workflow until a later time and date. The calling workflow then continues its run until it ends. Calling a scheduled workflow creates a task to start that workflow at the given time and date. When the calling workflow has run, you can view the scheduled workflow in the **Scheduler** and **My Orchestrator** views in the Orchestrator client.

Scheduled workflows only run once. You can schedule a workflow to run recurrently by calling the `Workflow.scheduleRecurrently` method in a scriptable task element in a synchronous workflow.

Nested Workflows

A workflow can start several workflows simultaneously by nesting several workflows in a single schema element. All the workflows listed in the nested workflow element start simultaneously when the calling workflow arrives at the nested workflows element in its schema. Significantly, each nested workflow starts in a different memory space from the memory space of the calling workflow. The calling workflow waits until all the nested workflows have completed their runs before it starts running the next element in its schema. The calling workflow can thus use the results of the nested workflows as input parameters when it runs its remaining elements.

Propagate Workflow Changes to other Workflows

If you call a workflow from another workflow, Orchestrator imports the input parameters of the child workflow in the parent workflow at the moment you add the workflow element to the schema.

If you modify the child workflow after you have added it to another workflow, the parent workflow calls on the new version of the child workflow, but does not import any new input parameters. To prevent changes to workflows affecting the behavior of other workflows that call them, Orchestrator does not propagate the new input parameters automatically to the calling workflows.

To propagate parameters from one workflow to other workflows that call it, you must find the workflows that call the workflow, and synchronize the workflows manually.

Prerequisites

Verify that you have a workflow that another workflow or workflows call.

Procedure

1 Modify and save a workflow that other workflows call.

2 Close the workflow editor.

3 Navigate to the workflow you changed in the hierarchical list in the **Workflows** view in the Orchestrator client.

4 Right-click the workflow, and select **References > Find Elements that Use this Element**.

A list of workflows that call this workflow appears.

5 Double-click a workflow in the list to highlight it in the **Workflows** view in the Orchestrator client.

6 Right-click the workflow, and select **Edit**.

The workflow editor opens.

7 Click the **Schema** tab in the workflow editor.

8 Right-click the workflow element for the changed workflow from the workflow schema and select **Synchronize > Synchronize Parameters**.

9 Select **Continue** in the confirmation dialog box.

10 Save and close the workflow editor.

11 Repeat Step 5 to Step 10 for all the workflows that use the modified workflow.

You propagated a changed workflow to other workflows that call it.

Propagate the Input Parameters and Presentation of a Child Workflow to the Parent Workflow

If you develop a workflow that calls other workflows, you can propagate the input parameters and the presentation of the child workflows to the parent workflow.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Run**.

2 Right-click the workflow that you want to modify and select **Edit**.

The workflow editor opens.

3 Select the **Schema** tab.

4 Right-click the element of the child workflow whose input parameters and presentation you want to propagate to the parent workflow and select **Synchronize > Synchronize Presentation**.

5 In the confirmation dialog, select **OK**.

6 (Optional) Repeat Step 4 and Step 5 for all child workflows whose input parameters and presentation you want to propagate to the parent workflow.

The input parameters of the child workflows are added to the input parameters of the parent workflow. The presentation of the parent workflow is extended with the presentations of the child workflows.

Call a Workflow Synchronously

Calling a workflow synchronously runs the called workflow as a part of the run of the calling workflow. The calling workflow can use the called workflow's output parameters as input parameters when it runs its subsequent schema elements.

You call workflows synchronously from another workflow by using the **Workflow** element.

Prerequisites

- Open a workflow for editing in the workflow editor.
- Add some elements to the workflow schema.

Procedure

1 Drag a **Workflow** element from the **Generic** menu to the appropriate position in the workflow schema.

The **Choose workflow** selection dialog box appears.

2 Search for ands select the workflow you want and click **OK**.

If the search returns a partial result, narrow your search criterion or increase the number of search results from the **Tools > User preferences** menu in the client.

3 Click the **Workflow** element to show its properties tabs in the bottom half of the **Schema** tab.

4 Click the **Edit** icon () of the **Workflow** element in the workflow schema.

5 Bind the required input parameters to the workflow in the **IN** tab of the workflow schema element.

6 Bind the required output parameters to the workflow in the **OUT** tab of the workflow schema element's.

7 Define the exception behavior of the workflow in the **Exceptions** tab.

8 Click **Close**.

9 Click **Save** at the bottom of the workflow editor.

You called a workflow synchronously from another workflow. When the workflow reaches the synchronous workflow during its run, the synchronous workflow starts, and the initial workflow waits for it to complete before continuing its run.

What to do next

You can call a workflow asynchronously from a workflow.

Call a Workflow Asynchronously

Calling a workflow asynchronously runs the called workflow independently of the calling workflow. The calling workflow continues its run without waiting for the called workflow to complete.

You call workflows asynchronously from another workflow by using the **Asynchronous Workflow** element.

Prerequisites

- Open a workflow for editing in the workflow editor.
- Add some elements to the workflow schema.

Procedure

1 Drag an **Asynchronous Workflow** element from the **Generic** menu to the appropriate position in the workflow schema.

The **Choose workflow** selection dialog box appears.

2 Search for and select the desired workflow from the list and click **OK**.

3 Click the **Edit** icon ( ) of the **Asynchronous Workflow** element in the workflow schema.

4 Bind the required input parameters to the workflow in **IN** tab of the asynchronous workflow element.

5 Bind the required output parameter in the **OUT** tab of the asynchronous workflow element.

You can bind the output parameter either to the called workflow, or to that workflow's result.

- Bind to the called workflow to return that workflow as an output parameter
- Bind to the workflow token of the called workflow to return the result of running the called workflow.

6 Define the exception behavior of the asynchronous workflow element in the **Exceptions** tab.

7 Click **Close**.

8 Click **Save** at the bottom of the workflow editor.

You called a workflow asynchronously from another workflow. When the workflow reaches the asynchronous workflow during its run, the asynchronous workflow starts, and the initial workflow continues its run without waiting for the asynchronous workflow to finish.

What to do next

You can schedule a workflow to start at a later time and date.

Schedule a Workflow

You can call a workflow from a workflow and schedule it to start at a later time and date.

You schedule workflows in another workflow by using the **Schedule Workflow** element.

Prerequisites

- Open a workflow for editing in the workflow editor.
- Add some elements to the workflow schema.

Procedure

1. Drag a **Schedule Workflow** element from the **Generic** menu to the appropriate position in the workflow schema.
2. Search for the workflow to call by typing part of its name in the text box.
3. Select the workflow from the list and click **OK**.
4. Click the **Edit** icon ( ) of the **Schedule Workflow** element in the workflow schema.
5. Click the **IN** property tab.

   A parameter named `workflowScheduleDate` appears in the list of properties to define, together with the input parameters of the calling workflow.
6. Click **Not set** for the `workflowScheduleDate` parameter to set the parameter.
7. Click **Create parameter/attribute in workflow** to create the parameter and set the parameter value.
8. Click **Not set** for **Value** to set the parameter value.
9. Use the calendar that appears to set the date and time to start the scheduled workflow and click **OK**.
10. Bind the remaining input parameters to the scheduled workflow in the **IN** tab of the scheduled workflow element.
11. Bind the required output parameters to the `Task` object in the **OUT** tab of the scheduled workflow element.
12. Define the exception behavior of the scheduled workflow element in the **Exceptions** tab.
13. Click **Close**.
14. Click **Save** at the bottom of the workflow editor.

You scheduled a workflow to start at a given time and date from another workflow.

What to do next

You can call multiple workflows simultaneously from a workflow.

Prerequisites for Calling a Remote Workflow from Within Another Workflow

If the workflow that you develop calls another workflow that resides on a remote Orchestrator server, certain prerequisites must be fulfilled so that the remote workflow can run successfully.

- All input parameters of the remote workflow must be resolvable on the remote Orchestrator server.
- All output parameters of the remote workflow must be resolvable on the local Orchestrator server.

To ensure that the parameters of the remote workflow are resolvable, the inventory objects that the workflow uses must be available both in the remote and the local Orchestrator servers. In case the remote workflow uses objects from a plug-in, the same plug-in must be available on both Orchestrator servers. The inventories of the remote plug-in and the local plug-in must be identical. In case the remote workflow uses system objects in Orchestrator, like workflows and actions, the same workflows and actions must exist in the inventories of the remote and the local Orchestrator servers.

For example, suppose that you insert the Rename virtual machine workflow in a Nested Workflow element in the Test workflow that you develop. You want to run the Rename virtual machine workflow in a remote Orchestrator server. When you run the Test workflow, the Rename virtual machine workflow is called within the run of the Test workflow. You specify a virtual machine to rename from the inventory of the local Orchestrator server. Because the Rename virtual machine workflow runs on the remote Orchestrator server, the same virtual machine must be available in the inventory of that server. Otherwise, the Rename virtual machine workflow cannot resolve its `vm` input parameter. Therefore, the vCenter Server plug-in on the local and the remote Orchestrator servers must be connected to the same vCenter Server instance.

Call Several Workflows Simultaneously

Calling several workflows simultaneously runs the called workflows synchronously as part of the run of the calling workflow. The calling workflow waits for all of the called workflows to complete before it continues. The calling workflow can use the results of the called workflows as input parameters when it runs its subsequent schema elements.

You call several workflows simultaneously from another workflow by using the **Nested Workflows** element. You can use nested workflows to run workflows with user credentials that are different from the credentials of the user of the calling workflow.

Prerequisites

- Open a workflow for editing in the workflow editor.
- Add some elements to the workflow schema.

Procedure

1 Drag a **Nested Workflows** element from the **Action & Workflow** menu to the appropriate position in the workflow schema.

The **Choose workflow** selection dialog box appears.

2 Search for and select a workflow to start and click **OK**.

3 Click the **Edit** icon () of the **Nested Workflows** element in the workflow schema.

4 Click the **Workflows** tab.

The workflow you selected in Step 2 appears in the tab.

5 Set the IN and OUT bindings for this workflow in the **IN** and **OUT** tabs in the right panel of the **Workflows** schema element properties tab.

6 Click the **Connection Info** tab in the right panel of the **Workflows** schema element properties tab.

The **Connection Info** tab allows you to access workflows stored in a different server to the local one, using the appropriate credentials.

7 To access workflows on a remote server, select **Remote** and click **Not set** to provide a host name or IP address for the remote server.

NOTE You can use the vCenter Orchestrator Multi-Node plug-in to call workflows on a remote server.

8 Define the credentials with which to access the remote server.

- Select **Inherit** to use the same credentials as the user who runs the calling workflow.
- Select **Dynamic** and click **Not set** to select a set of dynamic credentials that a parameter of the `credentials` type defines elsewhere in the workflow.
- Select **Static** and click **Not set** to enter the credentials directly.

9 Click the **Add Workflow** button in the **Workflows** tab to select more workflows to add to the nested workflow element.

10 Repeat Step 2 to Step 8 to define the settings for each of the workflows you add.

11 Click the nested workflow element in the workflow schema.

The number of workflows nested in the element appears as a numeral on the nested workflows element.

You called several workflows simultaneously from a workflow.

What to do next

You can define long-running workflows.

Running a Workflow on a Selection of Objects

You can automate repetitive tasks by running a workflow on a selection of objects. For example, you can create a workflow that takes a snapshot of all the virtual machines in a virtual machine folder, or you can create a workflow that powers off all the virtual machines on a given host.

You can use one of the following methods to run a workflow on a selection of objects.

- Run the **Library > vCenter > Batch > Run a workflow on a selection of objects** workflow.
- Create a workflow that calls the **Library > Orchestrator > Start workflows in a series** or **Start workflows in parallel** workflows.
- Create a workflow that obtains an array of objects and runs a workflow on each object in the array in a loop of workflow elements.
- Run a workflow from JavaScript by calling the `Workflow.execute()` method in a `For` loop in a scripted element in a workflow.

Which method you choose to run a workflow on a selection of objects depends on the workflow to run and can affect the performance of the workflow. For example, running the Run a workflow on a selection of objects workflow is the simplest way to run a workflow on multiple objects and requires no workflow development, but it can only run workflows that take a single input parameter.

Creating a workflow that calls the Start workflows in a series or Start workflows in parallel workflows allows you to run on multiple objects workflows that take more than one input parameter. The calling workflow must create a properties array to pass the input parameters to the Start workflows in a series or Start workflows in parallel workflow. These workflows are only for use in other workflows. Do not run them directly.

Running a workflow in a `For` loop in a scripted element is faster than running a workflow in a loop of workflow elements, but it is less flexible and limits the potential for reuse. Most importantly, running a workflow in a scripted loop loses the checkpointing that Orchestrator performs when it starts each element in a workflow run. As a consequence, if the Orchestrator server stops while the scripted loop is running, when the server restarts, the workflow will resume at the beginning of the scripted element, repeating the whole loop. If the Orchestrator server stops while running a workflow with a loop of workflow elements, the workflow will resume at the specific element in the loop that was running when the server stopped.

For more information about the Batch workflows, see *Using VMware vCenter Orchestrator Plug-Ins.*

How to create a workflow that runs a workflow on an array of objects in a loop of workflow elements is demonstrated in "Develop a Complex Workflow," on page 104.

How to run a workflow in a scripted `For` loop is demonstrated in "Workflow Scripting Examples," on page 138.

Implement the Start Workflows in a Series and Start Workflows in Parallel Workflows

You can use the Start workflows in a series and Start workflows in parallel workflows to run a workflow on a selection of objects.

You cannot run the Start workflows in a series and Start workflows in parallel workflows directly. You must include them in another workflow that you create. To use the Start workflows in a series and Start workflows in parallel workflows to run a workflow on a selection of objects, you must obtain the objects on which to run the workflow. You pass these objects and any other input parameters that the workflow requires to the workflow as an array of properties. The Start workflows in a series and Start workflows in parallel workflows emit the results of running the workflow on the selection of objects as an array of `WorkflowToken` objects.

You implement the Start workflows in a series and Start workflows in parallel workflows in the same way. The Start workflows in a series workflow runs the workflow on each object sequentially. The Start workflows in parallel workflow runs the workflow on all the objects simultaneously.

Prerequisites

Open a workflow for editing in the workflow editor.

Procedure

1 In the workflow schema, add a scriptable task element or an action to obtain a list of objects on which to run the workflow.

   For example, to run a workflow on all the virtual machines in a virtual machine folder, you can add the `getAllVirtualMachinesByFolder` action to the workflow.

2 Link the scripted element or action and bind the input and output of the scripted element or action to workflow inputs or attributes.

   For example, you can bind the `vmFolder` input of the `getAllVirtualMachinesByFolder` action to a workflow input parameter and the `actionResult` output to a workflow attribute in the calling workflow.

3 Add a scriptable task element to cast the list of objects into a properties array.

   For example, if the objects on which to run the workflow are an array of virtual machines, `allVMs`, returned by the `actionResult` output of the `getAllVirtualMachinesByFolder` action, you can write the following script to cast the objects into a properties array.

```
propsArray = new Array();

for each (var vm in allVMs) {
 var prop = new Properties();
 prop.put("vm", vm);
 propsArray.push(prop);
}
```

4 Bind the inputs and outputs of the scriptable task element to workflow attributes.

   In the example scriptable task element in Step 3, you bind the input to the `allVMs` array of virtual machines and you create the `propsArray` output attribute as an array of `Properties` objects.

5 Add a workflow element to the workflow schema.

6 Select either of the Start workflows in a series or Start workflows in parallel workflows and link the workflow element to the other elements.

7 Bind the `wf` input of the Start workflows in a series or Start workflows in parallel workflow to the workflow to run on the objects.

   For example, to remove any snapshots of all the virtual machines returned by the `getAllVirtualMachinesByFolder` action, select the Remove all snapshots workflow.

8 Bind the `parameters` input of the Start workflows in a series or Start workflows in parallel workflow to the array of `Properties` objects that contains the objects on which to run the workflow.

   For example, bind the `parameters` input to the `propsArray` attribute defined in Step 4.

9 (Optional) Bind the `workflowTokens` output of the Start workflows in a series or Start workflows in parallel workflow to an attribute in the workflow.

10 (Optional) Continue adding more elements that use the results of running the Start workflows in a series or Start workflows in parallel workflow.

You created a workflow that uses either of the Start workflows in a series or Start workflows in parallel workflows to run a workflow on a selection of objects.

Developing Long-Running Workflows

A workflow in a waiting state consumes system resources because it constantly polls the object from which it requires a response. If you know that a workflow will potentially wait for a long time before it receives the response it requires, you can add long-running workflow elements to the workflow.

Every running workflow consumes a system thread. When a workflow reaches a long-running workflow element, the long-running workflow element sets the workflow into a passive state. The long-running workflow element then passes the workflow information to a single thread that polls the system for all long-running workflow elements running in the server. Rather than each long-running workflow element constantly attempting to retrieve information from the system, long-running workflow elements remain passive for a set duration, while the long-running workflow thread polls the system on its behalf.

You set the duration of the wait in one of the following ways:

- Set a timer, encapsulated in a `Date` object, that suspends the workflow until a certain time and date. You implement long-running workflow elements that are based on a timer by including a **Waiting Timer** element in the schema.
- Define a trigger event, encapsulated in a `Trigger` object, that restarts the workflow after the trigger event occurs. You implement long-running workflow elements that are based on a trigger by adding a **Waiting Event** element or a **User Interaction** element in the schema.

Set a Relative Time and Date for Timer-Based Workflows

You can set the `timer.date` attribute of a Waiting Timer element to a relative time and date by binding it to a `Date` object. You define the `Date` object in a scripted function.

When the time on the given date arrives, the long-running workflow that is based on a timer reactivates and continues its run. For example, you can set the workflow to reactivate at midday on February 12. Alternatively, you can create a workflow element that calculates and generates a relative `Date` object according to a function that you define. For example, you can create a relative `Date` object that adds 24 hours to the current time.

Prerequisites

- Create a workflow.
- Open the workflow for editing in the workflow editor.

- Add some elements to the workflow schema.

Procedure

1. Drag a **Scriptable task** element from the **Generic** menu to the schema of a workflow, before the element that requires the relative `Date` object for its `timeout.date` attribute.
2. Click the **Edit** icon () of the **Scriptable task** element in the workflow schema.
3. Provide a name and description for the scripted workflow element in the **Info** properties tab.
4. Click the **OUT** properties tab, and click the **Bind to workflow parameter/attribute** icon ().
5. Click **Create parameter/attribute in workflow** to create a workflow attribute.
   a. Name the attribute `timerDate`.
   b. Select `Date` from the list of attribute types.
   c. Select **Create workflow ATTRIBUTE with the same name**.
   d. Leave the attribute value set to **Not set**, because a scripted function will provide this value.
   e. Click **OK**.
6. Click the **Scripting** tab for the scripted workflow element.
7. Define a function to calculate and generate a `Date` object named `timerDate` in the scripting pad in the **Scripting** tab.

   For example, you can create a `Date` object by implementing the following JavaScript function, in which the timeout period is a relative delay in milliseconds.

   ```
   timerDate = new Date();
   System.log( "Current date : '" + timerDate + "'" );
   timerDate.setTime( timerDate.getTime() + (86400 * 1000) );
   System.log( "Timer will expire at '" + timerDate + "'" );
   ```

   The preceding example JavaScript function defines a `Date` object that obtains the current date and time by using the `getTime` method and adds 86,400,000 milliseconds, or 24 hours. The **Scriptable Task** element generates this value as its output parameter.
8. Click **Close**.
9. Click **Save**.

You created a function that calculates and generates a `Date` object. A **Waiting Timer** element can receive this `Date` object as an input parameter, to suspend a long-running workflow until the date encapsulated in this object. When the workflow arrives at the **Waiting Timer** element, it suspends its run and waits for 24 hours before continuing.

What to do next

You must add a **Waiting Timer** element to a workflow to implement a long-running workflow that is based on a timer.

Create a Timer-Based Long-Running Workflow

If you know a workflow will have to wait for a response from an outside source for a predictable time, you can implement it as a timer-based long-running workflow. A timer-based long-running workflow waits until a given time and date before resuming.

You implement a workflow as a timer-based long-running workflow by using the **Waiting Timer** element.

Prerequisites

- Create a workflow.
- Open the workflow for editing in the workflow editor.
- Add some elements to the workflow schema.

Procedure

1 Drag a **Waiting Timer** element from the **Generic** menu to the position in the workflow schema at which to suspend the workflow's run.

If you implement a scriptable task to calculate the time and date, this element must precede the **Waiting Timer** element.

2 Click the **Edit** icon () of the **Waiting Timer** element in the workflow schema.

3 Provide a description of the reason for implementing the timer in the **Info** properties tab.

4 Click the **Attributes** properties tab.

The `timer.date` parameter appears in the list of attributes.

5 Click the `timer.date` parameter's **Not set** button to bind the parameter to an appropriate `Date` object.

The **Waiting Timer** selection dialog box opens, presenting a list of possible bindings.

- Select a predefined `Date` object from the proposed list, for example one defined by a **Scriptable Task** element elsewhere in the workflow.
- Alternatively, create a `Date` object that sets a specific date and time for the workflow to await.

6 (Optional) Create a `Date` object that sets a specific date and time that the workflow awaits.

a Click **Create parameter/attribute in workflow** in the **Waiting Timer** selection dialog box.

The **Parameter information** dialog box appears.

b Give the parameter an appropriate name.

c Leave the type set to `Date`.

d Click **Create workflow ATTRIBUTE with the same name**.

e Click the **Value** property's **Not set** button to set the parameter value.

A calendar appears.

f Use the calendar to set a date and time at which to restart workflow.

g Click **OK**.

7 Click **Close**.

8 Click **Save** at the bottom of the workflow editor.

You defined a timer that suspends a timer-based long-running workflow until a set time and date.

What to do next

You can create a long-running workflow that waits for a trigger event before continuing.

Create a Trigger Object

Trigger objects monitor event triggers that plug-ins define. For example, the vCenter Server plug-in defines these events as `Task` objects. When the task ends, the trigger sends a message to a waiting trigger-based long-running workflow element, to restart the workflow.

The time-consuming event for which a trigger-based long-running workflow waits must return a `VC:Task` object. For example, the `startVM` action to start a virtual machine returns a `VC:Task` object, so that subsequent elements in a workflow can monitor its progress. A trigger-based long-running workflow's trigger event requires this `VC:Task` object as an input parameter.

You create a `Trigger` object in a JavaScript function in a **Scriptable Task** element. This **Scriptable Task** element can be part of the trigger-based long-running workflow that waits for the trigger event. Alternatively, it can be part of a different workflow that provides input parameters to the trigger-based long-running workflow. The trigger function must implement the `createEndOfTaskTrigger()` method from the Orchestrator API.

IMPORTANT You must define a timeout period for all triggers, otherwise the workflow can wait indefinitely.

Prerequisites

- Create a workflow.
- Open the workflow for editing in the workflow editor.
- Add some elements to the workflow schema.
- In the workflow, declare a `VC:Task` object as an attribute or input parameter, such as a `VC:Task` object from a workflow or workflow element that starts or clones a virtual machine.

Procedure

1. Drag a **Scriptable Task** element from the **Generic** menu into the schema of a workflow.

   One of the elements that precedes the **Scriptable Task** must generate a `VC:Task` object as its output parameter.

2. Click the **Edit** icon ( ) of the **Scriptable task** element in the workflow schema.
3. Provide a name and description for the trigger in **Info** properties tab.
4. Click the **IN** properties tab.
5. Click the **Bind to workflow parameter/attribute** icon ( ).

   The input parameter selection dialog box opens.
6. Select or create an input parameter of the type **VC:Task**.

   This `VC:Task` object represents the time-consuming event that another workflow or element launches.
7. (Optional) Select or create an input parameter of the Number type to define a timeout period in seconds.
8. Click the **OUT** properties tab.
9. Click the **Bind to workflow parameter/attribute** icon ( ).

   The output parameter selection dialog box opens.

10 Create an output parameter with the following properties.

   a Create the Name property with the value `trigger`.

   b Create the Type property with the value `Trigger`.

   c Click **Create ATTRIBUTE with same name** to create the attribute.

   d Leave the value as **Not set**.

11 Define any exception behavior in the **Exceptions** properties tab.

12 Define a function to generate a `Trigger` object in the **Scripting** tab.

   For example, you could create a `Trigger` object by implementing the following JavaScript function.

   ```
   trigger = task.createEndOfTaskTrigger(timeout);
   ```

   The `createEndOfTaskTrigger()` method returns a `Trigger` object that monitors a `VC:Task` object named `task`.

13 Click **Close**.

14 Click **Save** at the bottom of the workflow editor.

You defined a workflow element that creates a trigger event for a trigger-based long-running workflow. The trigger element generates a `Trigger` object as its output parameter, to which a **Waiting Event** element can bind.

What to do next

You must bind this trigger event to a **Waiting Event** element in a trigger-based long-running workflow.

Create a Trigger-Based Long-Running Workflow

If you know a workflow will have to wait for a response from an outside source during its run, but do not know how long that wait will last, you can implement it as a trigger-based long-running workflow. A trigger-based long-running workflow waits for a defined trigger event to occur before resuming.

You implement a workflow as a trigger-based long-running workflow by using the **Waiting Event** element. When the trigger-based long-running workflow arrives at the **Waiting Event** element, it will suspend its run and wait in a passive state until it receives a message from the trigger. During the waiting period, the passive workflow does not consume a thread, but rather the long-running workflow element passes the workflow information to the single thread that monitors all long-running workflows in the server.

Prerequisites

- Create a workflow.
- Open the workflow for editing in the workflow editor.
- Add some elements to the workflow schema.
- Define a trigger event that is encapsulated in a `Trigger` object.

Procedure

1 Drag a **Waiting Event** element from the **Generic** menu to the position in the workflow schema at which you want to suspend the workflow's run.

   The scriptable task that declares the trigger must immediately precede the **Waiting Event** element.

2 Click the **Edit** icon () of the **Waiting Event** element in the workflow schema.

3 Provide a description of the reason for the wait in the **Info** properties tab.

4 Click the **Attributes** properties tab.

The `trigger.ref` parameter appears in the list of attributes.

5 Click the `trigger.ref` parameter's **Not set** link to bind the parameter to an appropriate `Trigger` object.

The **Waiting Event** selection dialog box opens, presenting a list of possible parameters to which to bind.

6 Select a predefined `Trigger` object from the proposed list.

This `Trigger` object represents a trigger event that another workflow or workflow element defines.

7 Define any exception behavior in the **Exceptions** properties tab.

8 Click **Close**.

9 Click **Save** at the bottom of the workflow editor.

You defined a workflow element that suspends a trigger-based long-running workflow, that waits for a specific trigger event before restarting.

What to do next

You can run a workflow.

Configuration Elements

A configuration element is a list of attributes you can use to configure constants across a whole Orchestrator server deployment.

All the workflows, actions, policies, and Web views running in a particular Orchestrator server can use the attributes you set in a configuration element. Setting attributes in configuration elements lets you make the same attribute values available to all the workflows, actions, policies, and Web views running in the Orchestrator server.

If you create a package containing a workflow, action, policy, or Web view that uses an attribute from a configuration element, Orchestrator automatically includes the configuration element in the package. If you import a package containing a configuration element into another Orchestrator server, you can import the configuration element attribute values as well. For example, if you create a workflow that requires attribute values that depend on the Orchestrator server on which it runs, setting those attributes in a configuration element lets you to export that workflow so that another Orchestrator server can use it. Configuration elements therefore allow you to exchange workflows, actions, policies, and Web views between servers more easily.

NOTE You cannot import values of a configuration element attribute from a configuration element exported from Orchestrator 5.1 or earlier.

Create a Configuration Element

Configuration elements allow you to set common attributes across an Orchestrator server. All elements that are running in the server can call on the attributes you set in a configuration element. Creating configuration elements allows you to define common attributes once in the server, rather than individually in each element.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Design**.

2 Click the **Configurations** view.

3 Right-click a folder in the hierarchical list of folders and select **New folder** to create a new folder.

4 Provide a name for the folder and click **Ok**.

5 Right-click the folder you created and select **New element**.

6 Provide a name for the configuration element and click **Ok**.

The configuration element editor opens.

7 Increment the version number by clicking the version digits in the **General** tab and providing a version comment.

8 Provide a description of the configuration element in the **Description** text box in the **General** tab.

9 Click the **Attributes** tab.

10 Click the **Add attribute** icon ( ) to create a new attribute.

11 Click the attribute values under **Name**, **Type**, **Value**, and **Description** to set the attribute name, type, value, and description.

12 Click the **Permissions** tab.

13 Click the **Add access rights** icon ( ) to grant permission to access this configuration element to a group of users.

14 Search for a user group in the **Filter** text box and select the relevant user group from the proposed list.

15 Check the appropriate check boxes to set the access rights for the selected user group.

You can set the following permissions on the configuration element.

Permission	Description
View	Users can view the configuration element, but cannot view the schemas or scripting.
Inspect	Users can view the configuration element, including the schemas and scripting.
Admin	Users can set permissions on the elements in the configuration element and have all other permissions.
Execute	Users can run the elements in the configuration element.
Edit	Users can edit the elements in the configuration element.

16 Click **Select**.

17 Click **Save and close** to exit the configuration element editor.

You defined a configuration element that sets common attributes across an Orchestrator server.

What to do next

You can use the configuration element to provide attributes to workflows or actions.

Workflow User Permissions

Orchestrator defines levels of permissions that you can apply to groups to allow or deny them access to workflows.

View The user can view the elements in the workflow, but cannot view the schema or scripting.

Inspect The user can view the elements in the workflow, including the schema and scripting.

Execute The user can run the workflow.

Edit	The user can edit the workflow.
Admin	The user can set permissions on the workflow and has all other permissions.

The **Admin** permission includes the **View**, **Inspect**, **Edit**, and **Execute** permissions. All the permissions require the **View** permission.

If you do not set any permissions on a workflow, the workflow inherits the permissions from the folder that contains it. If you do set permissions on a workflow, those permissions override the permissions of the folder that contains it, even if the permissions of the folder are more restrictive.

Set User Permissions on a Workflow

You set levels of permission on a workflow to limit the access that user groups can have to that workflow.

You can select the users and user groups for which to set permissions from the Orchestrator LDAP server.

Prerequisites

- Create a workflow.
- Open the workflow for editing in the workflow editor.
- Add some elements to the workflow schema.

Procedure

1. Click the **Permissions** tab.
2. Click the **Add access rights** icon () to define permissions for a new user group.
3. Search for a user group.

   The search results contain all the user groups from the Orchestrator LDAP server that match the search.
4. Select a user group and select the appropriate check boxes to set the level of permissions for this user group.

   To allow a user from this user group to view the workflow, inspect the schema and scripting, run and edit the workflow, and change the permissions, you must select all check boxes.
5. Click **Select**.

   The user group appears in the permissions list.
6. Click **Save and close** to exit the editor.

Validating Workflows

Orchestrator provides a workflow validation tool. Validating a workflow helps identify errors in the workflow and checks that the data flows from one element to the next correctly.

When you validate a workflow, the validation tool creates a list of any errors or warnings. Clicking an error in the list highlights the workflow element that contains the error.

If you run the validation tool in the workflow editor, the tool provides suggested quick fixes for the errors it detects. Some quick fixes require you to provide additional information or input parameters. Other quick fixes resolve the error for you.

Workflow validation checks the data bindings and connections between elements. Workflow validation does not check the data processing that each element in the workflow performs. Consequently, a valid workflow can run incorrectly and produce erroneous results if a function in a schema element is incorrect.

By default, Orchestrator always performs workflow validation when you run a workflow. You can change the default validation behavior in the Orchestrator client. See "Testing Workflows During Development," on page 15. For example, sometimes during workflow development you might want to run a workflow that you know to be invalid, for testing purposes.

Validate a Workflow and Fix Validation Errors

You must validate a workflow before you can run it. You can validate workflows in either the Orchestrator client or in the workflow editor. However, you can only fix validation errors if you have opened the workflow for editing in the workflow editor.

Prerequisites

Verify that you have a complete workflow to validate, with schema elements linked and bindings defined.

Procedure

1 Click the **Workflows** view.

2 Navigate to a workflow in the **Workflows** hierarchical list.

3 (Optional) Right-click the workflow and select **Validate workflow**.

If the workflow is valid, a confirmation message appears. If the workflow is invalid, a list of errors appears.

4 (Optional) Close the Workflow Validation dialog box.

5 Right-click the workflow and select **Edit** to open the workflow editor.

6 Click the **Schema** tab.

7 Click the **Validate** button in the **Schema** tab toolbar.

If the workflow is valid, a confirmation message appears. If the workflow is invalid, a list of errors appears.

8 For an invalid workflow, click an error message.

The validation tool highlights the schema element in which the error occurs by adding a red icon to it. Where possible, the validation tool displays a quick fix action.

- If you agree with the proposed quick fix action, click it to perform that action.
- If you disagree with the proposed quick fix action, close the Workflow Validation dialog box and fix the schema element manually.

IMPORTANT Always check that the fix that Orchestrator proposes is appropriate.

For example, the proposed action might be to delete an unused attribute, when in fact that attribute might not be bound correctly.

9 Repeat the preceding steps until you have eliminated all validation errors.

You validated a workflow and fixed the validation errors.

What to do next

You can run the workflow.

Debugging Workflows

Orchestrator provides a workflow debugging tool. You can debug a workflow to inspect the input and output parameters and attributes at the start of any activity, replace parameter or attribute values during a workflow run in edit mode, and resume a workflow from the last failed activity.

You can debug workflows from the standard workflow library and custom workflows. You can debug custom workflows while developing them in the workflow editor.

Debug a Workflow

You can debug elements of a workflow by adding breakpoints to the elements in the workflow schema.

When a breakpoint is reached, you have several options to continue the debugging process. When you debug an element from the workflow schema, you can view general information about the workflow run, modify the workflow variables, and view log messages.

Prerequisites

Log in to the Orchestrator client as a user who can run workflows.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Design**.

2 Click the **Workflows** view.

3 Select a workflow from the workflow library and click the **Schema** tab.

4 To add breakpoints to the schema elements that you want to debug, right-click a workflow element and select **Toggle breakpoint**.

You can enable or disable the toggled breakpoints.

5 Click the **Debug workflow** icon ().

If the workflow requires input parameters, you must provide them.

6 When the workflow run is paused after it reaches a breakpoint, select one of the available options.

Option	Description
 **Resume**	Resumes the workflow run until another breakpoint is reached.
 **Step into**	Lets you step into a workflow element. NOTE You cannot step into a nested workflow element when you debug a workflow in the workflow editor.
 **Step over**	Steps over the current element in the schema and pauses the workflow run on the next element.
 **Step return**	Exits the workflow element that you have stepped into.

7 (Optional) From the **Breakpoints** tab, modify the breakpoints.

You can enable, disable, or remove existing breakpoints.

8 (Optional) From the **Variables** tab, review the variables.

You can modify the values of some of the variables during the debugging process.

Example Workflow Debugging

You can debug a workflow from the standard workflow library.

For example, if you provide an incorrect recipient address, you can correct the value when you debug the Example interaction with email workflow.

Prerequisites

- Configure the Mail plug-in in the Orchestrator configuration interface.
- Log in to the Orchestrator client as a user who can run Mail workflows.

Procedure

1. From the drop-down menu in the Orchestrator client, select **Design**.
2. Click the **Workflows** view.
3. In the workflows hierarchical list, open **Library > Mail**.
4. Select the Example interaction with email workflow and click the **Schema** tab.
5. Right-click the **Email Send (Interaction)** workflow element and select **Toggle breakpoint**.
6. Click the **Debug workflow** icon ( ).
7. Provide the required information.
   a. In the **Destination address** text box, type an incomplete recipient address.

      For example, ***name@company.c***.
   b. Select an LDAP group of users who are authorized to answer the query.
   c. Click **Submit**.
8. When the breakpoint is reached, click the **Step into** icon ( ).
9. On the **Variables** tab, verify the values.
10. In the **toAddress** text box, type the correct recipient address value.

    For example, ***name@company.com***.
11. Click the **Resume** icon ( ) to continue the workflow run.

The workflow uses the value that you provided during the debugging process and continues the workflow run.

Running Workflows

An Orchestrator workflow runs according to a logical flow of events.

When you run a workflow, each schema element in the workflow runs according to the following sequence.

1. The workflow binds the workflow token attributes and input parameters to the schema element's input parameters.
2. The schema element runs.
3. The schema element's output parameters are copied to the workflow token attributes and workflow output parameters.
4. The workflow token attributes and output parameters are stored in the database.

5 The next schema element starts running.

This sequence repeats for each schema element until the end of the workflow.

Workflow Token Check Points

When a workflow runs, each schema element is a check point. After each schema element runs, Orchestrator stores workflow token attributes in the database, and the next schema element starts running. If the workflow stops unexpectedly, the next time the Orchestrator server restarts, the currently active schema element runs again, and the workflow continues from the start of the schema element that was running when the interruption occurred. However, Orchestrator does not implement transaction management or a rollback function.

End of Workflow

The workflow ends if the current active schema element is an end element. After the workflow reaches an end element, other workflows or applications can use the workflow's output parameters.

Run a Workflow in the Workflow Editor

You can run a workflow while you are developing it.

Running a workflow in the workflow editor lets you verify that the workflow runs correctly without interrupting the development process. You can view log messages that provide information about the workflow run. If the workflow run returns unexpected results, you can modify the workflow and run it again without closing the workflow editor.

Prerequisites

- Create a workflow.
- Open the workflow for editing in the workflow editor.
- Validate the workflow.

Procedure

1 Click the **Schema** tab.

2 Click **Run**.

3 (Optional) Review the messages in the **Logs** tab.

Run a Workflow

You can perform automated operations in vCenter Server by running workflows from the standard library or workflows that you create.

For example, you can create a virtual machine by running the Create simple virtual machine workflow.

Prerequisites

Verify that you have configured the vCenter Server plug-in. For details, see *Installing and Configuring VMware vCenter Orchestrator*.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Run** or **Design**.

2 Click the **Workflows** view.

3 In the workflows hierarchical list, open **Library > vCenter > Virtual machine management > Basic** to navigate to the Create simple virtual machine workflow.

4 Right-click the Create simple virtual machine workflow and select **Start workflow**.

5 Provide the following information into the **Start workflow** input parameters dialog box to create a virtual machine in a vCenter Server connected to Orchestrator.

Option	Action
Virtual machine name	Name the virtual machine `orchestrator-test`.
Virtual machine folder	a Click **Not set** for the **Virtual machine folder** value. b Select a virtual machine folder from the inventory. The **Select** button is inactive until you select an object of the correct type, in this case, `VC:VmFolder`.
Size of the new disk in GB	Type an appropriate numeric value.
Memory size in MB	Type an appropriate numeric value.
Number of virtual CPUs	Select an appropriate number of CPUs from the **Number of virtual CPUs** drop-down menu.
Virtual machine guest OS	Click the **Not Set** link and select a guest operating system from the list.
Host on which to create the virtual machine	Click **Not set** for the **Host on which to create the virtual machine** value and navigate through the vCenter Server infrastructure hierarchy to a host machine.
Resource pool	Click **Not set** for the **Resource pool** value and navigate through the vCenter Server infrastructure hierarchy to a resource pool.
The network to connect to	Click **Not set** for the **The network to connect to** value and select a network. Press Enter in the **Filter** text box to see all the available networks.
Datastore in which to store the virtual machine files	Click **Not set** for the **Datastore in which to store the virtual machine** value and navigate through the vCenter Server infrastructure hierarchy to a datastore.

6 Click **Submit** to run the workflow.

A workflow token appears under the Create simple virtual machine workflow, showing the workflow running icon.

7 Click the workflow token to view the status of the workflow as it runs.

8 Click the **Events** tab in the workflow token view to follow the progress of the workflow token until it completes.

9 Click the **Inventory** view.

10 Navigate through the vCenter Server infrastructure hierarchy to the resource pool you defined.

If the virtual machine does not appear in the list, click the refresh button to reload the inventory.

The `orchestrator-test` virtual machine is present in the resource pool.

11 (Optional) Right-click the `orchestrator-test` virtual machine in the **Inventory** view to see a contextual list of the workflows that you can run on the `orchestrator-test` virtual machine.

The Create simple virtual machine workflow ran successfully.

What to do next

You can log in vSphere Client and manage the new virtual machine.

Resuming a Failed Workflow Run

If a workflow fails, Orchestrator provides an option to resume the workflow run from the last failed activity.

You can change the parameters of the workflow and attempt to resume it, or retain the parameters and make changes to external components that affect the workflow run. For example, if a workflow run fails due to a problem in a third-party system, you can make changes to the system and resume the workflow run from the failed activity, without changing the workflow parameters and without repeating the successful activities.

Set the Behavior for Resuming a Failed Workflow Run

You can set the behavior for resuming a failed run for each custom workflow. The default workflows in the library use the default system setting for resuming a failed workflow run.

You can change the default system behavior by modifying a configuration file. See "Set Custom Properties for Resuming Failed Workflow Runs," on page 79.

Prerequisites

Verify that you have permissions to edit the workflow.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Design**.

2 Click the **Workflows** view.

3 Expand the workflows hierarchical list to navigate to the workflow for which you want to set the behavior.

4 Right-click the workflow and select **Edit**.

The workflow editor opens.

5 On the **General** tab, select an option from the **Resume from failed behavior** drop-down menu.

Option	Description
System default	Follows the default behavior.
Enabled	If a workflow run fails, a pop-up window displays an option to resume the workflow run.
Disabled	If a workflow run fails, it cannot be resumed.

6 Click **Save and close**.

Set Custom Properties for Resuming Failed Workflow Runs

By default, Orchestrator is not set up to resume failed workflow runs. You can enable Orchestrator to resume failed workflow runs and set a custom timeout period after which failed workflow runs cannot be resumed.

Procedure

1 On the Orchestrator server system, navigate to the folder that contains configuration files.

Option	Action
If you installed Orchestrator with the vCenter Server installer	Go to *install_directory*\VMware\Infrastructure\Orchestrator\app-server\conf.
If you installed the standalone version of Orchestrator	Go to *install_directory*\VMware\Orchestrator\app-server\conf.
If you downloaded and deployed the virtual appliance	Go to /etc/vco/app-server/.

2 Open the `vmo.properties` configuration file in a text editor.

3 Set Orchestrator to resume failed workflow runs by editing the following line in the `vmo.properties` file.

```
com.vmware.vco.engine.execute.resume-from-failed=true
```

4 Set a custom timeout period for resuming failed workflow runs by editing the following line in the `vmo.properties` file.

```
com.vmware.vco.engine.execute.resume-from-failed.timeout-sec=<seconds>
```

The value you set overrides the default timeout setting of 86400 seconds.

5 Save the `vmo.properties` file.

6 Restart the Orchestrator server.

Resume a Failed Workflow Run

You can resume a workflow run from the last failed activity, if resuming a failed run is enabled for the workflow.

When the option for resuming a failed workflow run is enabled, you can change the parameters of the workflow and try to resume it by using the options in the pop-up window that appears after the workflow fails. You can also retain the parameters and make changes to external components that affect the workflow run. If you do not select an option, the workflow run times out and cannot be resumed. For modifying the timeout period, see "Set Custom Properties for Resuming Failed Workflow Runs," on page 79.

Procedure

1 From the drop-down menu in the pop-up window, select **Resume** and click **Next**.

If you select **Cancel**, the workflow run cannot be resumed later.

2 (Optional) Modify the workflow parameters.

3 Click **Submit**.

Generate Workflow Documentation

You can export documentation in PDF format about a workflow or a workflow folder that you select at any time.

The exported document contains detailed information about the selected workflow or the workflows in the folder. The information about each workflow includes name, version history of the workflow, attributes, parameter presentation, workflow schema, and workflow actions. In addition, the documentation also provides the source code for the used actions.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Run** or **Design**.

2 Click the **Workflows** view.

3 Navigate to the workflow or workflow folder for which you want to generate documentation and right-click it.

4 Select **Generate documentation**.

5 Browse to locate the folder in which to save the PDF file, provide a file name, and click **Save**.

The PDF file containing the information about the selected workflow, or the workflows in the folder, is saved on your system.

Use Workflow Version History

You can use version history to revert a workflow to a previously saved state. You can revert the workflow state to an earlier or a later workflow version. You can also compare the differences between the current state of the workflow and a saved version of the workflow.

Orchestrator creates a new version history item for each workflow when you increase and save the workflow version. Subsequent changes to the workflow do not change the current saved version. For example, when you create a workflow version 1.0.0 and save it, the state of the workflow is stored in the version history. If you make any changes to the workflow, you can save the workflow state in the Orchestrator client, but you cannot apply the changes to workflow version 1.0.0. To store the changes in the version history, you must create a subsequent workflow version and save it. The version history is kept in the database along with the workflow itself.

When you delete a workflow, Orchestrator marks the element as deleted in the database without deleting the version history of the element from the database. This way, you can restore deleted workflows. See "Restore Deleted Workflows," on page 81.

Prerequisites

Open a workflow for editing in the workflow editor.

Procedure

1 Click the **General** tab in the workflow editor and click **Show version history**.

2 Select a workflow version and click **Diff Against Current** to compare the differences.

A window displays the differences between the current workflow version and the selected workflow version.

3 Select a workflow version and click **Revert** to restore the state of the workflow.

CAUTION If you have not saved the current workflow version, it is deleted from the version history and you cannot revert back to the current version.

The workflow state is reverted to the state of the selected version.

Restore Deleted Workflows

You can restore workflows that have been deleted from the workflow library.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Run** or **Design**.

2 Click the **Workflows** view.

3 Navigate to the workflow folder in which you want to restore deleted workflows.

4 Right-click the folder and select **Restore deleted workflows**.

5 Select the workflow or workflows that you want to restore and click **Restore**.

The restored workflows appear in the selected folder.

Develop a Simple Example Workflow

Developing a simple example workflow demonstrates the most common steps in the workflow development process.

The example workflow that you are about to create starts an existing virtual machine in vCenter Server and sends an email to the administrator to confirm that the virtual machine has started.

The example workflow performs the following tasks:

1 Prompts the user to select a virtual machine to start.

2 Prompts the user for an email address to which it can send notifications.

3 Checks whether the selected virtual machine is already powered on.

4 Sends a request to the vCenter Server instance to start the virtual machine.

5 Waits for vCenter Server to start the virtual machine, and returns an error if the virtual machine fails to start or if starting the virtual machine takes too long.

6 Waits for vCenter Server to start VMware Tools on the virtual machine, and returns an error if the virtual machine fails to start or if starting VMware Tools takes too long.

7 Verifies that the virtual machine is running.

8 Sends a notification to the provided email address, informing that the machine has started or that an error occurred.

The ZIP file of Orchestrator examples available for download from the landing page of the Orchestrator documentation contains a complete version of the Start VM and Send Email workflow.

The process for developing the example workflow consists of several tasks.

Prerequisites

Before you attempt to develop the simple example workflow, read "Key Concepts of Workflows," on page 13.

Procedure

1 Create the Simple Workflow Example on page 83

You must begin the workflow development process by creating the workflow in the Orchestrator client.

2 Create the Schema of the Simple Workflow Example on page 84

You can create a workflow's schema in the workflow editor. The workflow schema contains the elements that the workflow runs and determines the logical flow of the workflow.

3 (Optional) Create the Simple Workflow Example Zones on page 86

You can emphasize different zones in workflow by adding workflow notes of different colors. Creating different workflow zones helps to make complicated workflow schema easier to read and understand.

4 Define the Parameters of the Simple Workflow Example on page 87

In this phase of workflow development, you define the input parameters that the workflow requires to run. For the example workflow, you need an input parameter for the virtual machine to power on, and a parameter for the email address of the person to inform about the result of the operation. When users run the workflow, they will be required to specify the virtual machine to power on and an email address.

5 Define the Simple Workflow Example Decision Bindings on page 88

You bind a workflow's elements together in the **Schema** tab of the workflow editor. Decision bindings define how decision elements compare the input parameters received to the decision statement, and generate output parameters according to whether the input parameters match the decision statement.

6 Bind the Action Elements of the Simple Workflow Example on page 89

You can bind a workflow's elements together in the workflow editor. Bindings define how the action elements process input parameters and generate output parameters.

7 Bind the Simple Workflow Example Scripted Task Elements on page 92

You bind a workflow's elements together in the **Schema** tab of the workflow editor. Bindings define how the scripted task elements process input parameters and generate output parameters. You also bind the scriptable task elements to their JavaScript functions.

8 Define the Simple Workflow Example Exception Bindings on page 99

You define exception bindings in the **Schema** tab in the workflow editor. Exception bindings define how elements process errors.

9 Set the Read-Write Properties for Attributes of the Simple Workflow Example on page 100

You can define whether parameters and attributes are read-only constants or writeable variables. You can also set limitations on the values that users can provide for input parameters.

10 Set the Simple Workflow Example Parameter Properties on page 100

You can set the parameter properties in the workflow editor. Setting the parameter properties affects the behavior of the parameter, and places constraints on the possible values for that parameter.

11 Set the Layout of the Simple Workflow Example Input Parameters Dialog Box on page 102

You create the layout or presentation of the input parameters dialog box in the workflow editor. The input parameters dialog box opens when users run a workflow that needs input parameters to run.

12 Validate and Run the Simple Workflow Example on page 103

After you create a workflow, you can validate it to discover any possible errors. If the workflow contains no errors, you can run it.

Create the Simple Workflow Example

You must begin the workflow development process by creating the workflow in the Orchestrator client.

Prerequisites

Verify that the following components are installed and configured on the system.

- vCenter Server, controlling some virtual machines, at least one of which is powered off
- Access to an SMTP server
- A valid email address

For information about how to install and configure vCenter Server, see the *vSphere Installation and Setup* documentation. For information about how to configure Orchestrator to use an SMTP server, see *Installing and Configuring VMware vCenter Orchestrator*.

To write a workflow, you must have an Orchestrator user account with at least **View**, **Execute**, **Inspect**, **Edit**, and preferably **Admin** permissions on the server or on the workflow folder in which you are working.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Design**.

2 Click the **Workflows** view.

3 Right-click the root of the workflows list and select **Add folder**.

4 Name the new folder `Workflow Examples` and click **OK**.

5 Right-click the **Workflow Examples** folder and select **New workflow**.

6 Name the new workflow `Start VM and Send Email` and click **OK**.

The workflow editor opens.

7 In the **General** tab, click the version number digits to increment the version number.

Because this is the initial creation of the workflow, set the version to **0.0.1**.

8 Click the **Server restart behavior** value in the **General** tab to set whether the workflow resumes after a server restart.

9 Type a description of what the workflow does in the **Description** text box in the **General** tab.

For example, you can add the following description.

```
This workflow starts a virtual machine and sends a confirmation email to the Orchestrator
administrator.
```

10 Click **Save** at the bottom of the **General** tab.

You created a workflow called Start VM and Send Email, but you did not define its functions.

What to do next

Create the workflow's schema.

Create the Schema of the Simple Workflow Example

You can create a workflow's schema in the workflow editor. The workflow schema contains the elements that the workflow runs and determines the logical flow of the workflow.

Prerequisites

Complete the following tasks.

- "Create the Simple Workflow Example," on page 83.
- Open the workflow for editing in the workflow editor.

Procedure

1 Click the **Schema** tab in the workflow editor.

2 From the **Generic** menu, drag a decision element to the arrow that links the Start element and the `End` element in the schema.

3 Double-click the decision element and change its name to **`VM powered on?`**.

The decision element corresponds to a boolean function that checks whether the virtual machine is already powered on.

4 From the **Generic** menu, drag an action element to the red arrow that links the decision element and an `End` element.

The dialog box for action selection appears.

5 Type **`start`** in the **Filter** text box, select the **`startVM`** action from the filtered list of actions, and click **Select**.

6 Drag the following action elements, one after the other, to the blue arrow that links the `startVM` action element to an `End` element.

`vim3WaitTaskEnd`	Suspends the workflow run and pings an ongoing vCenter Server task at regular intervals, until that task is finished. The `startVM` action starts a virtual machine and the `vim3WaitTaskEnd` action makes the workflow wait while the virtual machine starts up. After the virtual machine starts, the `vim3WaitTaskEnd` lets the workflow resume.
`vim3WaitToolsStarted`	Suspends the workflow run and waits until VMware Tools starts on the target virtual machine.

7 From the **Generic** menu, drag a scriptable task element to the blue arrow that links the `vim3WaitToolsStarted` action element to an `End` element.

8 Double-click the scriptable task element and rename it to **`OK`**.

9 Drag another scriptable task element to the green arrow that links the `VM powered on?` decision element to an `End` element , and name this scriptable task element **`Already started`**.

10 Modify the linking of the `Already started` scriptable task element.

a Drag the `Already started` scriptable task element to the left of the `startVM` action element.

b Delete the blue arrow that connects the `Already started` scriptable task element to an `End` element.

c Link the `Already started` scriptable task element to the `vim3WaitToolsStarted` action element with a blue arrow.

11 From the **Generic** menu, drag the following scriptable task elements into the schema.

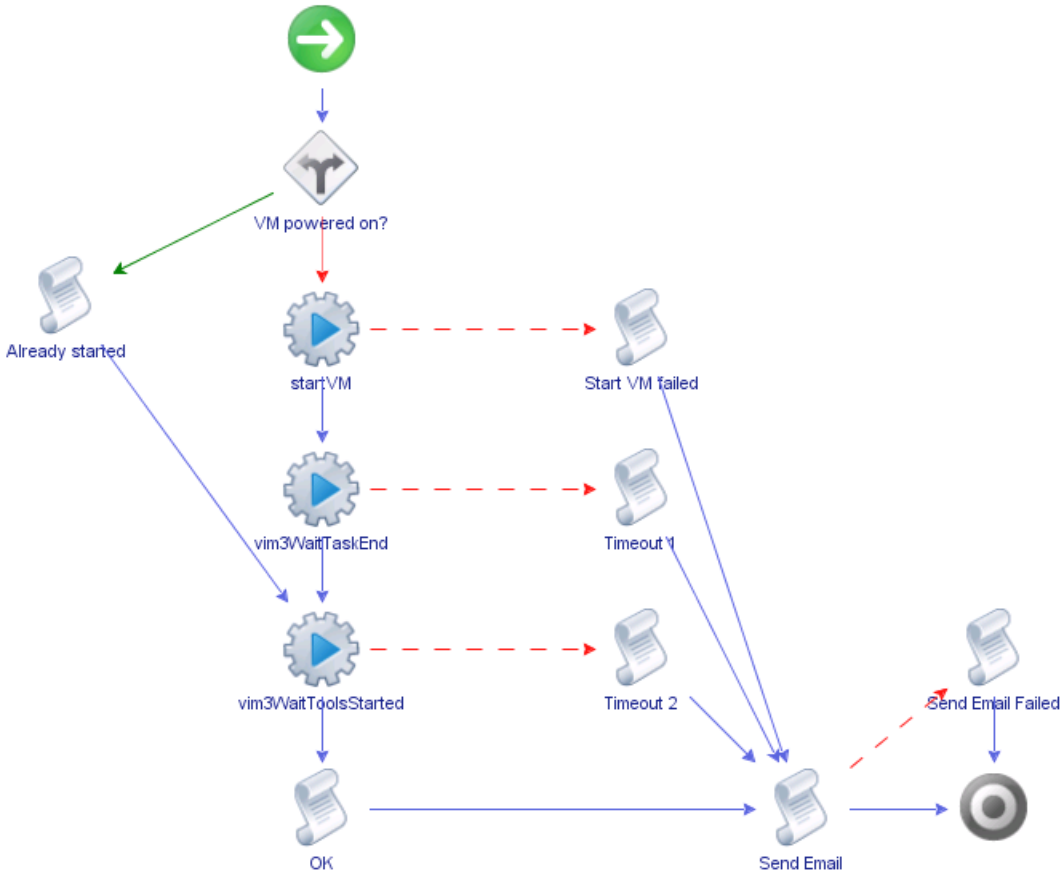
- Drag a scriptable task element to the `startVM` action element and name the scriptable task element **`Start VM Failed`**.
- Drag a scriptable task element to the `vim3WaitTaskEnd` action element and name the scriptable task element **`Timeout 1`**.
- Drag a scriptable task element to the `vim3WaitToolsStarted` action element and name the scriptable task element **`Timeout 2`**.
- Drag a scriptable task element to the blue arrow that links the `OK` scriptable task element to an `End` element, name the new scriptable task element **`Send Email`**, and drag it to the right of the `OK` scriptable task element.
- Link the `Start VM Failed`, `Timeout 1`, and `Timeout 2` scriptable task elements to the `Send Email` scriptable task element with blue arrows.
- Drag a scriptable task element to the `Send Email` scriptable task element, name the new scriptable task element **`Send Email Failed`**, drag it to the right of the `Timeout 2` scriptable task element, and link it to the `End` element with a blue arrow.

12 Drag the `End` element to the right of the `Send Email` scriptable task element.

13 Click **Save** at the bottom of the **Schema** tab.

The following figure shows the layout of the Start VM and Send Email workflow schema elements.

Figure 1-3. Linking the Elements of the Start VM and Send Email Example Workflow



What to do next

You can highlight different zones in the workflow.

(Optional) Create the Simple Workflow Example Zones

You can emphasize different zones in workflow by adding workflow notes of different colors. Creating different workflow zones helps to make complicated workflow schema easier to read and understand.

Prerequisites

Complete the following tasks.

- "Create the Simple Workflow Example," on page 83.
- "Create the Schema of the Simple Workflow Example," on page 84.
- Open the workflow for editing in the workflow editor.

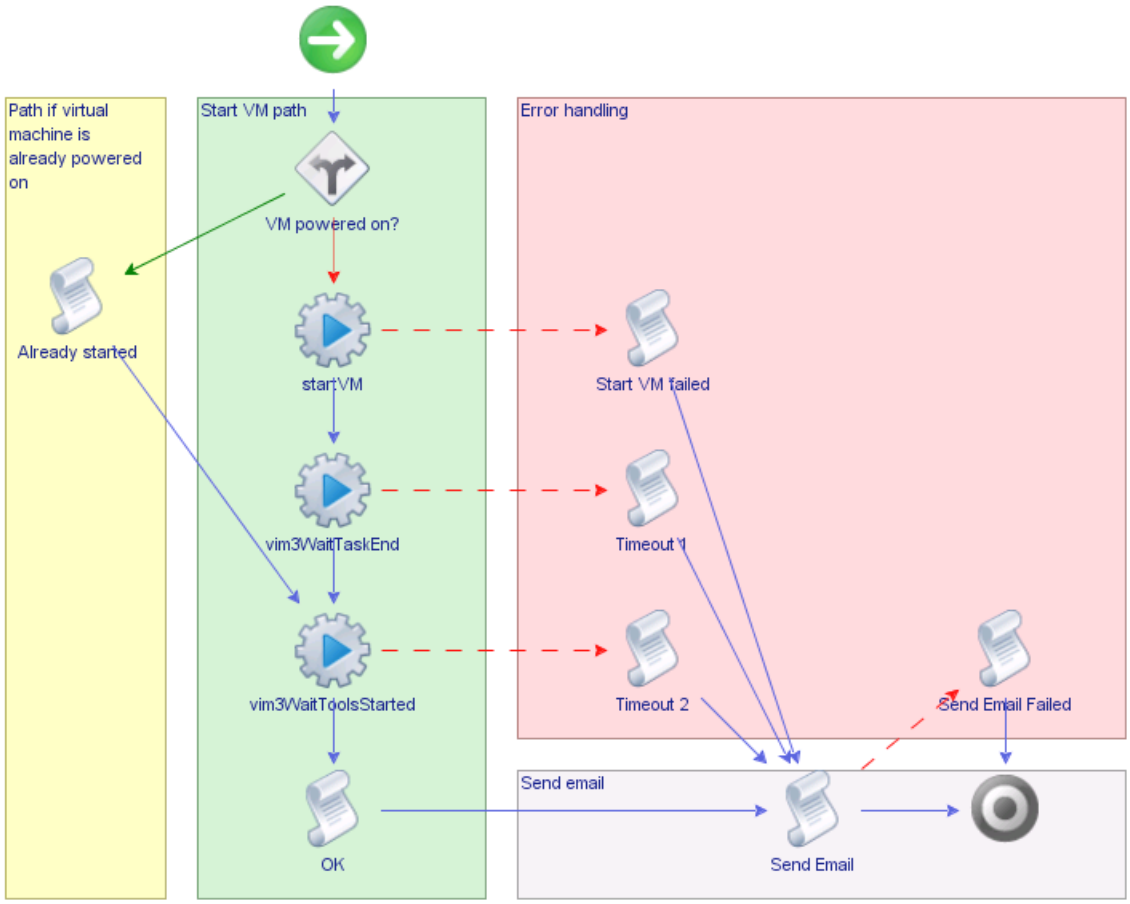
Procedure

1. Drag a workflow note element from the **Generic** menu into the workflow editor.
2. Position the workflow note over the `Already started` scriptable task element.
3. Drag the edges of the workflow note to resize it so that it surrounds the `Already started` scriptable task element.
4. Double-click the text and add a description.

   For example, **`Path if virtual machine is already powered on`**.
5. Press Ctrl+E to select the background color.
6. Repeat the preceding steps to highlight other zones in the workflow.
   - Place a note over the vertical sequence of elements from the `VM powered on?` decision element to the `OK` element. Add the description **`Start VM path`**.
   - Place a note over the `startVM failed`, both `Timeout` scriptable task elements and the `Send Email Failed` scriptable task element. Add the description **`Error handling`**.
   - Place a note over the `Send Email` scriptable task element. Add the description **`Send email`**.

The following figure shows what the example workflow zones should look like.

Figure 1-4. Start VM and Send Email Example Workflow Zones



What to do next

You must define the workflow's attributes and input and output parameters.

Define the Parameters of the Simple Workflow Example

In this phase of workflow development, you define the input parameters that the workflow requires to run. For the example workflow, you need an input parameter for the virtual machine to power on, and a parameter for the email address of the person to inform about the result of the operation. When users run the workflow, they will be required to specify the virtual machine to power on and an email address.

Prerequisites

Complete the following tasks.

- "Create the Simple Workflow Example," on page 83.
- Open the workflow for editing in the workflow editor.

Procedure

1 Click the **Inputs** tab in the workflow editor.

2 Right-click within the **Inputs** tab and select **Add Parameter**.

A parameter named `arg_in_0` appears in the **Inputs** tab.

3 Click **`arg_in_0`**.

4 Type the name **`vm`** in the Choose Attribute Name dialog box and click **OK**.

5 Click the **Type** text box and type **`vc:virtualm`** in the search text box in the parameter type dialog box.

6 Select **`VC:VirtualMachine`** from the proposed list of parameter types and click **Accept**.

7 Add a description of the parameter in the **Description** text box.

For example, type **`The virtual machine to power on`**.

8 Repeat Step 2 through Step 7 to create a second input parameter, with the following values.

- Name: `toAddress`
- Type: String
- Description: **`The email address to send the result of this workflow to`**

9 Click **Save** at the bottom of the **Inputs** tab.

You defined the workflow's input parameters.

What to do next

Define the bindings between the element parameters.

Define the Simple Workflow Example Decision Bindings

You bind a workflow's elements together in the **Schema** tab of the workflow editor. Decision bindings define how decision elements compare the input parameters received to the decision statement, and generate output parameters according to whether the input parameters match the decision statement.

Prerequisites

Complete the following tasks.

- "Create the Simple Workflow Example," on page 83.
- "Create the Schema of the Simple Workflow Example," on page 84.
- "Define the Parameters of the Simple Workflow Example," on page 87.
- Open the workflow for editing in the workflow editor.

Procedure

1 On the **Schema** tab, click the **Edit** icon () of the **`VM Powered On?`** decision element.

2 On the **Decision** tab, click the **Not set (NULL)** button and select `vm` as the decision element's input parameter from the list of proposed parameters.

3 Select the **state equals** statement from the list of decision statements proposed in the drop-down menu.

A **Not set** button appears in the value text box, which presents you with a limited choice of possible values.

4 Select **`poweredOn`**.

5 Click **Save** at the bottom of the workflow editor's **Schema** tab.

You have defined the true or false statement against which the decision element will compare the value of the input parameter it receives.

What to do next

You must define the bindings for the other elements in the workflow.

Bind the Action Elements of the Simple Workflow Example

You can bind a workflow's elements together in the workflow editor. Bindings define how the action elements process input parameters and generate output parameters.

Prerequisites

Complete the following tasks.

- "Create the Simple Workflow Example," on page 83.
- "Create the Schema of the Simple Workflow Example," on page 84.
- "Define the Parameters of the Simple Workflow Example," on page 87.
- "Define the Simple Workflow Example Decision Bindings," on page 88.
- Open the workflow for editing in the workflow editor.

Procedure

1 On the **Schema** tab, click the **Edit** icon () of the `startVM` action element.

2 Set the following general information on the **Info** tab.

Option	Action
Interaction	Select **No External interaction**.
Business Status	Select the check box and add the text **`Sending start VM`**.
Description	Leave the text Start / Resume a VM. Return the start task.

3 Click the **IN** tab.

The **IN** tab displays the two possible input parameters available to the `startVM` action, `vm` and `host`.

Orchestrator automatically binds the `vm` parameter to `vm[in-parameter]` because the `startVM` action can only take a `VC:VirtualMachine` as an input parameter. Orchestrator detects the `vm` parameter you defined when you set the workflow input parameters and so binds it to the action automatically.

4 Set `host` to **NULL**.

This is an optional parameter, so you can set it to null. However, if you leave it set to **Not set**, the workflow cannot validate.

5 Click the **OUT** tab.

The default output parameter that all actions generate, `actionResult`, appears.

6 For the `actionResult` parameter, click **Not set**.

7 Click **Create parameter/attribute in workflow**.

The Parameter information dialog box displays the values that you can set for this output parameter. The output parameter type for the `startVM` action is a `VC:Task` object.

8 Name the parameter **`powerOnTask`** and provide a description.

For example, **`Contains the result of powering on a VM`**.

9 Click **Create workflow ATTRIBUTE with the same name** and click **OK** to exit the Parameter information dialog box.

10 Repeat the preceding steps to bind the input and output parameters to the `vim3WaitTaskEnd` and `vim3WaitToolsStarted` action elements.

"Simple Workflow Example Action Element Bindings," on page 90 lists the bindings for the `vim3WaitTaskEnd` and `vim3WaitToolsStarted` action elements.

11 Click **Save** at the bottom of the workflow editor's **Schema** tab.

The action elements' input and output parameters are bound to the appropriate parameter types and values.

What to do next

Bind the scriptable task elements and define their functions.

Simple Workflow Example Action Element Bindings

Bindings define how the simple workflow example's action elements process input and output parameters.

When defining bindings, Orchestrator presents parameters you have already defined in the workflow as candidates for binding. If you have not defined the required parameter in the workflow yet, the only parameter choice is `NULL`. Click **Create parameter/attribute in workflow** to create a new parameter.

vim3WaitTaskEnd Action

The `vim3WaitTaskEnd` action element declares constants to track the progress of a task and a polling rate. The following table shows the input and output parameter bindings that the `vim3WaitTaskEnd` action requires.

Table 1-7. Binding Values of the vim3WaitTaskEnd Action

Parameter Name	Binding Type	Bind to Existing or Create Parameter?	Binding Values
`task`	IN	Bind	■ Local Parameter: `powerOnTask` ■ Source parameter: `task[attribute]` ■ Type: `VC:Task` ■ Description: **`Contains the result of powering on a VM.`**
`progress`	IN	Create	■ Local Parameter: `progress` ■ Source parameter: `progress[attribute]` ■ Type: Boolean ■ Value: No (false) ■ Description: **`Log progress while waiting for the vCenter Server task to complete.`**

Table 1-7. Binding Values of the vim3WaitTaskEnd Action (Continued)

Parameter Name	Binding Type	Bind to Existing or Create Parameter?	Binding Values
`pollRate`	IN	Create	■ Local Parameter: `pollRate` ■ Source parameter: `pollRate[attribute]` ■ Type: number ■ Value: **2** ■ Description: **`Polling rate in seconds at which vim3WaitTaskEnd checks the advancement of the vCenter Server task.`**
`actionResult`	OUT	Create	■ Local Parameter: `actionResult[attribute]` ■ Source parameter: `returnedManagedObject[attribute]` ■ Type: Any ■ Description: **`The returned managed object from the waitTaskEnd action.`**

vim3WaitToolsStarted Action

The `vim3WaitToolsStarted` action element waits until VMware Tools have installed on a virtual machine, and defines a polling rate and a timeout period. The following table shows the input parameter bindings the `vim3WaitToolsStarted` action requires.

The `vim3WaitToolsStarted` action element has no output, so requires no output binding.

Table 1-8. Binding Values of the vim3WaitToolsStarted Action

Parameter Name	Binding Type	Bind to Existing or Create Parameter?	Binding Values
`vm`	IN	Automatic binding	■ Local Parameter: `vm` ■ Source parameter: `vm[in-parameter]` ■ Type: `VC:VirtualMachine` ■ Value: Not editable, variable is not a workflow attribute. ■ Description: **`The virtual machine to start.`**
`pollingRate`	IN	Bind	■ Local Parameter: `pollRate` ■ Source parameter: `pollRate[attribute]` ■ Type: number ■ Description: **`The polling rate in seconds at which vim3WaitTaskEnd checks the advancement of the vCenter server task.`**
`timeout`	IN	Create	■ Local Parameter: `timeout` ■ Source parameter: `timeout[attribute]` ■ Type: number ■ Value: **`10`** ■ Description: **`The timeout limit that vim3WaitToolsStarted waits before throwing an exception.`**

Bind the Simple Workflow Example Scripted Task Elements

You bind a workflow's elements together in the **Schema** tab of the workflow editor. Bindings define how the scripted task elements process input parameters and generate output parameters. You also bind the scriptable task elements to their JavaScript functions.

Prerequisites

Complete the following tasks.

- "Create the Simple Workflow Example," on page 83.
- "Create the Schema of the Simple Workflow Example," on page 84.
- "Define the Parameters of the Simple Workflow Example," on page 87.
- "Define the Simple Workflow Example Decision Bindings," on page 88.
- Open the workflow for editing in the workflow editor.

Procedure

1 On the **Schema** tab, click the **Edit** icon () of the `Already Started` scriptable task element.

2 Set the following general information in the **Info** tab.

Option	Action
Interaction	Select **No External interaction**.
Business Status	Select the check box and add the text **`VM already powered on`**.
Description	Leave the text The VM is already powered on, bypassing startVM and waitTaskEnd, checking if the VM tools are up and running..

3 Click the **IN** tab.

Because this is a custom scriptable task element, no properties are predefined for you.

4 Click the **Bind to workflow parameter/attribute** icon ().

5 Select `vm` from the proposed list of parameters.

6 Leave the **OUT** and **Exception** tabs blank.

This element does not generate an output parameter or exception.

7 Click the **Scripting** tab.

8 Add the following JavaScript function.

```
//Writes the following event in the vCO database
Server.log("VM '"+ vm.name +"' already started");
```

9 Repeat the preceding steps to bind the remaining input parameters to the other scriptable task elements.

"Simple Workflow Example Scriptable Task Element Bindings," on page 93 lists the bindings for the `Start VM failed`, both `Timeout` or `Error`, `Send Email Failed`, and the `OK` scriptable task elements.

10 Click **Save** at the bottom of the workflow editor's **Schema** tab.

You have bound the scriptable task elements to their input and output parameters and provided the scripting that defines their function.

What to do next

You must define the exception handling.

Simple Workflow Example Scriptable Task Element Bindings

Bindings define how the simple workflow example's scriptable task elements process input parameters. You also bind the scriptable task elements to their JavaScript functions.

When defining bindings, Orchestrator presents parameters you have already defined in the workflow as candidates for binding. If you have not defined the required parameter in the workflow yet, the only parameter choice is `NULL`. Click **Create parameter/attribute in workflow** to create a new parameter.

Start VM Failed Scriptable Task

The Start VM Failed scriptable task element handles any exceptions that the `startVM` action throws by setting the content of an email notification about the failure to start the virtual machine, and writing the event in the Orchestrator log.

The following table shows the input and output parameter bindings that the Start VM Failed scriptable task element requires.

Table 1-9. Bindings of the Start VM Failed Scriptable Task Element

Parameter Name	Binding Type	Bind to Existing or Create Parameter?	Binding Values
vm	IN	Bind	■ Local Parameter: vm ■ Source parameter: vm[in-parameter] ■ Type: VC:VirtualMachine ■ Description: **The virtual machine to power on.**
errorCode	IN	Create	■ Local Parameter: errorCode ■ Source parameter: errorCode[attribute] ■ Type: string ■ Description: **Catch any exceptions while powering on a VM.**
body	OUT	Create	■ Local Parameter: body ■ Source parameter: body[attribute] ■ Type: string ■ Description: **The email body**

The Start VM Failed scriptable task element performs the following scripted function.

```
body = "Unable to execute powerOnVM_Task() on VM '"+vm.name+"', exception found: "+errorCode;
//Writes the following event  in the vCO database
Server.error("Unable to execute powerOnVM_Task() on VM '"+vm.name, "Exception found:
"+errorCode);
```

Timeout 1 Scriptable Task Element

The Timeout 1 scriptable task element handles any exceptions that the vim3WaitTaskEnd action throws by setting the content of an email notification about the failure of the task, and writing the event in the Orchestrator log.

The following table shows the input and output parameter bindings that the Timeout 1 scriptable task element requires.

Table 1-10. Bindings of the Timeout 1 Scriptable Task Element

Parameter Name	Binding Type	Bind to Existing or Create Parameter?	Binding Values
vm	IN	Bind	■ Local Parameter: vm ■ Source parameter: vm[in-parameter] ■ Type: VC:VirtualMachine ■ Description: **The virtual machine to start.**
errorCode	IN	Bind	■ Local Parameter: errorCode ■ Source parameter: errorCode[attribute] ■ Type: string ■ Description: **Catch any exceptions while powering on a VM.**
body	OUT	Bind	■ Local Parameter: body ■ Source parameter: body[attribute] ■ Type: string ■ Description: **The email body**

The Timeout 1 scriptable task element requires the following scripted function.

```
body = "Error while waiting for poweredOnVM_Task() to complete on VM '"+vm.name+"', exception
found: "+errorCode;
//Writes the following event in the vCO database
Server.error("Error while waiting for poweredOnVM_Task() to complete on VM '"+vm.name,
"Exception found: "+errorCode);
```

Timeout 2 Scriptable Task Element

The Timeout 2 scriptable task element handles any exceptions that the vim3WaitToolsStarted action throws by setting the content of an email notification about the failure of the task, and writing the event in the Orchestrator log.

The following table shows the input and output parameter bindings that the Timeout 2 scriptable task element requires.

Table 1-11. Bindings of the Timeout 2 Scriptable Task Element

Parameter Name	Binding Type	Bind to Existing or Create Parameter?	Binding Values
vm	IN	Bind	■ Local Parameter: vm ■ Source parameter: vm[in–parameter] ■ Type: VC:VirtualMachine ■ Description: **The virtual machine to power on.**
errorCode	IN	Bind	■ Local Parameter: errorCode ■ Source parameter: errorCode[attribute] ■ Type: string ■ Description: **Catch any exceptions while powering on a VM.**
body	OUT	Bind	■ Local Parameter: body ■ Source parameter: body[attribute] ■ Type: string ■ Description: **The email body**

The Timeout 2 scriptable task element requires the following scripted function.

```
body = "Error while waiting for VMware tools to be up on VM '"+vm.name+"', exception found:
"+errorCode;
//Writes the following event in the vCO database
Server.error("Error while waiting for VMware tools to be up on VM '"+vm.name, "Exception found:
"+errorCode);
```

OK Scriptable Task Element

The OK scriptable task element receives notice that the virtual machine has started successfully, sets the content of an email notification about the successful start of the virtual machine, and writes the event in the Orchestrator log.

The following table shows the input and output parameter bindings that the OK scriptable task element requires.

Table 1-12. Bindings of the OK Scriptable Task Element

Parameter Name	Binding Type	Bind to Existing or Create Parameter?	Binding Values
vm	IN	Bind	■ Local Parameter: vm ■ Source parameter: vm[in–parameter] ■ Type: VC:VirtualMachine ■ Description: **The virtual machine to power on.**
body	OUT	Bind	■ Local Parameter: body ■ Source parameter: body[attribute] ■ Type: string ■ Description: **The email body**

The OK scriptable task element requires the following scripted function.

```
body = "The VM '"+vm.name+"' has started successfully and is ready for use";
//Writes the following event in the vCO database
Server.log(body);
```

Send Email Failed Scriptable Task Element

The Send Email Failed scriptable task element receives notice that the sending of the email failed, and writes the event in the Orchestrator log.

The following table shows the input parameter bindings that the Send Email Failed scriptable task element requires.

Table 1-13. Bindings of the Send Email Failed Scriptable Task Element

Parameter Name	Binding Type	Bind to Existing or Create Parameter?	Binding Values
vm	IN	Bind	■ Local Parameter: vm ■ Source parameter: vm[in-parameter] ■ Type: VC:VirtualMachine ■ Description: **The virtual machine to power on.**
toAddress	IN	Bind	■ Local Parameter: toAddress ■ Source parameter: toAddress[in-parameter] ■ Type: string ■ Description: **The email address of the person to inform of the result of this workflow**
emailErrorCode	IN	Create	■ Local Parameter: emailErrorCode ■ Source parameter: emailErrorCode[attrbute] ■ Type: string ■ Description: **Catch any exceptions while sending an email**

The Send Email Failed scriptable task element requires the following scripted function.

```
//Writes  the following event in the vCO database
Server.error("Couldn't send result email to '"+toAddress+"' for VM '"+vm.name, "Exception found:
"+emailErrorCode);
```

Send Email Scriptable Task Element

The purpose of the Start VM and Send Email workflow is to inform an administrator when it starts a virtual machine. To do so, you must define the scriptable task that sends an email. To send the email, the Send Email scriptable task element needs an SMTP server, addresses for the sender and recipient of the email, the email subject, and the email content.

The following table shows the input and output parameter bindings that the Send Email scriptable task element requires.

Table 1-14. Bindings of the Send Email Scriptable Task Element

Parameter Name	Binding Type	Bind to Existing or Create Parameter?	Binding Values
vm	IN	Bind	■ Local Parameter: vm ■ Source parameter: vm[in-parameter] ■ Type: VC:VirtualMachine ■ Description: **The virtual machine to power on.**
toAddress	IN	Bind	■ Local Parameter: toAddress ■ Source parameter: toAddress[in-parameter] ■ Type: string ■ Description: **The email address of the person to inform of the result of this workflow**
body	IN	Bind	■ Local Parameter: body ■ Source parameter: body[attribute] ■ Type: string ■ Description: **The email body**
smtpHost	IN	Create	■ Local Parameter: smtpHost ■ Source parameter: smtpHost[attribute] ■ Type: string ■ Description: **The email SMTP server**
fromAddress	IN	Create	■ Local Parameter: fromAddress ■ Source parameter: fromAddress[attribute] ■ Type: string ■ Description: **The email address of the sender**
subject	IN	Create	■ Local Parameter: subject ■ Source parameter: subject[attribute] ■ Type: string ■ Description: **The email subject**

The Send Email scriptable task element requires the following scripted function.

```
//Create an instance of EmailMessage
var myEmailMessage = new EmailMessage() ;

//Apply methods on this instance that populate the email message
myEmailMessage.smtpHost = smtpHost;
myEmailMessage.fromAddress = fromAddress;
myEmailMessage.toAddress = toAddress;
myEmailMessage.subject = subject;
myEmailMessage.addMimePart(body , "text/html");
```

```
//Apply the method that sends the email message
myEmailMessage.sendMessage();
System.log("Sent email to '"+toAddress+"'");
```

Define the Simple Workflow Example Exception Bindings

You define exception bindings in the **Schema** tab in the workflow editor. Exception bindings define how elements process errors.

The following elements in the workflow return exceptions: `startVM`, `vim3WaitTaskEnd`, `Send Email`, and `vim3WaitToolsStarted`.

Prerequisites

Complete the following tasks.

- "Create the Simple Workflow Example," on page 83.
- "Create the Schema of the Simple Workflow Example," on page 84.
- "Define the Parameters of the Simple Workflow Example," on page 87.
- "Define the Simple Workflow Example Decision Bindings," on page 88.
- "Bind the Action Elements of the Simple Workflow Example," on page 89.
- "Bind the Simple Workflow Example Scripted Task Elements," on page 92.
- Open the workflow for editing in the workflow editor.

Procedure

1. On the **Schema** tab, click the **Edit** icon ( ) of the **`startVM`** action element.
2. Click the **Exception** tab.
3. Click the **Not set** button.
4. Select **`errorCode`** from the proposed list.
5. Repeat the preceding steps to set the exception binding to `errorCode` for both `vim3WaitTaskEnd` and `vim3WaitToolsStarted`.
6. Click the **Edit** icon ( ) of the **`Send Email`** scriptable task element.
7. Click the **Exception** tab.
8. Click the **Not set** button.
9. Select **`emailErrorCode`** from the proposed list.
10. Click **Save** at the bottom of the workflow editor's **Schema** tab.

You have defined the exception binding for the elements that return exceptions.

What to do next

You must set the read and write properties on the attributes and parameters.

Set the Read-Write Properties for Attributes of the Simple Workflow Example

You can define whether parameters and attributes are read-only constants or writeable variables. You can also set limitations on the values that users can provide for input parameters.

Setting certain parameters to read-only allows other developers to adapt the workflow or to modify it without breaking the workflow's core function.

Prerequisites

Complete the following tasks.

- "Create the Simple Workflow Example," on page 83.
- "Create the Schema of the Simple Workflow Example," on page 84.
- "Define the Parameters of the Simple Workflow Example," on page 87.
- "Define the Simple Workflow Example Decision Bindings," on page 88.
- "Bind the Action Elements of the Simple Workflow Example," on page 89.
- "Bind the Simple Workflow Example Scripted Task Elements," on page 92.
- "Define the Simple Workflow Example Exception Bindings," on page 99.
- Open the workflow for editing in the workflow editor.

Procedure

1 Click the **General** tab at the top of the workflow editor.

Under **Attributes** is a list of all the defined attributes, with check boxes next to each attribute. When you select these check boxes, you set attributes as read-only.

2 Select the check boxes to make the following attributes read-only constants:

- progress
- pollRate
- timeout
- smtpHost
- fromAddress
- subject

You have defined which of the workflow's attributes are constants and which are variables.

What to do next

Set the parameter properties and place constraints on the possible values for that parameter.

Set the Simple Workflow Example Parameter Properties

You can set the parameter properties in the workflow editor. Setting the parameter properties affects the behavior of the parameter, and places constraints on the possible values for that parameter.

Prerequisites

Complete the following tasks.

- "Create the Simple Workflow Example," on page 83.
- "Create the Schema of the Simple Workflow Example," on page 84.

- "Define the Parameters of the Simple Workflow Example," on page 87.
- "Define the Simple Workflow Example Decision Bindings," on page 88.
- "Bind the Action Elements of the Simple Workflow Example," on page 89.
- "Bind the Simple Workflow Example Scripted Task Elements," on page 92.
- "Define the Simple Workflow Example Exception Bindings," on page 99.
- Open the workflow for editing in the workflow editor.

Procedure

1 Click the **Presentation** tab in the workflow editor.

The two input parameters you defined for this workflow are listed.

2 Click the **`(VC:VirtualMachine)vm`** parameter.

3 Add a description in the **General** tab in the bottom half of the screen.

For example, type **`The virtual machine to start`**.

4 Click the **Properties** tab in the bottom half of the screen.

On this tab, you can set the properties for the `(VC:VirtualMachine)vm` parameter.

5 Click the **Add property** icon ( ).

6 From the list of proposed properties, select the **Mandatory input** property, click **Ok**, and set its value to **Yes**.

When you enable this property, users cannot run the Start VM and Send Email workflow without providing a virtual machine to start.

7 Click the **Add property** icon ( ).

8 From the list of proposed properties, select **Select value as**, click **Ok**, and select **list** from the list of possible values.

When you set this property, you set how the user selects the value of the `(VC:VirtualMachine)vm` input parameter.

9 Click the **`(string)toAddress`** parameter in the top half of the **Presentation** tab.

10 Add a description in the **Description** tab in the bottom half of the screen.

For example, type **`The email address of the person to notify`**.

11 Click the **Properties** tab for `(string)toAddress` and click the **Add property** icon ( ).

12 From the list of proposed properties, select the **Mandatory input** property, click **Ok**, and set its value to **Yes**.

13 Click the **Add property** icon ( ).

14 From the list of proposed properties, select **Matching regular expression** and click **Ok**.

This property allows you to set constraints on what users can provide as input .

15 Click the **Value** text box for **Matching regular expression** and set the constraints to **`[a-zA-Z0-9_%-+.]+@[a-zA-Z0-9-.]+\.[a-zA-Z]{2,4}`**.

Setting these constraints limits user input to characters that are appropriate for email addresses. If the user tries to input any other character for the email address of the recipient when they start the workflow, the workflow does not start.

You have made both parameters mandatory, defined how the user can select the virtual machine to start, and limited the characters that can be input for the recipient's email address.

What to do next

You must create the layout, or presentation, of the input parameters dialog box in which users specify a workflow's input parameter values when they run it.

Set the Layout of the Simple Workflow Example Input Parameters Dialog Box

You create the layout or presentation of the input parameters dialog box in the workflow editor. The input parameters dialog box opens when users run a workflow that needs input parameters to run.

The layout you define in the **Presentation** tab also defines the layout of the input parameter dialog boxes for workflows you run using a Web view.

Prerequisites

Complete the following tasks.

- "Create the Simple Workflow Example," on page 83.
- "Create the Schema of the Simple Workflow Example," on page 84.
- "Define the Parameters of the Simple Workflow Example," on page 87.
- "Define the Simple Workflow Example Decision Bindings," on page 88.
- "Bind the Action Elements of the Simple Workflow Example," on page 89.
- "Bind the Simple Workflow Example Scripted Task Elements," on page 92.
- "Define the Simple Workflow Example Exception Bindings," on page 99.
- "Set the Read-Write Properties for Attributes of the Simple Workflow Example," on page 100.
- "Set the Simple Workflow Example Parameter Properties," on page 100.
- Open the workflow for editing in the workflow editor.

Procedure

1 Click the **Presentation** tab in the workflow editor.

2 Right-click the **Presentation** node in the presentation hierarchical list and select **Create display group**.

A **New step** node and a **New group** sub-node appear under the **Presentation** node.

3 Right-click **New step** and select **Delete**.

Because this workflow has only two parameters, you do not need multiple layers of display sections in the input parameters dialog box.

4 Double-click **New group** to edit the group name and press Enter.

For example, name the display group **`Virtual Machine`**.

The text you enter here appears as a heading in the input parameter dialog box when users start the workflow.

5 In the **Description** text box of the **General** tab at the bottom of the **Presentation** tab, provide a description for the new display group.

For example, type **`Select the virtual machine to start`**.

The text you type here appears as a prompt in the input parameter dialog box when users start the workflow.

6 Drag the **(VC:VirtualMachine)vm** parameter under the **Virtual Machine** display group.

In the input parameters dialog box, a text box in which the user types the virtual machine name will appear under a Virtual Machine heading.

7 Repeat the preceding steps to create a display group for the `toAddress` parameter, setting the following properties:

a Create a display group and name it **Recipient's Email Address**.

b Add a description for the display group, for example, **Enter the email address of the person to notify when this virtual machine is powered-on**.

c Drag the **toAddress** parameter under the **Recipient's Email Address** display group.

You have set up the layout of the input parameters dialog box that appears when users run the workflow.

What to do next

You have completed the development of the simple workflow example. You can now validate and run the workflow.

Validate and Run the Simple Workflow Example

After you create a workflow, you can validate it to discover any possible errors. If the workflow contains no errors, you can run it.

Prerequisites

Complete the following tasks.

- "Create the Simple Workflow Example," on page 83.
- "Create the Schema of the Simple Workflow Example," on page 84.
- "Define the Parameters of the Simple Workflow Example," on page 87.
- "Define the Simple Workflow Example Decision Bindings," on page 88.
- "Bind the Action Elements of the Simple Workflow Example," on page 89.
- "Bind the Simple Workflow Example Scripted Task Elements," on page 92.
- "Define the Simple Workflow Example Exception Bindings," on page 99.
- "Set the Read-Write Properties for Attributes of the Simple Workflow Example," on page 100.
- "Set the Simple Workflow Example Parameter Properties," on page 100.
- "Set the Layout of the Simple Workflow Example Input Parameters Dialog Box," on page 102.
- Open the workflow for editing in the workflow editor.

Procedure

1 Click **Validate** in the **Schema** tab of the workflow editor.

The validation tool locates any errors in the definition of the workflow.

2 After you have eliminated any errors, click **Save and Close** at the bottom of the workflow editor.

You return to the Orchestrator client.

3 Click the **Workflows** view.

4 Select **Workflow Examples > Start VM and Send Email** in the workflow hierarchical list.

5 Right-click the **Start VM and Send Email** workflow and select **Start workflow**.

The input parameters dialog box opens and prompts you for a virtual machine to start and an email address to send notifications to.

6 Select a virtual machine to start from the vCenter Server inventory.

7 Type an email address to which to send email notifications.

8 Click **Submit** to start the workflow.

A workflow token appears under the Start VM and Send Email workflow.

9 Click the workflow token to follow the progress of the workflow as it runs.

If the workflow runs successfully, the virtual machine you selected is in the powered-on state, and the email recipient you defined receives a confirmation email.

What to do next

You can generate a document in which to review information about the workflow. See "Generate Workflow Documentation," on page 80.

Develop a Complex Workflow

Developing a complex example workflow demonstrates the most common steps in the workflow development process and more advanced scenarios, such as creating custom decisions and loops.

In the complex workflow exercise, you develop a workflow that takes a snapshot of all the virtual machines contained in a given resource pool. The workflow you create will perform the following tasks:

1 Prompts the user for a resource pool that contains the virtual machines of which to take snapshots.

2 Determines whether the resource pool contains running virtual machines.

3 Determines how many running virtual machines the resource contains.

4 Verifies whether an individual virtual machine running in the pool meets specific criteria for a snapshot to be taken.

5 Takes the snapshot of the virtual machine.

6 Determines whether more virtual machines exist in the pool of which to take snapshots.

7 Repeats the verification and snapshot process until the workflow has taken snapshots of all eligible virtual machines in the resource pool.

The ZIP file of Orchestrator examples that you can download from the landing page of the Orchestrator documentation contains a completed version of the Take a Snapshot of All Virtual Machines in a Resource Pool workflow.

Prerequisites

Before you attempt to develop this complex workflow, follow the exercises in "Develop a Simple Example Workflow," on page 81. The procedures to develop a complex workflow provide the broad steps of the development process, but are not as detailed as the simple workflow exercises.

Procedure

1 Create the Complex Workflow Example on page 105

You must begin the workflow development process by creating the workflow in the Orchestrator client.

2 Create a Custom Action for the Complex Workflow Example on page 106

The `Check VM` scriptable element calls on an action that does not exist in the Orchestrator API. You must create the `getVMDiskModes` action.

3 Create the Schema of the Complex Workflow Example on page 107

You can create a workflow's schema in the workflow editor. The workflow schema contains the elements that the workflow runs, and determines the logical flow of the workflow.

4 (Optional) Create the Complex Workflow Example Zones on page 109

Optionally, you can highlight different zones of the workflow by adding workflow notes. Creating different workflow zones helps to make complicated workflow schema easier to read and understand.

5 Define the Parameters of the Complex Workflow Example on page 111

You define workflow parameters in the workflow editor. The input parameters provide data for the workflow to process. The output parameters are the data the workflow returns when it completes its run.

6 Define the Bindings for the Complex Workflow Example on page 111

You can bind a workflow's elements together in the workflow editor. Bindings define the data flow of the workflow. You also bind the scriptable task elements to their JavaScript functions.

7 Set the Complex Workflow Example Attribute Properties on page 121

You set the attribute properties in the **General** tab in the workflow editor.

8 Create the Layout of the Complex Workflow Example Input Parameters on page 121

You create the layout, or presentation, of the input parameters dialog box in the **Presentation** tab of the workflow editor. The input parameters dialog box opens when users run a workflow, and is the means by which users enter the input parameters with which the workflow runs.

9 Validate and Run the Complex Workflow Example on page 122

After you create a workflow, you can validate it to detect any possible errors. If the workflow contains no errors, you can run it.

Create the Complex Workflow Example

You must begin the workflow development process by creating the workflow in the Orchestrator client.

For information about how to install and configure vCenter Server, see the *vSphere Installation and Setup* documentation. For information about how to configure Orchestrator, see *Installing and Configuring VMware vCenter Orchestrator*.

Prerequisites

Verify that the following components are installed and configured on the system.

- vCenter Server, controlling a resource pool that contains some virtual machines
- The `Workflow Examples` folder in the workflows hierarchical list, that you created in "Create the Simple Workflow Example," on page 83.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Design**.

2 Select **Workflows > Workflow Examples**.

3 Right-click the **Workflow Examples** folder and select **New workflow**.

4 Name the new workflow **`Take a Snapshot of All Virtual Machines in a Resource Pool`** and click **OK**.

The workflow editor opens.

5 On the **General** tab of the workflow editor, click the version number digits to increment the version number.

   For the initial creation of the workflow, set the version to **0.0.1**.

6 Click the **Server restart behavior** value to set whether the workflow resumes after a server restart.

7 In the **Description** text box, type a description of what the workflow does.

8 Click **Save** at the bottom of the **General** tab.

You created the Take a Snapshot of All Virtual Machines in a Resource Pool workflow.

What to do next

You must create a custom action.

Create a Custom Action for the Complex Workflow Example

The `Check VM` scriptable element calls on an action that does not exist in the Orchestrator API. You must create the `getVMDiskModes` action.

For more detail about creating actions, see Chapter 3, "Developing Actions," on page 141.

Prerequisites

Create the Take a Snapshot of All Virtual Machines in a Resource Pool workflow. See "Create the Complex Workflow Example," on page 105.

Procedure

1 Close the workflow editor by clicking **Save and Close**.

2 Click the **Actions** view in the Orchestrator client.

3 Right-click the root of the actions hierarchical list and select **New Module**.

4 Name the new module **`com.vmware.example`**.

5 Right-click the **com.vmware.example** module and select **Add Action**.

6 Create an action called `getVMDiskModes`.

7 Right-click **`getVMDiskModes`** and select **Edit**.

8 Increment the version number in the **General** tab in the actions editor by clicking the version digits.

9 Add the following description of the action in the **General** tab.

```
This action returns an array containing the disk modes of all disks on a VM.
The elements in the array each have one of the following string values:
- persistent
- independent_persistent
- nonpersistent
- independent_nonpersistent
Legacy values:
- undoable
- append
```

10 Click the **Scripting** tab.

11 Right-click in the top pane of the **Scripting** tab and select **Add Parameter** to create the following input parameter.

   - Name: `vm`

- Value: `VC:VirtualMachine`
- Description: **`The virtual machine for which to return the Disk Modes`**

12 Add the following scripting in the bottom of the **Scripting** tab.

The following code returns an array of disk modes for the disks of the virtual machine.

```
var devicesArray = vm.config.hardware.device;
var retArray = new Array();
if (devicesArray!=null && devicesArray.length!=0) {
    for (i in devicesArray) {
        if (devicesArray[i] instanceof VcVirtualDisk) {
            retArray.push(devicesArray[i].backing.diskMode);
        }
    }
}
return retArray;
```

13 Click **Save** and **Close** to exit the **Actions** palette.

You have defined the custom action the Take a Snapshot of All Virtual Machines in a Resource Pool workflow requires.

What to do next

Create the workflow's schema.

Create the Schema of the Complex Workflow Example

You can create a workflow's schema in the workflow editor. The workflow schema contains the elements that the workflow runs, and determines the logical flow of the workflow.

Prerequisites

Complete the following tasks.

- "Create the Complex Workflow Example," on page 105.
- "Create a Custom Action for the Complex Workflow Example," on page 106.
- Open the workflow for editing in the workflow editor.

Procedure

1 Click the **Schema** tab in the workflow editor.

2 Add the following schema elements to the workflow schema.

Element Type	Element Name	Position in Schema
Scriptable task	**`Initializing`**	Below the `Start` element
Decision	**`VMs to Process?`**	Below the `Initializing` scriptable task element
Scriptable task	**`Pool Has No VMs`**	Below the `VMs to Process?` custom decision element, linked with a red arrow
Custom decision	**`Remaining VMs?`**	Right of the `VMs to Process?` custom decision element, linked with a green arrow
Action	`getVMDiskModes`	Right of the `Remaining VMs?` custom decision element, linked with a green arrow
Custom decision	**`Create Snapshot?`**	Right of the `getVMDiskModes` action element, linked with a blue arrow

Element Type	Element Name	Position in Schema
Workflow	`Create a snapshot`	Above the `Create Snapshot?` custom decision element, linked with a green arrow
Scriptable task	**`VM Snapshots`**	Left of the `Create a snapshot` workflow, linked with a blue arrow
Scriptable task	**`Increment`**	Left of the `VM Snapshots` scriptable task element, linked with a blue arrow
Scriptable task	**`Set Output`**	Right of the `Pool Has No VMs` scriptable task element, linked with a blue arrow

3 Add a `Log Exception` scriptable task element.

   a Create an exception handling link from the Create a snapshot workflow to an `End` element.

   b Drag a scriptable task element to the red dashed arrow that links the Create a snapshot workflow to an `End` element.

   c Double-click the scriptable task element and rename it to **`Log Exception`**.

   d Move the `Log Exception` scriptable task element to above the `VM Snapshots` scriptable task element.

4 Unlink all `End` elements except the `End` element that is at the right of the `Set Output` scriptable task element.

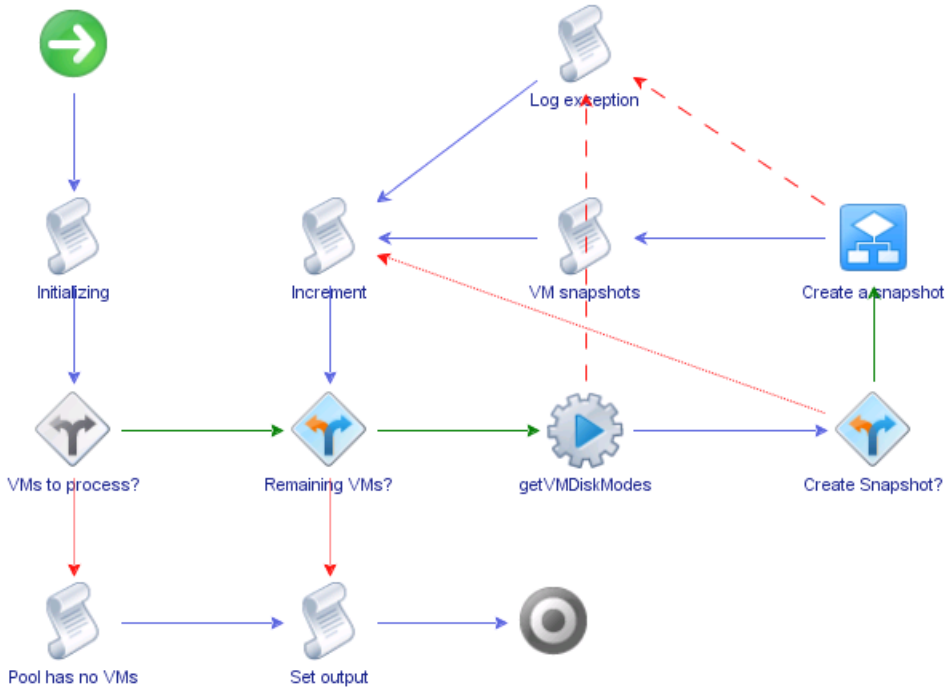
5 Link the remaining elements as described in the following table.

Element	Link to	Type of Arrow	Description
`getVMDiskModes` action element	`Log Exception` scriptable task element	Red dashed	Exception handling
`Create Snapshot?` custom decision element	`Increment` scriptable task element	Red	`False` result
`Log Exception` scriptable task element	`Increment` scriptable task element	Blue	Normal workflow progression
`Increment` scriptable task element	`Remaining VMs?` custom decision element	Blue	Normal workflow progression
`Remaining VMs?` custom decision element	`Set Output` scriptable task element	Red	`False` result

6 Click **Save** at the bottom of the **Schema** tab.

The following figure shows what the linked elements of the Take a Snapshot of All Virtual Machines in a Resource Pool workflow should look like.

Figure 1-5. Linking of the Take a Snapshot of All Virtual Machines in a Resource Pool Example Workflow



What to do next

You can optionally define workflow zones by using workflow notes.

(Optional) Create the Complex Workflow Example Zones

Optionally, you can highlight different zones of the workflow by adding workflow notes. Creating different workflow zones helps to make complicated workflow schema easier to read and understand.

Prerequisites

Complete the following tasks.

- "Create the Complex Workflow Example," on page 105.
- "Create the Schema of the Complex Workflow Example," on page 107.
- Open the workflow for editing in the workflow editor.

Procedure

1 Create the following workflow zones by using workflow notes.

Elements in Zone	Description
Start element; Initialize scriptable task; VMs to Process? custom decision	`Get an array of virtual machines from a resource pool, initialize the counter of the Array and set the first virtual machine to be treated, if any.`
Pool has no VMs scriptable task.	`Resource pool contains no virtual machines of which to take snapshots.`

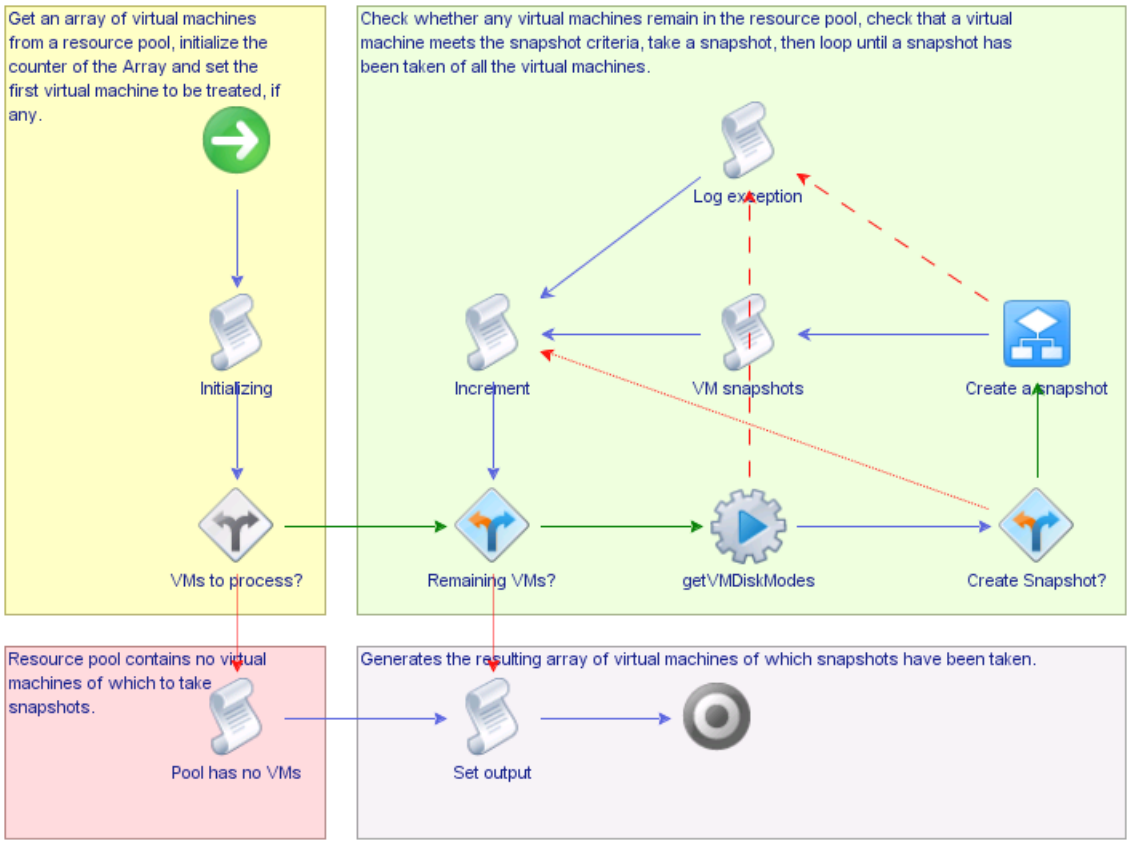
Elements in Zone	Description
VMs remaining? custom decision; getVMDisksModes action, Create Snapshot? decision; Create a snapshot workflow; VM Snapshots scriptable task; Increment scriptable task; Log Exception scriptable task	`Check whether any virtual machines remain in the resource pool, check that a virtual machine meets the snapshot criteria, take a snapshot, then loop until a snapshot has been taken of all the virtual machines.`
Set Output scriptable task; End element	`Generates the resulting array of virtual machines of which snapshots have been taken.`

2 Select a workflow note and press Ctrl+E to select the background color.

3 Click **Save** at the bottom of the workflow editor **Schema** tab.

Your workflow zones should look like the following diagram.

Figure 1-6. Schema Diagram for Take Snapshot of all Virtual Machines in a Resource Pool Example Workflow



What to do next

You must define the workflow's input and output parameters.

Define the Parameters of the Complex Workflow Example

You define workflow parameters in the workflow editor. The input parameters provide data for the workflow to process. The output parameters are the data the workflow returns when it completes its run.

Prerequisites

Complete the following tasks.

- "Create the Complex Workflow Example," on page 105.
- "Create the Schema of the Complex Workflow Example," on page 107.
- Open the workflow for editing in the workflow editor.

Procedure

1 Click the **Inputs** tab in the workflow editor.

2 Define the following input parameter.

- Name: `resourcePool`
- Type: `VC:ResourcePool`
- Description: **`The resource pool containing the virtual machines of which to take snapshots.`**

3 Click the **Outputs** tab in the workflow editor.

4 Define the following output parameter.

- Name: `snapshotVmArrayOut`
- Type: `Array/VC:VirtualMachine`
- Description: **`The Array of virtual machines of which snapshots have been taken.`**

You have defined the workflow's input and output parameters.

What to do next

You must define the bindings between the element parameters.

Define the Bindings for the Complex Workflow Example

You can bind a workflow's elements together in the workflow editor. Bindings define the data flow of the workflow. You also bind the scriptable task elements to their JavaScript functions.

Prerequisites

Complete the following tasks.

- "Create the Complex Workflow Example," on page 105.
- "Create the Schema of the Complex Workflow Example," on page 107
- "Define the Parameters of the Complex Workflow Example," on page 111
- Review the bindings that you must define. See "Complex Workflow Example Bindings," on page 112.
- Open the workflow for editing in the workflow editor.

Procedure

1 Click the **Schema** tab in the workflow editor.

2 Define the bindings.

3 Click **Save** at the bottom of the **Schema** tab.

All the input and output parameters of the elements are bound to the appropriate parameter types and values.

What to do next

Set the attribute properties.

Complex Workflow Example Bindings

Bindings define how the simple workflow example's action elements process input and output parameters.

The Take Snapshots of All Virtual Machines in a Resource Pool workflow requires the following input and output parameter bindings. You also define the JavaScript functions for the scriptable task elements.

In cases in which you bind to existing parameters, the binding inherits the type and description values from the original parameter.

Initializing Scriptable Task

The Initializing scriptable task element initializes the attributes of the workflow. The following table shows the input and output parameter bindings that the Initializing scriptable task element requires.

Table 1-15. Bindings of the Initializing Scriptable Task Element

Parameter Name	Binding Type	Bind to Existing or Create Parameter?	Binding Values
`resourcePool`	IN	Bind	■ Local parameter: `resourcePool` ■ Source parameter: `resourcePool[in-parameter]` ■ Type: `VC:ResourcePool` ■ Description: **`The resource pool containing the virtual machines of which to take snapshots`**
`allVMs`	OUT	Create	■ Local parameter: `allVMs` ■ Source parameter: `allVMs[attribute]` ■ Type: `Array/VC:VirtualMachine` ■ Description: **`The virtual machines in the resource pool.`**
`numberOfVMs`	OUT	Create	■ Local parameter: `numberOfVMs` ■ Source parameter: `numberOfVMs[attribute]` ■ Type: number ■ Description: **`The number of virtual machines found in the resourcePool`**

Table 1-15. Bindings of the Initializing Scriptable Task Element (Continued)

Parameter Name	Binding Type	Bind to Existing or Create Parameter?	Binding Values
vmCounter	OUT	Create	■ Local parameter: vmCounter ■ Source parameter: vmCounter[attribute] ■ Type: number ■ Description: **The counter of the virtual machines inside the array**
vm	OUT	Create	■ Local parameter: vm ■ Source parameter: vm[attribute] ■ Type: VC:VirtualMachine ■ Description: **The current virtual machine having a snapshot taken**
snapshotVmArray	OUT	Create	■ Local parameter: snapshotVmArray ■ Source parameter: snapshotVmArray[attribute] ■ Type: Array/VC:VirtualMachine ■ Description: **The Array of virtual machines of which snapshots have been taken**

The Initialize scriptable task element performs the following scripted function.

```
//Retrieve an array of virtual machines contained in the specified Resource Pool
allVMs = resourcePool.vm;
//Initialize the size of the Array and the first VM to snapshot
if (allVMs!=null && allVMs.length!=0) {
    numberOfVms = allVMs.length;
    vm = allVMs[0];
} else {
    numberOfVms = 0;
}
//Initialize the VM counter
vmCounter = 0;
//Initializing the array of VM snapshots
snapshotVmArray = new Array();
```

VMs to Process? Decision Element

The VMs to Process? decision element determines whether any virtual machines of which to take snapshots exist in the resource pool. The following table shows the bindings that the VMs to Process? decision element requires.

Table 1-16. Bindings of the VMs to Process? Decision Element

Parameter Name	Binding Type	Bind to Existing or Create Parameter?	Binding Values
`numberOfVMs`	Decision	Bind	■ Source parameter: `numberOfVMs[attribute]` ■ Decision statement: Greater than ■ Value: 0.0 ■ Description: **`The number of virtual machines found in the resourcePool`**

Pool Has No VMs Scriptable Task Element

The Pool Has No VMs scriptable task element logs the fact that the resource pool contains no eligible virtual machines in the Orchestrator database. The following table shows the bindings that the Pool Has No VMs scriptable task element requires.

Table 1-17. Bindings of the Pool Has No VMs Scriptable Task Element

Parameter Name	Binding Type	Bind to Existing or Create Parameter?	Binding Values
`resourcePool`	IN	Bind	■ Local parameter: `resourcePool` ■ Source parameter: `resourcePool[in-parameter]` ■ Type: `VC:ResourcePool` ■ Description: **`The resource pool containing the virtual machines of which to take snapshots.`**

The Pool Has No VMs scriptable task element performs the following scripted function.

```
//Writes the following event in the vCO database
Server.warn("The specified ResourcePool "+resourcePool.name+" does not contain any VMs.");
```

Remaining VMs? Custom Decision Element

The Remaining VMs? custom decision element determines whether any virtual machines of which to take snapshots remain in the resource pool. The following table shows the bindings that the Remaining VMs? custom decision element requires.

Table 1-18. Bindings of the Remaining VMs? Custom Decision Element

Parameter Name	Binding Type	Bind to Existing or Create Parameter?	Binding Values
`numberOfVMs`	IN	Bind	■ Source parameter: `numberOfVMs[attribute]` ■ Decision statement: Greater than ■ Value: 0.0 ■ Description: **`The number of virtual machines found in the resourcePool`**
`vmCounter`	IN	Bind	■ Local parameter: `vmCounter` ■ Source parameter: `vmCounter[attribute]` ■ Type: number ■ Description: **`The counter of the virtual machines inside the array`**

The Remaining VMs? custom decision element performs the following scripted function.

```
//Checks if the workflow has reached the end of the array of VMs
if (vmCounter < numberOfVms) {
    return true;
} else {
    return false;
}
```

getVMDisksModes Action Element

The `getVMDisksModes` action element obtains the modes of the disks running in a virtual machine. The following table shows the bindings that the `getVMDisksModes` action element requires.

Table 1-19. Bindings of the getVMDisksModes Action Element

Parameter Name	Binding Type	Bind to Existing or Create Parameter?	Binding Values
`vm`	IN	Bind	■ Local parameter: `vm` ■ Source parameter: `vm[attribute]` ■ Type: `VC:VirtualMachine` ■ Description: **`The current virtual machine having a snapshot taken`**
`actionResult`	OUT	Create	■ Local parameter: `actionResult` ■ Source parameter: `vmDisksModes[attribute]` ■ Type: Array/String ■ Description: **`The current Disks Modes of the virtual machine`**
`errorCode`	Exception	Create	Local parameter: `errorCode`

Create Snapshot? Custom Decision Element

The Create Snapshot? custom decision element determines whether to take snapshots of virtual machines, depending on the disk modes of the virtual machines. The following table shows the bindings that the Create Snapshot? custom decision element requires.

Table 1-20. Bindings of the Create Snapshot? Decision Element

Parameter Name	Binding Type	Bind to Existing or Create Parameter?	Binding Values
`vmDisksMode`	IN	Bind	■ Local parameter: `vmDisksMode` ■ Source parameter: `vmDisksMode[attribute]` ■ Type: Array/String ■ Description: **`The current Disks Modes of the virtual machine`**
`vm`	IN	Bind	■ Local parameter: `vm` ■ Source parameter: `vm[attribute]` ■ Type: `VC:VirtualMachine` ■ Description: **`The current virtual machine having a snapshot taken`**

The Create Snapshot? custom decision element custom decision element performs the following scripted function.

```
//A snapshot cannot be taken if one of its disks is in independent mode
// (independent-persistent or independent-nonpersistent)
var containsIndependentDisks = false;
if (vmDisksModes!=null && vmDisksModes.length>0) {
    for (i in vmDisksModes) {
        if (vmDisksModes[i].charAt(0)=="i") {
            containsIndependentDisks = true;
        }
    }
} else {
    //if no disk found no need to try to snapshot the VM
    System.warn("Won't snapshot '"+vm.name+"', no disks found");
    return false;
}
if (containsIndependentDisks) {
    System.warn("Won't snapshot '"+vm.name+"', independent disk(s) found");
    return false;
} else {
    System.log("Snapshoting '"+vm.name+"'");
    return true;
}
```

Create a snapshot Workflow Element

The Create a snapshot workflow element takes snapshots of virtual machines. The following table shows the bindings that the Create a snapshot workflow element requires.

Table 1-21. Bindings of the Create a snapshot Workflow Element

Parameter Name	Binding Type	Bind to Existing or Create Parameter?	Binding Values
vm	IN	Bind	■ Local parameter: vm ■ Source parameter: vm[attribute] ■ Type: VC:VirtualMachine ■ Description: **An active virtual machine of which to take a snapshot.**
name	IN	Create	■ Local parameter: name ■ Source parameter: snapshotName[attribute] ■ Type: string ■ Description: **The name for this snapshot. The name does not need to be unique for this virtual machine.**
description	IN	Create	■ Local parameter: description ■ Source parameter: snapshotDescription[attribute] ■ Type: string ■ Description: **A description for this snapshot.**
memory	IN	Create	■ Local parameter: memory ■ Source parameter: snapshotMemory[attribute] ■ Type: Boolean ■ Value: no ■ Description: **If TRUE, a dump of the internal state of the virtual machine (a memory dump) is included in the snapshot.**
quiesce	IN	Create	■ Local parameter: quiesce ■ Source parameter: snapshotQuiesce[attribute] ■ Type: Boolean ■ Value: yes ■ Description: **If TRUE and the virtual machine is powered on when the snapshot is taken, the VMware Tools are used to quiesce the file system in the virtual machine.**

Table 1-21. Bindings of the Create a snapshot Workflow Element (Continued)

Parameter Name	Binding Type	Bind to Existing or Create Parameter?	Binding Values
snapshot	OUT	Create	■ Local parameter: snapshot ■ Source parameter: NULL ■ Type: VC:VirtualMachineSnapshot ■ Description: **The snapshot taken.**
errorCode	Exception	Create	Local parameter: errorCode

VM Snapshots Scriptable Task Element

The VM Snapshots scriptable task element adds the snapshots to an array. The following table shows the bindings that the VM Snapshots scriptable task element requires.

Table 1-22. Bindings of the VM Snapshots Scriptable Task Element

Parameter Name	Binding Type	Bind to Existing or Create Parameter?	Binding Values
vm	IN	Bind	■ Local parameter: vm ■ Source parameter: vm[attribute] ■ Type: VC:VirtualMachine ■ Description: **An active virtual machine of which to take a snapshot.**
snapshotVmArray	IN	Bind	■ Local parameter: snapshotVmArray ■ Source parameter: snapshotVmArray[attribute] ■ Type: Array/VC:VirtualMachine ■ Description: **The Array of virtual machines of which snapshots have been taken**
snapshotVmArray	OUT	Bind	■ Local parameter: snapshotVmArray ■ Source parameter: snapshotVmArray[attribute] ■ Type: Array/VC:VirtualMachine ■ Description: **The Array of virtual machines of which snapshots have been taken**

The VM Snapshots scriptable task element performs the following scripted function.

```
//Writes the following event in the vCO database
Server.log("Successfully took snapshot of the VM '"+vm.name);
//Inserts the VM snapshot in an array
snapshotVmArray.push(vm);
```

Increment Scriptable Task Element

The Increment scriptable task element increments the counter that counts the number of virtual machines in the array. The following table shows the bindings that the Increment scriptable task element requires.

Table 1-23. Bindings of the Increment Scriptable Task Element

Parameter Name	Binding Type	Bind to Existing or Create Parameter?	Binding Values
`vmCounter`	IN	Bind	■ Local parameter: `vmCounter` ■ Source parameter: `vmCounter[attribute]` ■ Type: number ■ Description: **`The counter of the virtual machines inside the array`**
`allVMs`	IN	Bind	■ Local parameter: `allVMs` ■ Source parameter: `allVMs[attribute]` ■ Type: `Array/VC:VirtualMachine` ■ Description: **`The virtual machines in the resource pool.`**
`vmCounter`	OUT	Bind	■ Local parameter: `vmCounter` ■ Source parameter: `vmCounter[attribute]` ■ Type: number ■ Description: **`The counter of the virtual machines inside the array`**
`vm`	OUT	Bind	■ Local parameter: `vm` ■ Source parameter: `vm[attribute]` ■ Type: `VC:VirtualMachine` ■ Description: **`The current virtual machine having a snapshot taken`**

The Increment scriptable task element performs the following scripted function.

```
//Increases the array VM counter
vmCounter++;
//Sets the next VM to be snapshot in the attribute vm
vm = allVMs[vmCounter];
```

Log Exception Scriptable Task Element

The Log Exception scriptable task element handles exceptions from the workflow and action elements. The following table shows the bindings that the Log Exception scriptable task element requires.

Table 1-24. Bindings of the Log Exception Task Element

Parameter Name	Binding Type	Bind to Existing or Create Parameter?	Binding Values
vm	IN	Bind	■ Local parameter: vm ■ Source parameter: vm[attribute] ■ Type: VC:VirtualMachine ■ Description: **The current virtual machine having a snapshot taken**
errorCode	IN	Bind	■ Local parameter: errorCode ■ Source parameter: errorCode[attribute] ■ Type: string ■ Description: **An exception caught while taking a snapshot of a virtual machine**

The Log Exception scriptable task element performs the following scripted function.

```
//Writes  the following event in the vCO database
Server.error("Coudln't snapshot the VM '"+vm.name+"', exception: "+errorCode);
```

Set Output Scriptable Task Element

The Set Output scriptable generates the workflow's output parameter, that contains the array of virtual machines of which snapshots have been taken. The following table shows the bindings that the Set Output scriptable task element requires.

Table 1-25. Bindings of the Set Output Task Element

Parameter Name	Binding Type	Bind to Existing or Create Parameter?	Binding Values
snapshotVmArray	IN	Bind	■ Local parameter: snapshotVmArray ■ Source parameter: snapshotVmArray[attribute] ■ Type: Array/VC:VirtualMachine ■ Description: **The Array of virtual machines of which snapshots have been taken**
snapshotVmArrayOut	OUT	Bind	■ Local parameter: snapshotVmArrayOut ■ Source parameter: snapshotVmArrayOut[out-parameter] ■ Type: Array/VC:VirtualMachine ■ Description: **The Array of virtual machines of which snapshots have been**

The Set Output scriptable task element performs the following scripted function.

```
//Passes the value of the internal attribute to a workflow output parameter
snapshotVmArrayOut = snapshotVmArray;
```

Set the Complex Workflow Example Attribute Properties

You set the attribute properties in the **General** tab in the workflow editor.

Prerequisites

Complete the following tasks.

- "Create the Complex Workflow Example," on page 105.
- "Create the Schema of the Complex Workflow Example," on page 107.
- "Define the Bindings for the Complex Workflow Example," on page 111.
- Open the workflow for editing in the workflow editor.

Procedure

1 Click the **General** tab.

2 Select the read-only check box of the following attributes to make them read-only constants:

- snapshotName
- snapshotDescription
- snapshotMemory
- snapshotQuiesce

You have defined which of the workflow's attributes are constants and which are variables.

What to do next

You must create the workflow presentation, which creates the layout of the input parameters dialog box in which users specify a workflow's input parameter values when they run it.

Create the Layout of the Complex Workflow Example Input Parameters

You create the layout, or presentation, of the input parameters dialog box in the **Presentation** tab of the workflow editor. The input parameters dialog box opens when users run a workflow, and is the means by which users enter the input parameters with which the workflow runs.

Prerequisites

Complete the following tasks.

- "Create the Complex Workflow Example," on page 105.
- "Create the Schema of the Complex Workflow Example," on page 107.
- "Define the Parameters of the Complex Workflow Example," on page 111.
- "Define the Bindings for the Complex Workflow Example," on page 111.
- "Set the Complex Workflow Example Attribute Properties," on page 121.
- Open the workflow for editing in the workflow editor.

Procedure

1 Click the **Presentation** tab in the workflow editor.

The Take a Snapshot of All Virtual Machines in a Resource Pool workflow has only one input parameter, so creating the presentation is straightforward.

2 Right-click the **Presentation** node in the presentation hierarchical list and select **Create display group**.

3 Delete the **New step** element that appears above the **New group** element.

4 Double-click the **New group** element and change the group name to **`Resource Pool`**.

5 Provide a description of the **Resource Pool** display group in the **Description** text box on the **General** tab at the bottom of the **Presentation** tab.

For example,
`Enter the name of the resource pool that contains the virtual machines of which to take a snapshot.`

6 Click the `(VC:ResourcePool)resourcePool` parameter.

7 Click the **Properties** tab for `(VC:ResourcePool)resourcePool`.

8 Right-click within the **Properties** tab and select **Add Property > Mandatory input**.

9 Right-click within the **Properties** tab and select **Add Property > Select value as**.

When you set this property, you set how the user selects the value of the `(VC:ResourcePool)resourcePool` input parameter.

10 Drag the `(VC:ResourcePool)resourcePool` parameter under the **Resource Pool** display group.

You have created the layout of the dialog box that appears when users run the workflow.

What to do next

You have completed the development of the complex workflow example. You can now validate and run the workflow.

Validate and Run the Complex Workflow Example

After you create a workflow, you can validate it to detect any possible errors. If the workflow contains no errors, you can run it.

Prerequisites

Create a workflow, lay out its schema, define the links and bindings, define the parameter properties, and create the presentation of the input parameters dialog box.

Complete the following tasks.

- "Create the Complex Workflow Example," on page 105.
- "Create a Custom Action for the Complex Workflow Example," on page 106.
- "Create the Schema of the Complex Workflow Example," on page 107.
- "Define the Parameters of the Complex Workflow Example," on page 111.
- "Define the Bindings for the Complex Workflow Example," on page 111.
- "Set the Complex Workflow Example Attribute Properties," on page 121.
- "Create the Layout of the Complex Workflow Example Input Parameters," on page 121.
- Open the workflow for editing in the workflow editor.

Procedure

1 Click **Validation** in the **Schema** tab of the workflow editor.

The validation tool detects any errors in the definition of the workflow.

2 After you have eliminated any errors, click **Save and Close** at the bottom of the workflow editor.

You return to the Orchestrator client.

3 Click the **Workflows** view.

4 In the workflow hierarchical list, select **Workflow Examples > Take a Snapshot of All Virtual Machines in a Resource Pool**.

5 Right-click the **Take a Snapshot of All Virtual Machines in a Resource Pool** workflow and select **Start workflow**.

The input parameters dialog box opens and prompts you for a resource pool that contains the virtual machines of which to take a snapshot.

6 Click **Submit** to run the workflow.

A workflow token appears under the Take a Snapshot of All Virtual Machines in a Resource Pool workflow.

7 Click the workflow token to follow the progress of the workflow as it runs.

If the workflow runs successfully, the workflow takes a snapshot of all of the virtual machines in the selected resource pool.

What to do next

You can generate a document in which to review information about the workflow. See "Generate Workflow Documentation," on page 80.

Scripting 2

Orchestrator uses JavaScript to create building blocks from which you create actions, workflow elements, and policies that access the APIs of the technologies that you plug into Orchestrator.

Orchestrator uses the Mozilla Rhino 1.7R4 JavaScript engine as its scripting engine. The scripting engine provides variable type checking, name space management, automatic completion, and exception handling.

The Orchestrator workflow engine allows you to use basic JavaScript language features, such as if, loops, arrays, and strings. You can use objects in scripting that the Orchestrator API provides, or objects from any other API that you import into Orchestrator through a plug-in and that you map to JavaScript objects. For information about Rhino, see the Mozilla Rhino Web site.

This chapter includes the following topics:

Orchestrator Elements that Require Scripting

Not all Orchestrator elements require you to write scripts. To provide maximum flexibility to your applications, you can customize certain elements by adding JavaScript functions.

You can add scripts in the following Orchestrator elements.

Actions Actions are scripted functions. You can limit the scripting you write for an action to a single operation, to maximize the potential for action reuse by other elements, such as other workflows. Alternatively, an action can contain many operations, to limit the complexity of workflows, although this does reduce the capacity for reusing the action.

Policies You set policies by using scripts that watch for trigger events. When the trigger events occur, policies launch orchestration operations that you define in scripts.

Workflows The Scriptable Task workflow element allows you to write a custom scripted operation or sequence of operations that you can use in the workflows. You also define the Boolean decision statement for custom decision elements in scripts that return either `true` or `false`.

Limitations of the Mozilla Rhino Implementation in Orchestrator

Orchestrator uses the Mozilla Rhino 1.7R4 JavaScript engine. However, the implementation of Rhino in Orchestrator presents some limitations.

When writing scripts for workflows, you must consider the following limitations of the Mozilla Rhino implementation in Orchestrator.

- When a workflow runs, the objects that pass from one workflow element to another are not JavaScript objects. What is passed from one element to the next is the serialization of a Java object that has a JavaScript image. As a consequence, you cannot use the whole JavaScript language, but only the classes that are present in the API Explorer. You cannot pass function objects from one workflow element to another.
- Orchestrator runs the code in scriptable task elements in a context that is not the Rhino root context. Orchestrator transparently wraps scriptable task elements and actions into JavaScript functions, which it then runs. A scriptable task element that contains `System.log(this);` does not display the global object `this` in the same way as a standard Rhino implementation does.
- You can only call actions that return nonserializable objects from scripting, and not from workflows. To call an action that returns a nonserializable object, you must write a scriptable task element that calls the action by using the `System.getModule`*ModuleName*`.action()` method.
- Workflow validation does not check whether a workflow attribute type is different from an input type of an action or subworkflow. If you change the type of a workflow input parameter, for example from `VIM3:VirtualMachine` to `VC:VirtualMachine`, but you do not update any scriptable tasks or actions that use the original input type, the workflow validates but does not run.

Using the Orchestrator Scripting API

The Orchestrator API exposes as JavaScript objects and methods all of the objects and functions of the technologies that Orchestrator accesses through its plug-ins.

For example, you can access JavaScript implementations of the vCenter Server API through the Orchestrator API, to include vCenter operations in scripted elements that you create. You can also access JavaScript implementations of objects from all of the other plug-ins you install in the Orchestrator server. If you create a custom plug-in to a third-party application, you map the objects from its API to JavaScript objects that the Orchestrator API then exposes.

Procedure

1 Access the Scripting Engine from the Workflow Editor on page 127

The Orchestrator scripting engine uses the Mozilla Rhino 1.7R4 JavaScript engine to help you write scripts for scripted elements in workflows. You access the scripting engine for scripted workflow elements from the **Scripting** tab in the workflow editor.

2 Access the Scripting Engine from the Action or Policy Editor on page 128

The Orchestrator scripting engine uses the Mozilla Rhino JavaScript engine to help you write scripts for actions or policies. You access the scripting engine for actions and policies from the **Scripting** tabs in the action and policy editors.

3 Access the Orchestrator API Explorer on page 128

Orchestrator provides an API Explorer that you can use to search the Orchestrator API and see the documentation for JavaScript objects that you can use in scripted elements.

4 Use the Orchestrator API Explorer to Find Objects on page 128

The Orchestrator API exposes the API of all plugged-in technologies, including the entire vCenter Server API. The Orchestrator API Explorer helps you find the objects you need to add to scripts.

5 Writing Scripts on page 129

The Orchestrator scripting engine helps you to write scripts. Automatic insertion of functions and automatic completion of lines of scripting accelerates the scripting process and minimizes the potential for writing errors in scripts.

6 Add Parameters to Scripts on page 131

The Orchestrator scripting engine helps you to import available parameters into scripts.

7 Accessing the Orchestrator Server File System from JavaScript and Workflows on page 131

Orchestrator limits access to the Orchestrator server file system from JavaScript and Workflows to specific directories.

8 Accessing Java Classes from JavaScript on page 132

By default, Orchestrator restricts JavaScript access to a limited set of Java classes. If you require JavaScript access to a wider range of Java classes, you must set an Orchestrator system property to allow this access.

9 Accessing Operating System Commands from JavaScript on page 132

The Orchestrator API provides a scripting class, `Command`, that runs commands in the Orchestrator server host operating system. To prevent unauthorized access to the Orchestrator server host, by default, Orchestrator applications do not have permission to run the `Command` class.

Access the Scripting Engine from the Workflow Editor

The Orchestrator scripting engine uses the Mozilla Rhino 1.7R4 JavaScript engine to help you write scripts for scripted elements in workflows. You access the scripting engine for scripted workflow elements from the **Scripting** tab in the workflow editor.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Design**.

2 Right-click a workflow in the **Workflows** view of the Orchestrator client and select **Edit**.

3 Click the **Schema** tab in the workflows editor.

4 Add a Scriptable Task element or a Custom Decision element to the workflow schema.

5 Click on the scriptable element's **Scripting** tab.

You accessed the scripting engine to define the scripted functions of workflow elements. The **Scripting** tab allows you to navigate through the API, consult documentation about the objects, search for objects, and write JavaScript.

What to do next

Search the Orchestrator API using the API Explorer.

Access the Scripting Engine from the Action or Policy Editor

The Orchestrator scripting engine uses the Mozilla Rhino JavaScript engine to help you write scripts for actions or policies. You access the scripting engine for actions and policies from the **Scripting** tabs in the action and policy editors.

Procedure

1 Select an option from the drop-down menu in the Orchestrator client, depending on the type of the element whose scripting you want to edit.

Option	Description
Design	Select this option to edit the scripting of an action element.
Run	Select this option to edit the scripting of a policy.

2 Right-click an action or policy in the **Actions** or **Policies** views and select **Edit**.

3 Click the **Scripting** tab in the action or policy editor.

You accessed the scripting engine to define the scripted functions of action or policy elements. The **Scripting** tab allows you to navigate through the API, consult documentation about the objects, search for objects, and write JavaScript.

What to do next

Search the Orchestrator API using the API Explorer.

Access the Orchestrator API Explorer

Orchestrator provides an API Explorer that you can use to search the Orchestrator API and see the documentation for JavaScript objects that you can use in scripted elements.

You can consult an online version of the Scripting API for the vCenter Server plug-in on the Orchestrator documentation home page.

Procedure

1 Log in to the Orchestrator client.

2 Select **Tools > API Explorer**.

The API Explorer appears. You can use it to search all the objects and functions of the Orchestrator API.

What to do next

Use the API Explorer to write scripts for scriptable elements.

Use the Orchestrator API Explorer to Find Objects

The Orchestrator API exposes the API of all plugged-in technologies, including the entire vCenter Server API. The Orchestrator API Explorer helps you find the objects you need to add to scripts.

Prerequisites

Open the API Explorer.

Procedure

1 Enter the name or part of a name of an object in the API Explorer **Search** text box and click **Search**.

To limit your search to a particular object type, uncheck or check the **Scripting Class**, **Attributes & Methods** and **Types & Enumerations** check boxes.

2 Double-click the element in the proposed list.

The object is highlighted in the hierarchical list on the left. A documentation pane under the hierarchical list presents information about the object.

What to do next

Use the objects you find in scripts.

JavaScript Objects in the API Explorer

The Orchestrator API Explorer identifies and groups together the different kinds of JavaScript objects in the hierarchical tree on the left of the **Scripting** tab or API Explorer dialog box. The API Explorer uses icons to help you identify the different kinds of object.

The following table describes the objects of the Orchestrator API and shows their icon.

Table 2-1. JavaScript Objects in the Orchestrator API

Object	Icon in Hierarchical List	Description
Type		Types
Function set		Internal type that contains a set of static methods
Primitive		Primitive types
Object		Standard Orchestrator scripting objects
Attribute		JavaScript attributes
Method		JavaScript methods
Constructor		JavaScript constructors
Enumeration		JavaScript enumerations
String set		String set, default values
Module		A collection of actions
Plug-in	Image that plug-in defines	The APIs that plug-ins expose to Orchestrator

Writing Scripts

The Orchestrator scripting engine helps you to write scripts. Automatic insertion of functions and automatic completion of lines of scripting accelerates the scripting process and minimizes the potential for writing errors in scripts.

Prerequisites

Open a scripted element for editing and click its **Scripting** tab.

Procedure

1 Navigate through the hierarchical list of objects on the left of the **Scripting** tab, or use the API Explorer search function, to select a type, class, or method to add to the script.

2 Right-click the type, class, or method and select **Copy**.

If the scripting engine does not allow you to copy the element you selected, this object is not possible in the context of the script.

3 Right-click in the scripting pad, and paste the element you copied into the appropriate place in the script.

The scripting engine enters the element into the script, complete with its constructor and an instance name.

For example, if you copied the `Date` object, the scripting engine pastes the following code into the script.

```
var myDate = new Date();
```

4 Copy and paste a method to add to the script.

The scripting engine completes the method call, adding the required attributes.

For example, if you copied the `cloneVM()` method from the `com.vmware.library.vc.vm` module, the scripting engine pastes the following code into the script.

```
System.getModule("com.vmware.library.vc.vm").cloneVM(vm,folder,name,spec)
```

The scripting engine highlights those parameters that you already defined in the element. Any undefined parameters remain unhighlighted.

5 Place the cursor at the end of an element you pasted into the script and press Ctrl+space to select from a contextual list of possible methods and attributes that the object can call.

NOTE The automatic completion feature is currently experimental.

You added object and functions to the script.

What to do next

Add parameters to the script.

Color Coding of Scripting Keywords

When you add scripts on the **Scripting** tab of a scripted workflow element, certain types of keywords appear in different colors to enhance the readability of the code.

All scripting appears in standard black font unless stated otherwise.

Table 2-2. Color Coding of Scripting Keywords

Keyword Type	Text Color in Scripting Tab
Standard JavaScript keywords, for example `if`, `else`, `for`, and `new`	Bold black
Variable declarations, namely `var`	Green
Modifiers in loops, for example `in`	Red
Null variable values	Purple
Non-null variable values	Green
Code comments	Italic gray
Orchestrator plug-in object types, for example `VC:VirtualMachine` or `VC:Host`	Green

Table 2-2. Color Coding of Scripting Keywords (Continued)

Keyword Type	Text Color in Scripting Tab
Output text	Green
Workflow attributes	Pink
Workflow inputs	Pink
Workflow outputs	Pink

Add Parameters to Scripts

The Orchestrator scripting engine helps you to import available parameters into scripts.

If you have already defined parameters for the element you are editing, they appear as links in the **Scripting** tab toolbar.

Prerequisites

A scripted element is open for editing and its **Scripting** tab is open.

Procedure

1 Move the cursor to the appropriate position in a script in the scripting pad of the **Scripting** tab.

2 Click the parameter link in the **Scripting** tab toolbar.

Orchestrator inserts the parameter at the position of the cursor.

3 Insert a parameter with a null value into the script.

If you pass null values to primitive types such as integers, Booleans, and Strings, the Orchestrator scripting API automatically sets the default value for this argument.

You added parameters to the script.

What to do next

Add access to Java classes in scripts.

Accessing the Orchestrator Server File System from JavaScript and Workflows

Orchestrator limits access to the Orchestrator server file system from JavaScript and Workflows to specific directories.

JavaScript functions and workflows only have read, write, and execute permission in the permanent directory `c:\orchestrator`.

The Orchestrator administrator can modify the folders to which JavaScript functions and workflows have read, write, and execute access by setting a system property. See *Installing and Configuring VMware vCenter Orchestrator* for information about setting system properties.

JavaScript functions and workflows also have read, write, and execute permission in the server system default temporary I/O folder. Writing to the default temporary I/O folder is the only portable, guaranteed, and configuration-independent means of accessing the file system with full permissions. However, files that you write to the temporary I/O folder are lost when you reboot the server.

You obtain the default temporary I/O folder by calling the `System.getTempDirectory` method in JavaScript functions.

Access the Server File System Using the System.getTempDirectory Method

As an alternative to writing to the folders on the Orchestrator server system in which the administrator has set the appropriate permissions, you can write to the default temporary I/O folder.

Orchestrator has full read, write, and execute rights in the default temporary I/O folder by default. You obtain the default temporary I/O folder by using the `System.getTempDirectory` method in JavaScript functions

Procedure

- Include the following code line in JavaScript functions to access the `java.io.temp-dir` folder.

```
var tempDir = System.getTempDirectory()
```

Accessing Java Classes from JavaScript

By default, Orchestrator restricts JavaScript access to a limited set of Java classes. If you require JavaScript access to a wider range of Java classes, you must set an Orchestrator system property to allow this access.

By default, the Orchestrator JavaScript engine can access only the classes in the `java.util.*` package.

The Orchestrator administrator can allow access to other Java classes from JavaScript functions by setting a system property. See *Installing and Configuring VMware vCenter Orchestrator* for information about setting system properties.

Accessing Operating System Commands from JavaScript

The Orchestrator API provides a scripting class, `Command`, that runs commands in the Orchestrator server host operating system. To prevent unauthorized access to the Orchestrator server host, by default, Orchestrator applications do not have permission to run the `Command` class.

The Orchestrator administrator can allow access to the `Command` scripting class by setting the `com.vmware.js.allow-local-process=true` system property.

Exception Handling Guidelines

The Orchestrator implementation of the Mozilla Rhino JavaScript Engine supports exception handling, to allow you to process errors. You must use the following guidelines when writing exception handlers in scripts.

- Use the following European Computer Manufacturers Association (ECMA) error types. Use `Error` as a generic exception that plug-in functions return, and the following specific error types.
  - `TypeError`
  - `RangeError`
  - `EvalError`
  - `ReferenceError`
  - `URIError`
  - `SyntaxError`

  The following example shows a `URIError` definition.

```
try {
     ...
        throw new URIError("VirtualMachine with ID 'vm-0056'
                      not found on 'vcenter-test-1'") ;
```

```
    ...
} catch ( e if e instanceof URIError ) {

}
```

- All exceptions that scripts do not catch must be simple string objects of the form `<type>:SPACE<human readable message>`, as the following example shows.

  ```
  throw "ValidationError: The input parameter 'myParam' of type 'string' is too short."
  ```

- Write human readable messages as clearly as possible.
- Simple string exception type checking must use the following pattern.

  ```
  try {
        throw "VMwareNoSpaceLeftOnDatastore: Datastore 'myDatastore' has no space left" ;
  } catch ( e if (typeof(e)=="string" && e.indexOf("VMwareNoSpaceLeftOnDatastore:") == 0) ) {
        System.log("No space left on device") ;
        // Do something useful here
  }
  ```

- Simple string exception type checking, must use the following pattern in scripted elements in workflows.

  ```
  if (typeof(errorCode)=="string"
        && errorCode.indexOf("VMwareNoSpaceLeftOnDatastore:")
        == 0) {
        // Do something useful here
  }
  ```

Orchestrator JavaScript Examples

You can cut, paste, and adapt the Orchestrator JavaScript examples to help you write JavaScripts for common orchestration tasks.

- Basic Scripting Examples on page 134

  Workflow scripted elements, actions, and policies require basic scripting of common tasks. You can cut, paste, and adapt these examples into your scripted elements.

- Email Scripting Examples on page 135

  Workflow scripted elements can include scripting of common email-related tasks. You can cut, paste, and adapt these examples into your scripted elements.

- File System Scripting Examples on page 137

  Workflow scripted elements, actions, and policies require scripting of common file system tasks. You can cut, paste, and adapt these examples into your scripted elements.

- LDAP Scripting Examples on page 137

  Workflow scripted elements, actions, and policies require scripting of common LDAP tasks. You can cut, paste, and adapt these examples into your scripted elements.

- Logging Scripting Examples on page 138

  Workflow scripted elements, actions, and policies require scripting of common logging tasks. You can cut, paste, and adapt these examples into your scripted elements.

- Networking Scripting Examples on page 138

  Workflow scripted elements, actions, and policies require scripting of common networking tasks. You can cut, paste, and adapt these examples into your scripted elements.

- Workflow Scripting Examples on page 138

  Workflow scripted elements, actions, and policies require scripting examples of common workflow tasks. You can cut, paste, and adapt these examples into your scripted elements.

Basic Scripting Examples

Workflow scripted elements, actions, and policies require basic scripting of common tasks. You can cut, paste, and adapt these examples into your scripted elements.

Access XML Documents

The following JavaScript example allows you to access XML documents from JavaScript by using the ECMAScript for XML (E4X) implementation in the Orchestrator JavaScript API.

Note In addition to implementing E4X in the JavaScript API, Orchestrator also provides a Document Object Model (DOM) XML implementation in the XML plug-in. For information about the XML plug-in and its sample workflows, see the *Using vCenter Orchestrator Plug-Ins*.

```
var people = <people>
                <person id="1">
                    <name>Moe</name>
                </person>
                <person id="2">
                    <name>Larry</name>
                </person>
            </people>;

System.log("'people' = " + people);

// built-in XML type
System.log("'people' is of type : " + typeof(people));

// list-like interface System.log("which contains a list of " +
people.person.length() + " persons");
System.log("whose first element is : " + people.person[0]);

// attribute 'id' is mapped to field '@id'
people.person[0].@id='47';
// change Moe's id to 47
// also supports search by constraints
System.log("Moe's id is now : " + people.person.(name=='Moe').@id);

// suppress Moe from the list
delete people.person[0];
System.log("Moe is now removed.");

// new (sub-)document can be built from a string
people.person[1] = new XML("<person id=\"3\"><name>James</name></person>");
System.log("Added James to the list, which is now :");
for each(var person in people..person)

for each(var person in people..person){
    System.log("- " + person.name + " (id=" + person.@id + ")");
}
```

Setting and Obtaining Properties from a Hashtable

The following JavaScript example sets properties in a hashtable and obtains the properties from the hashtable. In the following example, the key is always a String and the value is an object, a number, a Boolean, or a String.

```
var table = new Properties() ;
table.put("myKey",new Date()) ;
// get the object back
var myDate= table.get("myKey") ;
System.log("Date is : "+myDate) ;
```

Replace the Contents of a String

The following JavaScript example replaces the content of a String and replaces it with new content.

```
var str1 = "'hello'" ;
var reg = new RegExp("(')", "g");
var str2 = str1.replace(reg,"\\'") ;
System.log(""+str2) ; // result : \'hello\'
```

Compare Types

The following JavaScript example checks whether an object matches a given object type.

```
var path = 'myurl/test';
if(typeof(path, string)){
  throw("string");
else {
  throw("other");
}
```

Run a Command in the Orchestrator Server

The following JavaScript example allows you to run a command line on the Orchestrator server. Use the same credentials as those used to start the server.

NOTE Access to the file system is limited by default. To access the file server from Orchestrator, see "Accessing the Orchestrator Server File System from JavaScript and Workflows," on page 131.

```
var cmd = new Command("ls -al") ;
cmd.execute(true) ;
System.log(cmd.output) ;
```

Email Scripting Examples

Workflow scripted elements can include scripting of common email-related tasks. You can cut, paste, and adapt these examples into your scripted elements.

When you run a mail workflow, it uses the default mail server configuration that you set in the Orchestrator configuration interface. You can override the default values by using input parameters, or by defining custom values in workflow scripted elements.

Obtain an Email Address

The following JavaScript example obtains the email address of the current owner of a running script.

```
var emailAddress = Server.getRunningUser().emailAddress ;
```

Send an Email

The following JavaScript example sends an email to the defined recipient, through an SMTP server, with the defined content.

```
var message = new EmailMessage() ;
message.smtpHost = "smtpHost" ;
message.subject= "my subject" ;
message.toAddress = "receiver@vmware.com" ;
message.fromAddress = "sender@vmware.com" ;
message.addMimePart("This is a simple message","text/html") ;
message.sendMessage() ;
```

Retrieve Email Messages

The following JavaScript example retrieves the messages of an email account, without deleting them, by using the scripting API provided by the `MailClient` class.

```
var myMailClient = new MailClient();

myMailClient.setProtocol(mailProtocol);
if(useSSL){
 myMailClient.enableSSL();
}

myMailClient.connect( mailServer, mailPort, mailUsername, mailPassword);
System.log("Successfully login!");

try {
    myMailClient.openFolder("Inbox");

    var messages = myMailClient.getMessages();
    System.log("Reading messages...!");
    if ( messages != null && messages.length > 0 ) {
      System.log( "You have " + messages.length + " email(s) in your inbox" );
     for (i = 0; i < messages.length; i++) {
      System.log("");
      System.log("-----MSG-------");
      System.log("Headers: ");
      var headerProp = messages[i].getHeaders();
      for each(key in headerProp.keys){
       System.log(key+": "+headerProp.get(key));
      }
      System.log("");

      System.log( "Message["+ i +"] with from: " + messages[i].from + " to: " + messages[i].to);
      System.log( "Message["+ i +"] with subject: " + messages[i].subject);
      var content = messages[i].getContent();
      System.log("Msg content as string: " + content);
     }
    } else {
      System.warn( "No messages found" );
    }
```

```
} finally {
    myMailClient.closeFolder();
    myMailClient.close();
}
```

File System Scripting Examples

Workflow scripted elements, actions, and policies require scripting of common file system tasks. You can cut, paste, and adapt these examples into your scripted elements.

NOTE Access to the file system is limited by default. To access the file server from Orchestrator, see "Accessing the Orchestrator Server File System from JavaScript and Workflows," on page 131.

Add Content to a Simple Text File

The following JavaScript example adds content to a text file.

```
var tempDir = System.getTempDirectory() ;
var fileWriter = new FileWriter(tempDir + "/readme.txt") ;
fileWriter.open() ;
fileWriter.writeLine("File written at : "+new Date()) ;
fileWriter.writeLine("Another line") ;
fileWriter.close() ;
```

Obtain the Contents of a File

The following JavaScript example obtains the contents of a file from the Orchestrator server host machine.

```
var tempDir = System.getTempDirectory() ;
var fileReader = new FileReader(tempDir + "/readme.txt") ;
fileReader.open() ;
var fileContentAsString = fileReader.readAll();
fileReader.close() ;
```

LDAP Scripting Examples

Workflow scripted elements, actions, and policies require scripting of common LDAP tasks. You can cut, paste, and adapt these examples into your scripted elements.

Convert LDAP Objects to Active Directory Objects

The following JavaScript example converts LDAP group elements to Active Directory user group objects, and the reverse.

```
var ldapGroup ;
// convert from ldap element to Microsoft:UserGroup object
var adGroup = ActiveDirectory.search("UserGroup",ldapGroup.commonName) ;
// convert back to LdapGroup element
var ldapElement = Server.getLdapElement(adGroup.distinguishedName) ;
```

Logging Scripting Examples

Workflow scripted elements, actions, and policies require scripting of common logging tasks. You can cut, paste, and adapt these examples into your scripted elements.

Persistent Logging

The following JavaScript example creates persistent log entries.

```
Server.log("This is a persistant message", "enter a long description here");
Server.warn("This is a persistant warning", "enter a long description here");
Server.error("This is a persistant error", "enter a long description here");
```

Non-Persistent Logging

The following JavaScript example creates non-persistent log entries.

```
System.log("This is a non-persistant log message");
System.warn("This is a non-persistant log warning");
System.error("This is a non-persistant log error");
```

Networking Scripting Examples

Workflow scripted elements, actions, and policies require scripting of common networking tasks. You can cut, paste, and adapt these examples into your scripted elements.

Obtain Text from a URL

The following JavaScript example accesses a URL, obtains text, and converts it to a string.

```
var url = new URL("http://www.vmware.com") ;
var htmlContentAsString = url.getContent() ;
```

Workflow Scripting Examples

Workflow scripted elements, actions, and policies require scripting examples of common workflow tasks. You can cut, paste, and adapt these examples into your scripted elements.

Return All Workflows Run by the Current User

The following JavaScript example obtains all workflow runs from the server and checks whether they belong to the current user. You can use this scripting with Webview components.

```
var allTokens = Server.findAllForType('WorkflowToken');
var currentUser = Server.getCredential().username;
var res = [];
for(var i = 0; i<res.length; i++){
    if(allTokens[i].runningUserName == currentUser){
      res.push(allTokens[i]);
    }
}
return res;
```

Access the Current Workflow Token

You can access the current workflow token by using the `workflow` variable. It is an object of type `WorkflowToken` that provides access to the current workflow run. The following JavaScript example gets the ID of the workflow token and its start date.

```
System.log("Current workflow run ID: " + workflow.id);
System.log("Current workflow run start date: "+workflow.startDate);
```

Schedule a Workflow

The following JavaScript example starts a workflow with a given set of properties, and then schedules it to start one hour later.

```
var workflowToLaunch = myWorkflow ;
// create parameters
var workflowParameters = new Properties() ;
workflowParameters.put("name","John Doe") ;
// change the task name
workflowParameters.put("__taskName","Workflow for John Doe") ;

// create scheduling date one hour in the future
var workflowScheduleDate = new Date() ;
var time = workflowScheduleDate.getTime() + (60*60*1000) ;
workflowScheduleDate.setTime(time) ; var scheduledTask =
workflowToLaunch.schedule(workflowParameters,workflowScheduleDate);
```

Run a Workflow on a Selection of Objects in a Loop

The following JavaScript example takes the array of virtual machines and runs a workflow on each one in a `For` loop. `VMs` and `workflowToRun` are workflow inputs.

```
var len=VMs.length;
for (var i=0; i < len; i++ )
{
 var VM = VMs[i];
 //var workflowToLaunch = Server.getWorkflowWithId("workflowId");
 var workflowToLaunch = workflowToRun;
 if (workflowToLaunch == null) {
 throw "Workflow not found";
 }
var workflowParameters = new Properties();
workflowParameters.put("vm",VM);
var wfToken = workflowToLaunch.execute(workflowParameters);
System.log ("Ran workflow on " +VM.name);
}
```

Developing Actions 3

Orchestrator provides libraries of predefined actions. Actions represent individual functions that you use as building blocks in workflows, Web views, and scripts.

Actions are JavaScript functions. They take multiple input parameters and have a single return value. They can call on any object in the Orchestrator API, or on objects in any API that you import into Orchestrator by using a plug-in.

When a workflow runs, an action takes its input parameters from the workflow's attributes. These attributes can be either the workflow's initial input parameters, or attributes that other elements in the workflow set when they run.

This chapter includes the following topics:

Reusing Actions

When you define an individual function as an action instead of coding it directly into a scriptable task workflow element, you expose it in the library. When an action is visible in the library, other workflows can use it.

When you define actions independently from the workflows that call on them, you can update or optimize the actions more easily. Defining individual actions also allows other workflows to reuse actions. When a workflow runs, Orchestrator caches each action only the first time that the workflow runs it. Orchestrator can then reuse the cached action. Caching actions is useful for recursive calls in a workflow, or fast loops.

You can duplicate actions, export them to other workflows or packages, or move them to a different module in the actions hierarchical list.

Access the Actions View

The Orchestrator client interface features an **Actions** view that provides access to the Orchestrator server's libraries of actions.

The **Actions** view of the Orchestrator client interface presents you with a hierarchical list of all the actions available in the Orchestrator server.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Design**.

2 Click the **Actions** view.

3 Browse the libraries of actions by expanding the nodes of the actions hierarchical list.

You can use the **Actions** view to view information about the actions in the libraries and create and edit actions.

Components of the Actions View

When you click an action in the actions hierarchical list, information about that action appears in the Orchestrator client's right pane.

The **Actions** view presents four tabs.

General	Displays general information about the action, including its name, its version number, the permissions, and a description.
Scripting	Shows the action's return types, input parameters, and the JavaScript code that defines the action's function.
Events	Shows all the events that this action encountered or triggered.
Permissions	Shows which users and user groups have permission to access this action.

Creating Actions

You can define individual functions as actions that other elements, such as workflows, can use. Actions are JavaScript functions with defined input and output parameters and permissions.

- Create an Action on page 142

  When you define an individual function as an action, instead of coding it directly into a scriptable task workflow element, you can expose it in the library for other workflows to use.

- Find Elements That Implement an Action on page 143

  If you edit an action and change its behavior, you might inadvertently break a workflow or application that implements that action. Orchestrator provides a function to find all of the actions, workflows, or packages that implement a given element. You can check whether modifying the element affects the operation of other elements.

- Action Coding Guidelines on page 144

  To optimize the performance of workflows and to maximize the potential to reuse actions, you should follow some basic coding guidelines when creating actions.

Create an Action

When you define an individual function as an action, instead of coding it directly into a scriptable task workflow element, you can expose it in the library for other workflows to use.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Design**.

2 Click the **Actions** view.

3 Expand the root of the actions hierarchical list and navigate to the module in which you want to create the action.

4 Right-click the module and select **Add action**.

5 Type a name for the action in the text box and click **OK**.

Your custom action is added to the library of actions.

6 Right-click the action and select **Edit**.

7 Click the **Scripting** tab.

8 To change the default return type, click the **void** link.

9 Add the action input parameters by clicking the arrow icon.

10 Write the action script.

11 Set the action permissions.

12 Click **Save and close**.

You created a custom action and added the action input parameters.

What to do next

You can use the new custom action in a workflow.

Find Elements That Implement an Action

If you edit an action and change its behavior, you might inadvertently break a workflow or application that implements that action. Orchestrator provides a function to find all of the actions, workflows, or packages that implement a given element. You can check whether modifying the element affects the operation of other elements.

IMPORTANT The **Find Elements that Use this Element** function checks all packages, workflows, and policies, but it does not check in scripts. Consequently, modifying an action might affect an element that calls this action in a script that the **Find Elements that Use this Element** function did not identify.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Design**.

2 Click the **Actions** view.

3 Expand the nodes of the actions hierarchical list to navigate to a given action.

4 Right-click the action and select **Find Elements that Use this Element**.

A dialog box shows all of the elements, such as workflows or packages, that implement this action.

5 Double-click an element in the list of results to show that element in the Orchestrator client.

You located all of the elements that implement an action.

What to do next

You can check whether modifying this element affects any other elements.

Action Coding Guidelines

To optimize the performance of workflows and to maximize the potential to reuse actions, you should follow some basic coding guidelines when creating actions.

Basic Action Guidelines

When you create an action, you must use basic guidelines.

- Every action must include a description of its role and function.
- Write short, elementary actions and combine them in a workflow.
- Avoid writing actions that perform multiple functions, because this limits the potential for reusing the action.
- Avoid actions that run for long periods of time. Instead, create a loop in the workflow and include a Waiting Event or Waiting Timer element after the action element.
- Do not write check points in actions. Workflows set a check point at the start and end of each element's run.
- Avoid writing loops in an action. Create loops in the workflow instead. If the server restarts, a running workflow resumes at its last check point, at the start of an element. If you write a loop inside an action and the server restarts while the workflow is running that action, the workflow resumes at the check point at the beginning of that action, and the loop starts again from the beginning.

Action Naming Guidelines

Use basic guidelines when you name actions.

- Write action names in English.
- Start action names with a lowercase letter. Use an uppercase letter at the beginning of each conjoined word in the name. For example, `myAction`.
- Make action names as explicit as possible, so that the function of the action is clear. For example, `backupAllVMsInPool`.
- Make module names as explicit as possible.
- Make module names unique.
- Use the inverse Internet address format for module names. For example, `com.vmware.myactions.myAction`.

Action Parameter Guidelines

Use basic guidelines when you write action parameter definitions.

- Write parameter names in English.
- Start parameter names with a lowercase letter.
- Make parameter names as explicit as possible.
- Preferably limit parameter names to a single word. If a name must contain more than one word, use an uppercase letter at the beginning of each conjoined word in the name. For example, `myParameter`.
- Use the plural form for parameters that represent an array of objects.
- Make variable names unambiguous, for example, `displayName`.
- Include a description for each parameter to describe its purpose.

- Do not use an excessive number of parameters in a single action.

Use Action Version History

You can use version history to revert an action to a previously saved state. You can revert the action state to an earlier or a later action version. You can also compare the differences between the current state of the action and a saved version of the action.

Orchestrator creates a new version history item for each action when you increase and save the action version. Subsequent changes to the action do not change the current version item. For example, when you create action version 1.0.0 and save it, the state of the action is stored in the database. If you make any changes to the action, you can save the action state in the Orchestrator client, but you cannot apply the changes to action version 1.0.0. To store the changes in the database, you must create a subsequent action version and save it. The version history is kept in the database along with the action itself.

When you delete an action, Orchestrator marks the element as deleted in the database without deleting the version history of the element from the database. This way, you can restore deleted actions. See "Restore Deleted Actions," on page 145.

Prerequisites

Open an action for editing.

Procedure

1 Click the **General** tab in the action editor.

2 Click **Show version history**.

A version history window appears.

3 Select an action version and click **Diff Against Current** to compare the differences.

A window that displays the differences between the current action version and the selected action version appears.

4 Select an action version and click **Revert** to restore the state of the action.

CAUTION If you have not saved the current action version, it is deleted from the version history and you cannot revert back to the current version.

The action state is reverted to the state of the selected version.

Restore Deleted Actions

You can restore actions that have been deleted from the library.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Design**.

2 Click the **Actions** view.

3 Navigate to the folder in which you want to restore a deleted action or actions.

4 Right-click the folder and select **Restore deleted actions**.

5 Select the action or actions that you want to restore and click **Restore**.

The action or actions appear in the selected folder.

Creating Resource Elements 4

Workflows and Web views can require as attributes objects that you create independently of Orchestrator. To use external objects as attributes in workflows or Web views, you import them into the Orchestrator server as resource elements.

Objects that workflows and Web views can use as resource elements include image files, scripts, XML templates, HTML files, and so on. Any workflows or Web views that run in the Orchestrator server can use any resource elements that you import into Orchestrator.

Importing an object into Orchestrator as a resource element allows you to make changes to the object in a single location, and to propagate those changes automatically to all the workflows or Web views that use this resource element.

You can organize resource elements into folders. The maximum size for a resource element is 16MB.

This chapter includes the following topics:

View a Resource Element

You can view existing resource elements in the Orchestrator client, to examine their contents and discover which workflows or Web views use this resource element.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Design**.

2 Click the **Resources** view.

3 Expand the hierarchical tree viewer to navigate to a resource element.

4 Click a resource element to show information about it in the right pane.

5 Click the **Viewer** tab to display the contents of the resource element.

6 Right-click the resource element and select **Find Elements that Use this Element**.

Orchestrator lists all the workflows and Web views that use this resource element.

What to do next

Import and edit a resource element.

Import an External Object to Use as a Resource Element

Workflows and Web views can require as attributes objects that you create independently of Orchestrator. To use external objects as attributes in workflows or Web views, you import them to the Orchestrator server as resource elements.

Prerequisites

Verify that you have an image file, script, XML template, HTML file, or other type of object to import.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Design**.

2 Click the **Resources** view.

3 Right-click a resource folder in the hierarchical list or the root and select **New folder** to create a folder in which to store the resource element.

4 Right-click the resource folder in which to import the resource element and select **Import resources**.

5 Select the resource to import and click **Open**.

Orchestrator adds the resource element to the folder you selected.

You imported a resource element into the Orchestrator server.

What to do next

Edit the general information of the resource element and set the user access permissions.

Edit the Resource Element Information and Access Rights

After you import an object into the Orchestrator server as a resource element, you can edit the resource element's details and permissions.

Prerequisites

Verify that you have imported an image, script, XML, or HTML file, or any other type of object into Orchestrator as a resource element.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Design**.

2 Click the **Resources** view.

3 Right-click the resource element and select **Edit**.

4 Click the **General** tab and set the resource element name, version, and description.

5 Click the **Permissions** tab and click the **Add access rights** icon () to define permissions for a user group.

6 Type a user group name in the **Filter** text box.

7 Select a user group and click **OK**.

8 Right-click the user group and select **Add access rights**.

9 Check the appropriate check boxes to set the level of permissions for this user group and click **OK**.

Permissions are not cumulative. To allow a user to view the resource element, use it in their workflows or Web views, and change the permissions, you must check all check boxes.

10 Click **Save and close** to exit the editor.

You edited the general information about the resource element and set the user access rights.

What to do next

Save the resource element to a file to update it, or add the resource element to a workflow or Web view.

Save a Resource Element to a File

You can save a resource element to a file on your local system. Saving the resource element as a file allows you to edit it.

You cannot edit a resource element in the Orchestrator client. For example, if the resource element is an XML configuration file or a script, you must save it locally to modify it.

Prerequisites

Verify that the Orchestrator server contains a resource element that you can save to a file.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Design**.

2 Click the **Resources** view.

3 Right-click the resource element and select **Save to file**.

4 Make the required modifications to the file.

You saved a resource element to a file.

What to do next

Update the resource element in the Orchestrator server.

Update a Resource Element

If a file or object that you have defined as a resource element changes, you can update the resource element in the Orchestrator server.

Prerequisites

Verify that you have imported an image, script, XML, or HTML file, or any other type of object into Orchestrator as a resource element.

Procedure

1 Modify the source file of the resource element in your local system.

2 From the drop-down menu in the Orchestrator client, select **Design**.

3 Click the **Resources** view.

4 Navigate through the hierarchical list to the resource element that you have updated.

5 Right-click the resource element and select **Update resource**.

6 (Optional) Click the **Viewer** tab to check that Orchestrator has updated the resource element.

You updated a resource element that the Orchestrator server contains.

Add a Resource Element to a Workflow

Resource elements are external objects that you can import to the Orchestrator server for workflows to use as attributes when they run. For example, a workflow can use an imported XML file that defines a map to convert one type of data to another, or a script that defines a function, when it runs.

Prerequisites

Verify that you have the following objects in your Orchestrator server:

- An image, script, XML, or HTML file, or any other type of object imported into Orchestrator as a resource element.
- A workflow that requires the resource element as an attribute.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Design**.

2 Click the **Workflows** view.

3 Expand the hierarchical tree viewer to navigate to the workflow that requires the resource element as an attribute.

4 Right-click the workflow and select **Edit**.

5 On the **General** tab, in the Attributes pane, click the **Add attribute** icon ( ).

6 Click the attribute name and type a new name for the attribute.

7 Click **Type** to set the attribute type.

8 In the **Select a type** dialog box, type `resource` in the **Filter** box to search for an object type.

Option	Action
Define a single resource element as an attribute	Select `ResourceElement` from the list.
Define a folder that contains multiple resource elements as an attribute	Select `ResourceElementCategory` from the list.

9 Click **Value** and type the name of the resource element or category of resource elements in the **Filter** text box.

10 From the proposed list, select the resource element or a folder containing resource elements and click **Select**.

11 Click **Save and close** to exit the editor.

You added a resource element or folder of resource elements as an attribute in a workflow.

Add a Resource Element to a Web View

Resource elements are external objects that you can import into the Orchestrator server for Web views to use as Web view attributes. Web view attributes identify objects with which Web view components interact.

Prerequisites

Verify that you have the following objects in your Orchestrator server:

- An image, script, XML, or HTML file, or any other type of object imported into Orchestrator as a resource element.
- A Web view that requires the resource element as an attribute.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Administer**.

2 Click the **Web Views** view.

3 If the Web view is running, right-click the Web view to which you want to add the resource element and select **Unpublish**.

4 Right-click the Web view and select **Edit**.

5 Click the **Attributes** tab.

6 Click the **Add attribute** icon ( ).

7 Click the attribute name and type a new name for the attribute.

8 Click **Type** to set the attribute type.

9 In the **Select a type** dialog box, type **`resource`** in the **Filter** box to search for an object type.

Option	Action
Define a single resource element as an attribute	Select `ResourceElement` from the list.
Define a folder that contains multiple resource elements as an attribute	Select `ResourceElementCategory` from the list.

10 Click **Value** and type the name of the resource element or category of resource elements in the **Filter** text box.

11 From the proposed list, select the resource element or a folder containing resource elements and click **Select**.

12 Click **Save and close** to exit the editor.

You added a resource element or folder of resource elements as an attribute in a Web view.

Creating Packages

5

Packages are used for transporting content from one Orchestrator server to another. Packages can contain workflows, actions, policies, Web views, configurations, or resources.

When you add an element to a package, Orchestrator checks for dependencies and adds any dependent elements to the package. For example, if you add a workflow that uses actions or other workflows, Orchestrator adds those actions and workflows to the package.

When you import a package, the server compares the versions of the different elements of its contents to matching local elements. The comparison shows the differences in versions between the local and imported elements. The administrator can decide whether to import the package, or can select specific elements to import.

Packages use digital rights management to control how the receiving server can use the contents of the package. Orchestrator signs packages and encrypts the packages for data protection. Packages can track which users export and redistribute elements by using X509 certificates.

IMPORTANT Packages that Orchestrator 3.2 generates are upwardly compatible with Orchestrator 4.x and 5.x. You can import a package from an Orchestrator 3.2 server to an Orchestrator 4.x or 5.x server. Packages from Orchestrator 4.x and 5.x are not backward compatible with Orchestrator 3.2. You cannot import a package generated by an Orchestrator 4.x or 5.x server to an Orchestrator 3.2 server.

For more information about using packages, see *Using the VMware vCenter Orchestrator Client*.

- Create a Package on page 154

  You can export workflows, policies, actions, plug-in references, resources, Web views, and configuration elements in packages. All elements that an element in a package implements are added to the package automatically, to ensure compatibility between versions. If you do not want to add the referenced elements, you can delete them in the package editor.

- Set User Permissions on a Package on page 155

  You set different levels of permission on a package to limit the access that different users or user groups can have to the contents of that package.

Create a Package

You can export workflows, policies, actions, plug-in references, resources, Web views, and configuration elements in packages. All elements that an element in a package implements are added to the package automatically, to ensure compatibility between versions. If you do not want to add the referenced elements, you can delete them in the package editor.

Prerequisites

Verify that the Orchestrator server contains elements such as workflows, actions, and policies that you can add to a package.

Procedure

1 From the drop-down menu in the Orchestrator client, select **Administer**.

2 Click the **Packages** view.

3 Right-click in the left pane and select **Add package**.

4 Type the name of the new package and click **Ok**.

The syntax for package names is *`domain.your_company.folder.package_name`*.

For example, `com.vmware.myfolder.mypackage`.

5 Right-click the package and select **Edit**.

The package editor opens.

6 On the **General** tab, add a description for the package.

7 On the **Workflows** tab, add workflows to the package.

- Click **Insert Workflows (list search)** to search for and select workflows in a selection dialog box.
- Click **Insert Workflows (tree browsing)** to browse and select folders of workflows from the hierarchical list.

8 On the **Policy Templates**, **Actions**, **Web View**, **Configurations**, **Resources**, and **Used Plug-Ins** tabs, add policy templates, actions, Web views, configuration elements, resource elements, and plug-ins to the package.

9 Click **Save and close** to exit the editor.

You created a package and added elements to it.

What to do next

Set user permissions for this package.

Set User Permissions on a Package

You set different levels of permission on a package to limit the access that different users or user groups can have to the contents of that package.

You can select the different users and user groups for which to set permissions from the users and user groups in the Orchestrator LDAP or vCenter Single Sign-On server. Orchestrator defines levels of permissions that you can apply to users or groups.

View	The user can view the elements in the package, but cannot view the schemas or scripting.
Inspect	The user can view the elements in the package, including the schemas and scripting.
Edit	The user can edit the elements in the package.
Admin	The user can set permissions on the elements in the package.

Prerequisites

Create a package, open it for editing in the package editor, and add the necessary elements to the package.

Procedure

1 Click the **Permissions** tab in the package editor.

2 Click the **Add access rights** icon () to define permissions for a new user or user group.

3 Search for a user or user group.

The search results show all of the users and user groups that match the search.

4 Select a user or user group.

5 Check the appropriate check boxes to set the level of permissions for this user and click **Select**.

To allow a user to view the elements, inspect the schema and scripting, run and edit the elements, and change the permissions, you must check all check boxes.

6 Click **Save and close** to exit the editor.

You created a package and set the appropriate user permissions.

Creating Plug-Ins by Using Maven 6

The Orchestrator Appliance provides a repository containing Maven artifacts, which you can use to create plug-in projects from archetypes.

The repository is hosted at `https://orchestrator_server:8281/vco-repo/` or `http://orchestrator_server:8280/vco-repo/`, in case your Maven version does not support the HTTPS protocol. This location is embedded in the `pom.xml` file of standard Orchestrator Maven plug-in projects.

This chapter includes the following topics:

Create an Orchestrator Plug-In with Maven from an Archetype

You can create a standard Orchestrator Maven plug-in from an archetype by running commands in the Windows Command Prompt.

Prerequisites

- Verify that you have installed Orchestrator Appliance 5.5.1 or later.
- Verify that you have installed Apache Maven 3.0.4 or 3.0.5.

Procedure

1 Create a project in interactive mode by choosing an archetype.

```
mvn archetype:generate -DarchetypeCatalog=https://orchestrator_server:8281/vco-repo/archetype-catalog.xml -DrepoUrl=https://orchestrator_server:8281/vco-repo -Dmaven.repo.remote=https://orchestrator_server:8281/vco-repo -Dmaven.wagon.http.ssl.insecure=true -Dmaven.wagon.http.ssl.allowall=true
```

2 (Optional) If you cannot access the repository over HTTPS, you can access it over HTTP. If you access the repository over HTTP or have a valid SSL certificate, you can create a project without using the `-Dmaven.wagon.http.ssl.allowall=true` flag.

```
mvn archetype:generate -DarchetypeCatalog=http://orchestrator_server:8280/vco-repo/archetype-catalog.xml -DrepoUrl=http://orchestrator_server:8280/vco-repo -Dmaven.repo.remote=http://orchestrator_server:8280/vco-repo -Dmaven.wagon.http.ssl.insecure=true
```

3 Navigate to the project directory and build the plug-in.

```
cd project_dir && mvn clean install –Dmaven.wagon.http.ssl.insecure=true –
Dmaven.wagon.http.ssl.allowall=true
```

If the build process is successful, the plug-in `.dar` file is generated in the DAR module's `target/` directory.

Maven Archetypes

You can use a set of predefined Maven archetypes as templates for developing Orchestrator plug-ins.

The following table describes the default Maven archetypes available in Orchestrator.

Table 6-1. Default Maven Archetypes

Archetype	Description
`com.vmware.o11n:o11n–plugin–archetype–simple`	A simple plug-in, which exposes a single scripting object and calls it.
`com.vmware.o11n:o11n–package–archetype`	A content-only Maven project, which can be used to keep packages in source form for better interaction with RCS, diff, post-processing, and so on.
`com.vmware.o11n:o11n–client–archetype–rest`	A simple command-line tool, which communicates with the Orchestrator REST API and calls a workflow.
`com.vmware.o11n:o11n–plugin–archetype–inventory`	A plug-in that demonstrates inventory use. The plug-in implements a repository, an adapter, and a factory for a single type. The inventory is stored in a file on a disk.
`com.vmware.o11n:o11n–archetype–inventory–annotation`	A plug-in whose `vso.xml` descriptor is generated on top of annotations.
`com.vmware.o11n:o11n–archetype–spring`	A plug-in that uses Spring-based SDK, provides a DI-enabled environment, and adds higher-level services in comparison to standard plug-in APIs.

Plug-In Development Best Practices

You can improve the process for delivering Orchestrator plug-ins created with Maven by performing a set of recommended activities.

Using a Repository Manager

If you are creating plug-ins in a larger organization, use an enterprise repository manager to set up the default Orchestrator Appliance repository to be added as a proxy repository. Using a central repository enables easier management and plug-in project collaboration. When you complete the first build in the new repository, the repository manager caches the artifacts from the Orchestrator Appliance repository and you can turn off the default repository.

Locking Workflows

After you verify that all workflows in your plug-in work as expected, lock them to prevent unauthorized modifications. Having locked workflows ensures that the basic functions of the plug-in cannot be compromised. If users need to modify a default workflow for a specific purpose, they can create a copy of the original workflow and edit it.

To produce release builds with locked workflows, pass the `–DallowedMask=vf` parameter to Maven.

Creating a Signing Certificate

Create a signing certificate, so that users can identify the plug-ins that you provide as trusted content.

If you do not create a signing certificate, the `.package` file is signed with the`archetype.keystore` file. Create a certificate by using the keytool from JDK and store the certificate in the keystore under the `_dunesrsa_alias_` alias. You can use the `–DkeystoreLocation=` and `–DkeystorePassword` parameters to provide the path to the keystore file and the password to Maven, or you can edit the `pom.xml` file to insert the values manually.

Index

X