Developing Workflows with VMware vRealize Orchestrator 8.x

February 2022

vRealize Orchestrator 8.2

vmware®

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

https://docs.vmware.com/

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

Contents

Developing Workflows with VMware vRealize Orchestrator

Developing Workflows with VMware vRealize Orchestrator provides information and instructions for developing custom VMware® vRealize Orchestrator workflows.

For more general information on the vRealize Orchestrator Client, see *Using the VMware vRealize Orchestrator Client*.

Intended Audience

This information is intended for developers who want to create custom vRealize Orchestrator workflows.

Developing Workflows

1

You develop workflows in the vRealize Orchestrator Client. Workflow development involves using the workflow editor, the vRealize Orchestrator APIs, and the JavaScript, Python, Node.js, and PowerShell scripting languages.

Note To use Python, Node.js, and PowerShell scripts in your vRealize Orchestrator Client, you must verify that your vRealize Orchestrator deployment uses a vRealize Suite or vRealize Automation license. For more information on the vRealize Orchestrator license-based feature enablement, see *vRealize Orchestrator Feature Enablement with Licenses* in *Installing and Configuring VMware vRealize Orchestrator*.

- Key Concepts of Workflows

  Workflows consist of a schema, workflow variables, and input and output 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 variables and parameters are the properties that workflows use to transfer data. vRealize Orchestrator saves a workflow token every time a workflow runs, recording the details of that specific workflow run.

- 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.

- Best Practices for Developing Workflows in a Clustered Environment

  You can follow best practices for developing vRealize Orchestrator workflows by multiple users and in a clustered environment.

- Using the Scripting API to Configure or Update a REST Host Authentication

  You can configure or update the OAuth 2.0 token sending strategy of a REST host by using either the query parameter or authorization header strategy.

- Creating and Editing Workflows

  You create and edit workflows in the vRealize Orchestrator Client workflow editor. The workflow editor is the Integrated Development Environment (IDE) of the vRealize Orchestrator Client for developing workflows.

- Workflow Editor Tabs

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

- Defining Variables and Parameters

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

- 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.

- (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.

- 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.

- 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.

- 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 might wait for a long time before it receives the response it requires, you can add long-running workflow elements to the workflow.

- Running Workflows

  A vRealize Orchestrator workflow runs according to a logical flow of events.

- Resuming a Failed Workflow Run

  If a workflow fails, vRealize Orchestrator provides an option to resume the workflow run from the last failed activity.

Key Concepts of Workflows

Workflows consist of a schema, workflow variables, and input and output 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 variables and parameters are the properties that workflows use to transfer data. vRealize Orchestrator saves a workflow token every time a workflow runs, recording the details of that specific workflow run.

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 runs. The user, an application, another workflow, or an action passes input parameters to a workflow for the workflow to process when it runs.

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

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. During a workflow run, the workflow 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 Variables

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

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

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

- Define variables when you create a workflow.
- Set the output parameter of a workflow element as a workflow variable.
- Inherit variables 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. You can configure the presentation of your workflows from the **Input Form** tab of the workflow editor in the vRealize Orchestrator Client.

Workflow Tokens

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

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 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 finished 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 variables 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 and out 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 variables.

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

7 Create the workflow presentation from the **Input Form** tab.

8 Validate the workflow.

Best Practices for Developing Workflows in a Clustered Environment

You can follow best practices for developing vRealize Orchestrator workflows by multiple users and in a clustered environment.

- Each developer has a dedicated test standalone vRealize Orchestrator instance for creating and developing workflows.
- Workflows are saved as maven projects on a shared source code control system.
- To ensure optimal performance of the vRealize Orchestrator production deployment, it is best to import workflows in a scheduled period.

Using the Scripting API to Configure or Update a REST Host Authentication

You can configure or update the OAuth 2.0 token sending strategy of a REST host by using either the query parameter or authorization header strategy.

Starting with vRealize Orchestrator 8.7, you can now use two different strategies for sending an OAuth 2.0 bearer access token when making an OAuth 2.0 authorized request. The default strategy, used in previous product versions, is to send the token in a `oauth_token` query parameter when making requests to the host. Using the query parameter strategy can introduce certain vulnerabilities, such as having the REST host server log the incoming requests.

Note The query parameter will be deprecated in a future vRealize Orchestrator release.

The newly introduced and recommended strategy is to use the Authorization header to send the token when making request to the host.

You can configure either token sending strategy by:

- Using the OAuth 2.0 tab used in built-in REST workflows such as **Add a REST Host** or **Update a REST Host**. For an example of using this option when running the **Add a REST Host** workflow, see Add a REST Host.
- Changing the authorization strategy by adding custom scripting to your workflow.

The scripting approach uses the following methods:

Token sending strategy	Scripting
Authorization header	`host.authentication = RESTAuthenticationManager.createAuthentication("OAuth 2.0", ["<token>", "Authorization header"]);`
Query parameter	`host.authentication = RESTAuthenticationManager.createAuthentication("OAuth 2.0", ["<token>", "Query parameter"]);`

Note The old scripting approach of creating an OAuth 2.0 authentication by passing only the token parameter without a token sending strategy still works, and for backwards compatibility preserves the past behavior of using the query parameter strategy.

```
host.authentication = RESTAuthenticationManager.createAuthentication("OAuth 2.0",
["<token>"]);
```

To demonstrate the scripting approach to managing your REST host authorization strategy, the following procedure presents a sample use case of creating a workflow which can be used for switching authorization strategies.

Procedure

1 Log in to the vRealize Orchestrator Client.

2 Navigate to **Library > Workflows** and select **New Workflow**.

3 Add a name for the workflow and click **Create**.

4 Navigate to the **Schema** tab and add a **Scriptable task** workflow element.

5 Add the following input and output parameters to the workflow element:

Input or Output	Name	Type
Input	**host**	**REST:RESTHost**
Input	**token**	**string**
Output	**result**	**REST:RESTHost**

6 Under the **Scripting** tab, add the following script:

```
var oldAuth = host.authentication
var ouath20type = "OAuth 2.0"
if (oldAuth.type !== ouath20type) {
    System.log ("REST host isn't using" + ouath20type);
    result = host;
} else {
    var oldStrategy = oldAuth.rawAuthProperties[1]
    if (oldStrategy === "Query parameter") {
        var newStrategy = "Authorization header";
    } else {
        var newStrategy = "Query parameter"
    }
```

```
    var newAuth = RESTAuthenticationManager.createAuthentication ("OAuth 2.0", [token,
newStrategy]);
    host.authentication = newAuth;
    result = RESTHostManager.updateHost(host);
}
```

Note In the preceding code sample, you can also use `oldAuth.getRawAuthProperty(1)` instead of `oldAuth.rawAuthProperties[1]`. Both function in the same way.

7 Save the workflow.

8 To change the token sending strategy of your REST host, run the workflow.

What to do next

You can verify what token sending strategy your REST host uses by navigating to **Inventory > REST-Host**, selecting your host, and checking the **Authorization** entry.

Creating and Editing Workflows

You create and edit workflows in the vRealize Orchestrator Client workflow editor. The workflow editor is the Integrated Development Environment (IDE) of the vRealize Orchestrator Client for developing workflows.

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
Summary	Provide general information about the workflow, such as the workflow name, description, tags, and version number. The **Summary** tab can also be used to configure the server restart behavior and workflow run failure behavior.
Variables	Define the variables for your workflow. Variables were previously known as attributes.
Input/Output	Define the input and output parameters of your workflow. The input parameters provide the data that the workflow processes during the workflow run. The output parameters are provided when the workflow run finishes.
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. By clicking an element in the schema diagram, you can define and edit the element's behavior.

Table 1-1. Workflow Editor Tabs (continued)

Tab	Description
Input Form	Define the layout of the user input dialog box that appears when users run a workflow. You arrange the parameters and variables into presentation steps and tabs 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. You can also add external validation for your workflow by using actions. For more information on the input form designer, see *vRealize Orchestrator Input Form Designer* in *Using the VMware vRealize Orchestrator Client*.
Version History	View and manage the version history of the workflow. Compare and restore versions and push and pull workflows to and from your integrated Git repository. For more information on using Git in the vRealize Orchestrator Client, see *How Can I Use Git Branching to Manage My vRealize Orchestrator Object Inventory* in *Using the VMware vRealize Orchestrator Client*.
Audit	View information about events related to the workflow such as, when it was saved, when a workflow run was performed, and when a workflow run was finished.

Defining Variables and Parameters

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

Workflow variables store data that workflows process internally. Unlike input and output parameters, variables values can be preset. Variables were previously known as attributes. 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 running.

- Create Workflow Parameters

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

  Workflow variables are the data that workflows process.

Create Workflow Parameters

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

You can create 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 finishes running.

Procedure

1 Log in to the vRealize Orchestrator Client.

2 Navigate to **Library > Workflows**, and select your workflow.

3 Select the **Inputs/Outputs** tab.

4 To add a parameter, click **New**.

5 Create a workflow parameter.

   a Select if you want to create an input or output parameter.

   b Enter a name.

   Note The parameter name must be a single word that can only contain letters, numbers, and the dollar ("$") and underscore ("_") symbols.

   c (Optional) Enter a description.

   d Set the parameter type.

   Note To create a composite type parameter, that includes multiple different parameter types, select **New Composite Type** from the **Type** text box.

   e (Optional) To create an array, click the **Array** check box.

   f To finish creating your parameter, click **Create**.

6 (Optional) Set a default value for your input parameters.

a Select the **Input Form** tab.

b Select your input parameter and click the **Values** tab.

c Select the value source for your input parameter, by expanding the **Default value** text box.

Value Source	Description
Constant	The default value source. Enter a constant default value for your input parameter.
Conditional value	Add one or more conditional expressions that define your input parameter value.
External source	Import an external action that configures the value of your input parameter. The filtering of available actions is done by parameter type.
Bind field	Bind the value of the select input parameter to another input parameter or variable.
Computed value	Configure the input parameter value by using operators. The type of available operators depends on the input parameter type. For example, for number type input parameters, you can select the arithmetic operators `Add`, `Subtract`, `Multiply`. **Note** **Compute value** is not available for SDK type input parameters, such as `AD:UserGroup`, `SSH:Host`, and so on.

7 (Optional) Configure output parameter exception handling.

a Select the **Schema** tab, and add a workflow element to the schema diagram.

b Add your output parameters to the workflow element.

c Expand the **Exception handling** menu item.

d Select a variable that you want to use to handle output parameter exceptions.

Results

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

Create Workflow Variables

Workflow variables are the data that workflows process.

Procedure

1 Log in to the vRealize Orchestrator Client.

2 Navigate to **Library > Workflows**, and select your workflow.

3 Select the **Variables** tab.

4 To add a variable, click **New**.

5 Create a workflow variable.

a Enter a name.

Note The variable name must be a single word that can only contain letters, numbers, and the dollar ("$") and underscore ("_") symbols.

b (Optional) Enter a description.

c Set the parameter type.

Note To create a composite type variable, that includes multiple different parameter types, select **New Composite Type** from the **Type** text box.

d (Optional) To create an array, click the **Array** check box.

e Add a value for your workflow variable.

f To finish creating your variable, click **Create**.

Results

You created a variable for the workflow.

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.

- 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.

- Schema Elements

  The workflow editor presents the workflow schema elements in menus on the **Schema** tab. You can use the schema elements available in the **Schema** tab to build a workflow.

- Schema Element Properties

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

- 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 variables.

- Decisions

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

- 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.

- 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 variables. 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.

- Add a Switch Activity to a Workflow

  You can add a basic switch activity to a workflow schema that defines the switch cases based on workflow variables or parameters.

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.

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

Procedure

1. Log in to the vRealize Orchestrator Client.
2. Navigate to **Library > Workflows**, and select your workflow.
3. Click the **Schema** tab.
4. Drag a schema element from the **Generic** menu in the left pane, to the workflow schema.

5 Click the element you dragged to the workflow schema and enter a name.

Note You should provide each element in your workflow with a unique name that describes its function. For example, your workflow might include two **Scriptable task** elements, one used to create a virtual machine, and another used to delete a virtual machine. In such a use case, you might want to name your elements, **Create VM** and **Delete VM**. You cannot rename the **End workflow** and **Throw exception** elements.

6 (Optional) Right-click an element in the schema and select either **Arrange vertically** or **Arrange horizontally** to manage the schema orientation around the specific element.

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

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

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

What to do next

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

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 variable that you have defined in the workflow editor. You can promote predefined variables only by replacing an input parameter with a variable of matching type.

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

Prerequisites

Open a workflow for editing in the workflow editor.

Procedure

1 Log in to the vRealize Orchestrator Client.

2 Navigate to **Library > Workflows**, and select your workflow.

3 Select the **Schema** tab.

4 Add a **Workflow element** or an **Action element** to the workflow schema.

5 Use the selected element to add a workflow or an action to the workflow.

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

```
No Inputs/Outputs were auto-binded. To create workflow-level variables/
parameters for other inputs/outputs, please promote them.
```

6 Click **Promote**.

7 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.
Variable	The argument is mapped to a workflow variable.

8 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.
Variable	The argument is mapped to a workflow variable.

9 Click **Promote**.

Results

You promoted parameters to the parent workflow.

Schema Elements

The workflow editor presents the workflow schema elements in menus on the **Schema** tab. You can use the schema elements available in the **Schema** tab to build a workflow.

Table 1-2. 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. A workflow can have only one start element. Start elements have one output and no input, and cannot be removed from the workflow schema.		Always present on the **Schema** tab
Scriptable task	General-purpose tasks you define. You write JavaScript functions in this element.		The **Generic** workflow palette

Table 1-2. Schema Elements and Icons (continued)

Schema Element Name	Description	Icon	Location in Workflow Editor
Decision	A 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 let the workflow 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.		The **Generic** workflow palette
Custom decision	A boolean function. Custom decisions can take several input parameters and process them according to custom scripts. Returns either `true` or `false`.		The **Generic** workflow palette
Decision activity	A boolean function. A decision activity runs a workflow and binds its output parameters to a `true` or a `false` path.		The **Generic** workflow palette
User interaction	Lets users pass new input parameters to 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. 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 must provide input. 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.		The **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.		The **Generic** workflow palette

Table 1-2. Schema Elements and Icons (continued)

Schema Element Name	Description	Icon	Location in Workflow Editor
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.		The **Generic** workflow palette
End workflow	The end point of a workflow. You can have multiple end elements in a schema, to represent the various 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.		The **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.		The **Generic** workflow palette
Workflow note	Lets you 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 workflow zones. Workflow notes provide only visual information, to help you understand the schema.		The **Generic** workflow palette
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.		The **Generic** workflow palette
Workflow element	Starts another workflow synchronously. When a workflow reaches a Workflow element in its schema, it runs that workflow as part of its own process. The original workflow continues only after the called workflow completes its run.		The **Generic** workflow palette
Foreach element	Runs a workflow on every element from an array. For example, you can run the Rename Virtual Machine workflow on all virtual machines from a folder.		The **Generic** workflow palette

Table 1-2. Schema Elements and Icons (continued)

Schema Element Name	Description	Icon	Location in Workflow Editor
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 complete.		The **Generic** workflow palette
Schedule workflow	Creates a task to run the workflow at a set time, and then the workflow continues its run.		The **Generic** workflow palette
Nested workflows	Starts several workflows simultaneously. You can decide 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 for all the nested workflows to complete before continuing its run.		The **Generic** workflow palette
Handle error	Handles an error for a specific workflow element. The workflow can handle the error by creating an exception, calling another workflow, or running a custom script.		The **Generic** workflow palette
Default error handler	Handles workflow errors that are not caught by standard error handlers. You can use any available schema elements to handle errors.		The **Generic** workflow palette

Table 1-2. Schema Elements and Icons (continued)

Schema Element Name	Description	Icon	Location in Workflow Editor
Switch	Switches to alternative workflow paths, based on a workflow variable or parameter.		The **Generic** workflow palette
Pre-Defined Task	Non-editable 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 ■ Send 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		The **Basic**, **Log**, and **Network** workflow palettes

Schema Element Properties

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

Using the Business Status Property for Schema Elements

You can define the business status of the workflow element from the **General** tab.

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.

Prerequisites

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

Procedure

1 Log in to the vRealize Orchestrator Client.

2 Navigate to **Library > Workflows**, and select your workflow.

3 Click the **Schema** tab.

4 Select an element to edit.

5 Click the **General** tab.

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

Schema Element Properties Tabs

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

Workflow Schema Element Properties Tabs

Certain workflow schema elements, such as the **Decision** element, do not include multiple tabs to manage their properties.

Schema Element Properties Tab	Description	Applies to Schema Element Type
General	Provide general information about the schema element, such as the name, description, input and output parameters, and exception handling.	■ **Scriptable task** ■ **Action element** ■ **Sleep** ■ **Change credential** ■ **Wait until date** ■ **Wait for custom event** ■ **Send custom event** ■ **Increase counter** ■ **Decrease counter** ■ **System log** ■ **System warning** ■ **System error** ■ **Server log** ■ **Server warning** ■ **Server error** ■ **System and Server log** ■ **System and Server warning** ■ **System and Server error** ■ **HTTP post** ■ **HTTP get**
Scripting	Enter the JavaScript script used in the schema element.	■ **Scriptable task** ■ **Action element** ■ **Sleep** ■ **Change credential** ■ **Wait until date** ■ **Wait for custom event** ■ **Send custom event** ■ **Increase counter** ■ **Decrease counter** ■ **System log** ■ **System warning** ■ **System error** ■ **Server log** ■ **Server warning** ■ **Server error** ■ **System and Server log** ■ **System and Server warning** ■ **System and Server error** ■ **HTTP post** ■ **HTTP get** ■ **Asynchrous element** ■ **Schedule workflow** ■ **Switch**

Schema Element Properties Tab	Description	Applies to Schema Element Type
Info	Has the same functionality as the **General** tab.	■ **Asynchrous element** ■ **Schedule workflow** ■ **Switch**
Cases	Define to which other element the **Switch** element switches to.	**Switch**

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 variables.

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.

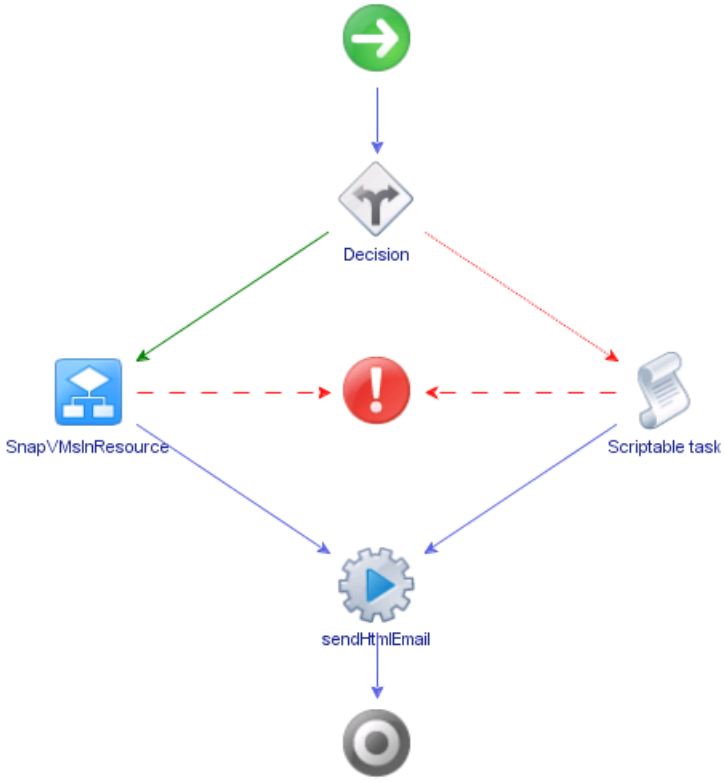
The standard path is the path that the workflow takes through the logical flow if all elements run as expected. The exception path is the path that the workflow takes through the logical flow if an element does not run as expected.

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 dashed arrow denotes the path that the workflow takes if a Boolean decision element returns `false`. Alternatively, 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.

Results

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 input and output bindings to bind incoming and outgoing data to workflow variables.

Data Flow of a Workflow

The data flow of a workflow is the manner in which the workflow element input and output parameters bind to workflow variables 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 variables that you set when you create the workflow, or by binding to an variable 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 variables that it creates. Other elements in the schema can bind to these new workflow variables as their input parameters. The workflow can generate the variables as its output parameters at the end of its run.

Element Bindings

You must bind all workflow input and output parameters. 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.

Input binding

Set a schema element's incoming data. For most schema elements, you can do this from the **Inputs/Outputs** menu of the **General** tab.

Output binding

Define the output parameters used when an element finishes its run. For most schema elements, you can do this from the **Inputs/Outputs** menu of the **General** tab.

Output Exception bindings

Link to exception handlers if the element encounters an exception when it runs.

Output exception bindings write values into the bound source parameter.

If the element changes the values of the input parameters that it receives when it runs, you must bind them to a workflow variable by using an output exception binding. Binding the element's output parameters to workflow variables 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 not to bind output parameter values to reflect the changes that the element makes to the workflow variables.

Important When you add an element that requires input and output parameters of a type that you have already defined in the workflow, vRealize Orchestrator sets the bindings to these parameters. You must verify that the parameters that vRealize 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. Log in to the vRealize Orchestrator Client.
2. Navigate to **Library > Workflows**, and select your workflow.
3. Select the **Schema** tab.
4. Select a workflow element.
5. On the **General** tab of the element configuration window, expand the **Inputs/Outputs** option.
6. Add the appropriate workflow input or output parameters from the list of existing parameters and variables.
7. To save your changes to the workflow schema, click **Save**.

Results

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

Results

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

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 when it encounters an error in that element.

Prerequisites

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

Procedure

1. Log in to the vRealize Orchestrator Client.
2. Navigate to **Library > Workflows**, and select your workflow.
3. Select the element for which you want to define exception binding.
4. On the **General** tab, expand the **Exception Handling** option.
5. Select a workflow variable you want to use for exception handling.
6. To finish editing the workflow, click **Save**.

Results

You defined how the element handles exceptions.

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 variables. 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 variables. 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 variables, you can run a workflow that iterates over these variables 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 variables 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 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 variables 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 must select the output parameters of the **Foreach element** in which to accumulate workflow outputs, so that the subworkflow can iterate over them as well.

Procedure

1. Log in to the vRealize Orchestrator Client.
2. Navigate to **Library > Workflows**, and select your workflow.
3. In the workflow editor, select the **Schema** tab.
4. From the **Generic** menu, drag a **Foreach element** in the workflow schema.
5. Create variables for the items in the array.
6. Under **Workflow**, select the workflow you want to add to the **Foreach element**.

   The input and output parameters of the selected workflow are added to the **Foreach element**.
7. Add an iteration error handler.
8. Add a variable you want to use for exception handling.

Results

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 variables that you have defined.

For parameters or variables 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.

Add a Switch Activity to a Workflow

You can add a basic switch activity to a workflow schema that defines the switch cases based on workflow variables or parameters.

Every switch activity can have multiple switch cases. Every switch case is defined by a condition related to an variable or a parameter. If the condition is fulfilled, the workflow run switches to a corresponding workflow element that you define. If none of the specified conditions are fulfilled, the workflow run switches to a default workflow element that you define.

Procedure

1. Log in to the vRealize Orchestrator Client.
2. Navigate to **Library > Workflows**, and select you workflow.
3. Drag a **Switch** element to the appropriate element in the workflow schema.
4. In the **Cases** tab, add or delete switch cases.
5. Define the condition for each switch case.
6. Select the corresponding workflow element for each switch case.
7. Select the default workflow element to switch to.
8. Click **Save**.

Results

You defined the switch case conditions and workflow paths.

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 variables for user interactions are `security.group` and `timeout.date`. When you set the `security.group` variable 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` variable, 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 (Optional) 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 required data.

2 (Optional) Set the User Interaction security.group Variable

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

3 (Optional) Set the timeout.date Variable to an Absolute Date

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

4 (Optional) 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.

5 (Optional) Set the timeout.date Variable to a Relative Date

You can set the `timeout.date` variable 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 (Optional) 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.

7 (Optional) 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.

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 required data.

Procedure

1. Log in to the vRealize Orchestrator Client.
2. Navigate to **Library > Workflows**, and select your workflow.
3. Select the **Schema** tab and drag a **User interaction** element to the appropriate position in the workflow schema.
4. (Optional) Enter a name and description for the user interaction.
5. (Optional) Under **Options**, configure the security group, security assignees, and timeout date variables for the user interaction.
6. To save your workflow, click **Save**.

Results

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.

Set the User Interaction security.group Variable

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

Prerequisites

Add a user interaction element to the workflow schema.

Procedure

1. Log in to the vRealize Orchestrator Client.
2. Navigate to **Library > Workflows**, and select your workflow.
3. Select the **Schema** tab.
4. Select the **User interaction** element.
5. Expand the **Options** menu.
6. Click **Select variable** for the `security.group` source parameter to set which users can respond to the user interaction.
7. Click **Create New**.
8. Name the parameter.
9. Under **Value**, enter the name of the LDAP user group you want to add to the element.

10 Click **Create**.

11 To save the changes to your workflow, click **Save**.

Results

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

What to do next

To configure the timeout period of the user interaction, set the `timeout.date` variable.

- To set the timeout to an absolute date and time, see Set the timeout.date Variable to an Absolute Date.
- 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.

Set the timeout.date Variable to an Absolute Date

You set the `timeout.date` variable 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.

Prerequisites

Add a user interaction element to the workflow schema.

Procedure

1 Log in to the vRealize Orchestrator Client.

2 Navigate to **Library > Workflows**, and select your workflow.

3 Select the **Schema** tab.

4 Select the **User interaction** element.

5 Expand the **Options** menu.

6 Click **Select variable** for the `timeout.date` source parameter to set which users can respond to the user interaction.

7 Click **Create New**.

8 Name the parameter.

9 Under **Value**, use the calendar to select an absolute date and time until which the workflow waits for the user to respond.

10 Click **Create**.

11 To save the changes to your workflow, click **Save**.

Results

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

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

Add a user interaction element to the workflow schema.

Procedure

1 Log in to the vRealize Orchestrator Client.

2 Navigate to **Library > Workflows**, and select your workflow.

3 Select the **Schema** tab.

4 Place a **Scriptable task** element before the workflow element that requires the relative `Date` object for its `timeout.date` variable.

5 Enter a name and description for the scripted workflow element.

6 Create a `Date` variable for the **Scriptable task** element.

   a Under **Inputs/Outputs**, create a variable.

   b Name the variable `timerDate`.

   c Select `Date` from the list of variable types.

   d Leave the value text box empty, because a scripted function provides this value.

   e Click **Save**.

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 To finish editing your workflow, click **Save**.

Results

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` variable. See Set the timeout.date Variable to a Relative Date.

Set the timeout.date Variable to a Relative Date

You can set the `timeout.date` variable 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` variable of a user interaction to this `Date` object. For example, if you bind the `timeout.date` variable 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.
- 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.

Procedure

1 Log in to the vRealize Orchestrator Client.

2 Navigate to **Library > Workflows**, and select your workflow.

3 Select the **Schema** tab.

4 Select your **User interaction** element.

5 Under **Options**, configure `timeout.date` to use the `timerDate` variable.

6 To finish editing your workflow, click **Save**.

Results

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

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.

Procedure

1. Log in to the vRealize Orchestrator Client.
2. Navigate to **Library > Workflows**, and select your workflow.
3. Select the **Schema** tab.
4. Select your **User interaction** element.
5. Create an exception handling variable.
   a. Under **Exception handling**, click **Select variable**.
   b. To create an exception handling variable, click **Create New**.
   c. Name the variable **`errorCode`**.
   d. Under Value, enter an appropriate error message.
   e. Click **Create**.
6. 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` variable from the user interaction.
7. Define the exception handling script.
   a. Enter an appropriate name for the **Scriptable task** element.

      For example, **`Log timeout`**.
   b. 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 vRealize Orchestrator log, write the following function:

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

8 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.

9 To finish editing your workflow, click **Save**.

Results

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.

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 input form designer of the user interaction element, not in the **Input Form** tab for the whole workflow. The **Input Form** tab of the whole workflow creates the layout of the input parameters dialog box that appears when you start a workflow. The input form designer 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.

Procedure

1 Log in to the vRealize Orchestrator Client.

2 Navigate to **Library > Workflows**, and select your workflow.

3 Select the **Schema** tab.

4 Select your **User interaction** element.

5 Under **Form Inputs**, click **Edit Input Form**.

6 Configure the appearance, values, and parameter constraints of the **User interaction** input parameters.

7 (Optional) To organize your input parameters under separate tabs, click **Add Tab**.

8 To finish editing your workflow, click **Save**.

Results

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

  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

  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

  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

  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

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

- 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 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

Managed by the **Workflow element** schema element. A workflow can start another workflow synchronously. The called workflow runs as a 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

Managed by the **Asynchronous element** schema element. 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

Managed by the **Schedule workflow** schema element. 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 under **Activity > Scheduled**.

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

Managed by the **Nested workflows** schema element.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. 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. As a result, the calling workflow can use the results of the nested workflows as input parameters when it runs its remaining elements.

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** schema element.

Procedure

1. Log in to the vRealize Orchestrator Client.
2. Navigate to **Library > Workflows**, and select your workflow.
3. Select the **Schema** tab.
4. Drag a **Workflow element** schema element from the **Generic** menu to the appropriate position in the workflow schema.
5. Under **Workflow**, select the workflow you want to call.
6. (Optional) If prompted, promote workflow input parameters or variables for use by the workflow.
7. Under **Inputs**, bind the required input parameters to the workflow.
8. Under **Outputs**, bind the required output parameters to the workflow.
9. Under **Exception handling**, define the exception behavior of the workflow.
10. To finish editing your workflow, click **Save**.

Results

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.

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.

Procedure

1. Log in to the vRealize Orchestrator Client.
2. Navigate to **Library > Workflows**, and select your workflow.
3. Select the **Schema** tab.

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

5 Under **Workflow**, select the workflow you want to call.

6 (Optional) If prompted, promote workflow input parameters or variables for use by the workflow.

7 Under **Inputs**, bind the required input parameters to the workflow.

8 Under **Options > Output Options**, bind the required output parameters to the workflow.

9 Under **Exception handling**, define the exception behavior of the workflow.

10 To finish editing your workflow, click **Save**.

Results

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.

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.

Procedure

1 Log in to the vRealize Orchestrator Client.

2 Navigate to **Library > Workflows**, and select your workflow.

3 Select the **Schema** tab.

4 Drag a **Schedule workflow** element from the **Generic** menu to the appropriate position in the workflow schema.

5 Under **Workflow**, select the workflow you want to call.

6 (Optional) If prompted, promote workflow input parameters or variables for use by the workflow.

7 Under **Inputs**, bind the required input parameters to the workflow.

8 Under **Options > Output Options**, bind the required output parameters to the workflow.

9 Under **Exception handling**, define the exception behavior of the workflow.

10 To finish editing your workflow, click **Save**.

Results

You scheduled a workflow to start at a given time and date from another 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 vRealize 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 vRealize Orchestrator server.
- All output parameters of the remote workflow must be resolvable on the local vRealize 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 vRealize Orchestrator servers. In case the remote workflow uses objects from a plug-in, the same plug-in must be available on both vRealize Orchestrator servers. The inventories of the remote plug-in and the local plug-in must be identical. In case the remote workflow uses library objects in vRealize Orchestrator, like workflows and actions, the same workflows and actions must exist in the inventories of the remote and the local vRealize 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 vRealize 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 vRealize Orchestrator server. Because the **Rename virtual machine** workflow runs on the remote vRealize 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 vRealize 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 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.

Procedure

1. Log in to the vRealize Orchestrator Client.
2. Navigate to **Library > Workflows**, and select your workflow.
3. Select the **Schema** tab.

4 Drag a **Nested workflows** element from the **Generic** menu to the appropriate position in the workflow schema.

5 Click **Add Workflow**, and add a workflow to the **Nested workflows** element.

6 Repeat for all the workflows you require.

7 To finish editing your workflow, click **Save**.

Results

You called several workflows simultaneously from a workflow.

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 **Run a workflow on a selection of objects** workflow. To access the workflow, navigate to **Library > Workflows** and enter the name of the workflow in the workflow search bar.
- Create a workflow that calls the **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 decide to run a workflow on a selection of objects depends on the specific workflow and can affect the performance of that 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 vRealize Orchestrator performs when it starts each element in a workflow run. As a consequence, if the vRealize Orchestrator

server stops while the scripted loop is running, when the server restarts, the workflow resumes at the beginning of the scripted element, repeating the whole loop. If the vRealize Orchestrator server stops while running a workflow with a loop of workflow elements, the workflow resumes at the specific element in the loop that was running when the server stopped.

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** or an **Action element** 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 variables.

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

   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 variable 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 To run on the objects, bind the `wf` input of the **Start workflows in a series** or **Start workflows in parallel** workflow to the workflow.

   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` variable defined in step 4.

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

Results

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 might 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 waiting period 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 including 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` variable 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 timer-based long-running workflow 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 in the left pane to the workflow schema, and place it before the element that requires the relative `Date` object for its `timeout.date` variable.
2. Click the **Scriptable task** element.
3. Enter a name and description for the scripted workflow element in the **Details** properties tab in the right pane.
4. On the **Inputs/Outputs** properties tab, create a workflow variable.
   a. Name the variable `timerDate`.
   b. Select `Date` from the list of variable types.

c Leave the variable value text box blank. A scripted function provides this value.

d Click **Create**.

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

6 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.

7 Click **Save** and confirm your selection.

Results

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 timer-based long-running workflow.

Create a Timer-Based Long-Running Workflow

If you know that a workflow has 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 in the left pane to the workflow schema, and position it where you want to suspend the workflow run.

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

2 Click the **Waiting timer** element.

3 Enter a description of the reason for implementing the timer in the **Details** properties tab in the right pane.

4 Click the **Inputs** tab.

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

5 Bind the `timer.date` parameter to an appropriate `Date` object.

   - 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 the **Select variable** text box, and then click **Create New**.

      The **New Variable** dialog box appears.

   b Enter a name and description for the variable.

   c Click the **Value** text box to set the variable value.

      A calendar appears.

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

   e Click **Create**.

7 Click **Save** and confirm your selection.

Results

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 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. The **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 vRealize 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 a variable 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 in the left pane to the workflow schema.

   One of the elements that precedes the **Scriptable task** must generate a `VC:Task` object as its output parameter.
2. Click the **Scriptable task** element.
3. Enter a name and description for the trigger in the **Details** properties tab in the right pane.
4. Click the **Inputs/Outputs** tab.
5. Select or create an input variable of the type **VC:Task**.

   This `VC:Task` object represents the time-consuming event that another workflow or element starts.
6. (Optional) Select or create an input parameter of the **Number** type to define a timeout period in seconds.
7. 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**.

8 Define any exception behavior in the **Exception handling** menu.

9 On the **Scripting** tab, define a function to generate a `Trigger` object.

For example, you can 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`.

10 Click **Save** and confirm your selection.

Results

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 that a workflow has to wait for a response from an outside source during its run, but do not know how long that wait is, 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 stops its run and waits 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 in the left pane to the workflow schema, and place it where you want to suspend the workflow run.

   The scriptable task that declares the trigger must immediately precede the **Waiting event** element.
2. Click the **Waiting event** element.
3. Enter a name and description of the waiting event in the **Details** properties tab in the right pane.
4. Click the **Inputs** tab.

   The `trigger.ref` parameter appears in the list of inputs.
5. Click the **Select variable** text box to bind the input parameter to an appropriate `Trigger` object.

   The `Trigger` object represents a trigger event that another workflow or workflow element defines.
6. Define any exception behavior in the **Exception handling** tab.
7. Click **Save** and confirm your selection.

Results

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.

Running Workflows

A vRealize 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 variables and input parameters to the input parameters of the schema element.
2. The schema element runs.
3. The output parameters of the schema element are copied to the workflow token variables and workflow output parameters.
4. The workflow token variables 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, vRealize Orchestrator stores workflow token variables in the database, and the next schema element starts running. If the workflow stops unexpectedly, the next time the vRealize 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, vRealize 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 output parameters of the workflow.

Resuming a Failed Workflow Run

If a workflow fails, vRealize 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.

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 creating a system property in the Control Center. See Set Custom Properties for Resuming Failed Workflow Runs.

Prerequisites

Verify that you have permissions to edit the workflow.

Procedure

1. Log in to the vRealize Orchestrator Client.
2. Navigate to **Library > Workflows**, and select your workflow.

3 On the **Summary** tab, select an option from the **Resume workflow 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.

4 Click **Save** and confirm your selection.

Set Custom Properties for Resuming Failed Workflow Runs

By default, vRealize Orchestrator is not set up to resume failed workflow runs. You can enable vRealize Orchestrator to resume failed workflow runs and set a custom timeout period after which failed workflow runs cannot be resumed.

Procedure

1 Log in to the vRealize Orchestrator Control Center as **root**.

2 Select **System Properties**, and click **New...**.

3 Under **Key**, enter **`com.vmware.vco.engine.execute.resume-from-failed.timeout-sec`**.

4 Under **Value**, enter the custom timeout period in seconds.

Note The value you set overrides the default timeout setting of 86400 seconds.

5 (Optional) Under **Description**, enter a description for the new system property.

6 Click **Add**.

Managing 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 resume failed workflow feature is enabled, you can change the parameters of the workflow and try to resume it. You can also retain the parameters and change 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.

In this use case, you create a simple workflow designed specifically to fail during its run. This workflow demonstrates the basic functionality of the resume failed workflow feature.

Procedure

1 Log in to the vRealize Orchestrator Client.

2 Navigate to **Library > Workflows**, and click **New Workflow**.

3 Enter the name **`resume workflow test`** and click **Create**.

4 On the **Summary** tab, click the **Resume workflow from failed behavior** drop-down menu.

5 Select **Enabled**.

6 On the **Inputs/Outputs** tab, click **New**.

7 Create an input parameter with the following properties.

Option	Description
Name	`fail`
Type	`boolean`

8 On the **Schema** tab, drag three **Scriptable task** elements into the schema diagram.

9 Name the schema elements **`before failure`**, **`failure`**, and **`after failure`**.

10 On the **Scripting** tab of the **`before failure`** element, enter **`System.log("Before failure");`**.

11 Add the `fail` input parameter to the **`failure`** element.

12 On the **Scripting** tab of the **`after failure`** element, enter **`System.log("After failure");`**.

13 Save the workflow, and click **Run**.

14 Toggle on the **fail** check-box, and click **Run**.

The workflow run reaches the **Waiting** state.

15 On the top-right, select **Answer**.

16 To cancel the workflow run, select **Cancel** and click **Answer**.

17 To resume the workflow run, select **Resume**, and navigate to the **Parameters** tab.

18 Toggle off the **fail** check-box, and click **Answer**.

The workflow run reaches the **Completed** state.

Results

You now know how to manage failed workflow runs.