# Universal Controller 6.9.x

# Variables and Functions

# Variables and Functions

## Variables



## Using Variables



## Functions



## Built-In Variables

The information on these pages also is located in the Universal Controller 6.9.x Variables and Functions.pdf.

# Variables and Functions Overview



## Variables and Functions

Variables and functions can be used in free-text fields within tasks and workflows. When a variable or function is specified in a free-text field, the Controller inserts its value into the field when the task or workflow is run.

Triggers can pass variables and functions into the tasks and workflows that they launch.

Additionally, email notifications for Controller resources (agents, OMS servers, and cluster nodes) can use Built-In Variables that are specific to that type of resource.

## Types of Variables

Universal Controller supports the following types of variables, all of which can be used in free text fields within tasks:

| | |
|---|---|
| User-Defined Variables | These variables are created by the user for use within:<br>• A single trigger, task, or workflow (that is a trigger-, task-, or workflow-specific variable ).<br>• All trigger, tasks, and workflows (that is, a Global variable). |
| Built-In Variables | These variables, maintained by the Controller, allow you to access information about task instances and other related data, such as task name, task status, and trigger name. |
| Functions | These variables calculate some value, such as current date and time, or perform some function, such as _replaceAll. |

## Setting Variables under Special Circumstances

The Controller also supports several features that allow you to set variables under special circumstances:

- Manually launch tasks and temporarily set user-defined variables.
- Manually launch all of the tasks associated with a trigger while supplying variable values used by the task(s) (see Triggering with Variables).
- Use the Set Variable action to set variables within a task or workflow.
- Use the ops-variable-set CLI function to set variables.

# User-Defined Variables



## Overview

User-defined Universal Controller variables are available for use in triggers, tasks, and Workflows.

You can define variables to be either:

- Available to a single trigger, task, or workflow; that is, **Local**.
- Available to all triggers, tasks, and workflows; that is, Global.

You define **Local** variables (variables specific to a single trigger, task, or workflow) on the **Variables** tab in the Details of that trigger, task, or workflow. These variables are stored in the ***ops_local_variable*** table.

You define Global variables either by:

- Selecting **Other > Variables** from the Automation Center navigation pane.
- Using the Set Variable action for a task or workflow.

Global variables are stored in the ***ops_variable*** table.

## Variable Naming Conventions

- Variable names must begin with a letter.
- Allowable characters are alphanumerics (upper or lower case), and underscore (_).
- White spaces are not permitted
- Variable names are not case-sensitive.

Warning

Do not define Controller variables with the prefix **ops**_. That prefix is reserved for built-in variables.

## Resolving User-Defined Variables

When the Controller creates a task instance from a task, it also resolves all variables specified in its free text fields. Because you can define variables at four different levels (trigger, task, workflow, and global), the Controller follows a prescribed formula to determine which variable takes precedence if duplicate variables have been defined. The general order of precedence, each of which may or may not exist in any given situation, is as follows:

1. Task trigger (highest precedence)
2. Task
3. Workflow trigger
4. Workflow
5. Global (lowest precedence)

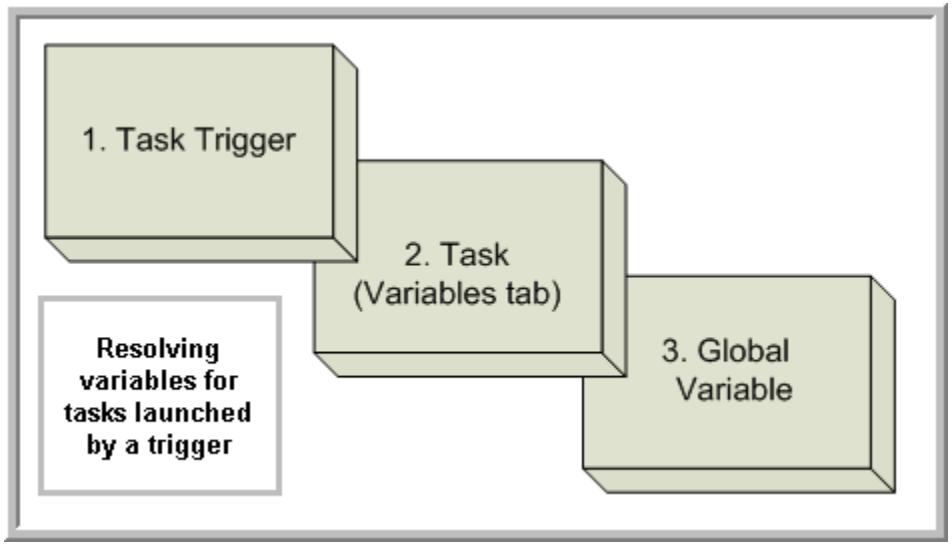
Note

You also can use the Set Variable Action of any task or workflow to define a variable. The Set Variable action explicitly states what scope you are setting the variable at, and under what circumstances.

The following scenarios provide more detailed information about how Controller variables are resolved.

## For Tasks Launched by a Trigger

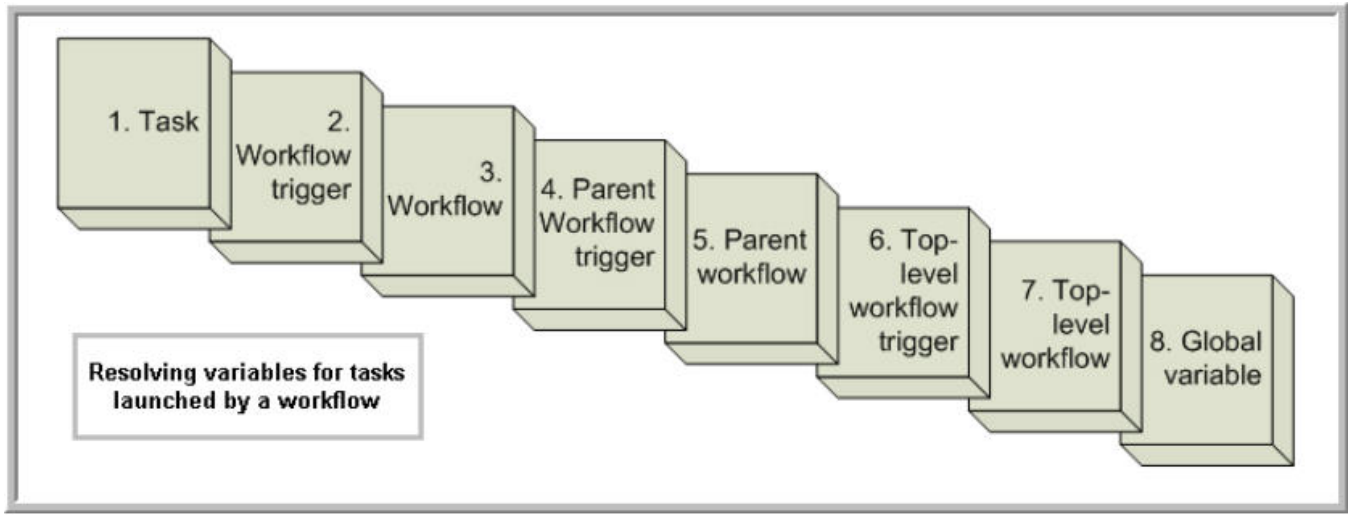
1. If the trigger defines the variable in the variables tab, that value is used to resolve the variable.
2. If the trigger does not define the variable, the value from the variable tab in the task Details is used.
3. If neither the trigger nor the task define the variable, the variable definition in the global variables table is used.
4. If the global variables table does not define the variable, the variable remains unresolved.




## For Tasks Launched by a Workflow

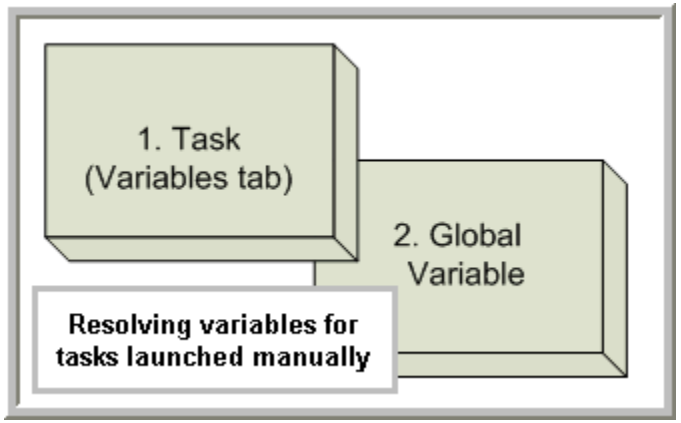
1. If the task defines the variable in the variables tab, that value is used to resolve the variable.
2. If the task does not define the variable, and the workflow was launched by a trigger, the value defined in the trigger is used.
3. If the workflow's trigger does not define the variable or the workflow was not launched by a trigger, the value defined in the workflow is used.
4. If the workflow does not define the variable, and there is a parent workflow, the value defined in the parent workflow's trigger is used.
5. If the parent workflow's trigger does not define the variable or if there is no trigger, the value defined in the parent workflow is used.
6. If the parent workflow does not define the variable, the Controller checks up a level for the trigger on the next parent workflow.

7. If that trigger does not define the variable, it checks for variables associated with the workflow. (This continues until the top level workflow is reached.)
8. If the top-level workflow does not define the variable, the variable definition in the global variables table is used.
9. If the global variables table does not define the variable, the variable remains unresolved.




## For Tasks Launched Manually

1. If the task defines the variable in the variables tab, that value is used to resolve the variable.
2. If the task does not define the variable, the variable definition in the global variables table is used.
3. If the global variables table does not define the variable, the variable remains unresolved.




# Format for Using Variables

When you enter a variable into a text field, precede the variable with the dollar sign ( $ ) and enclose the variable in curly braces ( { } ). You can enter a series of variables or nested variables.

Examples:

```
${variable_name}
${v1}${v2}
${${inner_variable}}
```

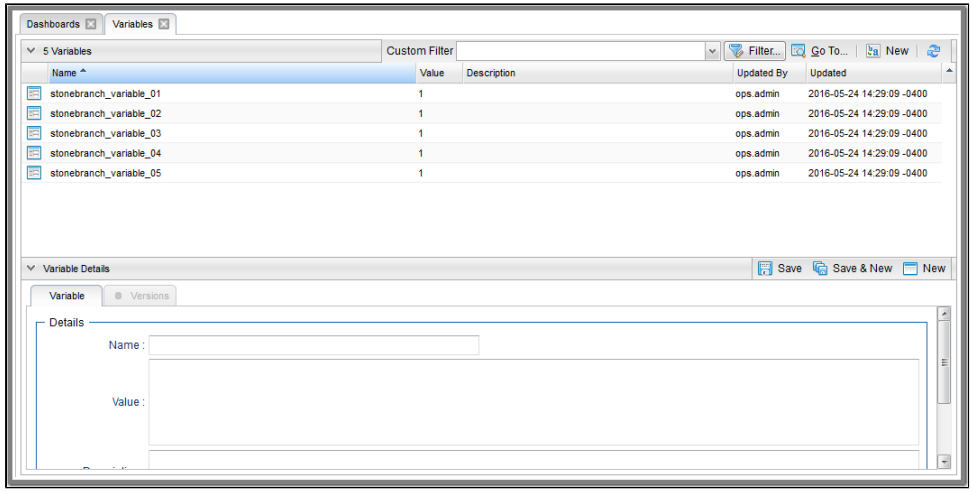
# Creating a Variable

You can create variables that are:

1. Available on a Global level; that is, available for all triggers, tasks, and Workflows.
2. Available only for a specific trigger, task, or Workflow.

# Creating a Global Variable

To create a Global variable that is available for all triggers, tasks, and Workflows:

| | |
|---|---|
| **Step 1** | From the Automation Center navigation pane, select **Other > Variables**. The Variables list displays a list of all Global variables. (You also can define a Global variable by using the Set Variable action for a task or workflow.)<br><br>Below the list, Variable Details for a new Global variable displays.<br><br>  |

| | |
|---|---|
| **Step 2** | Enter / select Details for a new Variable, using the field descriptions below as a guide.<br><br>• Required fields display in **boldface**.<br>• Default values for fields, if available, display automatically.<br><br>To display more of the Details fields on the screen, you can either:<br><br>• Use the scroll bar.<br>• Temporarily hide the list above the Details.<br>• Click the **New** button above the list to display a pop-up version of the Details. |
| **Step 3** | Click a **Save** button to save the record in the Controller database. |

Note

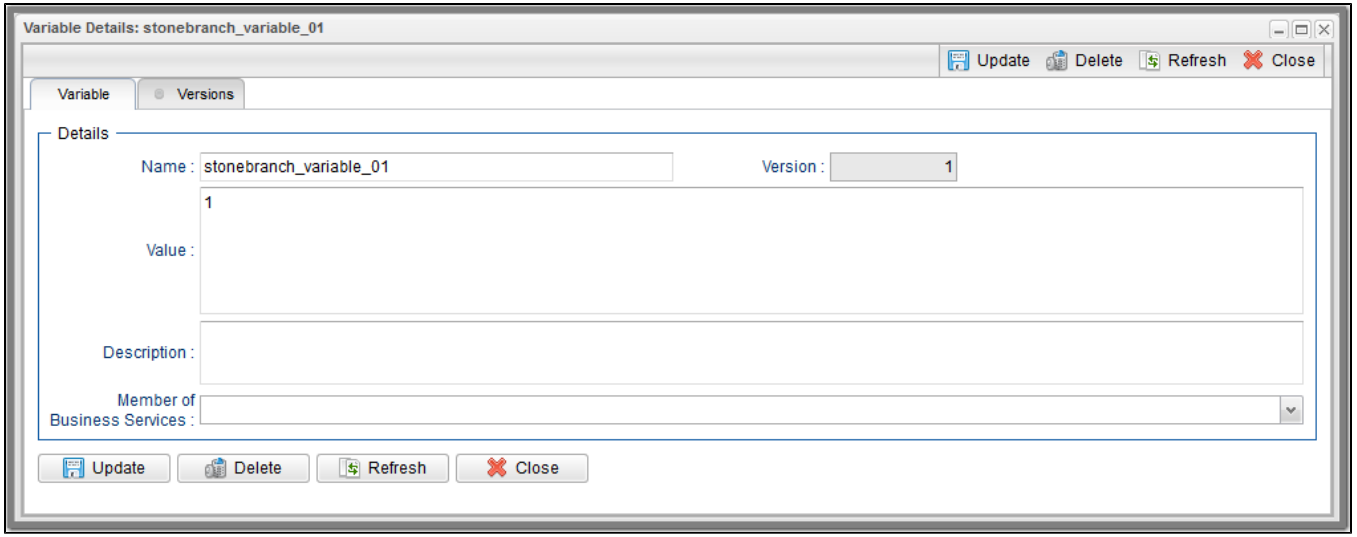
To open an existing record on the list, either:

- Click a record in the list to display its record Details below the list. (To clear record Details below the list, click the **New** button that displays above and below the Details.)
- Clicking the Details icon next to a record name in the list, or right-click a record in the list and then click **Open** in the Action menu that displays, to display a pop-up version of the record Details.
- Right-click a record in the a list, or open a record and right-click in the record Details, and then click **Open In Tab** in the Action menu that displays, to display the record Details under a new tab on the record list page (see Record Details as Tabs).

## Global Variable Details

The following Variable Details is for an existing Global Variable.

See the field descriptions below for a description of all fields that display in the Global Variable Details.




For information on how to access additional details - such as Metadata and complete database Details - for Variables (or any type of record), see Records.

## Global Variable Details Field Descriptions

The following table describes the fields and buttons in the Variables Details.

| Field Name | Description |
|---|---|
| Name | Name of the variable. Up to 128 alphanumerics. The name must begin with an alphabetic character and can consist of: alphas (a-z, A-Z), numerics 0-9, _ (underscore). White spaces are not permitted; names are not case-sensitive.<br><br>Important<br>Do not define variables with the prefix **ops**_. The **ops**_ prefix is reserved for built-in variables. |
| Version | System-supplied. The version number of the current record, which is incremented by the Controller every time a user updates a record. Click the Versions tab to view previous versions. For details, see Record Versioning. |
| Value | Value of the variable up to a maximum of 4000 characters. |
| Description | Optional. Description of this variable. |
| Member of Business Services | User-defined; allows you to select one or more Business Services that this record belongs to.<br><br>If the Business Service Visibility Restricted Universal Controller system property is set to true, depending on your assigned (or inherited) Permissions or Roles, Business Services available for selection may be restricted. |
| **Metadata** | This section contains Metadata information about this record. |
| UUID | Universally Unique Identifier of this record. |
| Updated By | Name of the user that last updated this record. |
| Updated | Date and time that this record was last updated. |
| Created By | Name of the user that created this record. |
| Created | Date and time that this record was created. |
| **Buttons** | This section identifies the buttons displayed above and below the Global Variable Details that let you perform various actions. |
| **Save** | Saves a new variable record in the Controller database. |
| **Save & New** | Saves a new record in the Controller database and redisplays empty Details so that you can create another new record. |
| **Save & View** | Saves a new record in the Controller database and continues to display that record. |
| **New** | Displays empty (except for default values) Details for creating a new record. |
| **Update** button | Saves updates to the record. |

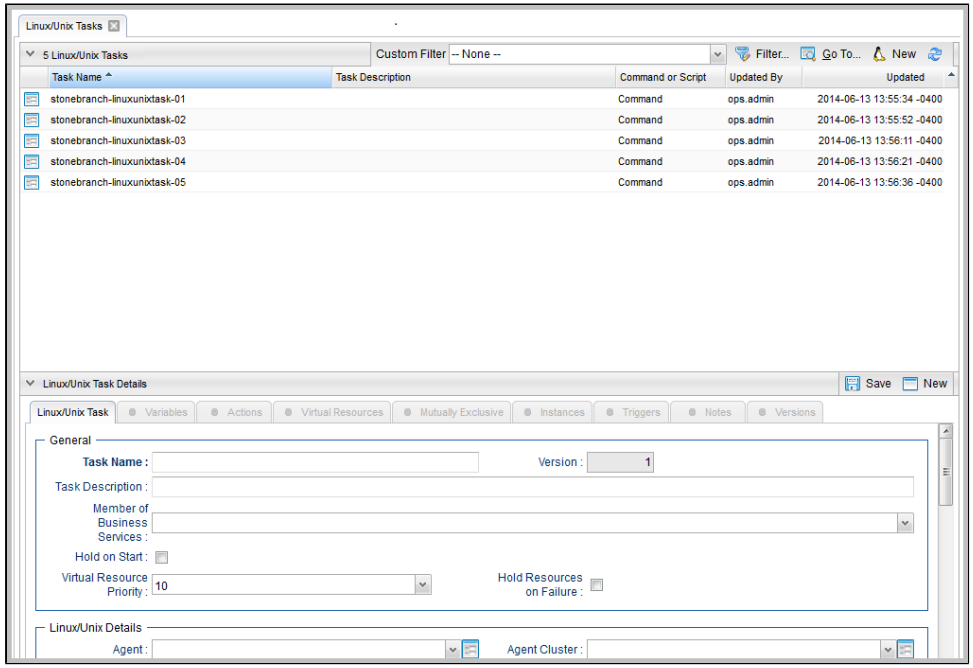
| | |
|---|---|
| **Delete** button | Deletes the current record. |
| **Refresh** | Refreshes any dynamic data displayed in the Details. |
| **Close** | For pop-up view only; closes the pop-up view of this task. |

# Creating a Variable Specific to a Trigger, Task, or Workflow

To create a variable that is specific to a single trigger, task, or Workflow:

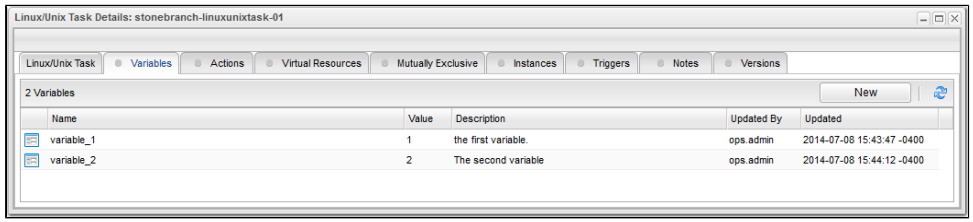
**Step 1**

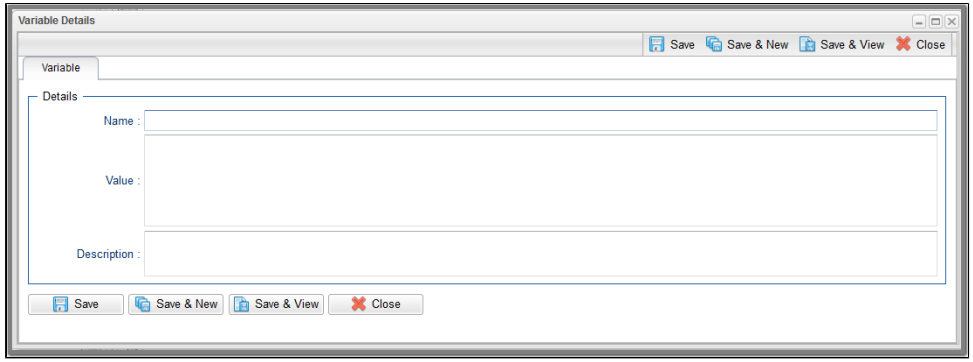
From the Automation Center navigation pane, select **Trigger ><trigger type>** or **Tasks ><task type>**. The records list for that trigger or task type displays. For example:




**Step 2**

Open a task on the list and click the **Variables** tab to display a list of any currently defined variables specific to that record.



| | |
|---|---|
| **Step 3** | Click the **New** button to display Variables Details for a new variable.<br>  |
| **Step 4** | Using the field descriptions provided for Global Variable Details as a guide, complete the fields as needed. |
| **Step 5** | Click the **Save** button, or right-click in the Details and click **Save**, to save the record. |
| **Step 6** | If appropriate, repeat these steps for any additional variables you want to add. |

# Automatically Incrementing a Variable

For example: To increment **`${counter}`**, use a Set Variable action to set **`${counter}`** with a value of **`${_trim(${__add("${counter}", "1")})}`**.

# Built-In Variables

# Overview

Built-in variables are maintained by Universal Controller and provide information about task instances, agents, Universal Message Service (OMS), and cluster nodes. They can be used in free text fields in triggers, tasks, task actions, and email notifications for agents, OMS servers, and cluster nodes.

Supported built-in variables and their descriptions are provided below. All built-in variables are prefixed with `ops_`.

# Built-In Variable Categories

Built-in variables are listed alphabetically within the following categories on this page:

# Agent Variables

The following agent variables can be used to pass information into an Agent notification.

## Agent Hostname

| | |
|---|---|
| **Description** | Resolves to the agent hostname. |
| **Syntax** | ${ops_agent_hostname} |
| **Example** | |

## Agent IP Address

| | |
|---|---|
| **Description** | Resolves to the agent IP address (see `${ops_agent_ip}`, below. |
| **Syntax** | ${ops_agent_ipaddr} |
| **Example** | |

## Agent IP Address

| | |
|---|---|
| **Description** | Resolves to the agent IP address. |
| **Syntax** | ${ops_agent_ip} |
| **Example** | |

## Agent Mode

| | |
|---|---|
| **Description** | Resolves to the agent operational mode (Active, Offline). |
| **Syntax** | ${ops_agent_mode} |
| **Example** | |

## Agent Name

| | |
|---|---|
| **Description** | Resolves to the agent name. |
| **Syntax** | ${ops_agent_name} |
| **Example** | |

### Agent Queue Name

| | |
|---|---|
| **Description** | Resolves to the agent queue name.<br><br>Note<br>In the user interface, the queue name is labelled **Agent Id**. |
| **Syntax** | ${ops_agent_id} |
| **Example** | |

Note

Although they have the same syntax, ${ops_agent_id}, this Agent Queue Name Agent variable resolves to a different value than the Agent sys_id Agent-based task instance variable.

## Agent-Based Task Instance Variables

The following variables can be used to pass agent information into agent-based task (Windows, Linux/Unix, z/OS, and SAP) notifications; see Creating Email Notifications and Creating SNMP Notifications.

### Agent Hostname

| | |
|---|---|
| **Description** | Resolves to the agent hostname. |
| **Syntax** | ${ops_agent_hostname} |
| **Example** | |

### Agent IP Address

| | |
|---|---|
| **Description** | Resolves to the agent IP address (see `${ops_agent_ip}`, below. |
| **Syntax** | ${ops_agent_ipaddr} |
| **Example** | |

### Agent IP Address

| | |
|---|---|
| **Description** | Resolves to the agent IP address. |
| **Syntax** | ${ops_agent_ip} |
| **Example** | |

### Agent Name

| | |
|---|---|
| **Description** | Resolves to the agent name. |
| **Syntax** | ${ops_agent_name} |
| **Example** | |

### Agent sys_id

| | |
|---|---|
| **Description** | Resolves to the sys_id of the agent. |
| **Syntax** | ${ops_agent_id} |
| **Example** | |

Note

Although they have the same syntax, ${ops_agent_id}, this Agent sys_id Agent-based task instance variable resolves to a different value than the Agent Queue Name Agent variable.

### Agent Queue Name

| | |
|---|---|
| **Description** | Resolves to the agent queue name.<br><br>Note<br>In the user interface, the queue name is labelled **Agent Id**. |
| **Syntax** | ${ops_agent_queue_name} |
| **Example** | |

## Agent Cluster Variables

The following agent cluster variables can be used to pass information into an Agent Cluster notification.

### Agent Cluster Name

| | |
|---|---|
| **Description** | Resolves to the agent cluster name. |
| **Syntax** | ${ops_agent_cluster_name} |
| **Example** | |

## Agent Cluster Distribution

| | |
|---|---|
| **Description** | Resolves to the Distribution type for the agent cluster. |
| **Syntax** | ${ops_agent_cluster_distribution} |
| **Example** | |

## Agent Cluster Task Execution Limit

| | |
|---|---|
| **Description** | Resolves to the type of Task Execution Limit for the agent cluster. |
| **Syntax** | ${ops_agent_cluster_limit_type} |
| **Example** | |

## Agent Cluster Suspended

| | |
|---|---|
| **Description** | Resolves to the current suspension status of the agent cluster. |
| **Syntax** | ${ops_agent_cluster_suspended} |
| **Example** | |

## Agent Cluster Task Execution Limit Amount

| | |
|---|---|
| **Description** | Resolves to the maximum number of tasks that can be running at the same time by Agents in this agent cluster. |
| **Syntax** | ${ops_agent_cluster_limit_max} |
| **Example** | |

## Agent Cluster Task Execution Current Limit

| | |
|---|---|
| **Description** | Resolves to the current number of tasks currently being run by the Agents in this agent cluster. |

| | |
|---|---|
| **Syntax** | ${ops_agent_cluster_limit_current} |
| **Example** | |

### Agent Cluster Network Alias

| | |
|---|---|
| **Description** | Resolves to the Network Alias of this agent cluster. |
| **Syntax** | ${ops_agent_cluster_network_alias} |
| **Example** | |

### Agent Cluster Network Alias Port

| | |
|---|---|
| **Description** | Resolves to the Agent Port of this agent cluster. |
| **Syntax** | ${ops_agent_cluster_network_alias_port} |
| **Example** | |

### Agent Cluster Notification State

| | |
|---|---|
| **Description** | Resolves to the Notification State for which the notification matched. |
| **Syntax** | ${ops_agent_cluster_notification_state} |
| **Example** | |

## Agent File Monitor Task Instance / Trigger Variables

When one or more tasks are launched by a Agent File Monitor trigger after the conditions in its associated Agent File Monitor task are met, the built-in variables described below are passed into the tasks being launched by the trigger.

For example, the Agent File Monitor trigger may specify the launch of a Windows task each time the associated Agent File Monitor task detects the creation of a specific file. The Windows task might use one of these built-in variables as a command argument. Or, if the Agent File Monitor task is not associated with a trigger but is running within a workflow, on completion you can propagate one or more of these built-in variable values to the parent workflow level using the Set Variable action. This allows you to pass information from the Agent File Monitor task to a successor task within the same workflow hierarchy.

### Base File Name

| | |
|---|---|
| **Description** | Resolves to the base file name. |

| | |
|---|---|
| **Syntax** | ${ops_trigger_file_name_simple} |

## File Directory

| | |
|---|---|
| **Description** | Resolves to the directory where the new file was created, but not the file itself. If the existence or non-existence of the final directory separator is a requirement, we recommend the use of `${ops_trigger_file_fullpath}` and `${ops_trigger_file_fullpath_no_separator}`, respectively. |
| **Syntax** | ${ops_trigger_file_path} |
| **Example** | |

## File Directory (with Final Directory Separator)

| | |
|---|---|
| **Description** | Resolves to the directory where the new file was created, but not the file itself; includes the final directory separator. |
| **Syntax** | ${ops_trigger_file_fullpath} |
| **Example** | |

## File Directory (without Final Directory Separator)

| | |
|---|---|
| **Description** | Resolves to the directory where the new file was created, but not the file itself; does not include the final directory separator. |
| **Syntax** | ${ops_trigger_file_fullpath_no_separator} |
| **Example** | |

## File Extension

| | |
|---|---|
| **Description** | Resolves to the file extension of a file. |
| **Syntax** | ${ops_trigger_file_name_extension} |
| **Example** | |

## Separator

| | |
|---|---|
| **Description** | Resolves to the separator appropriate to the platform where the agent is running. For Windows, resolves to a backslash ( \ ); for Linux/Unix, resolves to forward slash ( / ). This variable may be useful if you want to piece together a pathname using a combination of text and variables. |
| **Syntax** | ${ops_trigger_file_separator} |

| | |
|---|---|
| **Example** | `${ops_trigger_file_fullpath}sub_folder_name`<br>`${ops_trigger_file_separator}filename.txt` |

## Trigger File Date

| | |
|---|---|
| **Description** | Resolves to the file date of the file that fired the trigger. |
| **Syntax** | ${ops_trigger_file_date} |
| **Example** | |

## Trigger File Group

| | |
|---|---|
| **Description** | Resolves to the file group of the file that fired the trigger. |
| **Syntax** | ${ops_trigger_file_group} |
| **Example** | |

## Trigger File Name

| | |
|---|---|
| **Description** | Resolves to the name of the file that fired the trigger. |
| **Syntax** | ${ops_trigger_file_name} |
| **Example** | |

## Trigger File Name (No Path)

| | |
|---|---|
| **Description** | Resolves to the name of the file that fired the trigger, but without any path information. |
| **Syntax** | ${ops_trigger_file_name_nopath} |
| **Example** | |

## Trigger File Owner

| | |
|---|---|
| **Description** | Resolves to the file owner of the file that fired the trigger. |

| | |
|---|---|
| **Syntax** | ${ops_trigger_file_owner} |
| **Example** | |

### Trigger File Scan Result

| | |
|---|---|
| **Description** | Resolves to the result of the file scan: FOUND or NOT_FOUND. |
| **Syntax** | ${ops_trigger_file_scan} |
| **Example** | |

### Trigger File Size

| | |
|---|---|
| **Description** | Resolves to the file size of the file that fired the trigger. |
| **Syntax** | ${ops_trigger_file_size} |
| **Example** | |

## Application Monitor Trigger Variables

When a task is launched by an Application Monitor trigger, the following built-in variables are passed into the task being launched by the trigger:

### Trigger Application Name

| | |
|---|---|
| **Description** | Resolves to the name of the Application being monitored by the trigger. |
| **Syntax** | ${ops_trigger_appl_name} |
| **Example** | |

### Trigger Application Status

| | |
|---|---|
| **Description** | Resolves to the status of the Application being monitored by the trigger. |
| **Syntax** | ${ops_trigger_appl_status} |
| **Example** | |

### Trigger Application sys_id

| | |
|---|---|
| **Description** | Resolves to the sys_id of the application. |
| **Syntax** | ${ops_trigger_appl_id} |
| **Example** | |

### Trigger Application Type

| | |
|---|---|
| **Description** | Resolves to the type of Application being monitored by the trigger, as defined by the Application Type field. |
| **Syntax** | ${ops_trigger_appl_type} |
| **Example** | |

## Cluster Node Variables

The following cluster node variables allow you to pass information into a cluster node (Controller server) notification:

### Cluster Node Hostname

| | |
|---|---|
| **Description** | Resolves to the hostname of this cluster node. |
| **Syntax** | ${ops_cluster_hostname} |
| **Example** | `ops_cluster_hostname = MACHINEC19A` |

### Cluster Node ID

| | |
|---|---|
| **Description** | Resolves to the cluster node's internally-generated build ID. |
| **Syntax** | ${ops_cluster_id} |
| **Example** | `ops_cluster_id = MACHINEC19A:8080-uc` |

### Cluster Node IP Address

| | |
|---|---|
| **Description** | Resolves to the IP address of this cluster node. |
| **Syntax** | ${ops_cluster_ipaddr} |
| **Example** | `ops_cluster_ipaddr = 10.N.N.NN` |

## Cluster Node Mode

| | |
|---|---|
| **Description** | Resolves to the current mode of this cluster node: Offline, Active, Passive.<br>For more information, see Viewing Node Status. |
| **Syntax** | ${ops_cluster_mode} |
| **Example** | `ops_cluster_mode = Active` |

## Cluster Node Name

| | |
|---|---|
| **Description** | ${ops_cluster_name} is an alias for the ${ops_cluster_id} variable. |
| **Syntax** | ${ops_cluster_name} |
| **Example** | `ops_cluster_name = MACHINEC19A:8080-uc` |

## Cluster Node Running Time

| | |
|---|---|
| **Description** | Resolves to the numbers of days, hours, and minutes that this cluster node has been running since it was last started. |
| **Syntax** | ${ops_cluster_uptime} |
| **Example** | `ops_cluster_uptime = 7 Seconds` |

## Cluster Node Start Time

| | |
|---|---|
| **Description** | Resolves to the date and time the cluster node (server) was started. |

| | |
|---|---|
| **Syntax** | ${ops_cluster_start_time} |
| **Example** | `ops_cluster_start_time = 2011-09-26 17:35:01 -0400` |

# Common Variables

The following variable is available for Task Instances, Agents, OMS Servers, and Cluster Nodes.

## System Identifier

| | |
|---|---|
| **Description** | Resolves to the value of the System Identifier Universal Controller system property. |
| **Syntax** | ${ops_system_identifier} |
| **Example** | |

# Composite Trigger Variables

The following built-in variable is associated with the Composite Trigger type. This variable is only available for Composite Trigger components that have a Built-in Variable Prefix specified.

## Trigger Component Event Time

| | |
|---|---|
| **Description** | Resolves to the time when a Composite Trigger component fired. |
| **Syntax** | ${<prefix>_trigger_component_event_time} |
| **Example** | |

# Email Monitor Task Instance/Trigger Variables

When one or more tasks are launched by an Email Monitor trigger after the conditions in its associated Email Monitor task are met, the built-in variables described below are passed into the tasks being launched by the trigger.

For example, the Email Monitor trigger may specify the launch of an Email task each time the associated Email Monitor task detects the status in a Mailbox folder. The Windows task might use one of these built-in variables as a command argument. Or, if the Agent File Monitor task is not associated with a trigger but is running within a workflow, on completion you can propagate one or more of these built-in variable values to the parent workflow level using the Set Variable action. This allows you to pass information from the Email Monitor task to a successor task within the same workflow hierarchy.

## Body Field

| | |
|---|---|
| **Description** | Resolves to the Body field of the Email. |
| **Syntax** | ${ops_trigger_email_body} |
| **Example** | |

## Cc Field

| | |
|---|---|
| **Description** | Resolves to the Cc field of the Email. |
| **Syntax** | ${ops_trigger_email_cc} |
| **Example** | |

## From Field

| | |
|---|---|
| **Description** | Resolves to the From field of the Email. |
| **Syntax** | ${ops_trigger_email_from} |
| **Example** | |

## HTML Body Field

| | |
|---|---|
| **Description** | Resolves to the HTML Body field of the Email. |
| **Syntax** | ${ops_trigger_email_body_html} |
| **Example** | |

## Received Date Field

| | |
|---|---|
| **Description** | Resolves to the Received Date field of the Email. |
| **Syntax** | ${ops_trigger_email_received_date} |
| **Example** | |

## Reply To Field

| | |
|---|---|
| **Description** | Resolves to the Reply-To field of the Email. |
| **Syntax** | ${ops_trigger_email_reply_to} |
| **Example** | |

### Sent Date Field

| | |
|---|---|
| **Description** | Resolves to the Sent Date field of the Email. |
| **Syntax** | ${ops_trigger_email_sent_date} |
| **Example** | |

### Subject Field

| | |
|---|---|
| **Description** | Resolves to the Subject field of the Email. |
| **Syntax** | ${ops_trigger_email_subject} |
| **Example** | |

### To Field

| | |
|---|---|
| **Description** | Resolves to the To field of the Email. |
| **Syntax** | ${ops_trigger_email_to} |
| **Example** | |

## File Transfer Task Instance Variables

File Transfer variables are available for use in UDM scripts.

Note

These variables differ from all other built-in variables in that they are resolved by Universal Data Mover (UDM) on a UDM agent, not by the Universal Controller. File Transfer variables are sent to an agent unresolved and UDM performs all resolution for them. The resolved value is never available to the Controller.

Unlike the syntax of built-in variables resolved by Universal Controller - **`${<variable-name>}`** - the syntax of File Transfer variables is the same as all UDM variables - **`$(<variable-name>)`**.

The following example illustrates the correct way to code them:

```
open src=srcserver user=$(ops_src_cred_user) pwd=$(ops_src_cred_pwd) dst=dstserver user=$(ops_dst_cred_user) pwd=$(ops_dst_cred_pwd)
```

### Destination Password

| | |
|---|---|
| **Description** | Resolves to the destination password. |
| **Syntax** | $(ops_dst_cred_pwd) |
| **Example** | |

### Destination User ID

| | |
|---|---|
| **Description** | Resolves to the destination user ID. |
| **Syntax** | $(ops_dst_cred_user) |
| **Example** | |

### Source Password

| | |
|---|---|
| **Description** | Resolves to the source password. |
| **Syntax** | $(ops_src_cred_pwd) |
| **Example** | |

### Source User ID

| | |
|---|---|
| **Description** | Resolves to the source user ID. |
| **Syntax** | $(ops_src_cred_user) |
| **Example** | |

## OMS Server Variables

The following OMS Server variables allow you to pass information into an OMS Server notification.

## Last OMS Server Connected

| | |
|---|---|
| **Description** | Resolves to the last OMS Server connected to the Controller in an OMS HA cluster. |
| **Syntax** | ${ops_oms_last_connected} |
| **Example** | |

## Last OMS Server Connected Time

| | |
|---|---|
| **Description** | Resolves to the last time that the OMS Server connected to the Controller in an OMS HA cluster. |
| **Syntax** | ${ops_oms_last_connected_time} |
| **Example** | |

## OMS Server IP Address

| | |
|---|---|
| **Description** | Resolves to the OMS Server IP address. |
| **Syntax** | ${ops_oms_server_address} |
| **Example** | |

## OMS Server Status

| | |
|---|---|
| **Description** | Resolves to the current status of the OMS Server. |
| **Syntax** | ${ops_oms_status} |
| **Example** | |

## OMS Server sys_id

| | |
|---|---|
| **Description** | Resolves to the sys_id of the OMS server. |
| **Syntax** | ${ops_oms_id} |
| **Example** | |

## OMS Server Messaging Sessions Status

| | |
|---|---|
| **Description** | Resolves to the current status of the OMS Server messaging sessions (heartbeat, input, output): Operational, Impaired, None. |
| **Syntax** | ${ops_oms_session_status} |
| **Example** | |

# PeopleSoft Task and Task Instance Variables

The following built-in variables are available for PeopleSoft tasks and task instances:

## Distribution Status

(For task instances only.)

| | |
|---|---|
| **Description** | Resolves to the PeopleSoft task instance Distribution Status. |
| **Syntax** | ${ops_distribution_status} |
| **Example** | |

## Main Job Name

| | |
|---|---|
| **Description** | Resolves to the PeopleSoft Main Job Name. |
| **Syntax** | ${ops_main_job_name} |
| **Example** | |

## Main Schedule Name

| | |
|---|---|
| **Description** | Resolves to the PeopleSoft task/task instance Main Schedule Name. |
| **Syntax** | ${ops_main_schedule_name} |
| **Example** | |

## Process Instance

(For task instances only.)

| | |
|---|---|
| **Description** | Resolves to the PeopleSoft task instance Process Instance. |
| **Syntax** | ${ops_process_instance} |

| | |
|---|---|
| **Example** | |

### Process Name

| | |
|---|---|
| **Description** | Resolves to the PeopleSoft task/task instance Process/Job Name. |
| **Syntax** | ${ops_process_name} |
| **Example** | |

### Process Type

| | |
|---|---|
| **Description** | Resolves to the PeopleSoft task/task instance Process Type. |
| **Syntax** | ${ops_process_type} |
| **Example** | |

### Run Status

(For task instances only.)

| | |
|---|---|
| **Description** | Resolves to the PeopleSoft task instance Run Status. |
| **Syntax** | ${ops_run_status} |
| **Example** | |

## Recurring Task Instance Variables

The following built-in variables are available for Recurrent tasks and task instances:

### Next Recurrence Time

| | |
|---|---|
| **Description** | Resolves to the time when the Recurrence task next runs. |
| **Syntax** | ${ops_next_recurrence_time} |
| **Example** | |

### Recurrence Count

| | |
|---|---|
| **Description** | Resolves to the current count of task recurrences. |
| **Syntax** | ${ops_recurrence_count} |
| **Example** | |

### Source Instance ID

| | |
|---|---|
| **Description** | Resolves to the ID of the Recurrence task instance. |
| **Syntax** | ${ops_source_instance_id} |
| **Example** | |

### Source Instance Name

| | |
|---|---|
| **Description** | Resolves to the name of the Recurrence task instance. |
| **Syntax** | ${ops_source_instance_name} |
| **Example** | |

### Target Task ID

| | |
|---|---|
| **Description** | Resolves to the ID of the target task. |
| **Syntax** | ${ops_target_task_id} |
| **Example** | |

### Target Task Name

| | |
|---|---|
| **Description** | Resolves to the name of the target task. |
| **Syntax** | ${ops_target_task_name} |
| **Example** | |

## Remote File Monitor Task Instance Variables

The following built-in variables are available for Remote File Monitor task instances and provide information about the file or file(s) that matched the monitor's criteria.

You can use these variables in a Remote File Monitor action or in a successor task instance by propagating one or more of these built-in variable values to a parent workflow using the Set Variable action.

## Base Trigger File Name

| | |
|---|---|
| **Description** | Resolves to the base file name. |
| **Syntax** | ${ops_trigger_file_name_simple} |
| **Example** | |

## Files Matching Wildcard

| | |
|---|---|
| **Description** | Resolves to a comma-separated list of files that matched the wildcard, if one was specified in the Remote Filename field in the Remote File Monitor task. |
| **Syntax** | ${ops_trigger_files} |
| **Example** | `ops_trigger_files = COMPANY-2011-11-22.xls, COMPANY-2011-11-23.xls,COMPANY-2011-11-24.xls` |

## Remote Trigger File Name

| | |
|---|---|
| **Description** | Resolves to the remote file name. |
| **Syntax** | ${ops_trigger_file_name} |
| **Example** | |

## Remote Trigger File Name (No Path)

| | |
|---|---|
| **Description** | Resolves to the remote file name without any path information. |
| **Syntax** | ${ops_trigger_file_name_nopath} |
| **Example** | |

## Trigger File Directory

| | |
|---|---|
| **Description** | Resolves to the directory where the remote file is located, but not the file itself. ${ops_trigger_file_path} is an alias for ${ops_trigger_file_fullpath_no_separator}. |

| | |
|---|---|
| **Syntax** | ${ops_trigger_file_path} |
| **Example** | |

## Trigger File Directory (with Final Directory Separator)

| | |
|---|---|
| **Description** | Resolves to the directory where the remote file is located, but not the file itself; includes the final directory separator. |
| **Syntax** | ${ops_trigger_file_fullpath} |
| **Example** | |

## Trigger File Directory (without Final Directory Separator)

| | |
|---|---|
| **Description** | Resolves to the directory where the remote file is located, but not the file itself; does not include the final directory separator. |
| **Syntax** | ${ops_trigger_file_fullpath_no_separator} |
| **Example** | |

## Trigger File Extension

| | |
|---|---|
| **Description** | Resolves to the file extension of the file. |
| **Syntax** | ${ops_trigger_file_name_extension} |
| **Example** | |

## Trigger Wildcard

| | |
|---|---|
| **Description** | Resolves to the contents of the Remote Filename field in the Remote File Monitor task. |
| **Syntax** | ${ops_trigger_wildcard} |
| **Example** | `ops_trigger_wildcard = /home/prod/stonebranch/COMPANY*.xls` |

## Trigger Wildcard Path Only

| | |
|---|---|
| **Description** | Resolves to the path only, with the final slash but without the file name, from the Remote Filename field in the Remote File Monitor task. |

| | |
|---|---|
| **Syntax** | ${ops_trigger_wildcard_path} |
| **Example** | `ops_trigger_wildcard_path = /home/prod/stonebranch/` |

### Trigger Wildcard Path Only (without Final Slash)

| | |
|---|---|
| **Description** | Resolves to the path only, without the final slash and without the file name, from the Remote Filename field in the Remote File Monitor task. |
| **Syntax** | ${ops_trigger_wildcard_path_no_separator} |
| **Example** | `ops_trigger_wildcard_path_no_separator = /home/prod/stonebranch` |

## SAP Task Instance Variables

For an SAP task instance, where applicable, the following built-in variables resolve to the SAP jobname and SAP jobid of the job running in the SAP system. If you need to use the SAP jobname and/or the SAP jobid from one SAP task instance in a successor SAP task instance, you can use the Set Variable action to propagate these built-in variable values to the parent workflow.

### SAP InfoPackage Request ID

| | |
|---|---|
| **Description** | Resolves to the SAP InfoPackage Request ID. |
| **Syntax** | ${ops_sap_requestid} |
| **Example** | |

### SAP Job ID

| | |
|---|---|
| **Description** | Resolves to the SAP job ID. |
| **Syntax** | ${ops_sap_jobid} |
| **Example** | |

### SAP Job Name

| | |
|---|---|
| **Description** | Resolves to the SAP job name. |

| | |
|---|---|
| **Syntax** | ${ops_sap_jobname} |
| **Example** | |

### SAP Process Chain ID

| | |
|---|---|
| **Description** | Resolves to the SAP Process Chain ID. |
| **Syntax** | ${ops_sap_chainid} |
| **Example** | |

### SAP Process Chain Log ID

| | |
|---|---|
| **Description** | Resolves to the SAP Process Chain Log ID. |
| **Syntax** | ${ops_sap_logid} |
| **Example** | |

## SQL and Stored Procedure Task Instance Variables

The following built-in variables are used in SQL tasks and Stored Procedure tasks to collect SQLException data, if any:

### Error Message

| | |
|---|---|
| **Description** | Resolves to any error message generated by the database. |
| **Syntax** | ${ops_sql_error_msg} |
| **Example** | |

### Processed Rows

| | |
|---|---|
| **Description** | Resolves to the number of rows processed. |
| **Syntax** | ${ops_sql_rows} |
| **Example** | |

### Return Code for SQL Statement Outcome

| | |
|---|---|
| **Description** | Resolves to a return code that indicates the outcome of the most recently executed SQL statement. |
| **Syntax** | ${ops_sql_state} |
| **Example** | |

# SQL Task Instance Variables

The following built-in variable is available for SQL task instances.

## SQL Command Field

| | |
|---|---|
| **Description** | Resolves to the value of the SQL Command field. |
| **Syntax** | ${ops_sql_command} |
| **Example** | |

# Stored Procedure Task Instance Variables

The following built-in variable is available for Stored Procedure task instances and provides information about the stored procedure itself.

## Stored Procedure Name

| | |
|---|---|
| **Description** | Resolves to the value from the Stored Procedure Name field. |
| **Syntax** | ${ops_stored_proc_name} |
| **Example** | |

# System Monitor Task Instance Variables

The following System Monitor variables show the results for **Resource Available** and **Actual Available** that can be utilized in System Monitor tasks.

## Actual Size

| | |
|---|---|
| **Description** | Actual size determined by the agent. |
| **Syntax** | ${ops_sm_actual_size} |
| **Example** | |

### Actual Size (Rounded)

| | |
|---|---|
| **Description** | Same as ops_sm_actual_size, except rounded to the nearest integer. |
| **Syntax** | ${ops_sm_actual_int_size} |
| **Example** | |

### Actual Size (Scale)

| | |
|---|---|
| **Description** | Scale of the actual size determined by the agent. |
| **Syntax** | ${ops_sm_actual_scale} |
| **Example** | |

### Scale

| | |
|---|---|
| **Description** | Scale specified in the **By Scale** field for **Resource Available** of the System Monitor task definition. |
| **Syntax** | ${ops_sm_scale} |
| **Example** | |

### Size

| | |
|---|---|
| **Description** | Size specified in the **Resource Available** field of the System Monitor task definition. |
| **Syntax** | ${ops_sm_size} |
| **Example** | |

### Size (Rounded)

| | |
|---|---|
| **Description** | Same as ops_sm_size, except that ops_sm_int_size is rounded to the nearest integer. |
| **Syntax** | ${ops_sm_int_size} |
| **Example** | |

## Task Instance Variables

The following built-in variables are associated with task instances for all task types.

## Cluster Node Hostname

| | |
|---|---|
| **Description** | Resolves to the hostname of the Active cluster node. |
| **Syntax** | ${ops_cluster_hostname} |
| **Example** | `ops_cluster_hostname = MACHINEC19A` |

## Cluster Node ID

| | |
|---|---|
| **Description** | Resolves to the Active cluster node's internally-generated build ID. |
| **Syntax** | ${ops_cluster_id} |
| **Example** | `ops_cluster_id = MACHINEC19A:8080-uc` |

## Cluster Node IP Address

| | |
|---|---|
| **Description** | Resolves to the IP address of the Active cluster node. |
| **Syntax** | ${ops_cluster_ipaddr} |
| **Example** | `ops_cluster_ipaddr = 10.N.N.NN` |

## Cluster Node Mode

| | |
|---|---|
| **Description** | Resolves to the current mode of the cluster node: Offline, Active, Passive.<br>For more information, see Viewing Node Status. |
| **Syntax** | ${ops_cluster_mode} |

| | |
|---|---|
| **Example** | `ops_cluster_mode = Active` |

## Cluster Node Name

| | |
|---|---|
| **Description** | ${ops_cluster_name} is an alias for the ${ops_cluster_id} variable. |
| **Syntax** | ${ops_cluster_name} |
| **Example** | `ops_cluster_name = MACHINEC19A:8080-uc` |

## Cluster Node Running Time

| | |
|---|---|
| **Description** | Resolves to the numbers of days, hours, and minutes that the Active cluster node has been running since it was last started. |
| **Syntax** | ${ops_cluster_uptime} |
| **Example** | `ops_cluster_uptime = 7 Seconds` |

## Cluster Node Start Time

| | |
|---|---|
| **Description** | Resolves to the date and time the Active cluster node (server) was started. |
| **Syntax** | ${ops_cluster_start_time} |
| **Example** | `ops_cluster_start_time = 2011-09-26 17:35:01 -0400` |

## Command

| | |
|---|---|
| **Description** | For tasks that launch a command on a Windows or Linux/Unix machine; resolves to the task command. |
| **Syntax** | ${ops_cmd} |

| | |
|---|---|
| **Example** | |

## Command Parameters

| | |
|---|---|
| **Description** | For tasks that launch a command on a Windows or Linux/Unix machine; resolves to the task command parameters. |
| **Syntax** | ${ops_cmd_parms} |
| **Example** | |

## Custom Field 1

| | |
|---|---|
| **Description** | Resolves to the value of user-defined field #1. |
| **Syntax** | ${ops_custom_field1} |
| **Example** | |

## Custom Field 2

| | |
|---|---|
| **Description** | Resolves to the value of user-defined field #2. |
| **Syntax** | ${ops_custom_field2} |
| **Example** | |

## Description

| | |
|---|---|
| **Description** | Resolves to the value of the Task Description field. |
| **Syntax** | ${ops_description} |
| **Example** | |

## Duration

| | |
|---|---|
| **Description** | Resolves to the task instance Duration. |
| **Syntax** | ${ops_duration_text} |
| **Example** | ops_duration_text = 2 Minutes 10 Seconds |

## Duration In Seconds

| | |
|---|---|
| **Description** | Resolves to the task instance Duration In Seconds. |
| **Syntax** | ${ops_duration} |
| **Example** | ops_duration = 130000 |

## End Time

| | |
|---|---|
| **Description** | Resolves to the task ending time. |
| **Syntax** | ${ops_end_time} |
| **Example** | |

## End Time: Average Estimated

| | |
|---|---|
| **Description** | Resolves to the Average Estimated End Time in the server's time zone. |
| **Syntax** | ${ops_avg_estimated_end_time} |
| **Example** | ${ops_avg_estimated_end_time} > 2018-10-16 15:01:45 -0400 |

## End Time: Highest Estimated

| | |
|---|---|
| **Description** | Resolves to the Highest Estimated End Time in the server's time zone. |
| **Syntax** | ${ops_highest_estimated_end_time} |
| **Example** | ${ops_highest_estimated_end_time} > 2018-10-16 15:01:45 -0400 |

## End Time: Lowest Estimated

| | |
|---|---|
| **Description** | Resolves to the Lowest Estimated End Time in the server's time zone. |
| **Syntax** | ${ops_lowest_estimated_end_time} |
| **Example** | ${ops_lowest_estimated_end_time} > 2018-10-16 15:01:44 -0400 |

## End Time: User Estimated

| | |
|---|---|
| **Description** | Resolves to the User Estimated End Time in the server's time zone. |
| **Syntax** | ${ops_user_estimated_end_time} |
| **Example** | ${ops_user_estimated_end_time} > 2018-10-16 15:01:54 -0400 |

## Execution User ID

| | |
|---|---|
| **Description** | Resolves to the ID of the user who launched the task or to the ID of the user who enabled the trigger that launched the task. |
| **Syntax** | ${ops_execution_user} |
| **Example** | |

## Instance Number

| | |
|---|---|
| **Description** | Resolves to the sequentially assigned number, maintained per task, representing the creation order of the instance. For example, if you launch a task twice, the first task instance will have instance number 1, and the second task instance will have instance number 2. |
| **Syntax** | ${ops_instance_number} |
| **Example** | |

## Launch Time

| | |
|---|---|
| **Description** | Resolves to the task launch time. For workflows, all descendants will have the same launch time as the top-level workflow. |
| **Syntax** | ${ops_launch_time} |
| **Example** | |

## Maximum Retry Count

| | |
|---|---|
| **Description** | Resolves to the maximum retry count. |
| **Syntax** | ${ops_retry_maximum} |
| **Example** | |

## Parent Workflow Instance sys_id

| | |
|---|---|
| **Description** | Resolves to the sys_id of the parent workflow task instance. |

| | |
|---|---|
| **Syntax** | ${ops_workflow_id} |
| **Example** | |

## Parent Workflow Name

| | |
|---|---|
| **Description** | Resolves to the name of the parent workflow. |
| **Syntax** | ${ops_workflow_name} |
| **Example** | |

## Queued Time

| | |
|---|---|
| **Description** | Resolves to the date and time that the task was queued for processing. |
| **Syntax** | ${ops_queued_time} |
| **Example** | |

## Reference Id

| | |
|---|---|
| **Description** | Resolves to the sequentially assigned number, maintained per task, representing the creation order of the instance. For example, if you launch a task twice, the first task instance will have instance number 1, and the second task instance will have instance number 2.<br><br>Note<br>Although it still is supported, the Reference Id built-in variable has been superseded by the Instance Number built-in variable. |
| **Syntax** | ${ops_task_ref_count} |
| **Example** | |

## Retry Count

| | |
|---|---|
| **Description** | Resolves to the current retry count. |
| **Syntax** | ${ops_retry_count} |
| **Example** | |

## Retry Interval

| | |
|---|---|
| **Description** | Resolves to the retry interval (seconds). |
| **Syntax** | ${ops_retry_interval} |
| **Example** | |

## Script ID

| | |
|---|---|
| **Description** | For Windows, Linux/Unix, and SAP tasks where a Script or SAP Definition from Scripts is specified; resolves to the Controller system ID of the script. |
| **Syntax** | ${ops_script_id} |
| **Example** | |

## Script Name

| | |
|---|---|
| **Description** | For Windows, Linux/Unix, and SAP tasks where a Script or SAP Definition from Scripts is specified; resolves to the Controller name of the script. |
| **Syntax** | ${ops_script_name} |
| **Example** | |

## Script Parameters

| | |
|---|---|
| **Description** | For tasks that run a script on a Windows or Linux/Unix machine; resolves to the task script parameters. |
| **Syntax** | ${ops_script_parms} |
| **Example** | |

## Starting Time

| | |
|---|---|
| **Description** | Resolves to the task starting time. |
| **Syntax** | ${ops_start_time} |
| **Example** | |

## Task Instance Attempts

| | |
|---|---|
| **Description** | Resolves to the current task instance attempt count. Each Re-run operation increments the attempt. Initial attempt is 1. |

| | |
|---|---|
| **Syntax** | ${ops_attempt} |
| **Example** | |

## Task Instance Definition ID

| | |
|---|---|
| **Description** | Resolves to the task instance definition ID. |
| **Syntax** | ${ops_task_definition_id} |
| **Example** | |

## Task Instance Exit Code

| | |
|---|---|
| **Description** | Resolves to the task instance exit code, if any. |
| **Syntax** | ${ops_exit_code} |
| **Example** | |

## Task Instance Name

| | |
|---|---|
| **Description** | Resolves to the task instance name. |
| **Syntax** | ${ops_task_name} |
| **Example** | |

## Task Instance Status

| | |
|---|---|
| **Description** | Resolves to the current task instance status. |
| **Syntax** | ${ops_status} |
| **Example** | |

## Task Instance Status Description

| | |
|---|---|
| **Description** | Resolves to the task instance status description. |
| **Syntax** | ${ops_status_description} |

| | |
|---|---|
| **Example** | |

## Task Instance sys_id

| | |
|---|---|
| **Description** | Resolves to the sys_id of the task instance. |
| **Syntax** | ${ops_task_id} |
| **Example** | |

## Task Name at Instance Creation Time

| | |
|---|---|
| **Description** | Resolves to the name of the task at the time the task instance was created.<br><br>Note<br>If the name of the task contains variables, those variables contained in the task will be fully resolved when using this built-in variable, ${ops_task_security_name}. |
| **Syntax** | ${ops_task_security_name} |
| **Example** | |

## Task Type

| | |
|---|---|
| **Description** | Resolves to the task type. |
| **Syntax** | ${ops_task_type} |
| **Example** | |

## Time Zone (Task time zone)

| | |
|---|---|
| **Description** | Resolves to the time zone of the task instance, as specified by the Time Zone Preference field. |
| **Syntax** | ${ops_task_time_zone} |
| **Example** | |

## Time Zone (Trigger time zone)

| | |
|---|---|
| **Description** | Resolves to the time zone of the trigger that launched the task. If the task was launched by the Trigger Now/Launch Task command, the built-in variable will resolve to the command's time zone option, or if no time zone option was specified, the server time zone. |

| | |
|---|---|
| **Syntax** | ${ops_time_zone} |
| **Example** | |

### Top-Level Workflow Name

| | |
|---|---|
| **Description** | Resolves to the name of the top-level workflow task instance. |
| **Syntax** | ${ops_top_level_workflow_name} |
| **Example** | |

### Top-Level Workflow Task Instance ID

| | |
|---|---|
| **Description** | Resolves to the sys_id of the top-level workflow task instance. |
| **Syntax** | ${ops_top_level_workflow_id} |
| **Example** | |

### Virtual Resource Priority

| | |
|---|---|
| **Description** | Resolves to the value of the task instance field Virtual Resource Priority. |
| **Syntax** | ${ops_resource_priority} |
| **Example** | |

## Task Monitor Task Instance/Trigger Variables

When the conditions of a Task Monitor task are met and its associated Task Monitor trigger launches one or more tasks, the following built-in variables are passed into the task instances being launched by the trigger.

For example, the Task Monitor trigger may specify an Email task that will launch each time the conditions in the associated Task Monitor task are met. You might want to specify one or more of these variables in the body of the email.

If the Task Monitor task is not associated with a trigger but is running within a workflow, on completion you can propagate one or more of these built-in variable values to the parent workflow level by using the Set Variable action. This allows you to pass information from the Task Monitor task to a successor task within the same workflow hierarchy.

### Trigger Task Name

| | |
|---|---|
| **Description** | Resolves to the name of the task instance that fired the trigger. |
| **Syntax** | ${ops_trigger_task_name} |
| **Example** | |

### Trigger Task Status

| | |
|---|---|
| **Description** | Resolves to the status of the task instance that fired the trigger. |
| **Syntax** | ${ops_trigger_task_status} |
| **Example** | |

### Trigger Task sys_id

| | |
|---|---|
| **Description** | Resolves to the sys_id of the task instance that fired the trigger. |
| **Syntax** | ${ops_trigger_task_id} |
| **Example** | |

### Trigger Task Type

| | |
|---|---|
| **Description** | Resolves to the type of the task instance that fired the trigger. |
| **Syntax** | ${ops_trigger_task_type} |
| **Example** | |

### Trigger Workflow

| | |
|---|---|
| **Description** | Resolves to the name of the workflow instance that fired the trigger.<br><br>This variable is available only for a Task Monitor task that has a Workflow Condition specified. If a workflow condition is specified, **`${ops_trigger_workflow_name}`** will resolve to the name of the workflow instance that the workflow condition matched. |
| **Syntax** | ${ops_trigger_workflow_name} |
| **Example** | |

## Trigger Variables

The following built-in variables are associated with all trigger types.

When a task is launched by a trigger, the values of the following built-in variables, if they are specified in the task, are passed into the task instance.

## Trigger Name

| | |
|---|---|
| **Description** | Resolves to the name of the trigger that launched the task instance. |
| **Syntax** | ${ops_trigger_name} |
| **Example** | |

## Trigger Time

| | |
|---|---|
| **Description** | Resolves to the scheduled time of the trigger or, if the trigger is not scheduled, the actual trigger time.<br><br>If the task is triggered by date/time, it resolves to that specified date/time. |
| **Syntax** | ${ops_trigger_time} |
| **Example** | |

## Trigger Time (Trigger time zone)

| | |
|---|---|
| **Description** | Resolves to the trigger time in the time zone of the trigger. |
| **Syntax** | ${ops_trigger_time_tz} |
| **Example** | |

# Variable Monitor Task Instance/Trigger Variables

When the conditions of a Variable Monitor task are met and its associated Variable Monitor trigger launches one or more tasks, the following built-in variables are passed into the task instances being launched by the trigger.

For example, the Variable Monitor trigger may specify an Email task that will launch each time the conditions in the associated Variable Monitor task are met. You might want to specify one or more of these variables in the body of the email.

If the Variable Monitor task is not associated with a trigger but is running within a workflow, on completion you can propagate one or more of these built-in variable values to the parent workflow level by using the Set Variable action. This allows you to pass information from the Variable Monitor task to a successor task within the same workflow hierarchy.

## Trigger Variable Name

| | |
|---|---|
| **Description** | Resolves to the name of the variable being monitored. |

| | |
|---|---|
| **Syntax** | ${ops_trigger_variable_name} |
| **Example** | |

### Trigger Variable Value

| | |
|---|---|
| **Description** | Resolves to the current value of the variable being monitored. |
| **Syntax** | ${ops_trigger_variable_value} |
| **Example** | |

### Trigger Variable Previous Value

| | |
|---|---|
| **Description** | Resolves to previous value of the variable being monitored. |
| **Syntax** | ${ops_trigger_variable_prev_value} |
| **Example** | |

## Web Service Task Instance Variables

The following built-in variables are available for Web Service task instances:

### URL

| | |
|---|---|
| Description | Resolves to the entire encoded URL containing the host, port, path and query. |
| Syntax | ${ops_url} |
| Example | |

### Raw Value of URL

| | |
|---|---|
| Description | Resolves to the raw value of the URL field. |
| Syntax | ${ops_url_raw} |
| Example | |

### URL Host

| | |
|---|---|
| Description | Resolves to the URL host. |

| | |
|---|---|
| Syntax | ${ops_url_host} |
| Example | |

### URL Port

| | |
|---|---|
| Description | Resolves to the URL port. |
| Syntax | ${ops_url_port} |
| Example | |

### URL Path

| | |
|---|---|
| Description | Resolves to the encoded URL path. |
| Syntax | ${ops_url_path} |
| Example | |

### Unencoded URL Path

| | |
|---|---|
| Description | Resolves to the unencoded URL path. |
| Syntax | ${ops_url_path_unencoded} |
| Example | |

### URL Query

| | |
|---|---|
| Description | Resolves to the URL query. |
| Syntax | ${ops_url_query} |
| Example | |

### Unencoded URL Query

| | |
|---|---|
| Description | Resolves to the unencoded URL query. |
| Syntax | ${ops_url_query_unencoded} |
| Example | |

## z/OS Task Instance Variables

The following built-in variables are available for z/OS task instances:

## JCL Location

| | |
|---|---|
| **Description** | Resolves to the file and member name containing the JCL script. |
| **Syntax** | ${ops_jcl_location} |
| **Example** | |

## Job Number

| | |
|---|---|
| **Description** | Resolves to the job number assigned to the job by JES. |
| **Syntax** | ${ops_job_id} |
| **Example** | |

## Override JCL Location

| | |
|---|---|
| **Description** | Resolves to the file and member name of the JCL location containing a potential override JCL script. |
| **Syntax** | ${ops_override_jcl_location} |
| **Example** | |

## Submitted JCL Location

| | |
|---|---|
| **Description** | Resolves to the file and member name of the JCL location that was actually used for job submission. |
| **Syntax** | ${ops_submitted_jcl_location} |
| **Example** | |

# Launching With Variables

For information on how to launch a task with variables, see Launch a Task Manually with Temporary Variable Values on the Manually Running and Controlling Tasks page.

# Trigger With Variables

For information on how to use variables when manually launching tasks associated with a trigger, see Triggering with Variables (in the Triggers and Calendars section of this documentation).

# Creating a Set Variable Action within a Task or Workflow



## Overview

The Set Variable action allows you to set a variable to a specific value for a task or workflow, and to select a scope (level of usage) for that variable (see Variables and Variable Scope, below). Unless you set the scope of the variable to **GLOBAL**, which specifies that the variable can be accessed at any time by any task, workflow, or trigger, the value exists in memory only for the time that the task or workflow is running, or until another Set Variable action sets the variable to another value.

Note

Variables with a Variable Scope set to **GLOBAL** are added to the list of global variables on the Variables list (**Automation Center > Other > Variables**) after the task or workflow is run.

You can use the Set Variable action to create a new variable or modify an existing variable.

When creating a Set Variable action, you can trigger the Set Variable action based on one or more of the following:

- Status
- Exit codes
- Late start
- Late or early finish

## Variables and Variable Scope

A variable defined for a task under the **Variables** tab for that task is used only by that task.

A variable defined for a workflow under the **Variables** tab for that workflow is available for any task in that workflow; a task will use the variable value defined for the workflow unless the variable is defined for that task.

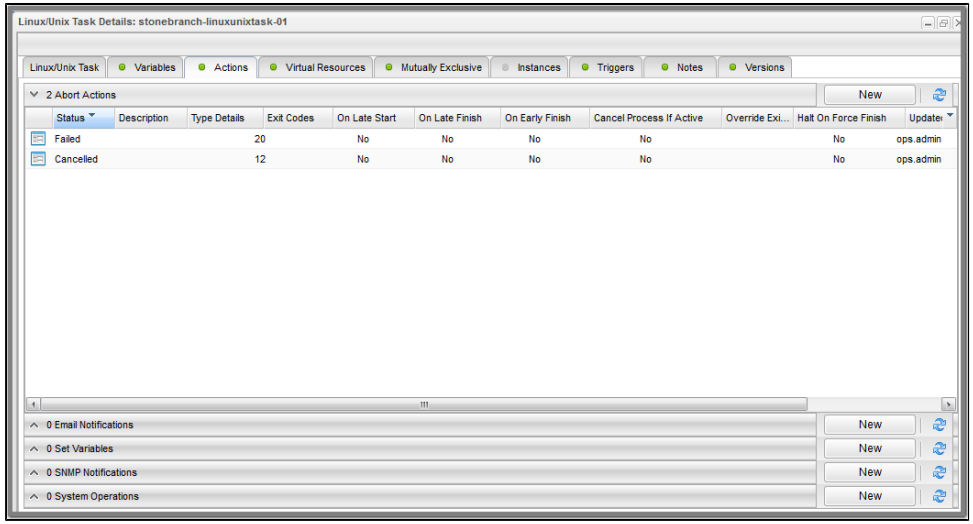
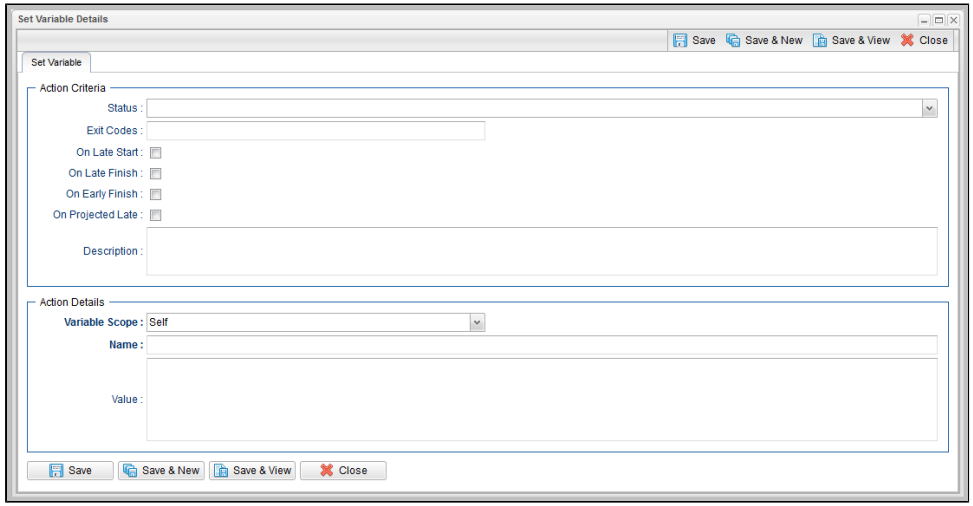
A variable defined for a task or workflow in the Set Variable Action Details lets you specify, in the Variable Scope field, the scope of that variable. You can specify that a variable be available for:

- Only the task where it is set.
- All tasks within the task's parent (immediate) workflow.
- All tasks within the task's top-level parent workflow.
- All tasks and workflow instances.

For example, if you set a variable for a task to be available within the scope of its parent workflow, the value of that variable is propagated up to the parent workflow level. As each task in the workflow is run, that value is available for that task.

## Creating a Set Variable Action

| | |
|---|---|
| **Step 1** | Display the Task Details of the task for which you are creating the Set Variable action. |

| | |
|---|---|
| **Step 2** | Click the **Actions** tab. A list of any defined Actions for that task displays.<br>  |
| **Step 3** | Click the **New** button that displays on the Set Variables row. The Set Variable Details pop-up displays.<br>  |
| **Step 4** | Using the field descriptions below as a guide, complete the fields as needed. |
| **Step 5** | Click a **Save** button to save the record in the Controller database. |
| **Step 6** | If appropriate, repeat these steps for any additional Set Variable actions you want to create. |

# Set Variable Details Field Descriptions

The table below describes the fields and buttons in the Set Variable Details.

| Field Name | Description |
|---|---|
| **Action Criteria** | This section contains criteria for performing the action. |
| Type Details | Displays - on the Set Variables actions list - the Variable Scope, Name, and Value for this action. |
| Action Inheritance | For Workflow tasks only; the records that this action applies to.<br><br>Options:<br><br>• Self<br>The action applies only to the workflow; it is not inherited by its children tasks. For example, if the action is defined for the Defined status, when the workflow where the action is specified transitions into the Defined status, the action will run for the workflow. When children tasks within this workflow transition into the Defined status, the action will not run.<br>• Self/Children<br>The action applies to the workflow and any children under the workflow (it is as if each child under the workflow had the action specified on itself). For example, if the workflow or any of its children transition into the Defined status, the action will run.<br>• Children<br>This action applies only to the children under the workflow and not the workflow itself. For example, if any child of this workflow transitions into the Defined status, the action will run. However, when the workflow where this action is specified transitions into the Defined status, this action will not run. |
| Status | The status of the task, by itself or together with an exit code, that will trigger this Set Variable action. You can specify as many statuses as needed. |
| Exit Codes | Specifies one or more exit codes that will trigger the event. If you specify an exit code, you must also specify at least one status. Use commas to separate multiple exit codes; use a hyphen to specify a range. Example: 1, 5, 22-30. |
| On Late Start | Generates the action or notification if the task started late, based on the Late Start Time specified in the task. |
| On Late Finish | Generates the action or notification if the task finishes late, based on the Late Finish time specified in the task. |
| On Early Finish | Generates the action or notification if the task finishes early, based on the Early Finish Time specified in the task. |
| On Projected Late | Execute the Action when the task instance is projected to be late based on critical path projected end times. Only applicable when a Late Start Time, Late Start Duration, or Late Finish Time is specified for the task instance.<br><br>Note<br>This field displays in the Details only if the Controller is configured for critical path calculations with an enabled Critical Path Calculations Permitted Universal Controller system property. |
| Description | Description of this action. |

| **Action Details** | This section contains additional details about the action. |
|---|---|
| Variable Scope | Applies to variables associated with a task in a workflow.<br><br>Options:<br><br>**Scope** / **Scope Value** / **Description**<br>**Self** / 1 / The variable is updated or created in the scope of the task instance running the action. If the task instance is a workflow, then any child of that workflow will be able to read that variable.<br>**Parent** / 2 / The variable is updated or created in the immediate parent workflow scope, allowing a child within a workflow to make a variable available to any other child in the same workflow (at the same level).<br>**Top Level Parent** / 3 / The variable is updated or created at the top-level workflow variable scope, allowing a child anywhere in the workflow hierarchy to make a variable available to any other child in the workflow hierarchy, regardless of which level in the workflow the task instances are running.<br>**Global** / 4 / A global variable will be updated and or created. Allows for variables to be shared across independent workflows. |
| System Notification | If Variable Scope = Global; Status of the Set Variable action that will trigger a system notification.<br><br>Options:<br><br>• None<br>• Operation Failure (default)<br>• Operation Success/Failure<br>• Operation Success<br><br>Note<br>The Controller must be configured for system notifications in order for system notifications to be triggered. |
| Name | Name of the variable. Up to 128 alphanumerics. The name must begin with an alphabetic character and can consist of: alphas (a-z, A-Z), numerics 0-9, _ (underscore). White spaces are not permitted; names are not case-sensitive.<br><br>Important<br>Do not define variables with the prefix **ops**_. The **ops**_ prefix is reserved for built-in variables. |
| Value | Value of the variable.<br><br>Note<br>While a global variable value can never exceed 4000 characters, a task instance variable value assigned dynamically at run time (for example, using a function to assign a variable value to Self, Parent, or Top Level Parent, using the Set Variable Action, can exceed the 4000 character limit. Keep in mind, however, that the Maximum Nested Variable Expansion Universal Controller system property prevents unlimited variable value expansion. |

| | |
|---|---|
| **Metadata** | This section contains Metadata information about this record. |
| UUID | Universally Unique Identifier of this record. |
| Updated By | Name of the user that last updated this record. |
| Updated | Date and time that this record was last updated. |
| Created By | Name of the user that created this record. |
| Created | Date and time that this record was created. |
| **Buttons** | This section identifies the buttons displayed above and below the Action Details that let you perform various actions. |
| **Save** | Saves a new Action record in the Controller database. |
| **Save & New** | Saves a new record in the Controller database and redisplays empty Details so that you can create another new record. |
| **Save & View** | Saves a new record in the Controller database and continues to display that record. |
| **New** | Displays empty (except for default values) Details for creating a new record. |
| **Update** | Saves updates to the record. |
| **Delete** | Deletes the current record. |
| **Refresh** | Refreshes any dynamic data displayed in the Details. |
| **Close** | Closes the Details pop-up of this action. |

# Listing and Setting Variables from the Command Line

To list and set variables from the command line, use the List Variables (ops-variable-list) and Set Variables (ops-variable-set) commands of the Universal Controller Command Line Interface (CLI).

# Functions

# Overview

Variables and functions can be used in free-text fields within tasks and workflows. When a variable or function is specified in a free-text field, the Controller inserts its value into the field when the task or workflow is run.

Also, triggers can pass variables and functions into the tasks and workflows they launch.

Universal Controller supports a number of functions that can be specified in free-text fields. They are resolved when a task instance runs or when a Set Variable action containing a function is executed.

Functions are entered using the following formats:

```
${_function}
${_function(arg1, ..., argN)}
```

# Formatting Rules

- Functions *must* be written either:
  - In all lower-case characters.
  - Exactly as shown in the tables on this page.
- Functions have zero, one, or multiple parameters.
- Each function parameter is one of three specific types:
  - String
  - Integer
  - Boolean
- String parameters *must* be enclosed in **single or double** quotation marks.
- Integer and Boolean parameters *can* be enclosed in **single or double** quotation marks.
- Optional parameters are identified on this page by being enclosed in [square brackets]. When copying a function from the documentation, be sure to remove the square brackets; otherwise, the function will not resolve.
- If a function has more than one optional parameter, any optional parameters preceding a specified optional parameter must be included in the function's parameter list. For example:
  - For function `${_responseJsonPath('pathExpression'[,'defaultValue','delimiter',prettyPrint])}`, usage `${_responseJsonPath('.outputData','','',true)}` would be valid, whereas `${_responseJsonPath('.outputData',,,true)}` would not be valid.
  - For function `${_formatDate(['date_time', 'format', day_offset, use_business_days, hour_offset, minute_offset, timezone])}`, usage `${_formatDate('2018-09-01','',0,true)}` would be valid, whereas `${_formatDate('2018-09-01','',,true)}` would not be valid.
- All functions allow nesting to two levels. That is, a function can be an argument to another function, which itself can be an argument to another function.
  - You must use a double underscore preceding the name of a first-level nested function.
  - You must use a triple underscore preceding the name of a second-level nested function.

For example, for 2nd day of next month less one Business Day:

```
${_formatDate('${__dayOfMonth(2,'${___dateadv('yyyy-MM-dd',0,1)}')}','',-1,true)}
```

# Function Categories

Functions are listed alphabetically within the following categories on this page:



# Conditional Functions

## Return Conditional Value Depending on Equality of String Parameters

| | |
|---|---|
| **Description** | Returns a conditional value depending on the equality of two string parameters.<br><br>(Returns `if_value` if string `value1` is equal to string `value2`; otherwise, `else_value` is returned.) |
| **Syntax** | **`${_ifEqual`**`('value1', 'value2', 'if_value', 'else_value'[, ignore_case])}` |
| **Parameters** | • `value1`<br>Required; First string.<br>• `value2`<br>Required; Second string.<br>• `if_value`<br>Required; Return value if `value1` equals `value2`.<br>• `else_value`<br>Required; Return value if `value1` does not equal `value2`.<br>• `ignore_case`<br>Optional; Specification (**true** or **false**) whether or not to ignore case when comparing `value1` and `value2`. Default is false. |
| **Examples** | `${_ifEqual('abc','def','YES','NO')}`<br>`${_ifEqual('abc','ABC','YES','NO',true)}`<br>`${_ifEqual('2015-08-15','${__date()}','17:00','18:00')}` |

### Return Conditional Value Depending on Value of Boolean Parameter

| | |
|---|---|
| **Description** | Returns a conditional value depending on the value of a boolean parameter.<br><br>Returns `if_value` if `value` is **true**; otherwise, `else_value` is returned. |
| **Syntax** | **`${_ifTrue`**`(value, 'if_value', 'else_value')}` |
| **Parameters** | • `value`<br>Required; Boolean value (**true** or **false**).<br>• `if_value`<br>Required; Return value if `value` is **true**.<br>• `else_value`<br>Required; Return value if `value` is **false**. |
| **Example** | `${_ifTrue(${__isToday('Mon', 'E')},'20:00','22:00')}` |

## Credential Functions

### Return Key Location of a Credential

| | |
|---|---|
| **Description** | Returns a token representing the Resolvable Credential Key Location that you want to embed. |
| **Syntax** | **`${_credentialKeyLoc`**`('<credential_name>')}` |
| **Parameters** | • `credential_name`<br>Required; Name of the Credential. |
| **Example** | `${_credentialKeyLoc('RCredentialXYZ')}  $(ops_unv_cred_key_loc_c89e7b2caf4247909bc46041df8a2643)` |

### Return Passphrase of a Credential

| | |
|---|---|
| **Description** | Returns a token representing the Resolvable Credential Passphrase that you want to embed. |
| **Syntax** | **`${_credentialPassphrase`**`('<credential_name>')}` |

| | |
|---|---|
| **Parameters** | • `credential_name`<br>Required; Name of the Credential. |
| **Example** | `${_credentialPassphrase('RCredentialXYZ')}  $(ops_unv_cred_passphrase_c89e7b2caf4247909bc46041df8a2643)` |

### Return User Name of a Credential

| | |
|---|---|
| **Description** | Returns a token representing the Resolvable Credential Runtime User that you want to embed. |
| **Syntax** | **`${_credentialUser`**`('<credential_name>')}` |
| **Parameters** | • `credential_name`<br>Required; Name of the Credential. |
| **Example** | `${_credentialUser('RCredentialXYZ')}  $(ops_unv_cred_user_c89e7b2caf4247909bc46041df8a2643)` |

### Return User Password of a Credential

| | |
|---|---|
| **Description** | Returns a token representing the Resolvable Credential Runtime Password that you want to embed. |
| **Syntax** | **`${_credentialPwd`**`('<credential_name>')}` |
| **Parameters** | • `credential_name`<br>Required; Name of the Credential. |
| **Example** | `${_credentialPwd('RCredentialXYZ')}  $(ops_unv_cred_pwd_c89e7b2caf4247909bc46041df8a2643)` |

## Date Functions

### Checks if Date Argument Equals Today's Date

| | |
|---|---|
| **Description** | Checks if a date argument is equal to today's date in the specified format.<br><br>Returns **true** if `date` is equal to today's date in the specified format; otherwise, **false** is returned. |
| **Syntax** | **`${_isToday`**`('date'[, 'format', is_relative])}` |

| | |
|---|---|
| **Parameters** | • `date`<br>Required; Date to compare to today's date.<br>• `format`<br>Optional; Format of today's date. Default is yyyy-MM-dd.<br>• `is_relative`<br>Optional; Specification (**true** or **false**) for whether today's date is relative to the trigger/launch time of the task instance. Default is false. |
| **Examples** | `${_isToday('Wed', 'E')}`<br>`${_isToday('${__dayOfMonth(1,'','',true)}')}` |

## Resolve to Current Date and Time

| | |
|---|---|
| **Description** | Resolves to the current date and time. |
| **Syntax** | **`${_date`**`(['format', day_offset, hour_offset, minute_offset])}` |
| **Parameters** | • `format`<br>Optional; Date format. Default format is yyyy-MM-dd HH:mm:ss Z. For details on the `format` parameter, see https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html<br>• `day_offset`<br>Optional; +/- number of days to offset.<br>• `hour_offset`<br>Optional; +/- number of hours to offset.<br>• `minute_offset`<br>Optional; +/- number of minutes to offset. |
| **Examples** | `${_date} --> 2012-07-14 12:43:06 -0400`<br>`${_date()} --> 2012-07-14 12:43:06 -0400`<br>`${_date('yyyy-MM-dd', 5)} --> 2012-07-19`<br>`${_date('yyyy-MM-dd HH:mm:ss', -2, -1)} --> 2012-07-12 11:43:06`<br>`${_date('', 0, 0, 10)} --> 2012-07-14 12:53:06 -0400` |

## Resolve to Current Date and Time (Advanced)

| | |
|---|---|
| **Description** | Resolves to the current date and time. |
| **Syntax** | **`${_dateadv`**`(['format', year_offset, month_offset, day_offset, hour_offset, minute_offset])}` |
| **Parameters** | |

| | |
|---|---|
| | • `format`<br>Date format. Default format is yyyy-MM-dd HH:mm:ss Z. For details on the `format` parameter, see https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html<br>• `year_offset`<br>Optional; +/- number of years to offset.<br>• `month_offset`<br>Optional; +/- number of months to offset.<br>• `day_offset`<br>Optional; +/- number of days to offset.<br>• `hour_offset`<br>Optional; +/- number of hours to offset.<br>• `minute_offset`<br>Optional; +/- number of minutes to offset. |
| **Examples** | `${_dateadv} --> 2012-07-29 09:31:42 -0700`<br>`${_dateadv('yyyy-MMM', -1)} --> 2011-Jul`<br>`${_dateadv('yyyy-MMM', 0, -1)} --> 2012-Jun` |

## Resolve to Current Unix Epoch Time

| | |
|---|---|
| **Description** | Resolves to the current time in milliseconds since Wed Dec 31 1969 19:00:00 GMT-0500 (EST) – the start of Unix epoch time. |
| **Syntax** | **`${_currentTimeMillis}`** |
| **Parameters** | n/a |

## Return Date with Offsets

| | |
|---|---|
| **Description** | Returns the date after applying offsets. Optionally, can specify the output format.<br><br>Whether a holiday is treated as a business day or a non-business day is specified by the Exclude Holidays for Business Days Universal Controller system property. |
| **Syntax** | **`${_formatDate`**`(['date_time', 'format', day_offset, use_business_days, hour_offset, minute_offset, timezone])}` |
| **Parameters** | • `date_time`<br>Date and time in any of the following formats:<br>  • yyyy-MM-dd<br>  • yyyy-MM-dd HH:mm<br>  • yyyy-MM-dd HH:mm:ss<br>  • yyyy-MM-dd HH:mm Z<br>  • yyyy-MM-dd HH:mm:ss Z.<br>  • yyyy-MM-dd HH:mm:ss.SSS<br>  • yyyy-MM-dd HH:mm:ss.SSS Z.<br>Default is the current date and time. |

| | |
|---|---|
| | • `format`<br>Format of returned date. If date_time specifies a time, the default format is yyyy-MM-dd HH:mm; otherwise, the default format is yyyy-MM-dd. For details on the `format` parameter, see https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html<br>• `day_offset`<br>+/- number of days to offset.<br>• `use_business_days`<br>Specification (`true` or `false`) for whether `day_offset` is for business days. Default is false.<br>• `hour_offset`<br>+/- number of hours to offset.<br>• `minute_offset`<br>+/- number of minutes to offset.<br>• `timezone`<br>Time Zone that the date is formatted in. |
| **Example** | `${_formatDate} --> 2018-08-24 15:37`<br>`${_formatDate()} --> 2018-08-24 15:37`<br>`${_formatDate('','MMddyyyy',5)} --> 08292018`<br>`${_formatDate('2018-09-01','',5)} --> 2018-09-06`<br>`${_formatDate('2018-09-01','',-5)} --> 2018-08-27`<br>`${_formatDate('2018-10-13 12:13:14 -0400','',5,true,0,0,'Australia/Sydney')} --> 2018-10-14 03:13:14 +1100` |

## Return Date with Offsets (Advanced)

| | |
|---|---|
| **Description** | Returns the date after applying offsets. Optionally, can specify the output format.<br><br>Whether a holiday is treated as a business day or a non-business day is specified by the Exclude Holidays for Business Days Universal Controller system property. |
| **Syntax** | **`${_formatDateAdv`**`(['date_time', 'format', year_offset, month_offset, day_offset, use_business_days, hour_offset, minute_offset, timezone])}` |
| **Parameters** | • `date_time`<br>Date and time in any of the following formats:<br>  • yyyy-MM-dd<br>  • yyyy-MM-dd HH:mm<br>  • yyyy-MM-dd HH:mm:ss<br>  • yyyy-MM-dd HH:mm Z<br>  • yyyy-MM-dd HH:mm:ss Z.<br>  • yyyy-MM-dd HH:mm:ss.SSS<br>  • yyyy-MM-dd HH:mm:ss.SSS Z.<br>Default is the current date and time.<br>• `format`<br>Format of returned date. If date_time specifies a time, the default format is yyyy-MM-dd HH:mm; otherwise, the default format is yyyy-MM-dd. For details on the `format` parameter, see https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html<br>• `year_offset`<br>Optional; +/- number of years to offset.<br>• `month_offset`<br>Optional; +/- number of months to offset.<br>• `day_offset`<br>Optional; +/- number of days to offset. |

| | |
|---|---|
| | • `use_business_days`<br>Optional; Specification (**true** or **false**) for whether `day_offset` is for business days. Default is false.<br>• `hour_offset`<br>+/- number of hours to offset.<br>• `minute_offset`<br>+/- number of minutes to offset.<br>• `timezone`<br>Time Zone that the date is formatted in. |
| **Examples** | `${_formatDateAdv} --> 2012-08-24 15:55`<br>`${_formatDateAdv()} --> 2012-08-24 15:55`<br>`${_formatDateAdv('','MMddyyyy',1)} --> 08242013`<br>`${_formatDateAdv('2012-09-01','',0,1)} --> 2012-10-01`<br>`${_formatDateAdv('2012-09-01','',0,-1)} --> 2012-08-01`<br>`${_formatDateAdv('2012-09-01','',0,0,5,false)} --> 2012-09-06` |

## Return Date with Time Zone

| | |
|---|---|
| **Description** | Returns the Date and Time in another time zone. |
| **Syntax** | **`${_formatDateTz`**`('date_time', 'target_time_zone'[, 'output_format'])}` |
| **Parameters** | • `date_time`<br>Date and time in any of the following formats:<br>  • yyyy-MM-dd HH:mm<br>  • yyyy-MM-dd HH:mm:ss<br>  • yyyy-MM-dd HH:mm Z<br>  • yyyy-MM-dd HH:mm:ss Z.<br>  • yyyy-MM-dd HH:mm:ss.SSS<br>  • yyyy-MM-dd HH:mm:ss.SSS Z.<br>• `target_time_zone`<br>Time zone in which to format the date and time.<br>• `output_format`<br>Optional; Format of the date and time in the other time zone. |
| **Examples** | `${_formatDateTz('2018-10-13 01:02:03 -0400', 'Australia/Sydney')} --> 2018-10-13 16:02:03 +1100`<br>`${_formatDateTz('2018-10-13 01:02:03 -0400', 'Australia/Sydney','yyyy-MM-dd HH:mm Z')} --> 2018-10-13 16:02 +1100`<br>`${_formatDateTz('${ops_launch_time}', '${ops_time_zone}')} = ${_formatDateTz('2018-06-13 15:35:00 -0400', 'Europe/Berlin')} = 2018-06-13 21:35:00 +0200` |

## Return Day of Week

| | |
|---|---|
| **Description** | Returns the day of week for the specified date as a number. |
| **Syntax** | **${_dayOfWeek**(['date', 'first_dow', first_dow_value])} |
| **Parameters** | • date<br>Date in any of the following formats:<br>  • yyyy-MM-dd<br>  • yyyy-MM-dd HH:mm<br>  • yyyy-MM-dd HH:mm:ss<br>  • yyyy-MM-dd HH:mm Z<br>  • yyyy-MM-dd HH:mm:ss Z.<br>  • yyyy-MM-dd HH:mm:ss.SSS<br>  • yyyy-MM-dd HH:mm:ss.SSS Z.<br>Default is the current date.<br>• first_dow<br>Optional; Specification for whether the week starts on Sunday or Monday. Values are **sun** and **mon** (not case-sensitive). Default is sun.<br>• first_dow_value<br>Optional; Starting value for the first day of week. Value must be a non-negative number. Default is 1. |
| **Example** | ```<br>${_dayOfWeek} --> 6<br>${_dayOfWeek()} --> 6<br>${_dayOfWeek('2012-07-04')} --> 4<br>${_dayOfWeek('2012-07-04', 'mon')} --> 3<br>``` |

## Return Days between Dates

| | |
|---|---|
| **Description** | Returns the number of days between date1 and date2.<br><br>• If return value is > 0, date2 is after date1.<br>• If return value is < 0, date2 is before date1.<br>• If return value is 0, date1 is equal to date2.<br><br>The start date is inclusive, but the end date is not. |
| **Syntax** | **${_daysBetween**('date1', 'date2')} |
| **Parameters** | • date1<br>Required.<br>• date2<br>Required.<br><br>date1 and date2 are specified in any of the following formats:<br><br>• yyyy-MM-dd<br>• yyyy-MM-dd HH:mm<br>• yyyy-MM-dd HH:mm:ss |

| | |
|---|---|
| | • yyyy-MM-dd HH:mm Z<br>• yyyy-MM-dd HH:mm:ss Z.<br>• yyyy-MM-dd HH:mm:ss.SSS<br>• yyyy-MM-dd HH:mm:ss.SSS Z. |
| **Example** | `${_daysBetween('2012-08-01','2012-09-01')} --> 31` |

## Return Non-Business Day of Month

| | |
|---|---|
| **Description** | Returns the Nth non-business day of month for the month of the date specified. Optionally, can start from the end of the month.<br><br>Whether a holiday is treated as a business day or a non-business day is specified by the Exclude Holidays for Business Days Universal Controller system property. |
| **Syntax** | **`${_nonBusinessDayOfMonth`**`(index, ['date', 'format', reverse])}` |
| **Parameters** | • `index`<br>Required; Nth non-business day of month.<br>• `date`<br>Date (and time) is specified in any of the following formats:<br>• yyyy-MM-dd<br>• yyyy-MM-dd HH:mm<br>• yyyy-MM-dd HH:mm:ss<br>• yyyy-MM-dd HH:mm Z<br>• yyyy-MM-dd HH:mm:ss Z.<br>• yyyy-MM-dd HH:mm:ss.SSS<br>• yyyy-MM-dd HH:mm:ss.SSS Z.<br>Default is the current date.<br>• `format`<br>Optional; Format of returned date. Default is yyyy-MM-dd. For details on the `format` parameter, see https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html<br>• `reverse`<br>Optional; Specification (**true** or **false**) for starting from the end of the month. Default is false. |
| **Examples** | `${_nonBusinessDayOfMonth(1)} --> 2012-08-04`<br>`${_nonBusinessDayOfMonth(1,'2012-09-01')} --> 2012-09-01`<br>`${_nonBusinessDayOfMonth(1,'2012-09-01','',true)} --> 2012-09-30` |

## Return Nth Business Day of Month

| | |
|---|---|
| **Description** | |

| | |
|---|---|
| | Returns the Nth business day of month for the month of the date specified. Optionally, can start from the end of the month.<br><br>Whether a holiday is treated as a business day or a non-business day is specified by the Exclude Holidays for Business Days Universal Controller system property. |
| **Syntax** | **`${_businessDayOfMonth`**`(index, ['date', 'format', reverse])}` |
| **Parameters** | • `index`<br>Required; Nth business day of month.<br>• `date`<br>Date (and time) is specified in any of the following formats:<br>  • yyyy-MM-dd<br>  • yyyy-MM-dd HH:mm<br>  • yyyy-MM-dd HH:mm:ss<br>  • yyyy-MM-dd HH:mm Z<br>  • yyyy-MM-dd HH:mm:ss Z.<br>  • yyyy-MM-dd HH:mm:ss.SSS<br>  • yyyy-MM-dd HH:mm:ss.SSS Z.<br>Default is the current date.<br>• `format`<br>Optional; Format of returned date. Default is yyyy-MM-dd. For details on the `format parameter`, see https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html<br>• `reverse`<br>Optional; Specification (`true` or `false`) for starting from the end of the month. Default is false. |
| **Examples** | `${_businessDayOfMonth(1)} --> 2012-08-01`<br>`${_businessDayOfMonth(1,'2012-09-01')} --> 2012-09-04`<br>`${_businessDayOfMonth(1,'2012-09-01','',true)} --> 2012-09-28` |

## Return Nth Day of Month

| | |
|---|---|
| **Description** | Returns the Nth day of month for the month of the date specified. Optionally, can start from the end of the month. |
| **Syntax** | **`${_dayOfMonth`**`(index, ['date', 'format', reverse])}` |
| **Parameters** | • `index`<br>Required; Nth day of month.<br>• `date`<br>Date (and time) is specified in any of the following formats:<br>  • yyyy-MM-dd<br>  • yyyy-MM-dd HH:mm<br>  • yyyy-MM-dd HH:mm:ss<br>  • yyyy-MM-dd HH:mm Z<br>  • yyyy-MM-dd HH:mm:ss Z.<br>  • yyyy-MM-dd HH:mm:ss.SSS<br>  • yyyy-MM-dd HH:mm:ss.SSS Z.<br>Default is the current date. |

| | |
|---|---|
| | • `format`<br>Optional; Format of returned date. Default is yyyy-MM-dd.<br>• `reverse`<br>Optional; Specification (**true** or **false**) for starting from the end of the month. Default is false. |
| **Examples** | `${_dayOfMonth(5)} --> 2012-08-05`<br>`${_dayOfMonth(15,'2012-09-01','MM/dd/yyyy')} --> 09/15/2012`<br>`${_dayOfMonth(1,'2012-09-01','',true)} --> 2012-09-30` |

## Return Number of Business Days between Dates

| | |
|---|---|
| **Description** | Returns the number of business days between date1 and date2.<br>• If return value is > 0, `date2` is after `date1`.<br>• If return value is < 0, `date2` is before `date1`.<br>• If return value is 0, `date1` is equal to `date2`.<br>The start date is inclusive, but the end date is not.<br>Whether a holiday is treated as a business day or a non-business day is specified by the Exclude Holidays for Business Days Universal Controller system property. |
| **Syntax** | **`${_businessDaysBetween`**`('date1', 'date2')}` |
| **Parameters** | • `date1`<br>Required.<br>• `date2`<br>Required.<br>date1 and date2 are specified in any of the following formats:<br>• yyyy-MM-dd<br>• yyyy-MM-dd HH:mm<br>• yyyy-MM-dd HH:mm:ss<br>• yyyy-MM-dd HH:mm Z<br>• yyyy-MM-dd HH:mm:ss Z.<br>• yyyy-MM-dd HH:mm:ss.SSS<br>• yyyy-MM-dd HH:mm:ss.SSS Z. |
| **Example** | `${_businessDaysBetween('2012-08-01','2012-09-01')} --> 23` |

# Mathematical Functions

## Add

| | |
|---|---|
| **Description** | Return the sum of the augend added with the addend. |
| **Syntax** | **${_add**(augend, addend)} |
| **Parameters** | • augend<br>Integer to which the addend is being added.<br>• addend<br>Integer being added to the augend. |
| **Example** | ${_add('77', '33')} --> 110<br><br>Using Variables for augend and addend (${augend} = 17, ${addend} = 5):<br><br>${_add('${augend}','${addend}')} --> 22 |

## Divide

| | |
|---|---|
| **Description** | Return the quotient of the dividend divided by divisor. |
| **Syntax** | **${_divide**(dividend, divisor)} |
| **Parameters** | • dividend<br>Integer being divided by the divisor.<br>• divisor<br>Integer being used to divide the dividend. |
| **Example** | ${_divide('7','20')} --> 0<br>${_divide('20','7')} --> 2<br>${_divide('20','5')} --> 4 |

| | |
|---|---|
| | Using Variables for dividend and divisor (${dividend} = 100, ${divisor} = 5)<br>`${_divide('${dividend}','${divisor}')} --> 20` |

## Multiply

| | |
|---|---|
| **Description** | Return the product of the multiplicand multiplied with the multiplier. |
| **Syntax** | **`${_multiply`**`(multiplicand, multiplier)}` |
| **Parameters** | • `multiplicand`<br>Integer being multiplied by the `multiplier`.<br>• `multiplier`<br>Integer being used to multiply the `multiplicand`. |
| **Example** | `${_multiply('7','20')} --> 140`<br>Using Variables for multiplicand and multiplier (${multiplicand} = 100, ${multiplier} = 5):<br>`${_multiply('${multiplicand}','${multiplier}')} --> 500` |

## Return Absolute Value

| | |
|---|---|
| **Description** | Return the absolute value of the parameter. |
| **Syntax** | **`{_abs`**`(parameter)}` |
| **Parameters** | • `parameter`<br>Integer (positive or negative value). |
| **Example** | |

```
${_abs('-1200')} --> 1200
${_abs('1200')} --> 1200
```

Using Variables for parameter (${parameter} = -100):

```
${_abs('${parameter}')} --> 100
```

## Return Modulo

| | |
|---|---|
| **Description** | Return the modulo (remainder) of the dividend divided by divisor. |
| **Syntax** | **${_mod**(dividend, divisor)} |
| **Parameters** | • dividend<br>Integer being divided by the divisor.<br>• divisor<br>Integer being used to divide the dividend. |
| **Example** | ```<br>${_mod('10', '2')} --> 0<br>${_mod('10', '3')} --> 1<br>${_mod('70', '65')} --> 5<br>```<br>Using Variables for dividend and divisor (${dividend} = 23, ${divisor} = 5):<br>```<br>${_mod('${dividend}','${divisor}')} --> 3<br>``` |

## Subtract

| | |
|---|---|
| **Description** | Return the difference of the subtrahend subtracted from the minuend. |
| **Syntax** | **${_subtract**(minuend, subtrahend)} |

| | |
|---|---|
| **Parameters** | • `minuend`<br>Integer from which the `subtrahend` is being subtracted.<br>• `subtrahend`<br>Integer being subtracted from the `minuend`. |
| **Example** | `${_subtract('77','33')} --> 44`<br>`${_subtract('33','77')} --> -44`<br><br>Using Variables for minuend and subtrahend (${minuend} = 100, ${subtrahend} = 5):<br><br>`${_subtract('${minuend}','${ subtrahend }')} --> 95` |

# Output Functions

(For Web Service output, see Web Service Functions.)

Note

A prerequisite for the use of these functions is that Automatic Output Retrieval and Wait For Output are selected at task level.

## Task Instance Output

| | |
|---|---|
| **Description** | Resolves to the output data, of the specified `outputType`, of the task instance that is resolving the function.<br>• If the output record of the specified `ouptutType` cannot be found, the function will remain unresolved. |
| **Syntax** | **`${_output`**`('outputType'[, 'defaultValue'])}` |
| **Parameters** | • `outputType`<br>Required; Type of output to resolve: STDOUT, STDERR, FILE, or JOBLOG.<br>• `defaultValue`<br>Optional; Default value to return if the output data is not found. Default is empty (''). |

## Sibling Task Instance Output

| | |
|---|---|
| **Description** | |

| | |
|---|---|
| | Resolves to the output data, of the specified `outputType`, of the task instance specified by the `siblingName` parameter.<br><br>The sibling task instance must be within the same workflow, and the Execution User of the task instance that is resolving the function must have Read permission for the sibling task instance.<br><br>• If the output record of the specified `outputType` cannot be found, the function will remain unresolved. |
| **Syntax** | **`${_outputFromTask`**`('siblingName', 'outputType'[, 'defaultValue'])}` |
| **Parameters** | • `siblingName`<br>Required; Name of a sibling task instance.<br>• `outputType`<br>Required; Type of output to resolve: STDOUT, STDERR, FILE, or JOBLOG.<br>• `defaultValue`<br>Optional; Default value to return if the output data is not found. Default is empty ('). |

## Task Instance Output by Specific Line(s)

| | |
|---|---|
| **Description** | Resolves to the specified line(s) of output data, of the specified `outputType`, of the task instance that is resolving the function.<br><br>• If the output record of the specified `ouptutType` cannot be found, the function will remain unresolved. |
| **Syntax** | **`${_outputLines`**`('outputType', startLine, numberOfLines[, 'defaultValue', 'resultDelimiter'])}` |
| **Parameters** | • `outputType`<br>Required; Type of output to resolve: STDOUT, STDERR, FILE, or JOBLOG.<br>• `startLine`<br>Required; Start line, where 1 is the first line and -1 is the last line.<br>• `numberOfLines`<br>Required; Number of lines to return starting from the `startLine`.<br>• `defaultValue`<br>Optional; Default value to return if no lines qualify. Default is empty (')<br>• `resultDelimiter`<br>Optional; Delimiter to use when concatenating matching lines. If not specified, "\n" or "\r\n" depending on original output line endings. |

## Sibling Task Instance Output by Specific Line(s)

| | |
|---|---|
| **Description** | Resolves to the specified line(s) of output data, of the specified `outputType`, of the task instance specified by the `siblingName` parameter.<br><br>The sibling task instance must be within the same workflow, and the Execution User of the task instance that is resolving the function must have Read permission for the sibling task instance.<br><br>• If the output record of the specified `outputType` cannot be found, the function will remain unresolved. |
| **Syntax** | **`${_outputLinesFromTask`**`('siblingName', 'outputType', startLine, numberOfLines[, 'defaultValue', 'resultDelimiter'])}` |
| **Parameters** | • `siblingName`<br>Required; Name of a sibling task instance. |

| | |
|---|---|
| | • `outputType`<br>Required; Type of output to resolve: STDOUT, STDERR, FILE, or JOBLOG.<br>• `startLine`<br>Required; Start line, where 1 is the first line and -1 is the last line.<br>• `numberOfLines`<br>Required; Number of lines to return starting from the `startLine`.<br>• `defaultValue`<br>Optional; Default value to return if no lines qualify. Default is empty (').<br>• `resultDelimiter`<br>Optional; Delimiter to use when concatenating matching lines. If not specified, "\n" or "\r\n" depending on original output line endings. |

## Task Instance Output by Line(s) Matching Regular Expression

| | |
|---|---|
| **Description** | Resolves to the line(s) of output data that match the specified regular expression, of the specified `outputType`, of the task instance that is resolving the function by specifying a regular expression.<br><br>• The complete outpout line is returned.<br>• If the output record of the specified `ouptutType` cannot be found, the function will remain unresolved. |
| **Syntax** | **`${_outputLinesByRegex`**`('outputType', 'regexPattern'[, maxCount, numberOfLinesBefore, numberOfLinesAfter, 'defaultValue', 'resultDelimiter'])}` |
| **Parameters** | • `outputType`<br>Required; Type of output to resolve: STDOUT, STDERR, FILE, or JOBLOG.<br>• `regexPattern`<br>Required; Regular expression used for determining if a line matches. Regular expression must match the whole line (see Example, below).<br>• `maxCount`<br>Optional; Maximum number of matching lines to return. Default is 1.<br>• `numberOfLinesBefore`<br>Optional; Number of lines before each matching line to return along with the matching line. Default is 0.<br>• `numberOfLinesAfter`<br>Optional; Number of lines after each matching line to return along with the matching line. Default is 0.<br>• `defaultValue`<br>Optional; Default value to return if no lines match the regular expression. Default is empty (').<br>• `resultDelimiter`<br>Optional; Delimiter to use when concatenating matching lines. If not specified, "\n" or "\r\n" depending on original output line endings. |
| **Example** | STDOUT contains:<br>*Some_Text*<br>*ABC=Some_String*<br>*More_Text*<br><br>${_outputLinesByRegex('STDOUT', '^ABC=')} Returns empty (the whole Line was not matched)<br>${_outputLinesByRegex('STDOUT', '^ABC=.*')} Returns ABC=Some_String |

## Sibling Task Instance Output by Line(s) Matching Regular Expression

| | |
|---|---|
| **Description** | Resolves to the line(s) of output data that match the specified regular expression, of the specified `outputType`, of the task instance specified by the `siblingName` parameter.<br><br>The sibling task instance must be within the same workflow, and the Execution User of the task instance that is resolving the function must have Read permission for the sibling task instance. |

| | |
|---|---|
| | • If the output record of the specified `outputType` cannot be found, the function will remain unresolved. |
| **Syntax** | **`${_outputLinesByRegexFromTask`**`('siblingName', 'outputType', 'regexPattern'[, maxCount, numberOfLinesBefore, numberOfLinesAfter, 'defaultValue', 'resultDelimiter'])}` |
| **Parameters** | • `siblingName`<br>Required; Name of a sibling task instance.<br>• `outputType`<br>Required; Type of output to resolve: STDOUT, STDERR, FILE, or JOBLOG.<br>• `regexPattern`<br>Required; Regular expression used for determining if a line matches.<br>• `maxCount`<br>Optional; Maximum number of matching lines to return. Default is 1.<br>• `numberOfLinesBefore`<br>Optional; Number of lines before each matching line to return along with the matching line. Default is 0.<br>• `numberOfLinesAfter`<br>Optional; Number of lines after each matching line to return along with the matching line. Default is 0.<br>• `defaultValue`<br>Optional; Default value to return if no lines match the regular expression. Default is empty ('').<br>• `resultDelimiter`<br>Optional; Delimiter to use when concatenating matching lines. If not specified, "\n" or "\r\n" depending on original output line endings. |

## Task Instance Output By XPath

| | |
|---|---|
| Description | Resolves to the XML output data of the task instance that is resolving the function, corresponding to the evaluated XPath expression.<br>• If the output record cannot be found, the function will remain unresolved.<br>• If the output record is found, but the path expression does not yield a result, the function will resolve to the default value. |
| Syntax | `${_`**`outputXPath`**`('outputType', 'xPathExpression'[, 'defaultValue', 'delimiter', prettyPrint])}` |
| Parameters | • `outputType`<br>Required; Type of output to resolve: STDOUT, STDERR, FILE, or JOBLOG.<br>• `xPathExpression`<br>Required; XPath expression. https://www.w3schools.com/xml/xpath_intro.asp<br>• `defaultValue`<br>Optional; Default value to return if the result is not found. Default is empty ('').<br>• `delimiter`<br>Optional; If `xPathExpression` evaluates to multiple results, the delimiter to be used to separate those results. Default is new line character (\n).<br>• `prettyPrint`<br>Optional; Specification (**true** or **false**) for whether or not XML output will be pretty printed (indented). Default is false. |
| Example | **`XML`**<br><br>`<message>`<br>`  <code>10</code>`<br>`</message>`<br><br>**`Function`** |

| | |
|---|---|
| | `${_outputXPath('STDOUT', '//code/text()')}`<br><br>**`Result`**<br><br>`10` |

## Sibling Task Instance Output By XPath

| | |
|---|---|
| Description | Resolves to the XML output data of the task instance specified by the `siblingName`, corresponding to the evaluated XPath expression.<br><br>The sibling task instance must be within the same workflow and the Execution User of the task instance that is resolving the function must have Read permission for the sibling task instance.<br><br>• If the output record cannot be found, the function will remain unresolved.<br>• If the output record is found but the path expression does not yield a result, the function will resolve to the default value. |
| Syntax | `${_outputXPathFromTask('siblingName', 'outputType', 'xPathExpression'[, 'defaultValue', 'delimiter', prettyPrint])}` |
| Parameters | • `siblingName`<br>Required; Name of a sibling task instance.<br>• `outputType`<br>Required; Type of output to resolve: STDOUT, STDERR, FILE, or JOBLOG.<br>• `xPathExpression`<br>Required; XPath expression. https://www.w3schools.com/xml/xpath_intro.asp<br>• `defaultValue`<br>Optional; Default value to return if the result is not found. Default is empty ('').<br>• `delimiter`<br>Optional; If `xPathExpression` evaluates to multiple results, the delimiter to be used to separate those results. Default is new line character (\n).<br>• `prettyPrint`<br>Optional; Specification (**true** or **false**) for whether or not XML output will be pretty printed (indented). Default is false. |
| Example | **`XML`**<br><br>`<message><code>10</code></message>`<br><br>**`Function`**<br><br>`${_outputXPathFromTask('Sibling_With_XML_Output','STDOUT','//code/text()')}`<br><br>**`Result`**<br><br>`10` |

## Task Instance Output By JsonPath

| | |
|---|---|
| Description | Resolves to the JSON output data of the task instance that is resolving the function, corresponding to the evaluated JsonPath expression.<br><br>• If the output record cannot be found, the function will remain unresolved.<br>• If the output record is found but the path expression does not yield a result, the function will resolve to the default value. |

| | |
|---|---|
| Syntax | `${_outputJsonPath('outputType', 'pathExpression'[, 'defaultValue', 'delimiter', prettyPrint])}` |
| Parameters | • `outputType`<br>Required; Type of output to resolve: STDOUT, STDERR, FILE, or JOBLOG.<br>• `PathExpression`<br>Required; JsonPath expression. https://github.com/json-path/Jsonpath<br>• `defaultValue`<br>Optional; Default value to return if the result is not found. Default is empty ('').<br>• `delimiter`<br>Optional; If `pathExpression` evaluates to multiple results, the delimiter to be used to separate those results. Default is new line character (\n).<br>• `prettyPrint`<br>Optional; Specification (**true** or **false**) for whether or not JSON output will be pretty printed (indented). Default is false. |
| Example | **JSON**<br><br>`{`<br>`  "code": 10`<br>`}`<br><br>**Function**<br><br>`${_outputJsonPath('STDOUT', '$.code')}`<br><br>**Result**<br><br>`10` |

## Sibling Task Instance Output By JsonPath

| | |
|---|---|
| Description | Resolves to the JSON output data of the task instance specified by the `siblingName`, corresponding to the evaluated JsonPath expression.<br><br>The sibling task instance must be within the same workflow and the Execution User of the task instance that is resolving the function must have Read permission for the sibling task instance.<br><br>• If the output record cannot be found, the function will remain unresolved.<br>• If the output record is found but the path expression does not yield a result, the function will resolve to the default value |
| Syntax | `${_outputJsonPathFromTask('siblingName', 'outputType', 'pathExpression'[, 'defaultValue', 'delimiter', prettyPrint])}` |
| Parameters | • `siblingName`<br>Required; Name of a sibling task instance.<br>• `outputType`<br>Required; Type of output to resolve: STDOUT, STDERR, FILE, or JOBLOG.<br>• `PathExpression`<br>Required; JsonPath expression. https://github.com/json-path/Jsonpath<br>• `defaultValue`<br>Optional; Default value to return if the result is not found. Default is empty ('').<br>• `delimiter`<br>Optional; If `pathExpression` evaluates to multiple results, the delimiter to be used to separate those results. Default is new line character (\n). |

| | |
|---|---|
| | • `prettyPrint`<br>Optional; Specification (**true** or **false**) for whether or not JSON output will be pretty printed (indented). Default is false. |
| Example | **JSON**<br><br>`{`<br>`  "code": 10`<br>`}`<br><br>**Function**<br><br>`${_outputJsonPathFromTask('Sibling_With_JSON_Output', 'STDOUT', '$.code')}`<br><br><br>**Result**<br><br>`10` |

## Task Instance Output By JsonPath As Array

| | |
|---|---|
| Description | Resolves to the JSON output data of the task instance that is resolving the function, corresponding to the evaluated JsonPath expression.<br><br>• If the output record cannot be found, the function will remain unresolved.<br>• If the output record is found but the path expression does not yield a result, the function will resolve to the default value. |
| Syntax | `${_outputJsonPathAsArray('outputType', 'pathExpression'[, 'defaultValue', prettyPrint])}` |
| Parameters | • `outputType`<br>Required; Type of output to resolve: STDOUT, STDERR, FILE, or JOBLOG.<br>• `PathExpression`<br>Required; JsonPath expression. https://github.com/json-path/Jsonpath<br>• `defaultValue`<br>Optional; Default value to return if the result is not found. Default is empty ('').<br>• `prettyPrint`<br>Optional; Specification (**true** or **false**) for whether or not JSON output will be pretty printed (indented). Default is false. |
| Example | **JSON**<br><br>`[`<br>`  {`<br>`  "message" : "Hello",`<br>`  "code" : 10`<br>`  },`<br>`  {`<br>`  "message" : "World!",`<br>`  "code" : 20`<br>`  }`<br>`]`<br><br>**Function** |

| | |
|---|---|
| | `${_outputJsonPathAsArray('STDOUT', '$[*].message', '', true)}`<br><br>**Result**<br><br>`[`<br>`  "Hello",`<br>`  "World!"`<br>`]` |

## Sibling Task Instance Output By JsonPath As Array

| | |
|---|---|
| Description | Resolves to the JSON output data of the task instance specified by the `siblingName`, corresponding to the evaluated JsonPath expression.<br><br>The sibling task instance must be within the same workflow and the Execution User of the task instance that is resolving the function must have Read permission for the sibling task instance.<br><br>• If the output record cannot be found, the function will remain unresolved.<br>• If the output record is found but the path expression does not yield a result, the function will resolve to the default value. |
| Syntax | `${_outputJsonPathFromTask('siblingName', 'outputType', 'pathExpression'[, 'defaultValue', prettyPrint])}` |
| Parameters | • `siblingName`<br>Required; Name of a sibling task instance.<br>• `outputType`<br>Required; Type of output to resolve: STDOUT, STDERR, FILE, or JOBLOG.<br>• `PathExpression`<br>Required; JsonPath expression. https://github.com/json-path/Jsonpath<br>• `defaultValue`<br>Optional; Default value to return if the result is not found. Default is empty (').<br>• `prettyPrint`<br>Optional; Specification (**true** or **false**) for whether or not JSON output will be pretty printed (indented). Default is false. |
| Example | **JSON**<br><br>`[`<br>`  {`<br>`  "message" : "Hello",`<br>`  "code" : 10`<br>`  },`<br>`  {`<br>`  "message" : "World!",`<br>`  "code" : 20`<br>`  }`<br>`]`<br><br>**Function**<br><br>`${_outputJsonPathAsArrayFromTask('Sibling_With_JSON_Output', 'STDOUT', '$[*].message')}`<br><br>**Result**<br>`["Hello","World!"]` |

**JSON**

# Script Functions

## Returns Path to Data Script

| | |
|---|---|
| **Description** | Returns a token representing the path to a Data Script that you want to embed. |
| **Syntax** | **${_scriptPath**('script_name')**}** |
| **Parameters** | • `script_name`<br>Required; Name of the Data Script. |
| **Example** | `${_scriptPath('myscriptdata')}` |

Note

**_scriptPath** requires Agent 6.4.0.0 or later.

# SQL/Stored Procedure Functions

## Return Column Names for SQL Results from Current Task

| | |
|---|---|
| **Description** | Returns the column names for the SQL results from the current SQL or Stored Procedure task. Column names are separated by the specified `separator`. |
| **Syntax** | **${_resultsColumnNames**(['separator'])} |
| **Parameters** | • `separator`<br>Optional; Column name separator. Default is comma (,). |

## Return Column Names for SQL Results from Sibling Task

| | |
|---|---|
| **Description** | Returns the column names for the SQL results from a sibling SQL or Stored Procedure task, within the same workflow. Column names are separated by the specified `separator`. |
| **Syntax** | **${_resultsColumnNamesFromTask**('name'[, 'separator'])} |
| **Parameters** | • `name`<br>Required; Name of the sibling task that the results should come from. The task must be within the same workflow. |

| | |
|---|---|
| | • `separator`<br>Optional; Column name separator. Default is comma (,). |

## Return SQL Results from Current Task

| | |
|---|---|
| **Description** | Returns all SQL results from the current SQL or Stored Procedure task. Columns are separated by the specified `separator` and rows are separated by a new line. |
| **Syntax** | **`${_resultsAll`**`(['separator', 'rowSeparator'])}` |
| **Parameters** | • `separator`<br>Optional; Column separator. Default is comma (,).<br>• `rowSeparator`<br>Optional; Overrides default New Line character. |

## Return SQL Results from Sibling Task

| | |
|---|---|
| **Description** | Returns all SQL results from a sibling SQL or Stored Procedure task, within the same workflow. Columns are separated by the specified `separator` and rows are separated by a new line. |
| **Syntax** | **`${_resultsAllFromTask`**`('name'[, 'separator', 'rowSeparator'])}` |
| **Parameters** | • `name`<br>Required; Name of the task that the results should come from. The task must be within the same workflow.<br>• `separator`<br>Optional; Column separator. Default is comma (,).<br>• `rowSeparator`<br>Optional; Overrides default New Line character. |

## Return SQL Warnings from Current Task

| | |
|---|---|
| **Description** | Returns all SQL warnings from the current SQL or Stored Procedure task. Columns are separated by the specified `separator` and rows are separated by a new line. |
| **Syntax** | **`${_SQLWarnings`**`(['separator'])}` |
| **Parameters** | • `separator`<br>Optional; Column separator. Default is comma (,). |

## Return SQL Warnings from Sibling Task

| | |
|---|---|
| **Description** | Returns all SQL warnings from a sibling SQL or Stored Procedure task, within the same workflow. Columns are separated by the specified `separator` and rows are separated by a new line. |
| **Syntax** | **`${_SQLWarningsFromTask`**`('name'[, 'separator'])}` |

| | |
|---|---|
| **Parameters** | • `name`<br>Required; Name of the sibling task that the warnings should come from. The task must be within the same workflow.<br>• `separator`<br>Optional; Column separator. Default is comma (,). |

## Return String Value of Row/Column by Column Name

| | |
|---|---|
| **Description** | Returns the string value of a row/column from a previously executed SQL task within the same workflow, or from the current SQL task. |
| **Syntax** | **`${_resultsColumn`**`('name', 'colname'[, rownum, 'default_value'])}` |
| **Parameters** | • `name`<br>Required; Name of a sibling SQL task within the same workflow from which you want the function to fetch results. If you want to execute the function against the current task, use an empty string for the name parameter.<br>• `colname`<br>Required; Name of column to retrieve.<br>• `rownum`<br>Optional; Numeric row number in result set to retrieve. Default is 1.<br>• `default_value`<br>Optional; Default value to return if result not found. |

## Return String Value of Row/Column by Column Number

| | |
|---|---|
| **Description** | Returns the string value of a row/column from a previously executed SQL task within the same workflow, or from the current SQL task. |
| **Syntax** | **`${_resultsColumnByNo`**`('name', colnum[, rownum, 'default_value'])}` |
| **Parameters** | • `name`<br>Required; Name of a sibling SQL task within the same workflow from which you want the function to fetch results. If you want to execute the function against the current task, use an empty string for the name parameter.<br>• `colnum`<br>Required; Number of column to retrieve. First column in result is 1, second is 2, and so on.<br>• `rownum`<br>Optional; Numeric row number in result set to retrieve. Default is 1.<br>• `default_value`<br>Optional; Default value to return if result not found. |

## Return String Values of Columns

| | |
|---|---|
| **Description** | Returns the string values of columns in a specific row in CSV (comma-separated values) format, from a previously executed SQL task within the same workflow, or from the current SQL task. |
| **Syntax** | **`${_resultsColumnsCSV`**`('name'[, rownum])}` |
| | |

| | |
|---|---|
| **Parameters** | • `name`<br>Required; Name of a sibling SQL task within the same workflow from which you want the function to fetch results. If you want to execute the function against the current task, use an empty string for the `name` parameter.<br>• `rownum`<br>Optional; Numeric row number in result set to retrieve. Default is 1. |

# String Functions

String Functions can accept:

- String content in a String parameter.
- Variable name in a String parameter (prefixed with _var) from which string content can be obtained.
- Integer and Boolean parameters.

For String functions that accept a String value parameter directly, the value parameter can be specified using hard-coded text, variables, functions, or any combination of the three.

Note

When using String functions that accept a String value parameter directly, you should be aware of expectations with respect to escape characters and escape sequences (see Escape Sequences, below).

For String functions that accept a variable name parameter, the fully resolved value of the variable by the specified name will be used as the String value argument. The variable must be fully resolvable and must not contain an unresolved function.

Note

Indexing functions use zero-based numbering; that is, the initial element is assigned the index 0.

## Escape Sequences

An escape character preceded by a backslash (\) is an escape sequence (see the following table for a list of escape sequences).

If you are using a String function to manipulate a String value that potentially may contain an escape sequence, you should use the String function that accepts a variable name parameter to allow for passing the value to the function without the escape sequence being interpreted.

| Escape Sequences | Escape Sequence Description |
|---|---|
| \t | Insert a tab in the text at this point. |
| \b | Insert a backspace in the text at this point. |
| \n | Insert a newline in the text at this point. |
| \r | Insert a carriage return in the text at this point. |
| \f | Insert a formfeed in the text at this point. |
| \' | Insert a single quote character in the text at this point. |
| \' | Insert a double quote character in the text at this point. |

| | |
|---|---|
| \\ | Insert a backslash character in the text at this point. |

## Convert Characters in Value to Lower Case

| | |
|---|---|
| **Description** | Converts all of the characters in the `value` to lower case using the rules of the default locale. |
| **Syntax** | **`${_toLowerCase`**`('value')}` |
| **Parameters** | • `value`<br>Required; String to convert to lower case. |

## Convert Characters in Variable to Lower Case

| | |
|---|---|
| **Description** | Converts all of the characters in the variable to lower case using the rules of the default locale. |
| **Syntax** | **`${_varToLowerCase`**`('variableName')}` |
| **Parameters** | • `variableName`<br>Required; Name of the variable being passed into the function. |

## Convert Characters in Value to Upper Case

| | |
|---|---|
| **Description** | Converts all of the characters in the `value` to upper case using the rules of the default locale. |
| **Syntax** | **`${_toUpperCase`**`('value')}` |
| **Parameters** | • `value`<br>Required; String to convert to upper case. |

## Convert Characters in Variable to Upper Case

| | |
|---|---|
| **Description** | Converts all of the characters in the variable to upper case using the rules of the default locale. |
| **Syntax** | **`${_varToUpperCase`**`('variableName')}` |
| **Parameters** | • `variableName`<br>Required; Name of the variable being passed into the function. |

## Escape Characters in Variable Using XML Entities

| | |
|---|---|
| **Description** | Escapes the characters in a variable value using XML entities. |
| **Syntax** | **`${_varEscapeXml`**`('variableName')}` |
| **Parameters** | • `variableName`<br>Required; Name of the variable being passed into the function. The variable value will be escaped for insertion into XML. |
| **Example** | Variable Name:<br>escape_me<br><br>Variable Value:<br>`1234567890\E-=[]\;',./ ~!@#$%^&*()_+{}\|:"<>?<br><br>`${_varEscapeXml('escape_me')} --> `1234567890\E-=[]\;',./ ~!@#$%^&*()_+{}\|:"<>?` |

## Escape Characters in Variable Using JSON String Rules

| | |
|---|---|
| **Description** | Escapes the characters in a variable value using JSON string values. |
| **Syntax** | **`${_varEscapeJson`**`('variableName')}` |
| **Parameters** | • `variableName`<br>Required; Name of the variable being passed into the function. The variable value will be escaped for insertion into JSON. |
| **Example** | Variable Name:<br>escape_me<br><br>Variable Value:<br>`1234567890\E-=[]\;',./ ~!@#$%^&*()_+{}\|:"<>?<br><br>`${_varEscapeJson('escape_me')} --> `1234567890\\E-=[]\\;',.\/ ~!@#$%^&*()_+{}\|:\"<>?` |

## Escape Characters in Variable Using JavaScript String Rules

| | |
|---|---|
| **Description** | Escapes the characters in a variable value using JavaScript String rules. |

| | |
|---|---|
| **Syntax** | **`${_varEscapeJavaScript`**`('variableName')}` |
| **Parameters** | • `variableName`<br>Required; Name of the variable being passed into the function. The variable value will be escaped for insertion into JavaScript. |
| **Example** | Variable Name:<br>escape_me<br><br>Variable Value:<br>`1234567890\E-=[]\;',./ ~!@#$%^&*()_+{}\|:"<>?<br><br>`${_varEscapeJavaScript('escape_me')} --> `1234567890\\E-=[]\\;\',.\/ ~!@#$%^&*()_+{}\|:\"<>?` |

## Escape Characters in Variable Using HTML Entities

| | |
|---|---|
| **Description** | Escapes the characters in a variable value using HTML entities. (Supports all known HTML 4.0 entities.) |
| **Syntax** | **`${_varEscapeHtml`**`('variableName')}` |
| **Parameters** | • `variableName`<br>Required; Name of the variable being passed into the function. The variable value will be escaped for insertion into HTML. |
| **Example** | Variable Name:<br>escape_me<br><br>Variable Value:<br>`1234567890\E-=[]\;',./ ~!@#$%^&*()_+{}\|:"<>?<br><br>`${_varEscapeHtml('escape_me')} --> `1234567890\E-=[]\;',./ ~!@#$%^&*()_+{}\|:"<>?` |

## Escape Characters in Variable as a Literal Pattern

| | |
|---|---|
| **Description** | Returns a literal regular expression pattern String for the value of the specified variable.<br><br>This method produces a String that can be used to create a Pattern that would match the String as if it were a literal pattern. |
| **Syntax** | **`${_varLiteralPattern`**`('variableName')}` |

| | |
|---|---|
| **Parameters** | • `variableName`<br>Required; Name of the variable being passed into the function. The variable value will be escaped for insertion into a regular expression as a literal pattern. |
| **Example** | Variable Name:<br>escape_me<br><br>Variable Value:<br>`1234567890\E-=[]\;',./ ~!@#$%^&*()_+{}\|:"<>?<br><br>`${_varLiteralPattern('escape_me')} --> \Q`1234567890\E\\E\Q-=[]\;',./ ~!@#$%^&*()_+{}\|:"<>?\E` |

## Randomly Generate a String

| | |
|---|---|
| **Description** | Randomly generates a String with a specified length. |
| **Syntax** | **`${_randomString`**`(length[, 'excludeCharacters', 'defaultCharacters'])}` |
| **Parameters** | • `length`<br>Required; String length.<br>• `excludeCharacters`<br>Optional; String containing characters to exclude from the default character set.<br>• `defaultCharacters`<br>Optional; String for overriding default character set.<br><br>Note<br>The following characters are included in the default character set, in addition to the space character.<br>ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890`-=~!@#$%^&*()_+[]\{}\|;':",./<>? |
| **Example** | `${_randomString(24, '', 'ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890@#$%*')} --> 5*L8T1RN#$AQWEKPA@BQ19JD` |

## Replace Substring of Value with Regular Expression

| | |
|---|---|
| **Description** | Replaces each substring of value that matches the specified regular expression, regex, with the specified replacement. |
| **Syntax** | **`${_replaceAll`**`('value', 'regex', 'replacement')}` |
| **Parameters** | • `value`<br>Required; Input string. |

| | |
|---|---|
| | • `regex`<br>Required; Regular expression.<br>• `replacement`<br>Required; Replacement string. |

## Replace Substring of Variable with Regular Expression

| | |
|---|---|
| **Description** | Replaces each substring of `variableName` that matches the specified regular expression, regex, with the specified replacement. |
| **Syntax** | **`${_varReplaceAll`**`('variableName', 'regex', 'replacement')}` |
| **Parameters** | • `variableName`<br>Required; Name of the variable being passed into the function.<br>• `regex`<br>Required; Regular expression.<br>• `replacement`<br>Required; Replacement string. |

## Return Base64 Encoded String

| | |
|---|---|
| **Description** | Returns the value of the specified variable encoded using the Base64 encoding scheme. |
| **Syntax** | **`${_varEncodeBase64`**`('variableName'[, 'charset'])}` |
| **Parameters** | • `variableName`<br>Required; Name of the variable whose value will be encoded using the Base64 encoding scheme.<br>• `charset`<br>Optional; Name of the charset; default UTF-8. |
| **Example** | Where Variable rawstring contains a value of "Test String":<br>`${_varEncodeBase64('rawstring')} --> VGVzdCBTdHJpbmc=` |

## Return Copy of Value with Whitespace Omitted

| | |
|---|---|
| **Description** | Returns a copy of `value`, with leading and trailing whitespace omitted. |
| **Syntax** | **`${_trim`**`('value')}` |
| **Parameters** | • `value`<br>Required; String to trim. |

## Return Copy of Variable with Whitespace Omitted

| | |
|---|---|
| **Description** | Returns a copy of `variableName`, with leading and trailing whitespace omitted. |
| **Syntax** | **`${_varTrim`**`('variableName')}` |
| **Parameters** | • `variableName`<br>Required; Name of the variable being passed into the function. |

## Return Index of Substring in String Value

| | |
|---|---|
| **Description** | Returns the index within the string value of the first occurrence of the specified substring, `str`. |
| **Syntax** | **`${_indexOf`**`('value', 'str')}` |
| **Parameters** | • `value`<br>Any string.<br>• `str`<br>Substring to search for. If the `str` argument occurs as a substring within the value, then the index of the first character of the first such substring is returned; if it does not occur as a substring, -1 is returned. |

## Return Index of Substring in String Variable

| | |
|---|---|
| **Description** | Returns the index within the string variable of the first occurrence of the specified substring, `str`. |
| **Syntax** | **`${_varIndexOf`**`('variableName', 'str')}` |
| **Parameters** | • `variableName`<br>Required; Name of the variable being passed into the function.<br>• `str`<br>Required; Substring to search for. If the `str` argument occurs as a substring within the variable, the index of the first character of the first such substring is returned; if it does not occur as a substring, -1 is returned. |

## Return Index of Substring Plus Offset in String Value

| | |
|---|---|
| **Description** | Returns the index within this string of the first occurrence of the specified substring plus the specified offset. The integer returned is the smallest value. |
| **Syntax** | **`${_indexOfWithOffset`**`('value', 'str', offset)}` |
| **Parameters** | • `value`<br>Required; Any string. |

| | |
|---|---|
| | • `str`<br>Required; Substring to search for. If the `str` argument occurs as a substring within the value, then the index of the first character of the first such substring is returned; if it does not occur as a substring, -1 is returned.<br>• `offset`<br>Required; Number (positive or negative) to offset the found index. |

## Return Index of Substring Plus Offset in String Variable

| | |
|---|---|
| **Description** | Returns the index within this string of the first occurrence of the specified substring plus the specified offset. The integer returned is the smallest variable. |
| **Syntax** | **`${_varIndexOfWithOffset`**`('variableName', 'str', offset)}` |
| **Parameters** | • `variableName`<br>Required; Name of the variable being passed into the function.<br>• `str`<br>Required; Substring to search for. If the `str` argument occurs as a substring within the variable, then the index of the first character of the first such substring is returned; if it does not occur as a substring, -1 is returned.<br>• `offset`<br>Required; Number (positive or negative) to offset the found index. |

## Return Index of Rightmost Occurrence of Substring in String Value

| | |
|---|---|
| **Description** | Returns the index within the string value of the rightmost occurrence of the specified substring, `str`. |
| **Syntax** | **`${_lastIndexOf`**`('value', 'str')}` |
| **Parameters** | • `value`<br>Required; Any string.<br>• `str`<br>Required; Substring to search for. If the `str` argument occurs one or more times as a substring within the value, then the index of the first character of the last such substring is returned. If it does not occur as a substring, -1 is returned. |

## Return Index of Rightmost Occurrence of Substring in String Variable

| | |
|---|---|
| **Description** | Returns the index within the string variable of the rightmost occurrence of the specified substring, `str`. |
| **Syntax** | **`${_varLastIndexOf`**`('variableName', 'str')}` |
| **Parameters** | • `variableName`<br>Required; Name of the variable being passed into the function.<br>• `str`<br>Required; Substring to search for. If the `str` argument occurs one or more times as a substring within the variable, then the index of the first character of the last such substring is returned. If it does not occur as a substring, -1 is returned. |

## Return Index of Rightmost Occurrence of Substring Plus Offset in String Value

| | |
|---|---|
| **Description** | Returns the index within this string of the rightmost occurrence of the specified substring, plus the specified offset. The returned index is the largest value. |
| **Syntax** | **`${_lastIndexOfWithOffset`**`('value', 'str', offset)}` |
| **Parameters** | • `value`<br>Required; Any string.<br>• `str`<br>Required; Substring to search for. If the `str` argument occurs as a substring within the value, then the index of the first character of the first such substring is returned; if it does not occur as a substring, -1 is returned.<br>• `offset`<br>Required; Number (positive or negative) to offset the found index. |

## Return Index of Rightmost Occurrence of Substring Plus Offset in String Variable

| | |
|---|---|
| **Description** | Returns the index within this string of the rightmost occurrence of the specified substring, plus the specified offset. The returned index is the largest variable. |
| **Syntax** | **`${_varLastIndexOfWithOffset`**`('variableName', 'str', offset)}` |
| **Parameters** | • `variableName`<br>Required; Name of the variable being passed into the function.<br>• `str`<br>Required; Substring to search for. If the `str` argument occurs as a substring within the variable, then the index of the first character of the first such substring is returned; if it does not occur as a substring, -1 is returned.<br>• `offset`<br>Required; Number (positive or negative) to offset the found index. |

## Return Length of Value

| | |
|---|---|
| **Description** | Returns the length of `value`. |
| **Syntax** | **`${_length`**`('value')}` |
| **Parameters** | • `value`<br>Required; Any string. |

## Return Length of Variable

| | |
|---|---|
| **Description** | Returns the length of `variableName`. |
| **Syntax** | **`${_varLength`**`('variableName'[, useEmptyForUndefined])}` |

| | |
|---|---|
| **Parameters** | • `variableName`<br>Required; Name of the variable being passed into the function.<br>• `useEmptyForUndefined`<br>Optional; Specification (**true** or **false**) for the handling of a missing variable name. Default is false.<br>  • If `useEmptyForUndefined` = **true**, the function will return 0.<br>  • If `useEmptyForUndefined` = **false**, the function will remain unresolved if the variable name does not exist. |

## Return New String that is Substring of Value

| | |
|---|---|
| **Description** | Returns a new string that is a substring of `value`. The substring begins at `beginIndex` and extends to the character at `endIndex` -1. |
| **Syntax** | **`${_substring`**`('value', beginIndex[, endIndex])}` |
| **Parameters** | • `value`<br>Required; String to make a substring from.<br>• `beginIndex`<br>Required; Beginning index, inclusive.<br>• `endIndex`<br>Optional; Ending index, exclusive. |
| **Example** | `${_substring('hamburger', 4, 8)} --> urge`<br>`${_substring('smiles', 1, 5)} --> mile` |

## Return New String that is Substring of Variable

| | |
|---|---|
| **Description** | Returns a new string that is a substring of `variableName`. The substring begins at `beginIndex` and extends to the character at `endIndex` -1. |
| **Syntax** | **`${_varSubstring`**`('variableName', beginIndex[, endIndex])}` |
| **Parameters** | • `variableName`<br>Required; Name of the variable being passed into the function.<br>• `beginIndex`<br>Required; Beginning index, inclusive.<br>• `endIndex`<br>Optional; Ending index, exclusive. |
| **Examples** | If the value of the `food` variable is **hamburger**, and the value of the `face` variable is **smiles**: |

```
${_varSubstring('food', 4, 8)} --> urge
${_varSubstring('face', 1, 5)} --> mile
```

### Return URL-Encoded String

| | |
|---|---|
| **Description** | Returns a URL-encoded string according to the ASCII Encoding Reference for UTF-8; all non-alphanumeric characters are replaced with their equivalent hexadecimal escape sequences. |
| **Syntax** | **`${_varEncodeUrl`**`('variableName')}` |
| **Parameters** | • `variableName`<br>Required; Name of the variable whose value will be converted to a URL encoded string. |
| **Example** | Where Variable rawstring contains a value of "`ABC$%^----DEF`":<br>`${_varEncodeUrl(‘rawstring’)} --> ABC%24%25%5E----DEF` |

## System Functions

### Display Variables

| | |
|---|---|
| **Description** | Displays all the defined and built-in variables associated with the task instance. |
| **Syntax** | **`${_scope}`** |
| **Parameters** | (none) |
| **Example** | `${_scope} --> {ops_workflow_id=, ops_task_type=Unix, ops_status=DEFINED, ops_retry_interval=60, ops_exit_code=0, ops_retry_maximum=0, ops_cmd_parms=, ops_cmd=ls -la; exit ${_random('9')};, ops_retry_count=0, ops_agent_id=67e4994143d2617201cdf4ba9df9ab0a, ops_task_id=84880af243d26172019aa1d25988a8f9, ops_task_name=uc - Linux Ls}` |

### Generate Random Number

| | |
|---|---|
| **Description** | Generates a random number between `max` (inclusive) and `min` (inclusive) |
| **Syntax** | **`${_random`**`([max, min])}` |
| **Parameters** | • `max`<br>Optional; Upper bound (inclusive) on the random number. Default is 9.<br>• `min`<br>Optional; Lower bound (inclusive) on the random number. Default is 0. |

## Resolve to GUID (Globally Unique ID)

| | |
|---|---|
| **Description** | Resolves to a 32-byte GUID (Globally Unique ID). |
| **Syntax** | **`${_guid}`** |
| **Parameters** | (none) |

## Resolve to Host Name

| | |
|---|---|
| **Description** | Resolves to the hostname of the machine running the Controller, if available. |
| **Syntax** | **`${_hostname}`** |
| **Parameters** | (none) |

## Resolve to IP Address

| | |
|---|---|
| **Description** | Resolves to the IP address of the machine running the Controller. |
| **Syntax** | **`${_ipaddress}`** |
| **Parameters** | (none) |

## Resolve to SYS_ID

| | |
|---|---|
| **Description** | Resolves to the sys_id of the first task instance found within the same workflow specified by the sibling name. |
| **Syntax** | **`${_siblingid`**`('sibling_name')}` |
| **Parameters** | • `sibling_name`<br>Required; Sibling name. |

| | |
|---|---|
| **Example** | `${_siblingid('Timer 60')} --> 5dbaaab943d26172015e10ab3e894e10` |

## Resolve to Variable Value

| | |
|---|---|
| **Description** | Locates the specified variable in the specified sibling task instance within the same workflow and resolves to the variable value. |
| **Syntax** | **`${_varLookup`**`('sibling_name', 'variable_name'[,'def'])}` |
| **Parameters** | • `sibling_name`<br>Required; Name of the sibling task instance from which the function is collecting the variable value.<br>• `variable_name`<br>Required; Name of the variable being collected by the function.<br>• `def`<br>Optional; Default value to return if the variable is not defined in the sibling task instance. |

## Resolve Variable

| | |
|---|---|
| **Description** | Resolves the variable specified by the `variable_name` parameter and substitutes the `default_value` if the variable cannot be resolved. |
| **Syntax** | **`${_resolve`**`('variable_name', 'default_value')}` |
| **Parameters** | • `variable_name`<br>Required; Variable name.<br>• `default_value`<br>Required; Default value to use if the variable cannot be resolved. |

## Resolve Variable (Advanced)

| | |
|---|---|
| **Description** | Resolves the variable specified by the `variable_name` parameter and substitutes the default value if the variable cannot be resolved. |
| **Syntax** | **`${_resolveadv`**`('variable_name', 'default_value', [use_default_if_blank])}` |
| **Parameters** | • `variable_name`<br>Required; Variable name.<br>• `default_value`<br>Required; Default value to use if the variable cannot be resolved.<br>• `use_default_if_blank`<br>Optional; Specification (**true** or **false**) for whether or not to use the default value if the variable is empty or blank. (If `use_default_if_blank` is false, `_resolveadv` behaves like _resolve.) |

# Universal Task Functions

## Convert Array Field Variable

| | |
|---|---|
| **Description** | |
| **Syntax** | **${_convertArrayFieldVariable**('arrayFieldVariableName'[,'delimiter', 'separator', 'keyQuote', 'valueQuote'])} |
| **Parameters** | • arrayFieldVariableName<br>Required; Name of the variable for the Array Field; for example, ***ops_af_p2***.<br>• delimeter<br>Optional; Value to be used to delimit the Name from the Value. Default is **=**.<br>• separator<br>Optional; Value to be used to separate one entry/row from the next. Default is comma (**,**).<br>• keyQuote<br>Optional; Quoting to be used around the Name. Default is "no quoting."<br>• valueQuote<br>Optional; Quoting to be used around the Value. Default is "no quoting." |
| **Example** | convertArrayFieldVariable(String arrayFieldVariableName, String delimiter, String separator, String keyQuote, String valueQuote)<br>==========<br><br>${_convertArrayFieldVariable('ops_af_p1')}<br>P1A=5,P1B=42<br>---------- |

```
${_convertArrayFieldVariable('ops_af_p1', '=')}
P1A=5,P1B=42
---------- 

${_convertArrayFieldVariable('ops_af_p1', ':')}
P1A:5,P1B:42
---------- 

${_convertArrayFieldVariable('ops_af_p1', ':', ';')}
P1A:5;P1B:42
---------- 

${_convertArrayFieldVariable('ops_af_p1', '::', ',', '\'')}
'P1A'::5,'P1B'::42
---------- 

${_convertArrayFieldVariable('ops_af_p1', '=', ',', '\'', '"')}
'P1A'="5",'P1B'="42"
---------- 

${_convertArrayFieldVariable('ops_af_p1', "", "", '\'', '\'')}
'P1A'='5','P1B'='42'
---------- 

${_convertArrayFieldVariable('ops_af_p1', ':', '\\n', '\'', '\'')}
'P1A':'5'
```

```
'P1B':'42'
---------------

${_convertArrayFieldVariable('ops_af_p2')}
P2A=Red,P2B=White,P2C=Blue
---------------

${_convertArrayFieldVariable('ops_af_p2', '=')}
P2A=Red,P2B=White,P2C=Blue
---------------

${_convertArrayFieldVariable('ops_af_p2', ':')}
P2A:Red,P2B:White,P2C:Blue
---------------

${_convertArrayFieldVariable('ops_af_p2', ':', ';')}
P2A:Red;P2B:White;P2C:Blue
---------------

${_convertArrayFieldVariable('ops_af_p2', '::', ',', '\'')}
'P2A'::Red,'P2B'::White,'P2C'::Blue
---------------

${_convertArrayFieldVariable('ops_af_p2', '=', ',', '\'', '"')}
'P2A'="Red",'P2B'="White",'P2C'="Blue"
---------------

${_convertArrayFieldVariable('ops_af_p2', '', '', '\'', '\'')}
'P2A'='Red','P2B'='White','P2C'='Blue'
---------------

${_convertArrayFieldVariable('ops_af_p2', ':', '\\n', '\'', '\'')}
'P2A':'Red'
'P2B':'White'
'P2C':'Blue'
---------------
```

## Get Array Field Variable Value

| | |
|---|---|
| **Description** | |
| **Syntax** | **`${_getArrayFieldVariableValue`**`('arrayFieldVariableName', 'name')}` |
| **Parameters** | • `arrayFieldVariableName`<br>Required; Name of the variable for the Array Field; for example, ***ops_af_p1***.<br>• `name`<br>Required; Name of the entry for which the value is to be returned. |
| **Example** | `ops_af_p1`<br>`----------` |

```
P1A=5
P1B=42
----------

ops_af_p2
----------
P2A=Red
P2B=White
P2C=Blue
----------

getArrayFieldVariableValue(String arrayFieldVariableName, String name)
==========================================================

P1A = ${_getArrayFieldVariableValue('ops_af_p1', 'P1A')}
----------
P1A=5
----------

P1B = ${_getArrayFieldVariableValue('ops_af_p1', 'P1B')}
----------
P1B=42
----------

P2A = ${_getArrayFieldVariableValue('ops_af_p2', 'P2A')}
----------
P2A=Red
----------

P2B = ${_getArrayFieldVariableValue('ops_af_p2', 'P2B')}
----------
P2B=White
----------

P2C = ${_getArrayFieldVariableValue('ops_af_p2', 'P2C')}
----------
P2C=Blue
----------
```

# Web Service Functions

All functions will remain unresolved if no Web Service output record can be found for the task instance, for the current attempt.

All functions will remain unresolved if a required parameter either is not specified or specified incorrectly.

## Raw Output from Web Service Task

| | |
|---|---|
| **Description** | Resolves to the raw output data of the Web Service task instance that is resolving the function.<br>• If the output record cannot be found, the function will remain unresolved.<br>• If the output record is found, but the path expression does not yield a result, the function will resolve to the default value. |
| | |

| | |
|---|---|
| **Syntax** | **${_responseRaw**(['default_value'])} |
| **Parameters** | • default_value<br>Optional; Default value to return if the result is not found. Default is empty ('). |

## Raw Output from Sibling Web Service Task

| | |
|---|---|
| **Description** | Resolves to the raw output data of the Web Service task instance specified by the siblingName parameter.<br><br>The sibling task instance must be within the same workflow, and the Execution User of the task instance that is resolving the function must have Read permission for the sibling task instance.<br><br>• If the output record cannot be found, the function will remain unresolved.<br>• If the output record is found but the path expression does not yield a result, the function will resolve to the default value. |
| **Syntax** | **${_responseRawFromTask**('siblingName'[,'defaultValue'])} |
| **Parameters** | • siblingName<br>Required; Name of a sibling task instance.<br>• default_value<br>Optional; Default value to return if the result is not found. Default is empty ('). |

## XML Output Data from Web Service Task

| | |
|---|---|
| **Description** | Resolves to the XML output data of the Web Service task instance that is resolving the function, corresponding to the evaluated xPath expression.<br><br>• If the output record cannot be found, the function will remain unresolved.<br>• If the output record is found, but the path expression does not yield a result, the function will resolve to the default value. |
| **Syntax** | **${_responseXPath**('xPathExpression'[,'defaultValue','delimiter',prettyPrint])} |
| **Parameters** | • xPathExpression<br>Required; xPath expression, https://www.w3schools.com/xml/xpath_intro.asp<br>• defaultValue<br>Optional; Default value to return if the result is not found. Default is empty (').<br>• delimiter<br>Optional; If xPathExpression evaluates to multiple results, the delimiter to be used to separate those results. Default is new line character (\n).<br>• prettyPrint<br>Optional; Specification (**true** or **false**) for whether or not XML output will be pretty printed (indented). Default is false. |
| **Examples** | If you want to obtain the info element text from the following Web Service Task XML output, you could use either of two examples for this Function.<br><br>`<?xml version="1.0" encoding="UTF-8" standalone="yes"?><command-response><type>set_variable</type><success>true</success><info>No changes detected for variable variableName, ignoring Set Variable command.</info><errors></errors></command-response>` |

**Example 1**

```
${_responseXPath('//info')}
```

Select the info node in the document no matter where it is.

**Example 2**

```
${_responseXPath('/command-response/info')}
```

Select the info node from a specific path in the document, starting from the root node.

Using either of these examples will resolve to the following: No changes detected for variable `variableName`, ignoring Set Variable command.

## XML Output Data From Sibling Web Service Task

| | |
|---|---|
| **Description** | Resolves to the XML output data of the Web Service task instance specified by the siblingName, corresponding to the evaluated xPath expression.<br><br>The sibling task instance must be within the same workflow and the Execution User of the task instance that is resolving the function must have Read permission for the sibling task instance.<br><br>• If the output record cannot be found, the function will remain unresolved.<br>• If the output record is found but the path expression does not yield a result, the function will resolve to the default value. |
| **Syntax** | **`${_responseXPathFromTask`**`('siblingName','xPathExpression'[,'defaultValue','delimiter',prettyPrint])}` |
| **Parameters** | • `siblingName`<br>Required; Name of a sibling task instance.<br>• `xPathExpression`<br>Required; xPath expression, https://www.w3schools.com/xml/xpath_intro.asp<br>• `defaultValue`<br>Optional; Default value to return if the result is not found. Default is empty ('').<br>• `delimiter`<br>Optional; If `xPathExpression` evaluates to multiple results, the delimiter to be used to separate those results. Default is new line character (\n).<br>• `prettyPrint`<br>Optional; Specification (**true** or **false**) for whether or not XML output will be pretty printed (indented). Default is false. |

## JSON Output Data From Web Service Task

| | |
|---|---|
| **Description** | Resolves to the JSON output data of the Web Service task instance that is resolving the function, corresponding to the evaluated JsonPath expression.<br><br>• If the output record cannot be found, the function will remain unresolved. |

| | |
|---|---|
| | • If the output record is found but the path expression does not yield a result, the function will resolve to the default value. |
| **Syntax** | **${_responseJsonPath**('pathExpression'[,'defaultValue','delimiter',prettyPrint])} |
| **Parameters** | • `pathExpression`<br>Required; JsonPath expression.<br>• `defaultValue`<br>Optional; Default value to return if the result is not found. Default is empty ('').<br>• `delimiter`<br>Optional; If `pathExpression` evaluates to multiple results, the delimiter to be used to separate those results. Default is new line character (\n).<br>• `prettyPrint`<br>Optional; Specification (**true** or **false**) for whether or not JSON output will be pretty printed (indented). Default is false. |

## JSON Output Data From Sibling Web Service Task

| | |
|---|---|
| **Description** | Resolves to the JSON output data of the Web Service task instance specified by the siblingName, corresponding to the evaluated JsonPath expression.<br><br>The sibling task instance must be within the same workflow and the Execution User of the task instance that is resolving the function must have Read permission for the sibling task instance.<br><br>• If the output record cannot be found, the function will remain unresolved.<br>• If the output record is found but the path expression does not yield a result, the function will resolve to the default value. |
| **Syntax** | **${_responseJsonPathFromTask**('siblingName','pathExpression'[,'defaultValue','delimiter',prettyPrint])} |
| **Parameters** | • `siblingName`<br>Required; Name of a sibling task instance.<br>• `pathExpression`<br>Required; JsonPath expression.<br>• `defaultValue`<br>Optional; Default value to return if the result is not found. Default is empty ('').<br>• `delimiter`<br>Optional; If `pathExpression` evaluates to multiple results, the delimiter to be used to separate those results. Default is new line character (\n).<br>• `prettyPrint`<br>Optional; Specification (**true** or **false**) for whether or not JSON output will be pretty printed (indented). Default is false. |

## JSON Output Data As Array From Web Service Task

| | |
|---|---|
| **Description** | Resolves to the JSON output data of the Web Service task instance that is resolving the function, corresponding to the evaluated JsonPath expression.<br><br>• If the output record cannot be found, the function will remain unresolved.<br>• If the output record is found but the path expression does not yield a result, the function will resolve to the default value. |
| **Syntax** | **${_responseJsonPathAsArray**('pathExpression'[,'defaultValue',prettyPrint])} |
| **Parameters** | • `pathExpression`<br>Required; JsonPath expression. |

| | |
|---|---|
| | • `defaultValue`<br>Optional; Default value to return if the result is not found. Default is empty ('').<br>• `prettyPrint`<br>Optional; Specification (**true** or **false**) for whether or not JSON output will be pretty printed (indented). Default is false. |

## JSON Output Data As Array From Sibling Web Service Task

| | |
|---|---|
| **Description** | Resolves to the JSON output data of the Web Service task instance specified by the siblingName, corresponding to the evaluated JsonPath expression.<br><br>The sibling task instance must be within the same workflow and the Execution User of the task instance that is resolving the function must have Read permission for the sibling task instance.<br><br>• If the output record cannot be found, the function will remain unresolved.<br>• If the output record is found but the path expression doesn't yield a result, the function will resolve to the default value. |
| **Syntax** | **`${_responseJsonPathAsArrayFromTask`**`('siblingName','pathExpression'[,'defaultValue',prettyPrint])}` |
| **Parameters** | • `siblingName`<br>Required; Name of a sibling task instance.<br>• `pathExpression`<br>Required; JsonPath expression.<br>• `defaultValue`<br>Optional; Default value to return if the result is not found. Default is empty ('').<br>• `prettyPrint`<br>Optional; Specification (**true** or **false**) for whether or not JSON output will be pretty printed (indented). Default is false. |