CA Process Automation

Content Designer Reference

Release 04.3.00

CA Technologies Product References

This document references the following CA Technologies products:

- CA Catalyst for CA Service Desk Manager (CA Catalyst Connector for CA SDM)
- CA Client Automation (formerly CA IT Client Manager)
- CA Configuration Automation (formerly CA Cohesion® Application Configuration Manager)
- CA Configuration Management Database (CA CMDB)
- CA eHealth®
- CA Embedded Entitlements Manager (CA EEM)
- CA Infrastructure Insight (formerly Bundle: CA Spectrum IM & CA NetQoS Reporter Analyzer combined)
- CA NSM
- CA Process Automation (formerly CA IT Process Automation Manager)
- CA Service Catalog
- CA Service Desk Manager (CA SDM)
- CA Service Operations Insight (CA SOI) (formerly CA Spectrum® Service Assurance)
- CA SiteMinder®
- CA Workload Automation AE

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services

- Information about user communities and forums

- Product and documentation downloads

- CA Support policies and guidelines

- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 3: Catalyst   59

Chapter 4: Command Execution   79

Chapter 5: Databases   143

Chapter 6: Date-Time   183

Chapter 7: Directory Services   187

Chapter 8: Email 249

Chapter 9: File Management   271

Chapter 10: File Transfer   301

Chapter 11: Java Management   323

Chapter 12: Network Utilities   335

Chapter 13: Process Control   361

Chapter 14: Utilities   381

Chapter 15: Web Services   407

Chapter 16: CA Process Automation System Functions   511

Index 561

Chapter 1: Introduction to Operators

This reference contains information about the operators that are included as part of CA Process Automation. Operators are grouped into categories. This guide groups the descriptions of operator information by these categories.

The *Content Designer Reference* also describes system functions. Use the system functions to write custom JavaScripts. These JavaScripts can be placed inside operators to manipulate the data that is used inside CA Process Automation.

Operator Overview

Containers for operator categories are represented as folders in the Operators palette in the Designer.

CA Process Automation contains the following categories of operators:

Standard (see page 43)

Standard operators include essential functionality operators that control workflows in processes. Simple functionality such as starting, stopping, linking, and commenting, are provided with the Standard operators. You can also set looping and reset options, and incorporate the lane changes using these operators.

Catalyst (see page 59)

Catalyst operators support the UCF create, read, update, delete (CRUD), and event subscription interfaces. These operators expose Unified Service Model (USM) object types and properties.

Command Execution (see page 79)

Command Execution operators run processes and scripts on the host operating environment.

Databases (see page 143)

Databases operators provide an avenue to communicate and run database queries against different database servers.

Date-Time (see page 183)

Date-Time operators manage the date and time for the CA Process Automation server.

Directory Services (see page 187)

Directory Services operators support the Lightweight Directory Access Protocol. All of these operators work with different LDAP servers except for operators specific to the Active Directory.

Email (see page 249)

Email operators automate tasks that are performed on emails and folders in an email server. Email operators read emails from the mail server through IMAP/POP3.

File Management (see page 271)

File Management operators monitor directories, files, and their contents. File Management operators can be run either locally or on a remote system. These operators create, delete, rename, compress and uncompress local files, and watch files on the touchpoint where the File Management category is running.

File Transfer (see page 301)

File Transfer operators let you use FTP and SFTP.

Java Management (see page 323)

Java Management operators provide a management interface for systems that support JMX.

Network Utilities (see page 335)

Network Utilities operators allow the user to communicate to other network devices through SNMP.

Process Control (see page 361)

Process Control operators run, monitor, and control CA Process Automation processes.

Utilities (see page 381)

Utilities operators invoke external JARS in CA Process Automation.

Web Services (see page 407)

Web Services operators provide various standard network protocol utilities to the automated business processes made possible by CA Process Automation.

Where Operators Can Run

An operator runs on an Orchestrator by default, but you can specify a specific location (target) in the execution settings.

Targets can be specified as a touchpoint, touchpoint group, agent ID, proxy touchpoint, or an IP address or FQDN. See How Targets for an Operator Can Be Specified in the *Content Designer Guide* for a description of each of these items. Each target ultimately resolves to an Orchestrator, one or more agents, or a remote host with an SSH connection to an agent. Where the operator runs is significant because in some cases the process that is being designed must operate on a specific host. Many operators can execute on either an Orchestrator or on agents. However, certain operators can execute only on Orchestrators; another can execute only on agents. Others require that the targeted remote host support SSH access.

We recommend that you dedicate an agent to running Invoke Java operators. This is the only CA Process Automation operator than cannot run on an Orchestrator.

Note: Agents acting as proxy touchpoints have different category requirements. Operations that target proxy touchpoints (or a remote host referenced by a host group) go through SSH.

Category	Operator	Orchestrator	Agent	Remote Host (SSH Target)
Catalyst (see page 59)	All	x		
Command Execution (see page 79)				
	Run Program	x	x	x
	Run SSH Command	x	x	
	Run SSH Script	x	x	
	Run Script	x	x	x
	Run Telnet Command	x	x	
	Run Telnet Script	x	x	
Databases (see page 143)				
	Bulk Insert into Database	x	x	x
	Delete from Database	x	x	x

Category	Operator	Orchestrator	Agent	Remote Host (SSH Target)
	Get Database Schema	x	x	x
	Get Free Space	x	x	x
	Get Stored Procedure	x	x	x
	Get Table	x	x	x
	Get Used Space	x	x	x
	Get Version	x	x	x
	Get View	x	x	x
	Insert into Database	x	x	x
	Query Database	x	x	x
	Select from Database	x	x	x
	Update in Database	x	x	x
Date-Time (see page 183)				
	Check Calendar	x		
	Check Date-Time	x		
Directory Services (see page 187)				
	Add Computer to Domain	x	x	
	Add User to Group	x	x	
	Create Group	x	x	
	Create Object	x	x	
	Create Organizational Unit	x	x	
	Create User	x	x	
	Delete Object	x	x	
	Get Domain Controller	x	x	
	Get Dormant Account	x	x	
	Get Object	x	x	

Category	Operator	Orchestrator	Agent	Remote Host (SSH Target)
	Get User	x	x	
	Move Object	x	x	
	Remove User from Group	x	x	
	Update Object Attributes	x	x	
	Update User Home Directory	x	x	
Email (see page 249)				
	Create Folder	x	x	
	Delete Email	x	x	
	Delete Folder	x	x	
	Get Email Content	x	x	
	Get Email Count	x	x	
	Get Email Envelope	x	x	
	Get Email List	x	x	
	Move Email	x	x	
	Purge Folder	x	x	
	Rename Folder	x	x	
	Send Email	x	x	
File Management (see page 271)				
	Compress File	x	x	x
	Copy File	x	x	x
	Create Folder	x	x	x
	Decompress File	x	x	x
	Delete File	x	x	x
	Get Directory Content	x	x	x
	Get File Attributes	x	x	x
	Monitor File	x	x	x

Category	Operator	Orchestrator	Agent	Remote Host (SSH Target)
	Read from File	x	x	x
	Rename File	x	x	x
	Search File Content	x	x	x
	Update File Ownership	x	x	x
	Update File Permission	x	x	x
	Update File Timestamp	x	x	x
	Write File	x	x	x
File Transfer (see page 301)				
	Create Directory	x	x	x
	Delete Directory	x	x	x
	Delete File	x	x	x
	Download File	x	x	x
	Get File Information	x	x	x
	Move File	x	x	x
	TFTP Download File	x	x	
	TFTP Upload File	x	x	
	Upload File	x	x	x
Java Management (see page 323)				
	Get MBean Attributes	x	x	
	Invoke MBean Method	x	x	
	Update MBean Attributes	x	x	
Network Utilities (see page 335)				

Category	Operator	Orchestrator	Agent	Remote Host (SSH Target)
	Get Local Network Interfaces	x	x	
	Get Network Services Status	x	x	
	Get SNMP Variable	x	x	
	Monitor SNMP Variable	x	x	
	Ping Host	x	x	
	Send SNMP Trap	x	x	
	Update SNMP Variable	x	x	
Process Control (see page 361)				
	Assign User Task	x		
	Evaluate Expression	x		
	Manage Resources	x		
	Monitor Event	x		
	Send Event	x		
	Start Process	x		
Standard (see page 43)				
	Start	x		
	Comment	x		
	Stop Success	x		
	Stop Failure	x		
	Process Progress	x		
	And	x		
	Or	x		
	Reset	x		
	Loop	x		
	Change Lane	x		

Category	Operator	Orchestrator	Agent	Remote Host (SSH Target)
	Exception	x		
Utilities (see page 381)				
	Apply Xpath	x		
	Apply XSLT	x		
	Delay	x		
	Invoke Java		x	
	Run JavaScript	x		
Web Services (see page 407)				
	HTTP Delete	x	x	
	HTTP Get	x	x	
	HTTP Head	x	x	
	HTTP Options	x	x	
	HTTP Post	x	x	
	HTTP Post Form	x	x	
	HTTP Put	x	x	
	HTTP Trace	x	x	
	Invoke SOAP Method	x	x	
	Invoke SOAP Method Async	x	x	

Common Properties of all Operators

Operator configuration options display in the Properties window when you double-click an operator in a process or schedule object.

Configuration for operator categories is described in the *Content Administrator Guide*.

Note: Verify that the Properties check box is selected if the properties for an operator do not display. Select View, Properties in the top right-hand corner of the Designer, then maximize the Properties window at the bottom of the screen. If the operators themselves do not display, click Operators under the View menu.

Execution Settings (All Operators)

Target

Specifies the target on which to run the operator. A target can be a touchpoint, touchpoint group, Agent ID, proxy touchpoint, IP address, or FQDN. Be sure that you enable the category for the operator on the touchpoint. To open the Object Browser dialog and select a touchpoint, click Select.

Target is a calculated expression

Specifies the target using an expression. To specify a target dynamically at runtime, use a calculated expression. Consider the following examples:

■ Use a string dataset variable containing the name of the touchpoint.

■ Use an Object Reference dataset variable that points to the touchpoint.

Important! When a process is destined for an import as a content package, specify the IP address or FQDN in a dataset. A dataset can be modified in the import environment, but the Target field cannot.

Match target in Host Groups only

Specifies how to resolve the target name.

Selected

Specifies that the Target field contains a host name or IP address that a host group references.

Note: If the Target field contains touchpoint, a proxy touchpoint, or an agent ID, the execution fails.

Cleared

Specifies that the Target field contains the name of a touchpoint, a proxy touchpoint, or an agent ID.

Note: If the Target field contains a valid host group reference, the processing succeeds but it is not optimized.

Loop

CA Process Automation lets you loop an operator until some condition is met. The Loop property specifies the number of times that an operator repeats.

The exit conditions and the connecting links from the operator that is running in a loop are evaluated only when the loop completes.

Note: For more detailed information about using loops in CA Process Automation, see the *Content Designer Guide*.

Repeat Count

Specifies the condition for looping. Two options are available:

- Specify an integer or a CA Process Automation expression that returns an integer at run time. The default value of 1 executes a loop on an operator a single time. To execute an infinite loop, click the Infinite loop check box.

- Specify a Boolean expression. The expression is evaluated after the operator executes. If the expression evaluates to true, the loop continues. If the expression evaluates to false, the loop completes.

Infinite Loop

When selected, Repeat Count is ignored and an infinite loop is created. The operator keeps repeating until either:

- The process is interrupted.

- The loop is stopped from a different branch of the process (by processing a stop loop command link to the Loop Operator (see page 51)).

Delay between iterations

Specifies the delay in seconds between each loop iteration (the default value is 0).

Timeout

Lets you set a timeout as part of every operator. If the operator does not finish by the specified time, this setting provides an exit strategy. The Timeout option provides the choice to either:

- End the operator and take the alternate timeout path.

- Let the operator continue while taking the timeout path.

- Reset the operator (run the operator again).

No Timeout

Specifies that the operator has no timeout. This check box is selected by default.

Type

Select either Duration or Target Date.

Duration/Target Date and Time

■ If you select Duration:

Enter a timeout duration in seconds.

The proper format for this field is anything that can be treated as an integer/long, or string literals (for example,"10").

The timeout is specific to each iteration of a loop. Therefore, a timeout occurs only if a single iteration takes longer than the timeout duration.

■ If you select Target Date:

Enter a date and time when you want the operator to time out.

When a string literal is entered in this field, it can be in one of the following formats:

■ JVM Format - The date/time format that the Java application understands. This format varies with the Java installation.

■ If the JVM format is unknown, enter as "MM/dd/yyyy HH:mm:ss".

■ Any other format must use a CA Process Automation library method named "parseDate". This method takes in two parameters: (a) Date as a string literal and (b) Format in which the string must be parsed. For example, parseDate("10/10/2010 10:10:10"," dd/MM/yyyy HH:mm:ss") returns this date: 10th of October 2010 10 hrs 10 mins and 10 secs.

If Target is selected as the Type, the Reset option (under Action) is disabled.

Note: Be sure to enter this data in the proper format or else the operator ignores these timeout settings.

Action

Notes:

- This option is not available for schedules.

- When an operator is in a loop and a timeout is configured, then the following options behave differently. See Loop and Timeout Scenarios (see page 31) for the actions that are taken.

Select from one of the following actions:

Continue

If selected, the process proceeds in the following manner after a timeout:

1. The operator remains in running mode only.

2. The timeout path is taken.

3. The post-execution code only runs when the operator completes, not when the timeout path was taken.

Reset

If selected, the process proceeds in the following manner after a timeout:

1. The operator is reset (that is, the operator starts executing again).

2. The timeout path is taken.

3. The post-execution code executes only if the operator completes (not when the operator was reset).

Note: If Reset is selected, then the Target Date option is disabled in the Type drop-down list.

Abort

If selected, the process proceeds in the following manner after a timeout:

1. The operator aborts.

2. The post-execution code executes.

3. The timeout path is taken.

Abandon

If selected, the process proceeds in the following manner after a timeout:

1. The operator times out.

2. The process continues to run in detached mode.

Note: An instance of a process started in detached mode has no parent relationship to the process that started it. A detached process instance is the root process in any call sequence originating from that process.

3. The post-execution code executes immediately.

4. The timeout path is taken.

Loop and Timeout Scenarios

If an operator is in a loop and the timeout is configured, then the following scenarios take place for the selected actions:

Action

Select from one of the following actions:

Continue

If selected, the process proceeds in the following manner after a timeout:

1. The next iteration executes.

2. The post-execution code only runs if the iteration is complete.

Using the OverallLoopDuration dataset variable to continue looping an operator that times out:

At the end of execution, the OverallLoopDuration contains the number of seconds from the start of the first iteration until the end of the last iteration. If the operator times out, the OverallLoopDuration does not contain the number of seconds from the start of the first iteration until the time the operator times out.

Reset

If selected, the process proceeds in the following manner after a timeout:

1. The iteration resets (that is, the particular iteration starts executing again).

2. The post-execution code executes only when the current iteration completes (not when the iteration was reset).

3. The next iteration executes only when the iteration completes.

Note: If Reset is selected, then the Target Date option is disabled in the Type drop-down list.

Using the OverallLoopDuration dataset variable to reset a looping operator that times out:

If you set an operator to loop with a timeout action of Reset, CA Process Automation checks the loop condition when moving from one iteration to another. The loop condition is *not* checked when resetting an iteration. Also, the OverallLoopDuration contains the number of seconds from the start of the first iteration, including the time spent in all the reset iterations. Iteration resets do not affect the OverallLoopDuration.

Abort

If selected, the flow proceeds in the following manner after a timeout:

1. The iteration aborts.

2. The post-execution code executes.

3. The next iteration executes.

Abandon

If selected, the process proceeds in the following manner after a timeout:

1. The iteration continues to run in detached mode.

Note: An instance of a process started in detached mode has no parent relationship to the process that started it and is the root process in any call sequence originating from that process.

2. The post-execution code executes.

3. The next iteration executes.

Common Properties of Operators in Processes

All operators have properties that configure their appearance and behavior when added to a process.

A process does not have a limit on the number of operators it can include. However, CA Technologies recommends that a process contains approximately 40-50 operators for maximum performance. If a process starts to grow larger than 40-50 operators, consider splitting the process into smaller components.

The properties described here are displayed in the Execution Settings, Simulation, and Information properties only for an operator in a process.

Note: Operator-specific properties override the properties that are defined at the category level.

Processing Group

The following properties define conditions to meet before you run an operator, and actions to perform before and after the operator runs. You can find these properties in the operator Execution Settings.

Pre-execution Code

Lets you add code that runs before an operator runs. You can run any JavaScript code. JavaScript code runs before the operator runs. Pre-execution code manipulates the operator and process dataset in such a way that the dataset can be used as input for the operator. In other words, you can manipulate the output parameters from a previous operator dataset and then use them as input for a later operator.

Pre-execution code can perform various tasks. For example, the following code sets a Process-level variable:

```
if(Process.username==null)
{
Process.username="testuser";
}
```

Note: For more information about adding code, see the [Run JavaScript operator](#) (see page 406).

Before the pre-execution code finishes and the operator runs, the operator has to reach the code CanExecute = 1. The process variable CanExecute is added by default to the operator. The default CanExecute value is 1. If you do not change the default, the operator runs. This requirement lets you verify external conditions and proceed only when an expected condition is met.

If you change the CanExecute value to 0 (Process.CanExecute =0), the operator does not run. CA Process Automation waits 30 seconds, then reruns the pre-execution code.

When there is no pre-execution code, the operator runs immediately. For example, you could use pre-execution code to set up loop variables or other variables to use as part of the operator.

For the code that runs in the operator, you can use the following syntax to access the operator dataset:

```
Process[OpName].fieldname
```

For example, the following code creates an operator dataset variable named iNow that contains the following data:

- The name of the host
- The current date
- The current time in a single string

```
Process[OpName].iNow = System.Host + ":" + System.Date + ":" + System.Time;
```

Post-execution Code

Lets you add code that runs after an operator completes. For example, you could use post-execution code to modify loop variables or to process the results of an operator.

For the code that runs in the operator, you can use the following syntax to access the operator dataset:

```
Process[OpName].fieldname
```

For example, the following code copies the value of the operator dataset variable named Result to variable named iResult:

```
Process.iResult = Process[OpName].Result;
```

Run as Caller User

Specifies that the selected operator in a process must run under the identity of the authorized user who started the process. This requirement is true whether the entire process is running as the owner or not. Run as Caller User lets process designers run processes that:

■   Deliver a self-contained automation object (run as the owner)

■   Require control of access rights to parts of the process (such as child processes and touchpoints)

Simulation Properties

The Simulation properties let you configure how to simulate execution of an operator in a process. Simulation can be used for testing branches of a process or to allow normal processing to skip an operator without having to reroute the process.

You configure Simulation properties for a new process. For each operator in a process, you can specify to inherit the Mode setting configured at the process level or specify a different Mode setting. Double-click an operator to display the operator properties pane. Then expand the Simulation tab to display the settings to configure.

Mode

Simulation modes are available when you select the Override simulation option in Process check box.

Inherit from Process

Specifies to use the setting (Off, Local, or Distant) configured for the parent process of the operator.

Off

Turns off simulation and enables normal processing of the operator. The End Condition is set to Completed, Delay is set to 0, and Evaluate Pre-execution and Post-execution Code is cleared.

Local

Disables the operator so that it is not processed. CA Process Automation does not call the associated operator or monitor the operator parameters. Parameter checks include looking for an application program or validating the execution touchpoint for an operator.

Distant

Causes the engine to call the associated operator. The operator examines the parameters before returning the result but does not actually run the operator. If the parameters are incorrect, the simulated operator fails regardless of the specified outcome. If the parameters are correct, the operator returns the specified result.

Delay

Specifies the number of seconds to delay a process to simulate the time that the operator uses during normal processing.

End Condition

Specifies the exit condition for the simulated operator. You can use this option to test or troubleshoot different branches in a process. The actual conditions depend on the operator.

■ The following end conditions apply when Mode is Off: Completed.

■ The following end conditions apply when Mode is Local: Completed, Failed, Timeout.

■ The following conditions apply when Mode is Distant: Completed, Failed, Timeout, Custom Result.

Completed

Causes the standard successful outcome exit link from the operator to process. The Result variable in the operator dataset is set to 1. Any positive integer value activates a standard successful exit link.

Failed

Causes the standard failed outcome exit link from the operator to process. The Result variable in the operator dataset is set to 0. Zero or any negative integer value activates the standard failed link.

Timeout

Causes the operator to take the timeout path when the given time is elapsed. The Result variable in the operator dataset displays as timeout.

Custom Result

Specifies the integer value that the Result variable in the operator dataset returns. You can set this parameter to any value (positive or negative) to activate a custom port that tests for a particular value.

This option is available only when Custom Result is selected for End condition.

Evaluate Pre-execution and Post-execution Code

Indicates whether to evaluate the pre-execution and post-execution code during operator simulation.

Selected

Specifies to evaluate the code and prevent side effects of ignoring the code.

Cleared

Specifies to not evaluate the code.

Information Properties

The Information properties determine the name of the operator and visual representation of the operator and its comments.

Override Object Preferences

Select this check box to override the default process settings and configure settings specific to that operator.

Icon

Name

Specifies the name of an operator. The Name property is especially important when configuring an operator that other operators reference. The Name property is used with the following syntax in an expression to access variables in the operator dataset of another operator in a process:

`Process.Operator_name.variable_name`

Operator names must be unique within the same process.

Use Default Icon

If checked, the operator uses the default icon. Clear this check box to use a custom icon object for the operator instead of the default.

Browse

Click to select the custom icon object that you want to use for this instance of the operator.

Label Display

Show Labels

Displays any icon information for the operator that you enter in the Name field. Select one of the following options:

Truncated

Displays only a partial amount of the Name field next to the operator icon.

Long

Displays the entire length of the Name field next to the operator icon.

Off

No label displays.

Label Source

Object Name

Select to display the name of the operator as its label.

Comments

Select to display the operator comments (that you enter in the Comments text box) as its label.

Label Colors

Click Choose Color to select a color for the following parts of the label:

- Text Color

- Background Color

- Border Color

Font

Configure the font properties of the operator icon label: Font Family, Font Style, and Font Size.

Note: Select fonts that are generally available on computers hosting CA Process Automation.

Preview

View your Font selections for the operator icon label before applying them.

Comments

Enter the comments that you want to display for the operator. If the value of Label Source is Comments, this text displays in the label next to the operator.

Common Properties of Operators in Schedules

Operators in a schedule are started according to specified calendar and time conditions. Properties groups for any operator added to a schedule associate calendar rules and other time conditions with the operator. Click the Properties icon in the schedule editor to view operator properties. General scheduling properties display on the General tab. Operator-specific properties display on the Specific tab.

Specify the time to start the operator in the first drop-down menu in the Start Time drop-down list. Specify repeating intervals through the Repeat Interval (minutes) check box. Select this check box to execute the operator at fixed intervals. Select a time to stop repeating the operator in the End Time drop-down.

Note: See the *Content Designer Guide* for more information about schedules.

Calendar Settings

The Calendar Settings properties let you include dates from a predefined calendar in a schedule.

Include Calendar

Click the calendar icon to select a predefined calendar to include in your schedule.

Exclude Calendar

Click the calendar icon to select a predefined calendar to exclude in your schedule.

Days per Shift

The number of days to shift a scheduled date when the scheduled date falls on a closed date. The shift can be negative or zero. When this value is negative the date shifts forward. When this value is zero, closed dates are simply skipped without rescheduling the task.

No excluded days

Select this check box to only count open days when shifting a scheduled date to avoid a closed date.

Maximum Shift

When an original scheduled date falls on a closed day and the task is rescheduled, the new date could also fall on a closed date. This parameter defines the maximum number of shifts that are allowed.

Only manually selected

When a calendar is not specified in a schedule, CA Process Automation considers the item scheduled every day. The exception is when you select this option. When this option is selected, schedule the run dates (on the Manually Included Dates tab).

Manually Included Dates

The Manually Included Dates properties let you manually add dates to a schedule. These properties display a list of dates that are manually scheduled. They also display a list of dates that the calendar rules specify. Dates added here override closed days that the Manually Excluded Dates properties specify.

When a schedule does not specify a calendar, the Manually Included Dates properties can schedule dates; select the Only manually selected check box in the Calendar Settings properties.

Click the appropriate option buttons to add, remove, or rearrange list items.

Add Item

Adds a date to include in the list. To set the date, click the entry, then click the calendar icon to select the dates to include.

Delete Item

Removes a selected date from the list.

Move Up

Moves up a selected date in the list.

Move Down

Moves a selected date down in the list.

Manually Excluded Dates

The Manually Excluded Dates properties let you manually remove dates from a schedule. The Manually Excluded Dates properties list dates that you do not schedule under any circumstances, regardless of all other rules or conditions.

Click the appropriate option button to add, remove, or rearrange list items.

Add Item

Adds dates to exclude to the list. To set the date, click the entry, then click the calendar icon to select the dates to include.

Delete Item

Removes a selected date from the list.

Move Up

Moves up a selected date in the list.

Move Down

Moves down a selected date in the list.

Task Name

The Task Name specifies the name of the user defined task. When you add any operator or process to the schedule, you can specify the custom name in this field. The customized task name applies to corresponding runtime task instances.

Common Output Parameters for all Operators

All operators contain the following output properties. Any further output parameters are specified for each operator.

StartTime

The time the operator began in the process or schedule.

StartDate

The date the operator began in the process or schedule.

Reason

Specifies the reason if the operator fails after execution.

Result

Specifies the result of the operator execution.

Chapter 2: Standard Operators

Use the Standard operators to control workflows in processes.

Start Operator

Use the Start operator to start a workflow in a process. The Start operator is automatically included in a process by default.

You can add more than one Start operator to a process. Each Start operator in a process starts its own workflow when an Orchestrator starts the process.

You can also add a Start operator to terminate a cyclical sequence of operators. A Start operator that terminates a workflow reinitializes the operators in the workflow. The operator then loops processing back to the initial Start icon for the sequence of operators that were executed between the Start operators.

Input Parameters

Double-click the Start operator to configure its name and appearance using the Information (see page 37) properties.

Comment Operator

The Comment operator adds comments to the process. Comments are important for documenting steps in a process and allow more space than labels.

Follow these steps::

1. Drag the Comment operator from the Standard folder to a location on one of the editor tabs (main, exception handler, lane change handler).

2. Double-click the comment text to display the comment properties.

Input Parameters

Background

Border Color

Click the Choose Color drop-down to select the color for the comment border.

Background Color

Select the color and transparency of the background for the comment text.

- Transparent for a transparent background.

- Opaque for a colored background. When you select this option, you can also change the Color setting for the background.

Comments

Lets you enter or change the text of a comment.

- Configure the font properties: font, font style, size, effects (bold, italic, and underline), color, and highlighting.

  Select fonts that are likely to be installed on computers that host CA Process Automation.

- Configure the alignment: left, right, or center horizontal alignment of the comment text. Bulleted and numbered lists are also available.

- Click Hyperlink to turn selected text into a hyperlink.

- Click Source Edit to switch to source editing mode.

Stop Success Operator

The Stop Success operator terminates a process and determines it a success. A Stop Success operator can terminate a process:

- At the end of a sequence of operators on the Main Editor tab.

- In an exception on the Exception Handler tab.

- In a lane change on the Lane Change Handler tab.

A Stop Success operator can be configured as either a Stop Success or a Stop Failure operator (through its properties). When a process is run, the Stop Success operator sets the Result variable for an operator dataset to 1 by default. You can override the positive default to a negative one to change the Stop Success operator to a failure.

Input Parameters

Result

Specifies a result parameter. The result parameter is an integer expression which is used to determine whether the flow ended correctly (positive value) or incorrectly (zero or negative).

The default when you select a Stop Success end type is 1. The default when you select Stop Failure is -1. You can also enter some other integer value or enter an expression that returns a calculated value for the result code at run time.

The Result value for the Stop Success operator that terminates an instance of a process is saved to the Result variable in the process dataset of the instance.

End Type

Select one of the following options:

Stop Success

Processes a normal end for a workflow. This option sets the Result code to 1. If you change the Result value, use a positive integer to be consistent with a normal finish.

Stop Failure

Processes an abnormal end for a flow. This option sets the Result value to -1. If you change the Result value, enter a negative integer to be consistent with an abnormal finish.

Break Calling Loop

When the flow is invoked from another process, select this check box to break a calling loop. Clearing this check box allows a calling loop to continue. This check box only applies if the flow was called from within a loop in *another* process.

Ignore Running Tasks (Immediate Stop)

Ends a flow immediately without waiting for other operators to finish processing. Clear this check box to wait for any operators still processing to finish before ending the flow.

Stop Failure Operator

The Stop Failure operator terminates a process and determines it a failure. A Stop Failure operator can terminate a process:

- At the end of a sequence of operators on the Main Editor tab.

- In an exception on the Exception Handler tab.

- In a lane change on the Lane Change Handler tab.

A Stop Failure operator can be configured as either a Stop Success or a Stop Failure operator (through its properties). When a process is run, the Stop Failure operator sets the Result variable for an operator dataset to -1 by default. You can override the negative default to a positive one to change the Stop Failure operator to a success.

Input Parameters

Result

Specifies a result parameter. The result parameter is an integer expression which is used to determine whether the flow ended correctly (positive value) or incorrectly (zero or negative).

The default when you select a Stop Failure end type is -1. The default when you select Stop Success is 1. You can also enter some other integer value or enter an expression that returns a calculated value for the result code at run time.

The Result value for the Stop Failure operator that terminates an instance of a process is saved to the Result variable in the process dataset of the instance.

End Type

Select one of the following options:

Stop Success

Processes a normal end for a workflow. This option sets the Result code to 1. If you change the Result value, use a positive integer to be consistent with a normal finish.

Stop Failure

Processes an abnormal end for a flow. This option sets the Result value to -1. If you change the Result value, enter a negative integer to be consistent with an abnormal finish.

Break Calling Loop

When the flow is invoked from another process, select this check box to break a calling loop. Clearing this check box allows a calling loop to continue. This check box only applies if the flow was called from within a loop in *another* process.

Ignore Running Tasks (Immediate Stop)

Ends a flow immediately without waiting for other operators to finish processing. Clear this check box to wait for any operators still processing to finish before ending the flow.

Process Progress Operator

The Process Progress operator lets you set the progress at different stages of a process. Privileged users can monitor the progress of the process from any one of the following ways:

- Process Dataset

- User-defined Reports

- Process Instances table in the Operations tab

Note: For more information about the privileged users, see the *Permissions by Tab* section in the *Content Administrator Guide*.

Input Parameters

Process Progress

Specifies the completion progress of a process as a percentage in the range 0 through 100. If the execution of a process reaches a Stop Success operator, the process progress is 100. If a process fails after reaching a progress of 50, the Process Dataset displays 50.

Output

Progress

Displays the percentage completion of a process under Progress (variable name) in the Process Dataset.

And Operator

The And operator defines a synchronization point between all entry links to it. Exit links from an And operator are activated only after all its entry links are activated. Use the And operator to synchronize multiple branches of a process with a logical And condition when two or more separate branches of a flow must all be completed before beginning one or more additional branches.

You can include an And operator in a process in various ways.

Follow these steps::

1. Drag the And operator from the Standard folder to one of the editor tabs (Main, Exception Handler, Lane Change handler).

2. Link one or more input operators that the And operator can synchronize.

3. Link one or more output operators to follow completion of the And operator.

Parameters

The And operator does not contain any parameters.

Or Operator

The Or operator defines a synchronization point between all entry links to it. Exit links from an Or operator are activated after at least one entry link to the operator has been activated. At least one of two or more separate branches of a flow leading to an Or operator be completed before beginning one or more exit branches.

The Or operator can be added to sequences of operators in a process on one of the editor tabs (Main, Exception Handler, Lane Change Handler). Place an Or operator in a process by dragging it from the Standard folder to any of those three editors. Link one or more input operators that the Or operator can synchronize and link one or more output operators to follow completion of the Or operator.

Parameters

The Or operator does not contain any parameters.

Reset Operator

Use the Reset operator to reset selected operators (typically an operator that caused an exception) in a suspended process to their initial states. These reset operators act as if they had not been executed and the process continues.

This operator also lets a user ignore an exception and continue with a process anyway. The Reset operator lets a user set an operator in simulate mode and continue the process with that operator simulated.

Input Parameters

Operators List

Click Add to add an operator to reset. A drop-down menu lets you select one of the available operator names in the current process. Multiple operators can be added, which can then be added, deleted, or sorted.

You can also enter an expression (instead of choosing an operator from the drop-down menu) which resolves to a String (an operator name) or ValueArray (a list of operator names) at runtime.

Continue with Result

Selected

The End Condition drop-down menu and the Evaluate pre-execution and post-execution code check box become available.

Cleared

When an error-condition is met at runtime, CA Process Automation resets the selected operators. CA Process Automation then continues with the process flow

End Condition

When you select the Continue with Result check box, the End Condition drop-down menu becomes available with the following options:

Successful

CA Process Automation assumes that the selected operators are successful if no error condition is met at runtime. CA Process Automation then continues with the rest of the process flow.

Unsuccessful

CA Process Automation assumes that the selected operators have failed when an error condition is met at runtime. CA Process Automation then continues with the rest of the process flow.

Evaluate Pre-execution and Post-execution Code

Select this check box to evaluate pre-execution and post-execution code.

How the Reset Operator Works with the Loop Operator

The Reset operator works with the Loop operator as follows.

1. The Reset operator allows the Loop operator to be reset. The Reset operator resets the Loop operator as follows:

   ■ Resets all operators inside the Loop operator.

   ■ Resets the Loop operator to the first iteration.

2. After the reset, the Loop operator restarts from the first iteration.

Note: The Loop operator does not support simulation. The Reset operator always resets a Loop operator regardless of the values of the fields:

■ Continue with Result

■ End Condition

■ Evaluate pre-execution and post-execution code

Loop Operator

The Loop operator loops an enclosed sequence of operators in a process either a specified number of times or indefinitely. You can place it in a process and can resize the box to accommodate any number of operators in the sequence.
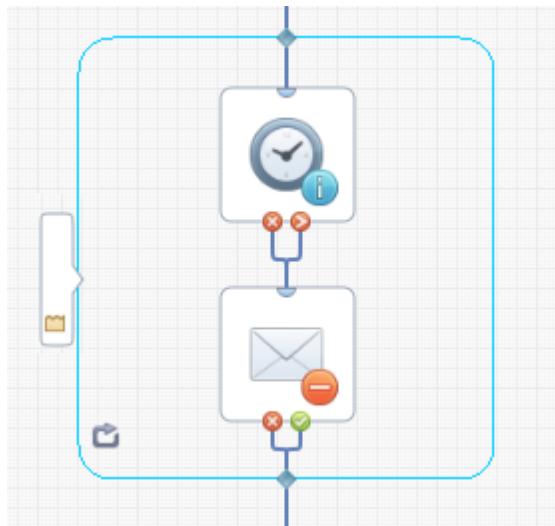
The Loop operator can enclose a sequence of operators in a process on the Main Editor pane, the Exception Handler pane, or the Lane Change Handler pane of the Designer tab.

To place a Loop operator in a process

1. Drag the Loop operator from the Standard folder to an editor.

2. Drag one or more input links to the input portal, and one or more output links from its output portal.

3. Add looped operators inside the Loop box.

4. Link the input portal to the first operator in the looped sequence.

5. Link the last operator in the sequence to the output portal.

Input Parameters

Repeat Count

Specifies the number of times that an operator repeats. The following two options are available:

■ This value can be specified with an integer or a CA Process Automation expression that returns an integer at run time. The default value of 1 executes a loop on an operator a single time in a workflow. To execute an infinite loop, click the Infinite Loop check box.

■ A Boolean expression can also be used. As long as the expression evaluates to true, an operator in a workflow executes a continual loop. If the expression is false, the operator exits.

This value can also be specified using the loop variables in the dataset of the Loop operator:

− CurrentLoopIteration: A loop counter that starts at 0 during the first iteration of the loop and increments by 1 for each additional iteration. This variable is updated at the beginning and end of every iteration.

If the operator is configured to loop three times, at the end of execution of all iterations, CurrentLoopIteration is equal to:

- 0 in iteration 1

- 1 in iteration 2

- 2 in iteration 3

- 3 in the last iteration, which is not executed as it violates the loop condition.

– OverallLoopDuration: A loop counter that specifies the amount of time (in seconds) that has passed since the start of the first iteration of the loop. This variable is updated at the beginning and end of every iteration and includes any delay that is set between iterations of the loop.

Set the Repeat Count to:

Process[OpName].CurrentLoopIteration < x

where

x is the number of times to run the operator.

Or, set Repeat Count to:

Process[OpName].OverallLoopDuration < x

where

x is the number of seconds to loop the operator. The operator does not stop at the number of seconds specified when it is in the middle of an iteration. Instead, if OverallLoopDuration is greater than the number of seconds specified, the operator does not execute the next iteration.

CA Process Automation checks the loop condition between iterations.

Infinite Loop

When selected, Repeat Count is ignored and an infinite loop is created. The operator keeps repeating until either:

■ The process is interrupted.

■ The loop is stopped from a different branch of the process (by processing a stop loop command link to the Loop Operator).

Delay between iterations

Specifies the delay in seconds between each loop iteration.

While loop

When selected, the Loop operator behaves as a *while* loop. If unselected, the Loop operator behaves as a *do while* loop.

While loop

The Loop operator checks the loop condition specified in the Repeat Count field before it executes any iteration, including the first iteration.

Do while loop

The Loop operator checks the loop condition specified in the Repeat Count field at the end of every iteration, so it is guaranteed to execute at least the first iteration of the loop.

Note: All existing loop operators that are imported from CA Process Automation before v4 have the While Loop field unchecked. These existing operators continue to work as Do while loops, as they did in previous versions.

Pre- and Post execution Code

Use these fields to execute JavaScript code to execute with each iteration of the loop.

The processing sequence of any Pre and Post condition depends on the type of loop. See the *Content Designer Guide* to learn more about the logical sequence of a loop.

Output Parameters

CurrentLoopIteration

A loop counter that starts at 0 during the first iteration of the loop and increments by 1 for each additional iteration. This variable is updated at the beginning and end of every iteration. If the Loop operator is configured to loop three times, at the end of execution of all iterations, CurrentLoopIteration is equal to:

- 0 in iteration 1

- 1 in iteration 2

- 2 in iteration 3

- 3 in the last iteration, which is not executed as it violates the condition of the Loop operator.

OverallLoopDuration

A loop counter that specifies the amount of time (in seconds) that has passed since the start of the first iteration of the loop. This variable is updated at the beginning and end of every iteration and includes any delay that is set between iterations of the loop.

Reset the Loop Operator Manually in a Process

See the *Content Designer Guide* for details on how to reset the Loop operator manually in a process.

Change Lane Operator

The Change Lane operator initiates a series of lane changing rules in the Lane Change Handler pane of the Designer.

To place the Change Lane operator in a process

Drag the Change Lane operator from the Standard folder onto one of the editors.

Input Parameters

Name

This option displays the name of the lane change. You can change the name by editing the Name property under the Information properties group for the lane change.

Source

Specifies the source lane for the lane change. Select All for a lane change from any lane.

Destination

Specifies the destination lane for the lane change. Select All for a lane change to any lane.

Exception Operator

Use the Exception operator to initiate an exception, such as a termination due to system errors or unidentified exit conditions. To place the Exception operator in a process, drag it from the Standard folder onto the Exception Handler editor.

Input Parameters

Name

Displays the name of the exception. To change the name, edit the Name property under the Information properties group for the exception.

Exception type

Select System Error, Unidentified Response, Aborted, or Timeout from the drop-down list to categorize the exception.

Links

Links define the structure of a process by creating sequences of operators.

To create a link

Click an exit link on an operator and drag it to the subsequent operator in the sequence.

If the link you want does not display, right-click the operator and then click the link type (such as Failed, Completed, or Custom) on the shortcut menu.

Link Properties

Link properties display when you right-click a link in a process, then select Link Properties.

Weight

Specifies the thickness of lines between operators.

Color

Opens the Choose Link Color dialog, in which you can change the color of links in the process.

Shapes

Specifies the line shape for links between operators:

Straight

Creates straight links between operators.

Orthogonal

Creates right-angled links between operators.

Dashed

Click this check box to create a dashed (dotted) link.

Chapter 3: Catalyst

The Catalyst operators include create, read, update, delete, and event subscription operators that can be invoked on any Catalyst connector. All operator parameters can contain expressions for maximum flexibility in building content. CA Process Automation processes can be created using any combination of these operators to construct integrations across multiple products. In addition, the Catalyst operators could be used as base operators to build custom operators for product-specific solutions.

All Catalyst connectors expose objects that comply with the Unified Service Model (USM model). This common model facilitates cross product integrations.

The Catalyst operators contain are generic to any USM type. These operators also contains operators that are specific to each USM type. See the Connector guide that is provided with the applicable Catalyst connector for more information.

Catalyst nodes contain a broker, which is a directory of connectors. In design mode, the Catalyst operators query the broker and display the connector names in the MdrProdInstance list.

Generic USM Operators

The following operators are the more commonly used operators that apply to all USM types.

Create Operator

The Create operator supports the CRUD create and update operations on any USM type.

Input Parameters

UCFBrokerURL

Defines the UCF Broker URL of your Catalyst server. Defaults to the UCF Broker URL in the Catalyst configuration.

Specify the Broker URL of the Catalyst Broker Service as:

`http://<hostname>:7000/ucf/BrokerService`

When using secure Catalyst communications, specify the secure broker URL as:

`https://<hostname>:7443/ucf/BrokerService`

MdrProduct

Unique identifier of the connecting product.

MdrProdInstance

Unique identifier of the instance of the connecting product as registered in the UCF Broker. CA Process Automation queries the UCF Broker for the list of connectors available and populate this field. You can then select your connector from the drop-down list.

Create

Indicates whether the operator:

■ Creates an object

Or

■ Updates an existing object.

Itemtype

Specifies the USM type of the object that is created or updated.

Values:

Alert, ComputerSystem, Router, Service, and so on.

For example:

```
itemtype=ComputerSystem
```

Properties

The operator parameters contain the properties of the USM type.

Customizing the Properties

The Properties form can be customized using the "Product property configuration file name". If the MdrProduct and itemtype values match an entry in the "Product property configuration file name", then the form displays according to the following rules:

■ If the property is not defined in the USM type, then it is a custom property and that property is added to the form.

■ If the property is defined in the USM type, then it is added to the form.

■ If the property is defined in the USM type and it has an alias name, then it is added to the form using the alias name.

For example, this entry displays the Alert form as shown.

```
<!-- SCOM  -->
<MdrTypes MdrProduct="CA:00031">
    <TypeMap name="Alert">
      <Mapping propName="MdrProdInstance" aliasName="siloHost" />
      <Mapping propName="MdrElementId" aliasName="Id" />
      <Mapping propName="UrlParams" aliasName="" />
      <Mapping propName="SeverityTrend" aliasName="" />
      <Mapping propName="RelatedAlerts" aliasName="" />
      <Mapping propName="AlertedMdrProdInstance" aliasName="siloHost" />
      <Mapping propName="AlertedMdrElementID" aliasName="MonitoringObjectId"
/>
      <Mapping propName="Summary" aliasName="Name" />
      <Mapping propName="Message" aliasName="Description" />
      <Mapping propName="Assignee" aliasName="Owner" />
    </TypeMap>

</MdrTypes>
```

The tooltip of properties with alias names indicates the USM property name.

Delete Operator

The Delete operator supports the CRUD delete operation on any USM type. The parameters identify the MDR and the object to delete.

Input Parameters

UCFBrokerURL

Defines the UCF Broker URL of your Catalyst server. Defaults to the UCF Broker URL in the Catalyst configuration.

Specify the Broker URL of the Catalyst Broker Service as:

`http://<hostname>:7000/ucf/BrokerService`

When using secure Catalyst communications, specify the secure broker URL as:

`https://<hostname>:7443/ucf/BrokerService`

MdrProduct

Unique identifier of the connecting product.

MdrProdInstance

Unique identifier of the instance of the connecting product as registered in the UCF Broker. CA Process Automation queries the UCF Broker for the list of connectors available and populate this field. You can then select your connector from the drop-down list.

MdrElementID

Unique identifier of the object in the connecting product.

ClassName

Class name of the object (Alert, ComputerSystem, and so on.)

Execute Operator

The Execute operator supports custom operations on any UCF Connector. The parameters identify the MDR, the operation, and the operation parameters.

Input Parameters

CatalystBrokerURL

Defines the UCF Broker URL of your Catalyst server. This value defaults to the UCF Broker URL in the Catalyst configuration.

Specify the Broker URL of the Catalyst Broker Service as:

```
"http://hostname:7000/ucf/BrokerService"
```

When using secure Catalyst communications, specify the secure broker URL as:

```
"https://hostname:7443/ucf/BrokerService"
```

MdrProduct

Defines the unique identifier of the connecting product.

CA Process Automation is identified as:

```
"CA:00074 (CA Process Automation)"
```

MdrProdInstance

Defines the unique identifier of the instance of the connecting product as registered in the UCF Broker. CA Process Automation queries the Catalyst Broker for the list of available connectors and populates this field so you can select your connector from the drop-down list.

CA Process Automation is identified as:

```
"CA:00074:01"
```

Operation Category

Specifies the loaded connector descriptors for the Catalyst operators. Select a descriptor from the drop-down list.

Operation

Specifies the connector descriptor operations after you select an Operation Category. Select a connector descriptor from the drop-down list.

ParameterExpression

Defines a value map that matches the expected parameter structure.

Use this operator when the Execute operator is used as a base operator of a custom operator and the pre-execution code creates the value dynamically. You can use the dataset of the Execute base operator as a reference to construct the value.

ParameterNamespaceExpression

Defines the namespaces that are used in the parameter expression.

Use this operator when the Execute operator is used as a base operator of a custom operator and the pre-execution code creates the value dynamically. You can use the dataset of the Execute base operator as a reference to construct the value.

Parameters

After you select an Operation, click Parameters to specify the input parameters for the query. These fields vary according to the selected operation. Do not enter data for them if you have already entered data in the Parameter Expression and Parameter Namespace Expression fields.

Parameters for each operation can be found here.

Operation-Specific Parameters

Enter the following data about the selected operation.

Note: Do not enter data for them if you have already entered data in the Parameter Expression and Parameter Namespace Expression fields.

AbortStartRequest

Instance

Specifies the instance of the start request to abort.

Cancel

MdrProduct

Specifies the unique identifier of the connecting product that you want to cancel the request with.

Example: CA:00074 (CA Process Automation)

MdrProdInstance

Specifies the unique identifier of the instance of the connecting product (as registered in the Catalyst Broker) that you want to cancel the request with.

Example: CA:00074:01

MdrElementID

Specifies the unique ID of the product element instances.

Example: Process instance ID to cancel.

DelegateTask

TaskID

Specifies the ID of the task that you want to cancel.

Export

Specifies information about the exported content.

ExportRequest

- Folder Name: Specifies the name of the folder that holds the exported content.
- IsAbsolute
- SealModifiableReleaseVersions

ExportFileName

Specifies

- ExportLocation
- OverwriteFile

Filter

ObjectTypes: Lets you filter a specific object type.

Hold

MdrProduct

Specifies the unique identifier of the connecting product.

CA:00074 (CA Process Automation)

MdrProdInstance

Specifies the unique identifier of the instance of the connecting product (as registered in the Catalyst Broker).

Example: CA:00074:01

MdrElementID

Specifies the unique ID of the product element instances.

Example: Process instance ID to be put on hold.

Import

ImportLocation

Specifies the CA Process Automation library location to import the .xml.

Example: /Testimport/

SourceLocation

Specifies the location from where to import the .xml.

Example: C:\test_import\import_test.xml

OverwriteAction

Import

Increase the version of the imported object if the same object exists in the specified import folder. Overrides the release version if the same release version exists.

DoNotImport

Do not import objects with the same name as an existing object.

ImportAndReplace

Import and replace the existing object. Deletes the previous versions of the object.

SetCurrent

Boolean: true (sets the imported version to current) or false (set the imported version as not current).

MakeAvailable

Boolean: true (sets the imported version as available) or false (sets the imported version as not available).

QueryDatasetNames

PathName

Specifies the path that contains the name of the dataset that you want to run a query on.

QueryDatasetParameters

PathName

Specifies the path that contains the name of the dataset that you want to run a query on.

QueryModuleConfigProperties

ModuleName

Specifies the name of the CA Process Automation module properties that you want to configure.

QueryModuleConfigs

(Optional) ModuleName

Specifies the name of the CA Process Automation module.

(Optional) The <ModuleType>

Specifies whether the standard module configuration or custom module group configuration is returned. By default, the QueryModuleConfigs method returns all the available modules. The <ModuleType> attribute takes the following values:

All

Returns the standard modules and the custom modules configuration.

Standard

Returns only standard modules configuration.

Custom

Returns only custom modules configuration.

QueryStartRequestForms

LookUpPath

Specifies the path or the CA Process Automation library folder to query the start request forms.

Example: /SRF_test

IsRecursive

Values:

■ True: Specify true to search recursively over the child folders under the specified folder.

■ False: Specify false to not to search over the child folders under the specified folder.

Keywords

Provide the start request forms Tags property as keywords to filter the list of start request forms.

QueryStartRequests

Instance

Specifies the start request instance to query.

IsArchived

Specifies whether the start request instance is archived.

Values:

■ True: CA Process Automation checks if the provided start request instance is available in the archived table database. If the start request instance is not in the archived table database, the provided instance name returns all of the available archived instances from the database.

■ False: CA Process Automation checks if the provided start request instance is available in the current table database. If the start request instance is not in the current table database, the provided instance name returns all of the available unarchived instances from the database.

QueryTasks

TaskID

Specifies the ID of the task that you want to query.

RootUUID

Specifies the ROOTUUID of the process instance.

ProcessID

Specifies the process instance ID.

State

Specifics the state of the task, such as pending, taken, completed, failed, and timeout.

Release

MdrProduct

Specifies the unique identifier of the connecting product.

Example: CA:00074 (CA Process Automation)

MdrProdInstance

Specifies the unique identifier of the instance of the connecting product (as registered in the Catalyst Broker).

Example: CA:00074:01

MdrElementID

Specifies the unique ID of the product element instances.

Example: Process instance ID to be released.

ReplyTask

TaskID

Specifies the ID of the task that you want to reply to.

IsApproved

Specifies the ID of the task that you want to approve.

Values:

- True: Specifies approved.
- False: Specifies rejected.

Arguments

Specifies the arguments/parameters and their value for the forms that is attached to the task.

ReturnTask

TaskID

Specifies the ID of the task that you want to return.

Start

MdrProduct

Specifies the unique identifier of the connecting product.

Example: CA:00074 (CA Process Automation)

MdrProdInstance

Specifies the unique identifier of the instance of the connecting product (as registered in the Catalyst Broker).

Example: CA:00074:01

MdrElementID

Specifies the unique ID of the product element instances.

Example: Path to the process to start from the library.

Argument

Specifies the parameters and its values to be initialized when the process starts.

SubmitStartRequestForm

Name

Specifies the name of the start request form to submit.

Path

Specifies the path of the start request form to submit.

Arguments

Specifies the parameters and their values to assign during when you submit the start request forms.

StartDate

Specifies the start date of the start request form to submit.

StartTime

Specifies the time that the start request form you want to submit began.

Priority

Specifies the priority of the start request form that you want to submit.

TakeTask

TaskID

Specifies the ID of the task that you want to take.

TransferTask

TaskID

Specifies the ID of the task that you want to transfer.

Users

Specifies the user to transfer the task to.

Example: pamadmin or pamusers

Groups

Specifies the groups to transfer the task to.

Example: pamadmin groups, pamuser groups, pamdesigner groups

UpdateDatasetParameters

PathName

Specifies the name of the path for the dataset that contains the parameters that you want to update.

Parameters

Specify a new parameter value for the dataset that you want to update.

Get Operator

The Get operator supports the CRUD read operation on any USM type. The parameters identify the MDR and the UCF filter values (entitytype, itemtype, recursive, id, and updatedAfter). In addition, the MaxNumberOfObjects parameter restricts the number of objects that the operator returns.

Input Parameters

UCFBrokerURL

Defines the UCF Broker URL of your Catalyst server. Defaults to the UCF Broker URL in the Catalyst configuration.

Specify the Broker URL of the Catalyst Broker Service as:

`http://<hostname>:7000/ucf/BrokerService`

When using secure Catalyst communications, specify the secure broker URL as:

`https://<hostname>:7443/ucf/BrokerService`

MdrProduct

Unique identifier of the connecting product.

MdrProdInstance

Unique identifier of the instance of the connecting product as registered in the UCF Broker. CA Process Automation queries the UCF Broker for the list of connectors available and populate this field. You can then select your connector from the drop-down list.

entitytype

Specifies the type of the entity. Values can be "Alert", "Item" or "Relationship".

For example:

`entitytype=Item`

itemtype

Specifies the type of item. If not specified, then all types are retrieved.

For example:

`itemtype=ComputerSystem`

recursive

Specifies if the connector recursively includes the item and its constituent children and relationships.

id

Specifies a specific object identifier (same as the MdrElementID)

updatedAfter

Specifies only objects that are updated after a specific time.

MaxNumberOfObjects

Specifies the maximum number of objects to retrieve before the operator completes.

SubscribeToChanges Operator

The SubscribeToChanges operator supports event subscriptions on any USM type using UCF filters. The parameters identify the MDR and the UCF filter values (entitytype, itemtype, recursive, id, and updatedAfter). In addition, the MaxNumberOfObjects parameter restricts the number of objects that the operator returns. The timeOut parameter specifies the number of seconds after which the subscription expires. The operator completes when either the number of objects are returned or the timeout expires.

Input Parameters

This operator takes the following input parameters:

UCFBrokerURL

Defines the UCF Broker URL of your Catalyst server. Defaults to the UCF Broker URL in the Catalyst configuration.

Specify the Broker URL of the Catalyst Broker Service as:

```
http://<hostname>:7000/ucf/BrokerService
```

When using secure Catalyst communications, specify the secure broker URL as:

```
https://<hostname>:7443/ucf/BrokerService
```

MdrProduct

Unique identifier of the connecting product.

MdrProdInstance

Unique identifier of the instance of the connecting product as registered in the UCF Broker. CA Process Automation queries the UCF Broker for the list of connectors available and populate this field. You can then select your connector from the drop-down list.

entitytype

Specifies the type of the entity. Values can be "Alert", "Item" or "Relationship".

For example:

```
entitytype=Item
```

itemtype

Specifies the type of item. If not specified, then all types are retrieved.

For example:

```
itemtype=ComputerSystem
```

recursive

Specifies if the connector recursively includes the item and its constituent children and relationships.

id

Specifies a specific object identifier (same as the MdrElementID).

updatedAfter

Specifies only objects that are updated after a specific time.

timeOut

Specifies the number of seconds after which the subscription expires.

MaxNumberOfObjects

Specifies the maximum number of objects to retrieve before the operator completes.

Example

The Catalyst operators can be used directly in processes to build generic content. You can also use the operators as base operators of custom operators for product-specific content.

This example describes how to create Service Desk Incidents from SCOM Alerts.

To build a simple solution

1. Add the SubscribeToChanges operator to a process.

2. Select the SCOM Connector from the MdrProduct/MdrProdInstance lists and Alert from the entitytype list.

3. Add the IncidentUpdate operator to the process. Select the Create check box and the Service Desk Connector from the list.
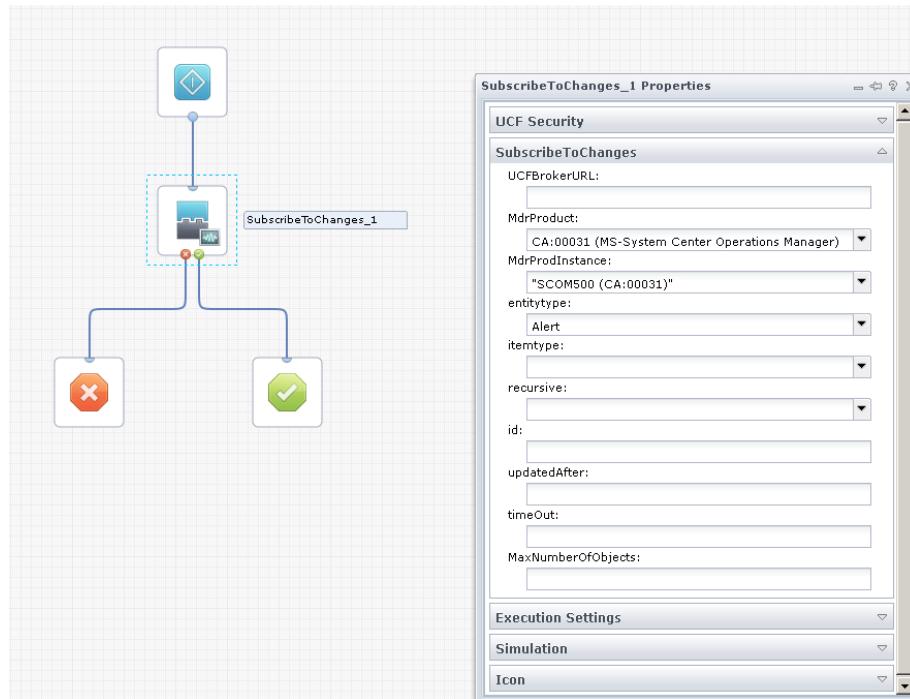
4. Set the values of the Incident properties using the properties of the SCOM Alert object as variable expressions.
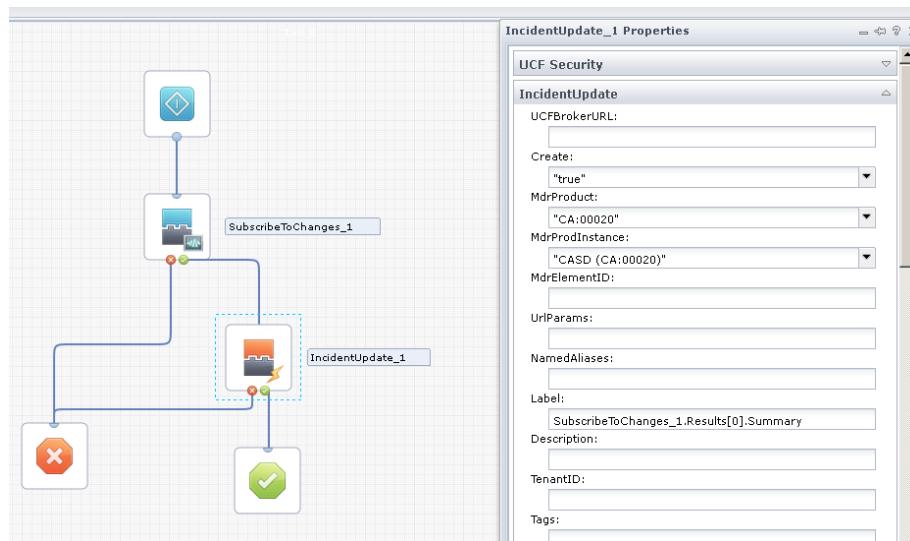
The content is now available for use.

Non-Generic USM Operators

In addition to the generic CRUD operators, there are specific Create/Update operators for each USM type. The operator parameters are created from the properties of the USM types. These operators are dynamically constructed from the USM schema during the initialization of the Catalyst initialization.

The following operators are generated specifically from USM types:

- Alert Update

- ApplicationServerUpdate

- ApplicationSystemUpdate

- ApplicationUpdate

- AssetUpdate

- BackgroundProcessUpdate

- BinaryRelationshipsUpdate

- BootSoftwareUpdate

- BusinessProcessServerUpdate

- BusinessTransactionUpdate

- ChangeOrderUpdate

- ChangePackageUpdate

- ClusterUpdate

- CommentUpdate

- CommunicationServerUpdate

- ComplianceStatusUpdate

- ComputerSystemUpdate

- ConnectorIDUpdate

- ConnectorUpdate

- ContractUpdate

- DatabaseInstanceUpdate

- DatabaseUpdate

- DirectoryServerUpdate

- DiskPartitionUpdate

- EntityIDUpdate

- EntityUpdate

- EnvironmentSensorUpdate

- ExtensionEntityUpdate

- ExtensionRunningHardwareUpdate

- FileUpdate

- GenericIPDeviceUpdate

- GroupUpdate

- HypervisorManagerUpdate

- IncidentUpdate

- InterfaceCardUpdate

- IPConfigUpdate

- ITActivityProfileUpdate

- ITActivityTemplateUpdate

- ITActivityUpdate

- LatestUsmBuildUpdate

- LocationUpdate

- MailServerUpdate

- ManagedAccesssUpdate

- ManagementAgentUpdate

- MediaDriveUpdate

- MemoryUpdate

- MessageServerUpdate

- MultiFunctionEntityUpdate

- NetworkServerUpdate

- NetworkUpdate

- OperatingSystemUpdate

- OrganizationalEntityUpdate

- PersonUpdate

- PhysicalContainerUpdate

- PortUpdate

- PowerSupplyUpdate

- PrinterUpdate

- PrintServerUpdate

- ProblemUpdate

- ProcessorUpdate

- ProjectUpdate
- ProvisionedSoftwareUpdate
- RequestUpdate
- ResourceServerUpdate
- RouterUpdate
- RunningHardwareUpdate
- RunningSoftwareUpdate
- SecurityServerUpdate
- ServiceSpecificationUpdate
- ServiceUpdate
- SnmpV1AccessUpdate
- SnmpV3AccessUpdate
- SoftwareComponentUpdate
- StorageArrayUpdate
- StoragePoolUpdate
- StorageVolumeUpdate
- SwitchUpdate
- TablespaceUpdate
- TransactionContextUpdate
- TransactionSegmentUpdate
- TransactionServerUpdate
- VirtualizationManagerUpdate
- VirtualSystemUpdate
- VMDataStoreUpdate

Catalyst Security Parameters

Every Catalyst operator includes Catalyst Security parameters. These parameters support authentications at the Catalyst level and at the connector level.

After access is granted to Catalyst nodes, you can use claims to get connector-specific security information. See the Connector guide that is provided with the applicable Catalyst connector for information about connector-specific claims.

Username

Defines the user ID that accesses Catalyst nodes.

Password

Defines the password that is associated with the Username.

Because you specify the password as an expression, the text you enter is visible. Avoid using literal strings and specify an expression that references a password variable in a global dataset.

Claims

These claims are not password-protected.

Click Add and enter the first claim name with its value. Repeat this step for each claim. Use the up and down arrows to sequence or delete the claims as necessary.

Claim Name

Defines the name of the claim.

Claim Value

Defines the value of the named claim.

Passwordclaims

These claims are password-protected. CA Process Automation encrypts the password values.

Click Add and enter the first password claim name with its value. Repeat this step for each password claim. Use the up and down arrows to sequence or delete the claims as necessary.

Claim Name

Defines the name of the claim.

Claim Value

Defines the value of the named claim.

Chapter 4: Command Execution

The Command Execution operators run processes and scripts on host operating environments.

Run Program Operator

The Run Program operator starts and monitors a program.
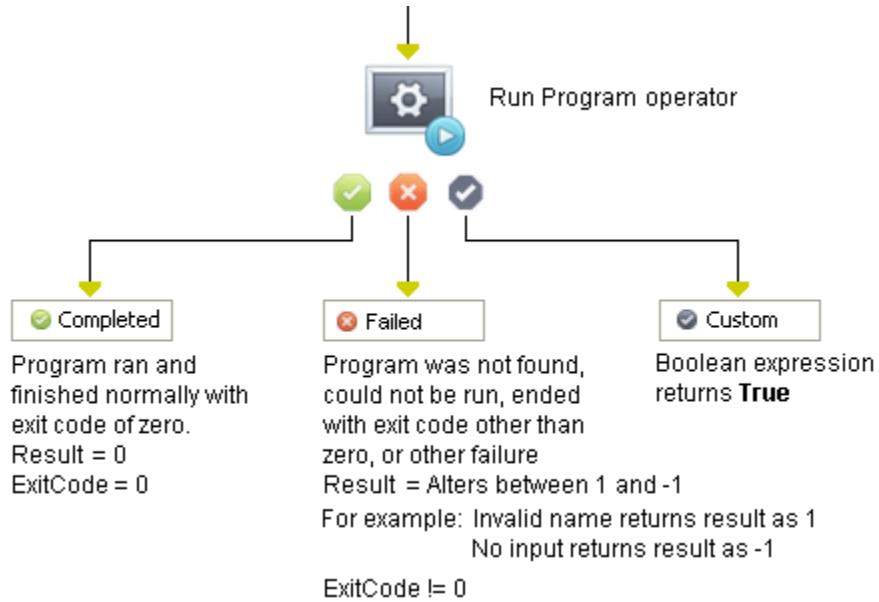
Run Program operator

| ✓ Completed | ✗ Failed | ✓ Custom |

Program ran and finished normally with exit code of zero.
Result = 0
ExitCode = 0

Program was not found, could not be run, ended with exit code other than zero, or other failure
Result = Alters between 1 and -1
For example: Invalid name returns result as 1
No input returns result as -1

ExitCode != 0

Boolean expression returns **True**

Note: For almost all operators, Result is set to 1 upon successful execution and a number other than 1 upon a failed execution. For this specific operator, Result is set to 0 upon successful execution and a number other than 0 upon a failed execution.

Input Parameters

Program name

■ UNIX target: The program must be either a binary file or a shell script following UNIX conventions (the first line of the file must have the full path of the shell, as in #!/bin/ksh).

■ Windows target: The program must be an executable file or a script. The file is typically specified by:

– The name of the file to execute in the working directory for the operator

Or

– In one of the directories specified by the PATH environment variable on the target host.

The path relative to the Working directory specified for the UNIX Command Execution operators.

Profile

UNIX

Specifies a host system profile to be "sourced" to define parts of the execution context of the process.

For example:

`"/home/username/appli_1_profile"`

If no profile is specified here, the profile specified in the UNIX Command Execution property settings (at the category level) is used. If no profile is specified in either the operator or operator settings, then only the user profile that runs the process is used.

Windows

Specifies the path to a batch (.bat) file in Windows that sets environment variables to run by the process.

The variable settings that are defined by the batch file specified here are in addition to any variable settings defined for the user profile that runs the process. If no file is specified for this option, the operator uses the settings defined by the Shell profile parameter set for the Windows process service.

Working directory

Typically, this field specifies the folder that contains the program file or some related files required by the program. Any file that is specified without an explicit path is created or looked for in this directory.

The default if you do not specify a working directory is the home or working directory for the user account that runs the operator.

UNIX

Specifies the working directory for the operator.

For example: "/home/user1"

User ID

Specifies the system user name under which to run the program. The user must have execute permissions on the file.

If you leave this field blank, the default is the default user specified in the configuration settings at the category level.

User names and passwords can be specified at the category level, or stored in named dataset variables so they can be updated centrally without changing process values.

Password

Specifies the current password for the specified user ID.

Specifying the password as a literal string value is considered a bad practice. A much better method is to have the password kept in a dataset variable of type password and to pass that variable.

Parameters

Specifies parameters to pass to the process.

The parameters are passed to the process in the same order that they are listed here. Use the buttons to add, remove, or reorder parameters.

Program parameters are passed individually to the program on startup.

Standard out file

Specifies the file to capture text that the program writes to STDOUT.

For example:

/tmp/trace.log

You can specify the same file for both the standard error and standard out files. However, no order is maintained for the different types of output.

Standard error file

Specifies the file to capture text that the program writes to STDERR.

For example:

```
/tmp/trace.err
```

You can specify the same file for both the standard error and standard out files. However, no order is maintained for the different types of output.

Post output to logs

Logs process output to the global log files.

Post output to dataset variable

Copies output of an operation (stdout and stderr) to the operator dataset variable processOutput.

Truncate log file used for standard out

Replaces an existing log file with the same name every time new output is written. Clear this check box to append output to an existing error log file with the same name.

This check box also replaces an existing file even if it is also used for standard error and the Truncate log file used for standard error check box is not selected.

Truncate log file used for standard error

Replaces an existing error file with the same name every time new output is written. Clear this check box to append output to an existing error file with the same name.

This check box also replaces an existing file even if it is also used for standard output and the Truncate log file used for standard out check box is not selected.

Load OS user profile

Loads the operating system (typically Windows) profile that is associated with the user account. The User ID specifies the profile (in addition to the profile specified by Profile, which specifies environment variables). Typically this profile is not used except to establish associations and similar Windows registry-based constructs for a particular user. A performance penalty is associated with downloading user information from a Domain server.

Kill process on flow end

The OS process running the specified program is killed (if it has not already terminated) once the CA Process Automation process completes.

Output Parameters

programName

profile

workingDir

userID

password

parameters

stdOutFileName

stdErrFileName

isPostToGlobalLog

isPostToOutVar

isTruncateForStdOut

isTruncateForStdErr

isLoadOSProfile

isKillProcessOnFlowEnd

processOutput

ExitCode

PID

Warnings

Run Script Operator

The Run Script operator runs a script on a touchpoint host computer.
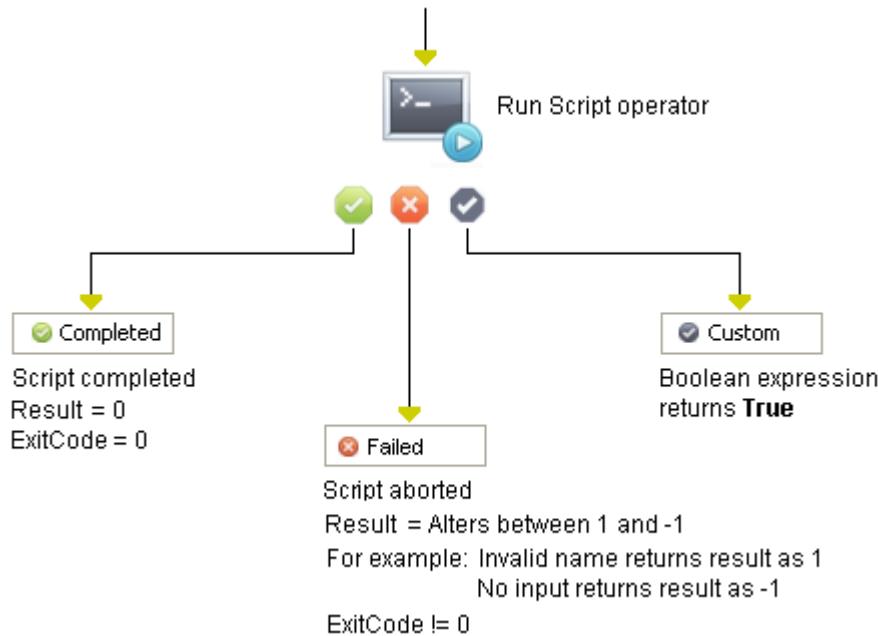
Note: For almost all operators, Result is set to 1 upon successful execution and a number other than 1 upon a failed execution. For this specific operator, Result is set to 0 upon successful execution and a number other than 0 upon a failed execution.

Input Parameters

Script extension

For Windows, specifies the extension that indicates the type of script. Select an option from the drop-down list or type an extension.

Inline script

Specifies a subscript for the script to run. Click the (...) button to open the editor to enter the script to run.

UNIX

The script must be a script that can run according to UNIX protocols. The first line of the script must indicate the full path of the shell that is used to interpret the script (for example, #!/bin/sh).

Windows

The script must be a script that can run according to the Windows extension specified in the Script extension field.

Note: See the *Content Designer Guide* for more information about using the CA Process Automation Code Editor.

Profile

UNIX

Specifies a host system profile to use as the source to define parts of the processing context for the script (for example, /home/username/appli_1_profile).

If you do not specify a profile, the product uses the profile specified in the Command Execution category property settings. The product uses the user file that runs the process when the Command Execution category property settings do not specify a profile.

Windows

Specifies the path to a batch (.bat) file in Windows that sets environment variables for the process to run. The environment variable definitions in the batch file are in the following format:

```
SET SOME_ENV_VAR=/tmp/PAM.exe
SET ANOTHER_ENV_VAR=/tmp/aaaa
```

The variable settings that the specified batch file defines are in addition to settings that are defined for the user profile that runs the process. If this option does not specify a file, the operator uses the settings that the Command Execution category Shell profile parameter value defines.

Working directory

UNIX

Specifies the working directory for the operator (for example, /home/user1). Typically, the working directory is the folder that contains the program file or related files that the program requires. The product looks in this directory for files that are specified without explicit paths.

If you do not specify a working directory, the default value is the home directory of the user account running the script.

Windows

Specifies the working directory for the operator. Typically, the working directory is the folder that contains the script file or related files that the script requires.

If you do not specify a working directory, the value defaults to the working directory of the user account that runs the script.

User ID

Specifies the user name under which to run the script. The expression must have run permissions on the file (for example, Process.Appli_1.User). If you leave the User ID field blank, the value defaults to the user that the Command Execution category configuration specifies.

User names (and their associated passwords) are typically stored in named dataset variables so users can update them centrally without changing process values.

Password

Specifies the current password that is associated with the specified user ID. For example, the following input sets the password to the value of the Process variable Password:

```
Process.Password
```

The product typically evaluates the password against system information. However, where nonstandard security mechanisms are defined on the target host, administrators can deactivate this checking.

Because you specify the password as an expression, the text you enter must be visible. Avoid using literal strings and refer instead to password dataset variables.

Parameters

Specifies parameters as in the following example to pass to the program:

```
/tmp/input_file
/tmp/output_file
```
The product passes parameters to the process in the order that they are listed.

Program parameters are passed individually to the program on startup (that is, they are not concatenated with spaces between them). For example, entering the following expression on one line returns the single parameter "P1P2":

```
P1 + P2
```

Entering the following expressions on two lines returns the two parameters "P1" and "P2":

```
"P1"
"P2"
```

Standard out file

Specifies the standard output file for the script. If you do not specify the full path, the Working directory parameter value defines the root directory for the path (for example, /tmp/trace.log).

The Command Execution category directs the stdout stream from the process to the specified file. You can specify the same file for both the standard error and standard out files. However, the product does not maintain a relative order for the different output types.

Standard error file

Specifies the standard error file for the script. If you do not specify the full path, the Working directory parameter value defines the root directory for the path (for example, /tmp/trace.err).

The Command Execution category directs the stderr stream from the process to the specified file. You can specify the same file for both the standard error and standard out files. However, the product does not maintain a relative order for the different output types.

Post output to logs

Logs process output to the global log files.

Post output to dataset variable

Copies output of an operation (stdout and stderr) to an operator dataset variable (for example, scriptOutput).

Truncate log file used for standard out

Select this check box to have the product replace an existing log file that has the same name when it writes new output.

With the check box selected, the product replaces an existing file even when the following items are true:

- The file is also used for standard error output

- The Truncate log file used for standard error check box is cleared

If the check box is cleared, the product appends output to an existing error log file with the same name.

Truncate log file used for standard error

Select this check box to have the product replace an existing error file that has the same name when it writes new output.

With the check box selected, the product replaces an existing file even when the following items are true:

- The file is also used for standard output

- The Truncate log file used for standard out check box is cleared

If the check box is cleared, the product appends output to an existing error file with the same name.

Load OS user profile

Loads the operating system profile (typically Windows) that is associated with the following items:

- The user account that the User ID specifies

- The profile that Profile specifies, which defines environment variables

The OS user profile is typically used only to establish associations and similar Windows registry-based constructs for a specific user. Downloading user information from a Domain server carries a performance penalty.

Kill process on flow end

If you select this option, the product ends the process when the process flow finishes.

PowerShell Execution Policy

To run PowerShell scripts, Windows imposes a security in terms of its execution policy. The Windows PowerShell execution policy determines whether scripts are allowed to run and, if they can run, whether they must be digitally signed. It also determines whether configuration files can be loaded.

The default execution policy of PowerShell on Windows is Restricted. To run a PowerShell script, change the execution policy to any one of the following:

- RemoteSigned

- AllSigned

- Unrestricted

CA Process Automation provides an option during the installation of the agent or Orchestrator to set the execution policy of the PowerShell script to Remote Signed (meaning downloaded scripts must be signed by a trusted publisher before they can execute). However, you always have an option to change the execution policy through command prompt using the following PowerShell command:

```
Set-ExecutionPolicy
```

...followed by the appropriate policy name. For example, this command sets the execution policy to AllSigned:

```
Set-ExecutionPolicy AllSigned
```

Output Parameters

scriptType

inLineScript

profile

workingDir

userID

password

parameters

stdOutFileName

stdErrFileName

isPostToGlobalLog

isPostToOutVar

isTruncateForStdOut

isTruncateForStdErr

isLoadOSProfile

isKillProcessOnFlowEnd

processOutput

StartDate

StartTime

Result

ExitCode

PID

Reason

Warnings

Run SSH Command Operator

The Run SSH Command operator is designed for use with targets such as network devices or other non-server devices.

For executing on remote servers using SSH, it can be simpler to use the proxy touchpoint or host group concepts.

Note: This operator does not require the user to specify the login sequence.

The Run SSH Command operator takes the following actions:

- Opens an SSH connection to the remote host.

- Sends one command at a time.

- Reads the output of the command until it sees the prompt to indicate that the command is completed.

- Sends the next command.

You can set the maximum amount of time to wait for the prompt before failing the operator. Verify that this setting is greater than the execution time of the longest command that this operator can execute.

You can set this operator to switch to a different user (including root) after login and before executing the commands. Switching users allows the commands to be executed under a different user. Switching to a different user is done interactively.

Input Parameters

For all input that can be specified as a regular expression in this operator, the operator matches the entire reply data against the pattern. The Run SSH Command operator does not match the pattern as a substring of the reply data). A dot '.' matches a new line terminator (can be used to match multiline reply data.

Commands

Remote Hostname

The host name or IP of the computer to connect to.

Use Indexed String Variable for Commands?

If this check box is not selected, you can enter commands in the Commands field. Select this field to specify the commands as indexed String variables in the Commands Indexed String Variable field.

Commands

List of commands to execute on the remote host. Do not end the list with an 'exit' command, as the operator automatically exits the SSH session after the last command executes.

Commands Indexed String Variable

Name of the dataset variable that contains a list of commands to execute on the remote host. Do not end the list with an 'exit' command, as the operator automatically exits the SSH session after the last command executes.

Save Output to Dataset Variable?

Select this check box to copy the output of each command to the dataset of the operator. The output of each command is stored in the SSHCommandsOutput variable.

Commands Output Size limit

Specifies the maximum number of bytes of each command's output to save in the dataset variable of the operator. If this number is not specified, the operator uses value: 4096.

User Command Prompt

This field serves two purposes:

■ Indicates that the user is logged in. The operator looks for this prompt after the user logs in.

■ Indicates that a command (in the list of Commands or Commands Array executed on the SSH session under this user) has finished. The operator can then send the next command in the list.

This field is generally an indication of the command prompt of the user. The field is typically specified as "#", "$", ">", and so on, but must be specified as a regular expression. For example: ".*[$>?:#]" to match any input (including new lines) followed by $ or > or ? or : or #. Specify all the prompts that you expect to see during the execution of the commands.

Important! Start the regular expression with .* to match all data returned by the command until the prompt shows up. This regular expression should match all output from the command until the next prompt.

Note: The brackets are required around the $ to indicate the $ character. $ has a special meaning in regular expressions if it is not surrounded by brackets.

Time to Wait for Prompts (sec):

The amount of time (in seconds) to wait for a prompt before giving up on the prompt to send the commands. If this field is left blank, the operator uses value: 60.

This field applies to the prompts expected after executing each command specified in the operator. The operator cannot tell if a command that executed in the SSH session returned all its data. The operator keeps reading the output of the command until it matches the specified User or Switch User Command prompt or until this timeout is up (whichever comes first). The operator then proceeds to process the output of the command before moving to the next command or failing the operator.

Important! Set this time to be greater than the execution time of the longest command that the operator executes.

Remote Login Information

Pseudo Terminal Type

The type of pseudo terminal to request on the SSH connection. This field overrides the value specified at the category level. If the field is left blank, the operator uses the default value set at the category level. If that value is blank, the operator defaults to VT100.

- VT100 works typically with most computers (especially Linux-based).

- VT400 works typically with Windows-based computers. VT400 is required for Windows platforms, especially when the output retrieved from the SSH server (commands output) contains control characters in the place of spaces. For example, [19;1H in the place of a space in the output. VT400 interprets the spaces correctly for Windows.

Other terminal types can be used. Ensure that you test them before moving the operator into production. Some pseudo terminal types are:

- dumb

- xterm

- vt220

- vt320

- gogrid

Check your SSH server's installation and configuration for the supported pseudo terminals. Some SSH servers list the supported pseudo terminals in the TermInfo folder.

The type of pseudo terminal controls how space characters appear in the commands output. You should test this operator against the pseudo terminals supported by the SSH server to find an appropriate pseudo terminal that returns the spaces appropriately. If the spaces are not returned appropriately, and no pseudo terminal is available that could remedy this issue, do the following:

- Modify the operator's input to accommodate this SSH server's limitation.

- Use JavaScript to polish/extract the output of the commands.

If you request a pseudo terminal that is not supported, some SSH servers return an error while others ignore the requested pseudo terminal type and use another. Review the SSH server's logs for the pseudo terminal used when the operator is running.

Port

The port to connect to on the remote host. This field overrides the value specified at the operator category level. If this field is left blank, the operator uses the default value set at the operator category level. If that default operator category value is blank, the operator uses value: 22.

User name

The user name to use for logging in to the remote host. This field overrides the value specified at the operator category level. If it is left blank, the operator uses the default value set at the operator category level.

Use Private Key for Login?

Specifies if a private key should be used to log in to the remote host (rather than the password information). This field overrides the value specified at the operator category level. If it is left blank, the operator uses the default value set at the operator category level. Select one of the following:

True

Prompts the operator to use a private key.

Selecting true enables the following fields: Private Key Input Source, Private Key Inline Content, Private Key Expression, Private Key File Path, Passphrase for key. The Password field is disabled.

False

Prompts the operator to use password information.

Selecting false disables the following fields: Private Key Input Source, Private Key Inline Content, Private Key Expression, Private Key File Path, Passphrase for key. The Password field is enabled.

Entering any other value prompts the operator to use false and enables all fields (to accommodate the user entering an expression).

Password

The password used for logging in to the remote host. This field overrides the value specified at the operator category level. If it is left blank, the operator uses the default value set at the operator category level.

Private Key Input Source

Specifies how to provide the private key. Select from one of the following:

Inline Content

Enables Private Key Inline Content and disables Private Key Expression and Private Key File Path.

File Path

Enables Private Key File Path and disables Private Key Inline Content and Private Key Expression.

Expression

Enables Private Key Expression and disables Private Key Inline Content and Private Key File Path.

Private Key Inline Content

The content of the private key for logging in to the remote host. This field overrides the value specified at the operator category level. If it is left blank, and the Private Key Input Source is set to Inline Content, the operator uses the default value set at the operator category level.

Private Key Path

The path to the private key for logging in to the remote host. This field overrides the value specified at the operator category level. If it is left blank, and the Private Key Input Source is set to File Path, the operator uses the default value set at the operator category level.

Private Key Expression

The dataset variable that contains the content of the private key for logging in to the remote host.

Note: Ensure that the dataset variable is a multiline String.

Passphrase for key

Optional passphrase to unlock the content of the private key. This field is required if the private key was created with a passphrase. This field overrides the value specified at the operator category level.

A blank Passphrase for Key does not automatically prompt the operator to inherit the Passphrase for Key value from the operator category settings. In fact, the Passphrase for Key field is tied to the Private Key Inline Content, Private Key Path, or Private Key Expression field as follows:

■ If the operator's Passphrase for Key is specified, it is used by the operator.

■ If the operator's Passphrase for Key is blank, Private Key Inline Content is specified (not blank), and the Private Key Input Source is set to Inline Content, then the operator uses a blank Passphrase for key (passphrase not set).

■ If the operator's Passphrase for Key is blank, Private Key File Path is specified (not blank), and the Private Key Input Source is set to File Path, then the operator uses a blank Passphrase for key (passphrase not set).

■ If the operator's Passphrase for Key is blank, the Private Key Expression is specified (not blank), and the Private Key Input Source is set to Expression, then the operator uses a blank Passphrase for key (passphrase not set).

■ For all other cases, the Run SSH Command operator uses the Default Passphrase for key.

Note: The creation of SSH private/public keys is described in the *Administration Guide*.

Switch User Information

Run Commands/Script as Another User?

Should the specified commands be run as a different user? Select true to switch users upon login or false to continue execution as the login user.

This field overrides the value specified at the operator category level. If it is left blank, the operator uses the default value set at the operator category level.

Switch User Command

The command to switch the user on the remote host. This is generally:

■ su - username

or

■ sudo su - username

This field overrides the value specified at the operator category level. If it is left blank, the operator uses the default value set at the operator category level.

Switch User Password Text Prompt

The text prompt that indicates that the remote host requires a password for switching the user to another user. This is generally:

■ Password:

or

■ password:

This parameter must be specified as a regular expression. For example, ".*assword: " to match any input (including new lines) followed by "assword: ".

This field overrides the value specified at the operator category level. If it is left blank, the operator uses the default value set at the operator category level.

If a password is not required for switching to another user (for example, when switching from root to another user), you can enter any value in this field. The operator attempts to match the data read from the SSH session after submitting the Switch User Command against the Switch User Password Text Prompt first, and if it does not match, it then attempts to match the data against and the Switch User Command Prompt to check if a password is required.

Switch User Password

The password to switch the user to another user. This field overrides the value specified at the operator category level. If it is left blank, the operator uses the default value set at the operator category level.

This field is not used if switching to another user does not require a password.

Switch User Command Prompt

This field serves two purposes:

■ To indicate that the user switch occurred.

■ To indicate that a command in the list of Commands or Commands Array executed on the SSH session under the new user (that the operator switched to) has finished, and the operator can send the next command in the list.

This field is generally an indication of the command prompt of the new user (that the operator switched to). It is generally "#", "$", ">", etc. This field must be specified as a regular expression. For example: ".*[$>?:#]" to match any input (including new lines) followed by $ or > or ? or : or #. Specify all the prompts that you expect to see during the execution of the commands. Start the regular expression with .* to match all data returned by the command until the prompt shows up. This regular expression matches all output from the command until the next prompt.

Note: The brackets are required around the $ to indicate the $ character. $ has a special meaning in regular expressions if it is not surrounded by brackets.

Be careful with the RegEx to avoid false positives, for example:

The user enters a bad password when switching to root:

```
# su — root
Password:
```

The answer for a bad password ends with #:

```
su: Sorry
#
```

Which gives the same prompt as when the user enters a good password, where the answer also ends with #:

```
Sun Microsystems Inc.   SunOS 5.10      Generic January 2005
#
```

This field overrides the value specified at the operator category level. If it is left blank, the operator uses the default value set at the operator category level.

Output Parameters

SSHSwitchUserPasswordPrompt

The actual Switch User Pasword Text Prompt read from the SSH session within the time to wait for prompts. This is the data that was matched against the pattern specified in the Switch User Pasword Text Prompt field. This field is empty if switching the user did not require a password, such as when switching from Root to any other user, or if the operator is not set up to switch users.

SSHSwitchUserCommandPrompt

The actual Switch User Command Prompt read from the SSH session (within the time to wait for prompts) the first time after switching to the new user. This is the data that was matched against the pattern specified in the Switch User Command Prompt field. This field is empty if the operator is not set up to switch users.

SSHCommandsOutput

An indexed String containing the output read from each command (within the time to wait for prompts) that ran on the remote host. The output for each command is truncated to the value specified in Commands Output Size limit.

The full output of each command contains the next prompt.

SSHHost

CommandsAsAnArray

parameters

CommandsArray

isPostToOutVar

PostToOutVarSizeLimit

SSHHostUserCommandPrompt

SSHHostWaitTimeout

SSHPseudoTerminalType

SSHPort

SSHUsername

SSHUsePrivateKey

SSHPassword

SSHPrivateKeyOption

SSHPrivateKey

SSHPrivateKeyVar

SSHPrivateKeyPath

SSHPrivateKeyPassphrase

SSHSwitchUser

SSHHostSwitchUserCommand

SSHHostSwitchUserPasswordPrompt

SSHHostSwitchUserPassword

SSHHostSwitchUserCommandPrompt

Operator Ports

Success

■ The operator completed successfully.

■ The operator depends on the patterns specified in User Command Prompt and Switch User Command Prompt to decide if a command (and the operator) succeeds or fails.

For example, a pattern of ".*#" succeeds the following command: 'badCmd':

1. The operator executes it:

   # badCmd

2. The output is read (which matches .*# prompt):

   badCmd: not found
   #

3. Execute the next command.

Failure

■ The user provides invalid input; for example, negative prompts timeout, blank user command prompt, and so on.

■ Unable to establish a connection to the remote host. Check the remote host, user name, password, and keys provided to the operator.

■ The user provides an unknown host in the remote host.

■ Unable to authenticate the user on the remote host:

   – User/password authentication: user name or password is invalid

   – User/private key authentication:

   - A bad passphrase is provided for a passphrase-protected private key: the operator indicates it cannot read the key due to an internal IO problem

   - The passphrase is missing for a passphrase-protected private key: the operator indicates that the passphrase required for key authentication is not supplied.

   - A bad private key is provided: the operator indicates that it is unable to parse the private key, the key format is not supported, or that it cannot read the key due to an internal IO problem.

   - The path to the private key file is invalid, non-existent, or cannot be read.

■ Unable to switch user. Could not match the data read from the SSH session (within the time to wait for prompts), with the specified Switch User Command Prompt pattern.

■ Unable to switch user. Could not match the data read from the SSH session (within the time to wait for prompts), with the specified Switch User Password Text Prompt pattern.

■   The user name/password provided are not allowed to log in through SSH.

■   The operator could not match the command prompt (specified in User Command Prompt, or Switch User Command Prompt) with the output read from a command within the time to wait for prompts. In this case either:

–   The command finished execution and returned data that did not match the pattern in User Command Prompt, or Switch User Command Prompt.

–   The command's execution is taking longer than the time to wait for prompts and the operator stopped reading the output data before the command finished. As a result, it did not match the data read with the pattern in User Command Prompt, or Switch User Command Prompt. In this case, you must increase the Time to Wait for Prompts to be greater than the execution time of the longest command in the list.

Note: In this case, any subsequent commands in the list are not executed and the SSHCommandsOutput only contains the output of the commands that were executed before (including the output read for the current command).

Custom Ports

If set by the user during the process design.

Example

This example explains how the Run SSH Command operator operates interactively.

Follow these steps::

1.   Open an SSH connection to the remote host.

2.   Log in non-interactively using one of the following:

■   A user name and password.

■   A user name and private key.

■   A user name, private key, and passphrase, if the private key was created with a passphrase.

3.   Read from the SSH session until one of the following occurs:

■   You match the pattern specified in *User Command Prompt*. Continue to step 4.

■   The Time to Wait for Prompts expires without matching the pattern specified in User Command Prompt. Fail the operator.

4.   Determine whether the operator is set to *Run Commands as Another User*.

■   If so, go to step 5.

■   If not, go to step 6.

5. Send the *Switch User Command* and read from the SSH session until one of the following occurs:

   ■ You match the pattern specified in *Switch User Password Text Prompt*, which requires a password to switch users.

      – Send the Switch User Password.

      – Read from the SSH session until one of the following occurs:

         – You match the pattern specified in *Switch User Command Prompt*. Continue to step 6.

         – The Time to Wait for Prompts expires without matching the Switch User Command Prompt. Fail the operator.

   ■ You match the pattern specified in *Switch User Command Prompt*, which requires no password to switch users. Continue to step 6.

   ■ The Time to Wait for Prompts expires without matching the *Switch User Password Text Prompt* and the *Switch User Command Prompt*. Fail the operator.

6. The Run SSH Command operator has logged in and switched the user, if applicable. The operator is now ready to execute the commands.

7. Loop through the commands, sending one command at a time, and read from the SSH session until one of the following occurs:

   ■ You match the pattern specified in *User Command Prompt*, if we did not switch to another user. Repeat step 7 and send the next command.

   ■ You match the pattern specified in *Switch User Command Prompt*, if we switched to another user. Repeat step 7 and send the next command.

   ■ The Time to Wait for Prompts expires without matching the User Command Prompt or the Switch User Command Prompt (whichever is applicable). Fail the operator.

Run SSH Script Operator

The Run SSH Script operator works in interactive mode to accommodate network devices, where the presence of a file system is unknown.

Note: For non-interactive SSH communication, use a proxy touchpoint or a host group.

The operator uses the login credentials that you specify to do the following:

■ Open an SSH connection to the remote host.

■ Build a "conn" object.

> **Note:** When you specify a script, either bean shell or JavaScript, the "conn" object is made available in the scope of this script.

> You can leverage the public methods of the "conn" object in the script. You can use these public methods to automate operations executed on an SSH pseudo terminal. Examples include sending commands to the remote SSH host, waiting for the prompt after sending each command, and retrieving the output of each command from the server through SSH.

Unlike the Run Telnet Script operator, the Run SSH Script operator provides you with the "conn" object after logging in to the SSH host.

The difference between the Run SSH Script operator and the Run SSH Commands operator is that in the Run SSH Commands operator:

■ The output of **all** commands can be automatically saved in the operator's dataset.

■ The regular expression specified in the user command prompt field is used to match the prompt after execution of all commands in the list.

■ The time to wait for prompts applies to **all** commands in the list, hence it should be greater than the execution time of the longest command in the list.

While in the Run SSH Script:

■ You can specify which command output to view or save in the operator's dataset by calling the following commands in this sequence:

   1. 'conn.sendLine()'

   2. 'conn.waitFor()'

   3. 'conn.getLastOutput()'

■ You can specify a different regular expression to match the prompt after execution of each command.

■ You can specify a different time to wait for the prompt after execution of each command.

Input Parameters

Input parameters for the Run SSH Script operator include the following.

SSH Script Attributes

Remote Host name

The host name or IP of the computer to connect to.

Script Type:

The type of the script specified in the Inline Script field. Select from bean shell script (.bsh) and JavaScript (.js). If this field is left blank, the operator defaults to .bsh.

Inline Script?

This operator provides two methods to provide the script: inline or as an expression. Select this check box to provide the script inline.

Inline Script

The script, written in bean shell or javascript, uses the conn object and its API as follows:

- Send a command to the remote host

- Wait for the command to terminate

- Retrieve the output of the last command

The APIs that the conn object exposes are detailed in Run SSH Script Operator Inline Script APIs (see page 106).

Script as Expression

Provides the script as an expression. See the Inline Script field for information about the script itself.

Parameters

The CA Process Automation parameters to pass to the script. Only simple CA Process Automation parameter types can be passed to the script as follows:

- PAM Boolean is passed as a Boolean object.

- PAM Date is passed as a Date object.

- PAM Double is passed as a Double object.

- PAM Integer is passed as an Integer object.

- PAM Long is passed as a Long object.

- PAM String is passed as a String object.

- PAM Object Reference is passed as a String object.

Complex CA Process Automation parameters types (indexed types, ValueMaps, and so on) cannot be passed to the script.

The script can access these objects through the args array of objects, where args[0] corresponds to the first parameter in the list, args[1] corresponds to the second parameters, and so on.

Output Variable Names

The names of the variables, created in the script, to save in the operator dataset at the end of the execution of the script.

The variables must be defined in the scope of the script so they are visible at the end of the execution and can be saved in the operator's dataset.

The output variables are saved as follows:

■ Boolean object saves as a PAM Boolean.

■ Date object saves as a PAM Date.

■ Integer object saves as a PAM Integer.

■ Number object saves as a PAM Long or Double object.

■ String object saves as a PAM string.

■ Character object saves as a PAM string.

■ An array of objects saves as an indexed PAM type, where the PAM type is defined by the type of the first object in the array of objects.

■ Undefined saves as a PAM string with undefined as its value (the variable has not been assigned a value).

Run SSH Script Operator Inline Script APIs

The product uses a script that is written in bean shell or JavaScript for the Run SSH Script operator Inline Script field. The script uses the conn object, which exposes the following APIs:

```
void send (String str, boolean log) throws Exception
```

This method sends data to the remote host.

String str

Defines the data to send to the remote host.

boolean log

Exposes or hides the data sent to remote host in the CA Process Automation logs.

To debug, follow the interaction between the operator and the remote host. Set the following code in the *<install_dir>*/server/conf/log4j.xml file:

```
<category name="com.company.c2o.servicegroup.netutils">


    <priority value="DEBUG" />
</category>
```

Also, set the CA Process Automation log file (c2o.log) to accept the DEBUG statements in log4j.xml.

When you set the debug level, the Command Execution operator category starts logging to the CA Process Automation log file (c2o.log) at the DEBUG level. The CA Process Automation logs expose any data sent to the remote host through send or sendLine.

Values:

True: Logs the String str value when logging at the DEBUG level.

False: Does not log the String str value.

Note: c2o.log is the CA Process Automation log file, *not* the process logs. The operators do not write messages to the process log.

This method has no return values. The product generates an exception if the API cannot write the data to the remote host.

```
public void sendLine (String str, boolean log) throws Exception
```

This method appends a new line character to the data and sends it to the remote host. To force the remote host to start running the command that is sent in the parameter, use this method.

This method has the same parameters and values as void send (String str, boolean log) throws Exception.

```
void send (String str) throws Exception
```

This method is equivalent to Send (String str, true).

```
void sendLine (String str) throws Exception
```

This method is equivalent to SendLine (String str, true).

```
public boolean waitFor(String pattern, int timeout) throws Exception
```

This method reads the output from the remote host and stops when either of the following results occurs:

- The output matches the pattern that is specified in the parameters.

- The timeout interval expires.

The product stores the output from the remote host by each call to the waitFor method in a buffer accessible through the getLastOutput() method. Each waitFor call overrides the buffer content from the previous call.

The next waitFor call starts reading the output from where the previous waitFor call stopped reading. This is relevant when you use the method with getLastOutput.(). For example, if a waitFor call does not match the entire command output, the next call can contain the remaining output from the previous command.

Important! To avoid mixing the previous command output with the current command output, call waitFor after each sendLine call.

The Telnet script and SSH script operators use different mechanisms to read data from the remote host:

- The Telnet waitFor starts reading data directly from the host and matches as it reads from the host.

- The SSH waitFor retrieves the data read so far from a buffer and matches it against the pattern.

- Calling waitFor after each sendLine call makes the SSH and Telnet operators behave alike.

Parameters include:

String pattern

> Defines a regular expression to match against the data that the product reads from the remote host. Typically, this pattern matches any data up to the next prompt (for example, ".*[$]"). Because you can match (and retrieve) the command output data (including new lines up to the next prompt), start the pattern with .*.

> **Note**: The method matches the entire data read (during this waitFor call) against the pattern. The method does not match the pattern as a substring of the data read. Also, a dot (.) can match a new line terminator (it can be used to match multiline reply data).

Int timeout

> Defines the interval (in seconds) during which to read data from the remote host and match it against the pattern.

Returns are Boolean:

True

> The data read before the timeout expired matches the pattern.

False

> The data read before the timeout expired does not match the pattern.

Note: The method returns a value as soon as it matches the pattern or the timeout expires.

The product generates an exception in the following cases:

- The Int timeout parameter is less than or equal to 0.

- The String pattern parameter is null or empty.

- The product cannot read data from the SSH session.

- The pattern contains invalid syntax.

- An error occurs when matching the pattern to the data received.

public String getLastOutput()

This method returns the content of the buffer where the last waitFor call saved the data that it read from the remote host. This data does not always match the waitFor pattern. The buffer stores whatever the most recent waitFor call read, whether waitFor returned true or false.

This method has no parameters.

This method does not generate exceptions.

Remote Login Information

Pseudo Terminal Type

The type of pseudo terminal to request on the SSH connection. This field overrides the value specified at the operator category level. If the field is left blank, the operator uses the default value set at the operator category level. If that value is blank, the operator defaults to VT100.

■ VT100 works typically with most computers (especially Linux-based).

■ VT400 works typically with Windows-based computers. VT400 is required for Windows platforms, especially when the output retrieved from the SSH server (commands output) contains control characters in the place of spaces. For example, [19;1H in the place of a space in the output. VT400 interprets the spaces correctly for Windows.

Other terminal types can be used. Ensure that you test them before moving the operator into production. Some pseudo terminal types are:

■ dumb

■ xterm

■ vt220

■ vt320

■ gogrid

Check your SSH server's installation and configuration for the supported pseudo terminals. Some SSH servers list the supported pseudo terminals in the TermInfo folder.

The type of pseudo terminal controls how space characters appear in the commands output. You should test this operator against the pseudo terminals supported by the SSH server to find an appropriate pseudo terminal that returns the spaces appropriately. If the spaces are not returned appropriately, and no pseudo terminal is available that could remedy this issue, do the following:

■ Modify the operator's input to accommodate this SSH server's limitation.

■ Use JavaScript to polish/extract the output of the commands.

If you request a pseudo terminal that is not supported, some SSH servers return an error while others ignore the requested pseudo terminal type and use another. Review the SSH server's logs for the pseudo terminal used when the operator is running.

Port

> The port to log in to on the remote host. This field overrides the value specified at the operator category level. If this field is left blank, the operator uses the default value set at the operator category level; if that default operator category value is blank, the operator uses value: 22.

User name

> The user name used for logging into the remote host. This field overrides the value specified at the operator category level. If it is left blank, the operator uses the default value set at the operator category level.

Use Private Key for Login?

> Specifies if a private key should be used to log in to the remote host (rather than the password information). This field overrides the value specified at the operator category level. If it is left blank, the operator uses the default value set at the operator category level. Select one of the following:

> **True**

>> Prompts the operator to use a private key.

>> Selecting true enables the following fields: Private Key Input Source, Private Key Inline Content, Private Key Expression, Private Key File Path, Passphrase for key. The Password field is disabled.

> **False**

>> Prompts the operator to use password information.

>> Selecting false disables the following fields: Private Key Input Source, Private Key Inline Content, Private Key Expression, Private Key File Path, Passphrase for key. The Password field is enabled.

> Entering any other value prompts the operator to use false and enables all fields (to accommodate the user entering an expression).

Password

> The password used for logging in to the remote host. This field overrides the value specified at the operator category level. If it is left blank, the operator uses the default value set at the operator category level.

Private Key Input Source

Specifies how to provide the private key. Select from one of the following:

Inline Content

Enables Private Key Inline Content and disables Private Key Expression and Private Key File Path

Expression

Enables Private Key Expression and disables Private Key Inline Content and Private Key File Path

File Path

Enables Private Key File Path and disables Private Key Inline Content and Private Key Expression

Private Key Inline Content

The content of the private key for logging in to the remote host. This field overrides the value specified at the operator category level. If it is left blank, and the Private Key Input Source is set to Inline Content, the operator uses the default value set at the operator category level.

Private Key as Expression

The dataset variable that contains the content of the private key for logging in to the remote host.

Note: Ensure that the dataset variable is a multiline String.

Private Key File Path

The path to the private key for logging in to the remote host. This field overrides the value specified at the operator category level. If it is left blank, and the Private Key Input Source is set to File Path, the operator uses the default value set at the operator category level.

Passphrase for key

Optional passphrase to unlock the content of the private key. This field is required if the private key was created with a passphrase. This field overrides the value specified at the operator category level.

A blank Passphrase for key does not automatically prompt the operator to inherit the Passphrase for Key value from the operator category. In fact, the Passphrase for key field is tied to the Private Key Inline Content, Private Key File Path, or Private Key as Expression field as follows:

- If the operator's Passphrase for key is specified, the operator uses it.

- The operator uses a blank Passphrase for key (passphrase not set) if the following conditions are set:

    - The operator's Passphrase for Key is blank

    - Private Key Inline Content is specified (not blank)

    - Private Key Input Source is set to Inline Content

- The operator uses a blank Passphrase for key (passphrase not set) if the following conditions are set:

    - The operator's Passphrase for key is blank

    - Private Key File Path is specified (not blank)

    - Private Key Input Source is set to File Path

- The operator uses a blank Passphrase for key (passphrase not set) if the following conditions are set:

    - The operator's Passphrase for Key is blank

    - The Private Key Expression is specified (not blank)

    - The Private Key Input Source is set to Expression

For all other cases, the Run SSH Script operator uses the Default Passphrase for key.

Note: The creation of SSH private/public keys is described in the CA Process Automation *Content Administrator Guide*.

Output Parameters

Each variable in the Output Variable Names list is created with the corresponding CA Process Automation type.

If a variable name in the Output Variable Names does not exist in the script, the operator creates the corresponding variable as an empty String.

The bean shell interpreter provides a robust environment where if the script throws an exception or contains an error, the variables defined and initialized in the script before the occurrence of the error can be retrieved with their values. On the other hand, the javascript interpreter does not allow for any variable to be retrieved with its value if the script throws an exception or contains an error.

Output parameters include:

SSHHost

inLineScriptLanguage

inlineScriptType

inLineScript

scriptExpression

parameters

outputVariables

SSHPseudoTerminalType

SSHPort

SSHUsername

SSHUsePrivateKey

SSHPassword

SSHPrivateKeyOption

SSHPrivateKey

SSHPrivateKeyVar

SSHPrivateKeyPath

SSHPrivateKeyPassphrase

Operator Ports

Success

The operator completed successfully.

Failure

The operator fails for any of the following reasons

- The user provides invalid input; for example: empty inline script, empty remote host, negative port, empty user name, and so on.

- The user specifies an Inline Script Type other than '.bsh' and '.js'.

- Unable to establish a connection to the remote host. Verify the remote host and port provided to the operator.

- The user provides an unknown host in the remote host.

- Unable to authenticate the user on the remote host:

    - User/password authentication: user name or password is invalid.

- User/private key authentication:

  - A bad passphrase is provided for a passphrase-protected private key: the operator indicates it cannot read the key due to an internal IO problem

  - The passphrase is missing for a passphrase-protected private key: the operator indicates that the passphrase required for key authentication is not supplied.

  - A bad private key is provided: the operator indicates that it is unable to parse the private key, the key format is not supported, or that it cannot read the key due to an internal IO problem.

  - The path to the private key file is invalid, non-existent, or cannot be read.

- The username/password provided are not allowed to log in through SSH.

- The user provides a complex data type in the list of parameters. Complex value types (arrays, ValueMaps, and so on) cannot be passed to the script. Use simple value types such as Double, Integer, Long, String, Date, and Boolean.

- When executing a '.bsh' or '.js' script:

  - Parse or syntax error while evaluating the script.

  - The script threw an exception.

  - An error occurred when executing the script.

  - Error when retrieving a variable from the script's scope (at the end of execution).

Custom Ports

If set by the user during the process design.

Example

The following procedure is an example of how to use the Run SSH Script operator.

Follow these steps::

1. The Run SSH Script operator reads the login credentials you specify and uses them as follows:

    ■   To connect and authenticate to the remote SSH host.

    ■   To create the "conn" object.

    An example of a completed Remote Login Information panel for this operator follows:
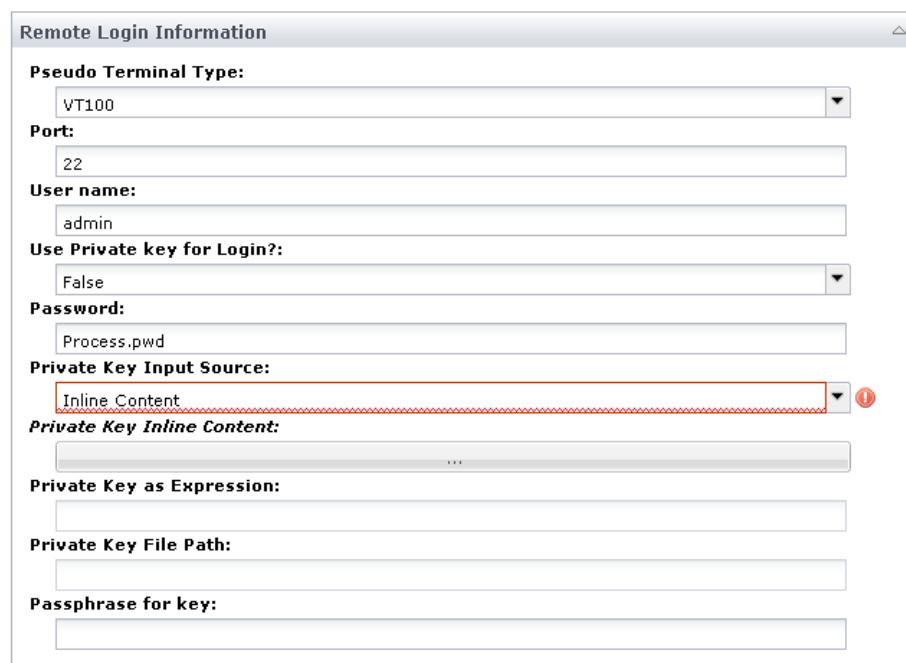
2. Complete the Script parameters as follows:

   a. Specify the remote host name.

   b. Specify the parameters to pass to the script, where the values in this example follow:

      A string whose value is the word: "date".

   c. Specify the name of the output variables that you create in the script (bean shell or javascript), and that you want saved to the operator's dataset at the end of execution (here, "svrDate").

      Note: The creation of dataset variables directly from the script is currently not supported.
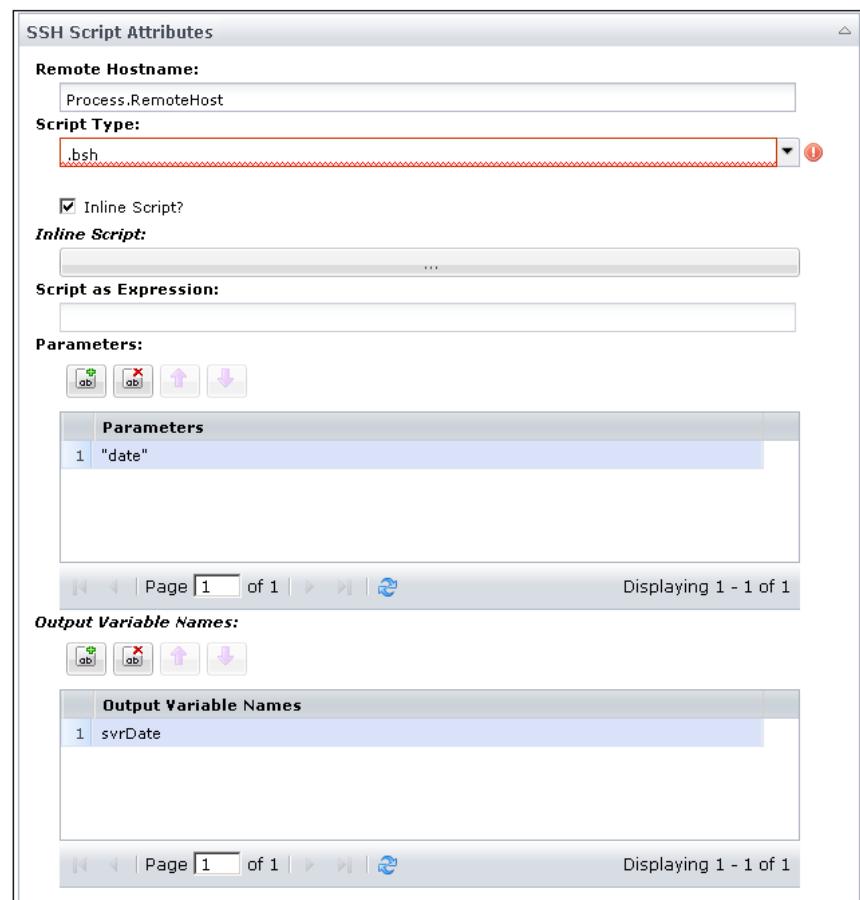
3. In the inline script, you leverage the "conn" object as shown in the following bean shell script example:

```
Private Key Inline Content                              ✕

String svrDate - "";

if (conn.waitFor(".*[$] ",10 )) {
    conn.sendLine(args[0]);
    if(conn.waitFor(".*[$] ",10 )) {
      svrDate = conn.getLastOutput();
    }
}

                                       Cancel      OK
```
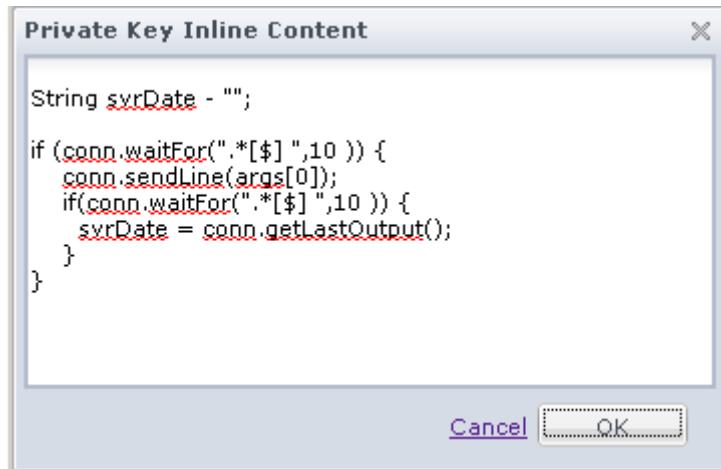
a. Create the svrDate variable to be visible at the script scope, so it can be saved to the dataset of the operator at the end of execution.

b. Use "conn.waitFor()" to wait for the first prompt ".*[$]" (Reg Ex) up to 10 seconds.

c. If the prompt is found within 10 seconds, then use "conn.sendLine()" to send the value of the first parameter passed to the script, followed by a new line character. In this example, the value is: args[0] = the word "date".

d. Use "conn.waitFor()" to wait for the next prompt ".*[$]" (Reg Ex) up to 10 seconds.

e. If the prompt is found within 10 seconds, use 'conn.getLastOutput()' to retrieve the output read during the last call of the method waitFor and store it in svrDate.

At the end of execution, the operator saves the String object svrDate as a CA Process Automation string in the dataset of the operator.

Note: The prompt that was matched appears in the output returned by conn.getLastOutput(). Some SSH servers return this prompt twice in the output, while others return it once.

Run Telnet Command Operator

The Run Telnet Command operator takes the following actions:

- Opens a Telnet connection to the remote host.

- Sends one command at a time.

- Reads the output of the command until it sees the prompt to indicate that the command was completed.

- Sends the next command.

Note: The Run Telnet Command operator and the Run SSH Command operator log in to the remote host differently. The Run Telnet Command operator performs the login in an interactive way.

You can set the maximum amount of time to wait for the prompt before failing the operator. Verify that this setting is greater than the execution time of the longest command that this operator executes.

You can set the Run Telnet Command operator to switch to a different user (including root) after login and before executing the commands. Switching users allows subsequent commands to be executed under a different user. Switching to different user is done interactively.

Input Parameters

Notes:

- For all input that can be specified as a regular expression in the Run Telnet Command operator:

    – The operator matches the entire reply data against the pattern.

    – The operator does not match the pattern as a substring of the reply data.

- A dot '.' matches a new line terminator (it can be used to match multiline reply data).

Commands

Remote Hostname

The host name or IP of the computer to connect to.

Use Indexed String Variable for Commands?

If this check box is not selected, you can enter commands in the Commands field. Select this option to specify the commands as indexed String variables in the Commands Indexed String Variable field.

Commands

List of commands to execute on the remote host. Do not end the list with an exit command, as the operator automatically exits the Telnet session after the last command executes.

Commands Indexed String Variable

Name of the dataset variable that contains a list of commands to execute on the remote host. Do not end the list with an 'exit' command. The operator automatically exits the Telnet session after the last command executes.

Save Output to Dataset Variable?

Select this check box to copy the output of each command to the dataset of the operator. The output of each command is stored in the TelnetCommandsOutput variable.

Commands Output Dataset Variable Size Limit (bytes)

Specify the maximum number of bytes of each command's output to save in the dataset variable of the operator. If this number is not specified, the operator uses value 4096.

Remote Login Information

Pseudo Terminal Type

The type of pseudo terminal to request on the Telnet connection. This field overrides the value specified at the operator category level. If the field is left blank, the operator uses the default value set at the operator category level. If that value is blank, the operator defaults to VT100.

- VT100 works typically with most computers (especially Linux-based).

- VT400 works typically with Windows-based computers. VT400 is required for Windows platforms, especially when the output retrieved from the Telnet server (commands output) contains control characters in the place of spaces. For example, [19;1H in the place of a space in the output. VT400 interprets the spaces correctly for Windows.

Other terminal types can be used. Ensure that you test them before moving the operator into production. Some pseudo terminal types are:

■ dumb

■ xterm

■ vt220

■ vt320

■ gogrid

Check your Telnet server's installation and configuration for the supported pseudo terminals.

The type of pseudo terminal controls how space characters appear in the commands output. You should test this operator against the pseudo terminals supported by the Telnet server to find an appropriate pseudo terminal that returns the spaces appropriately. If the spaces are not returned appropriately, and no pseudo terminal is available that could remedy this issue, do the following:

■ Modify the operator's input to accommodate this Telnet server's limitation.

■ Use JavaScript to polish/extract the output of the commands.

If you request a pseudo terminal that is not supported, some Telnet servers return an error while others ignore the requested pseudo terminal type and use another. Review the Telnet server's logs for the pseudo terminal used when the operator is running.

Remote Port

The port to connect to on the remote host. This field overrides the value specified at the operator category level. If this field is left blank, the operator uses the default value set at the operator category level; if that default category value is blank, the operator uses value: 23.

Connection Timeout (sec)

The connection timeout in seconds before giving up on the connection. This field overrides the value specified at the operator category level. If this field is left blank, the operator uses the default value set at the operator category level; if that default category value is blank, the operator uses value: 20.

Login Scheme

The login scheme; select from one of the following:

0

Prompts the operator to use user name and password

1

Prompts the operator to use password only, which disables the following fields: Password Text Prompt, and Password

2

Prompts the operator to use no user name and no password, which disables the following fields: User login Text Prompt, User name, Password Text Prompt, and Password

This field overrides the value specified at the operator category level. If you do not specify a value and leave the field blank, the operator uses the value set at the operator category level. Any other value prompts the operator to use user name and password.

User Login Text Prompt

The text prompt that indicates that the remote host requires a login ID for logging in. This is generally:

■ Login:

or

■ login:

This parameter must be specified as a regular expression. For instance: ".*ogin: " to match any input (including new lines) followed by "ogin: ".

This field overrides the value specified at the operator category level. If it is left blank, the operator uses the default value set at the category level.

User name

The user name to be used for logging into the remote host. This field overrides the value specified at the operator category level. If it is left blank, the operator uses the default value set at the category level.

User Password Text Prompt

The text prompt that indicates that the remote host requires a password for the user logging in. This is generally:

■ Password:

or

■ password:

This parameter must be specified as a regular expression. For instance: ".*assword: " to match any input (including new lines) followed by "assword: ". This field overrides the value specified at the operator category level. If it is left blank, the operator uses the default value set at the category level.

Password

The password used for logging into the remote host. This field overrides the value specified at the operator category level. If it is left blank, the operator uses the default value set at the category level.

User Command Prompt

This field serves two purposes:

■ To indicate that the user is logged in.

■ To indicate that a command in the list of Commands or Commands Array executed on the Telnet session under this user has finished, and the operator can send the next command in the list.

This field is generally an indication of the command prompt of the user. It is generally "#", "$", ">", and so on, but must be specified as a regular expression. For example, ".*[$>?:#]" to match any input (including new lines) followed by $ or > or ? or : or #. You should specify all the prompts that you expect to see during the execution of the commands. The regular expression should start with .* to be able to match all data returned by the command until the prompt shows up. This regular expression should be able to match all output from the command until the next prompt.

This field overrides the value specified at the operator category level. If it is left blank, the operator uses the default value set at the category level.

Note: The brackets are required around the $ to indicate the $ character. $ has a special meaning in regular expressions if not surrounded by brackets.

Time to Wait for Prompt

The amount of time (in seconds) to wait for a prompt before giving up on the prompt to send the commands. This field overrides the value specified at the operator category level. If it is left blank, the operator uses the default value set at the category level. If that default value is blank, the operator uses value: 60.

This field applies to the prompts expected after each command in the login and switch user commands, and also the prompts expected after executing each command specified in the operator. The operator cannot tell if a command executed in the Telnet session returned all its data; hence it keeps reading the output of the command until it matches the specified User or Switch User Command prompt or until this timeout is up (whichever comes first). It then proceeds to process the output of the command before moving to the next command or failing the operator.

Important! Set this time to be greater than the execution time of the longest command to be executed by the operator.

Switch User Information

Run Commands/Script as Another User?

Should the script or the specified commands be run as a different user? Select True or False.

■ If true, the current logged in user switches to another user before executing the commands.

■ If false, the following fields are disabled: Switch User Command, Switch User Password Text Prompt, Switch User Password, and Switch User Command Prompt.

This field overrides the value specified at the operator category level. If it is left blank, the operator uses the default value set at the category level.

Switch User Command

The command to switch the user on the remote host. This is generally:

■ su - username

   or

■ sudo su - username

This field overrides the value specified at the operator category level. If it is left blank, the operator uses the default value set at the category level.

Switch User Password Text Prompt

The text prompt that indicates that the remote host requires a password for switching the user to another user. This is generally:

■ Password:

   or

■ password:

This parameter must be specified as a regular expression. For example, ".*assword: " to match any input (including new lines) followed by "assword: ".

This field overrides the value specified at the operator category level. If it is left blank, the operator uses the default value set at the category level.

If a password is not required for switching to another user (for example, when switching from root to another user), you can enter any value in this field. The operator attempts to match the data read from the Telnet session after submitting the Switch User Command against the Switch User Password Text Prompt first, and if it does not match, it then attempts to match the data against the Switch User Command Prompt to check if a password is required.

Switch User Password

The password to switch the user to another user. This field overrides the value specified at the operator category level. If it is left blank, the operator uses the default value set at the category level.

This field is not used if switching to another user does not require a password.

Switch User Command Prompt

This field serves two purposes:

- To indicate that the user switch occurred.

- To indicate that a command in the list of Commands or Commands Array executed on the Telnet session under the new user (that the operator switched to) has finished, and the operator can send the next command in the list.

This field is generally an indication of the command prompt of the new user (that the operator switched to). It is generally "#", "$", ">", etc. It must be specified as a regular expression.

For example: ".*[$>?:#]" to match any input (including new lines) followed by $ or > or ? or : or #. Specify all the prompts that you expect to see during the execution of the commands. The regular expression should also start with .* to match all data returned by the command until the prompt shows up. This regular expression should be able to match all output from the command until the next prompt.

Note: The brackets are required around the $ to indicate the $ character. $ has a special meaning in regular expressions if not surrounded by brackets.

Be careful with the RegEx to avoid false positives, for instance:

The user enters a bad password when switching to root:

```
# su – root
Password:
```

The answer for a bad password ends with #:

```
su: Sorry
#
```

Which gives the same prompt as when the user enters a good password, where the answer also ends with #:

```
Sun Microsystems Inc.   SunOS 5.10      Generic January 2005
#
```

This field overrides the value specified at the operator category level. If it is left blank, the operator uses the default value set at the category level.

Output Parameters

TelnetUserLoginPrompt

The actual User Login Text Prompt that was read from the Telnet session within the time to wait for prompts. This is the data that was matched against the pattern specified in the User Login Text Prompt field.

TelnetUserPasswordPrompt

The actual User Password Text Prompt that was read from the Telnet session within the time to wait for prompts. This is the data that was matched against the pattern specified in the User Password Text Prompt field.

TelnetUserCommandPrompt

The actual User Command Prompt that was read from the Telnet session (within the time to wait for prompts) the first time, either after login or after connection (if no login is required). This is the data that was matched against the pattern specified in the User Command Prompt field.

TelnetSwitchUserPasswordPrompt

The actual Switch User Password Text Prompt that was read from the Telnet session within the time to wait for prompts. This is the data that was matched against the pattern specified in the Switch User Password Text Prompt field.

This field is empty if switching the user did not require a password; for example, when switching from Root to any other user, or if the operator is not set up to switch users.

TelnetSwitchUserCommandPrompt

The actual Switch User Command Prompt that was read from the Telnet session (within the time to wait for prompts) the first time after switching to the new user. This is the data that was matched against the pattern specified in the Switch User Command Prompt field. This field is empty if the operator is not set up to switch users.

TelnetCommandsOutput

An indexed String containing the output read from each command (within the time to wait for prompts) that ran on the remote host. The output for each command is truncated to the value specified in Commands Output Size limit.

The full output of each command contains the next prompt.

TelnetHost

CommandsAsAnArray

parameters

CommandsArray

isPostToOutVar

PostToOutVarSizeLimit

TelnetPseudoTerminalType

TelnetPort

TelnetHostConnectTimeout

TelnetloginScheme

TelnetHostUserLoginPrompt

TelnetUsername

TelnetHostUserPasswordPrompt

TelnetPassword

TelnetHostUserCommandPrompt

TelnetHostWaitTimeout

TelnetSwitchUser

TelnetHostSwitchUserCommand

TelnetHostSwitchUserPasswordPrompt

TelnetHostSwitchUserPassword

TelnetHostSwitchUserCommandPrompt

TelnetSwitchUserPasswordPrompt

TelnetSwitchUserCommandPrompt

TelnetUserLoginPrompt

TelnetUserPasswordPrompt

TelnetUserCommandPrompt

Operator Ports

Success

- The operator completed successfully.

- The operator depends on the patterns specified in User Command Prompt and Switch User Command Prompt to decide if a command (and the operator) succeeds or fails.

  For example, a pattern of ".*#" succeeds the following command: 'badCmd':

  1. The operator executes it:

     # badCmd

  2. The output is read (which matches .*# prompt):

     badCmd: not found
     #

  3. Execute the next command.

Failure

- The user provides invalid input; for example, negative remote port, negative connection timeout, 0 or negative prompts timeout, blank user command prompt, and so on.

- Unable to log in. Could not match the data read from the Telnet session (within the time to wait for prompts), with the specified User Login Text Prompt pattern.

- Unable to log in. Could not match the data read from the Telnet session (within the time to wait for prompts), with the specified User Password Text Prompt pattern.

- Unable to log in. Could not match the data read from the Telnet session (within the time to wait for prompts), with the specified User Command Prompt pattern.

- Unable to switch user. Could not match the data read from the Telnet session (within the time to wait for prompts), with the specified Switch User Command Prompt pattern.

- Unable to switch user. Could not match the data read from the Telnet session (within the time to wait for prompts), with the specified Switch User Password Text Prompt pattern.

- The user provided incorrect username/password login credentials.

- The user provided incorrect username/password credentials to switch the user.

- Unable to switch to another user unless the username/password scheme is used to log in to the Telnet session.

- The user provided an unknown remote host.

- Telnet connection to the remote host is refused (Telnet is not allowed).

- The username/password provided are not allowed to log in through Telnet.

- The operator could not match the command prompt (specified in User Command Prompt, or Switch User Command Prompt) with the output read from a command within the time to wait for prompts. In this case either:

  - The command finished execution and returned data that did not match the pattern in User Command Prompt, or Switch User Command Prompt.

  - The command's execution is taking longer than the time to wait for prompts and the operator stopped reading the output data before the command finished. As a result, it did not match the data read with the pattern covered in User Command Prompt, or Switch User Command Prompt. In this case, you must increase the Time to Wait for Prompts to be greater than the execution time of the longest command in the list.

  Note: In this case, any subsequent commands in the list are not executed and the TelnetCommandsOutput only contains the output of the commands that were executed before (including the output read for the current command).

Custom Ports

If set by the user during the process design.

Example

Use the Run Telnet Command operator interactively

The Run Telnet Command operator operates in the following interactive manner:

1. Open a Telnet connection to the remote host.

2. Do one of the following:

   - If no login is required, go to step 8.

   - If password only login is required, go to step 4.

   - If user name and password login is required, then read from the Telnet session until one of the following occurs:

     - You match the pattern specified in *User Login Text Prompt*. (Go to Step 3.)

     - The Time to Wait for Prompts is up. If this time elapses, the operator fails.

3. Send the User name.

4. Read from the Telnet session until one of the following occurs:

   - You match the pattern specified in User Password Text Prompt (go to Step 5).

   - The Time to Wait for Prompts expires. If this occurs, fail the operator.

5. Send the password, then:

    a. Read from the Telnet session until one of the following occurs:

        –   You match the pattern specified in *User Command Prompt,* and continue.

        –   The Time to Wait for Prompts expires, then fail the operator.

    b. Determine if the operator is set to Run Commands as Another User:

        –   If so, go to step 6.

        –   If not, go to step 7.

6. Send the *Switch User Command* and then do the following:

    a. Read from the Telnet session until one of the following occurs:

        –   You match the pattern specified in *Switch User Password Text Prompt*.

        –   You match the pattern specified in *Switch User Command Prompt*.

        –   The Time to Wait for Prompts expires.

    b. Take one of the following actions, based on the outcome:

        –   If you match the pattern for *Switch User Password Text Prompt* (password required to switch user), send the *Switch User Password,* and read from the Telnet session until one of the following occurs:

            - You match the pattern specified in Switch User Command Prompt. Go to Step 7.

            - The Time to Wait for Prompts expires and the operator fails.

        –   If you match the pattern for *Switch User Command Prompt* (no password required to switch user), go to Step 7.

        –   If the Time to Wait for Prompts expires, fail the operator.

7. The Run Telnet Command operator has logged in and switched the user, if applicable. The operator is now ready to execute the commands.

8. Loop through the commands, send one command at a time, and read from the Telnet session until one of the following occurs:

    ■   You match the pattern specified in User Command Prompt (if we did not switch to another user). Repeat step 8 and send the next command.

    ■   You match the pattern specified in Switch User Command Prompt (if we switched to another user). Repeat step 8 and send the next command.

    ■   The Time to Wait for Prompts expires without matching the User Command Prompt or the Switch User Command Prompt (whichever is applicable). Fail the operator.

Run Telnet Script Operator

The Run Telnet Script operator uses the remote host and port you specify to do the following:

- Open a Telnet connection to the remote host.

- Build a "conn" object.

  Note: When you specify a script, either bean shell or javascript, the "conn" object is made available in the scope of this script.

  You can leverage the public methods of the "conn" object in the script. The public methods are used to authenticate on the Telnet session and automate operations executed on a Telnet pseudo terminal. Examples of automated operations include the following:

  - Sending commands to the remote Telnet host.

  - Waiting for the prompt after sending each command.

  - Retrieving the output of each command.

Unlike the Run SSH Script operator, the Run Telnet Script operator does not authenticate a user ID on the Telnet connection. Instead, you must leverage the 'conn' object's methods to authenticate on the Telnet connection at the beginning of your script.

Unlike the Run Telnet Command operator, the Run Telnet Script operator:

- Lets you specify which command output to view or save in the dataset of the operator. You call the following methods in this sequence:

  1. 'conn.sendLine()'

  2. 'conn.waitFor()'

  3. 'conn.getLastOutput()'

- Lets you specify a different regular expression to match the prompt after the execution of each command.

- Lets you specify a different time to *wait for the prompt* after the execution of each command.

Input Parameters

Input parameters for the Run Telnet Script operator are as follows.

Script

Remote Host name

The host name or IP of the computer to connect to.

Script Type:

The type of the script specified in the Inline Script field. Select from bean shell script (.bsh) and JavaScript (.js). If this field is left blank, the operator defaults to .bsh.

Inline Script?

This operator provides two methods to provide the script: inline or as an expression. Select this check box to provide the script inline.

Inline Script

The script, written in bean shell or javascript, uses the conn object and its API as follows:

- Send a command to the remote host

- Wait for the command to terminate

- Retrieve the output of the last command

The APIs that the conn object exposes are detailed in Run Telnet Script Operator's Inline Script APIs (see page 133).

Script as Expression

Provides the script as an expression. See the Inline Script field for information on the script itself.

Parameters

The CA Process Automation parameters to pass to the script. Only simple CA Process Automation parameter types can be passed to the script as follows:

- PAM Boolean is passed as a Boolean object

- PAM Date is passed as a Date object

- PAM Double is passed as a Double object.

- PAM Integer is passed as an Integer object.

- PAM Long is passed as a Long object.

- PAM String is passed as a String object.

- PAM Object Reference is passed as a String object.

Complex CA Process Automation parameters types (indexed types, ValueMaps, and so on) cannot be passed to the script.

The script can access these objects through the args array of objects, where args[0] corresponds to the first parameter in the list, args[1] corresponds to the second parameters, and so on.

Output Variable Names

The names of the variables, created in the script, to save in the operator's dataset at the end of the execution of the script.

The variables must be defined in the scope of the script so they are visible at the end of the execution and can be saved in the operator's dataset.

The output variables are saved as follows:

■ Boolean object saves as a PAM Boolean

■ Date object saves as a PAM Date

■ Integer object saves as PAM Integer

■ Number object saves as PAM Long or Double object

■ String object saves as PAM string

■ Character object saves as PAM string

■ An array of objects saves as an indexed PAM type, where the PAM type is defined by the type of the first object in the array of objects.

■ Undefined saves as a PAM string with 'undefined' as its value.

Run Telnet Script Operator Inline Script APIs

The product uses a script that is written in bean shell or JavaScript for the Run Telnet Script operator Inline Script field. The script uses the conn object, which exposes the following APIs:

```
void send (String str, boolean log) throws Exception
```

This method sends data to the remote host.

String str

Defines the data to send to the remote host.

boolean log

Exposes or hides the data sent to remote host in the CA Process Automation logs.

To debug, follow the interaction between the operator and the remote host. Set the following code in the *<install_dir>*/server/conf/log4j.xml file:

```
<category name="com.company.c2o.servicegroup.netutils">


    <priority value="DEBUG" />
</category>
```

Also, set the CA Process Automation log file (c2o.log) to accept the DEBUG statements in log4j.xml.

When you set the debug level, the Command Execution operator category starts logging to the CA Process Automation log file (c2o.log) at the DEBUG level. The CA Process Automation logs expose any data sent to the remote host through send or sendLine.

Values:

True: Logs the String str value when logging at the DEBUG level.

False: Does not log the String str value.

Note: c2o.log is the CA Process Automation log file, *not* the process logs. The operators do not write messages to the process log.

This method has no return values. The product generates an exception if the API cannot write the data to the remote host.

```
public void sendLine (String str, boolean log) throws Exception
```

This method appends a new line character to the data and sends it to the remote host. To force the remote host to start running the command that is sent in the parameter, use this method.

This method has the same parameters and values as void send (String str, boolean log) throws Exception.

```
void send (String str) throws Exception
```

This method is equivalent to Send (String str, true).

```
void sendLine (String str) throws Exception
```

This method is equivalent to SendLine (String str, true).

```
public boolean waitFor(String pattern, int timeout) throws Exception
```

This method reads the output from the remote host and stops when either of the following results occurs:

- The output matches the pattern that is specified in the parameters.

- The timeout interval expires.

The product stores the output from the remote host by each call to the waitFor method in a buffer accessible through the getLastOutput() method. Each waitFor call overrides the buffer content from the previous call.

The next waitFor call starts reading the output from where the previous waitFor call stopped reading. This is relevant when you use the method with getLastOutput.(). For example, if a waitFor call does not match the entire command output, the next call can contain the remaining output from the previous command.

Important! To avoid mixing the previous command output with the current command output, call waitFor after each sendLine call.

The Telnet script and SSH script operators use different mechanisms to read data from the remote host:

- The Telnet waitFor starts reading data directly from the host and matches as it reads from the host.

- The SSH waitFor retrieves the data read so far from a buffer and matches it against the pattern.

- Calling waitFor after each sendLine call makes the SSH and Telnet operators behave alike.

Parameters include:

String pattern

> Defines a regular expression to match against the data that the product reads from the remote host. Typically, this pattern matches any data up to the next prompt (for example, ".*[$]"). Because you can match (and retrieve) the command output data (including new lines up to the next prompt), start the pattern with .*.

> **Note**: The method matches the entire data read (during this waitFor call) against the pattern. The method does not match the pattern as a substring of the data read. Also, a dot (.) can match a new line terminator (it can be used to match multiline reply data).

Int timeout

> Defines the interval (in seconds) during which to read data from the remote host and match it against the pattern.

Returns are Boolean:

True

> The data read before the timeout expired matches the pattern.

False

> The data read before the timeout expired does not match the pattern.

Note: The method returns a value as soon as it matches the pattern or the timeout expires.

The product generates an exception in the following cases:

■ The Int timeout parameter is less than or equal to 0.

■ The String pattern parameter is null or empty.

■ The product cannot read data from the Telnet session.

■ The pattern contains invalid syntax.

■ An error occurs when matching the pattern to the data received.

public String getLastOutput()

This method returns the content of the buffer where the last waitFor call saved the data that it read from the remote host. This data does not always match the waitFor pattern. The buffer stores whatever the most recent waitFor call read, whether waitFor returned true or false.

This method has no parameters.

This method does not generate exceptions.

Remote Login Information

Pseudo Terminal Type

The type of pseudo terminal to request on the Telnet connection. This field overrides the value specified at the module level. If the field is left blank, the operator uses the default value set at the module level. If that value is blank, the operator defaults to VT100.

- VT100 works typically with most computers (especially Linux-based).

- VT400 works typically with Windows-based computers. VT400 is required for Windows platforms, especially when the output retrieved from the Telnet server (commands output) contains control characters in the place of spaces. For example, [19;1H in the place of a space in the output. VT400 interprets the spaces correctly for Windows.

Other terminal types can be used. Ensure that you test them before moving the operator into production. Some pseudo terminal types are:

- dumb

- xterm

- vt220

- vt320

- gogrid

Check your Telnet server's installation and configuration for the supported pseudo terminals.

The type of pseudo terminal controls how space characters appear in the commands output. You should test this operator against the pseudo terminals supported by the Telnet server to find an appropriate pseudo terminal that returns the spaces appropriately. If the spaces are not returned appropriately, and no pseudo terminal is available that could remedy this issue, do the following:

- Modify the operator's input to accommodate this Telnet server's limitation.

- Use JavaScript to polish/extract the output of the commands.

If you request a pseudo terminal that is not supported, some Telnet servers return an error while others ignore the requested pseudo terminal type and use another. Review the Telnet server's logs for the pseudo terminal used when the operator is running.

Remote Port

The port to connect to on the remote host. This field overrides the value specified at the module level. If this field is left blank, the operator uses the default value set at the module level. If that default module value is blank, the operator uses value: 23.

Connection Timeout (sec)

The connection timeout in seconds before giving up on the connection. This field overrides the value specified at the module level. If it is left blank, the operator uses the default value set at the module level. If that default module value is blank, the operator uses value: 20.

Output Parameters

Each variable in the Output Variable Names list is created with the corresponding CA Process Automation type.

If a variable name in the Output Variable Names does not exist in the script, the Run Telnet Script operator creates the corresponding variable as an empty string.

The bean shell interpreter provides a robust environment for the script. For example, if the bean shell script throws an exception or contains an error, the variables defined and initialized in the script, before the occurrence of the error, can be retrieved with their values. Alternatively, if the javascript script throws an exception or contains an error, the javascript interpreter does not allow for any variable to be retrieved with its value.

Output parameters include the following:

TelnetHost

inLineScriptLanguage

inlineScriptType

inLineScript

scriptExpression

parameters

outputVariables

TelnetPseudoTerminalType

TelnetPort

TelnetHostConnectTimeout

Operator Ports

Success

The operator completed successfully.

Failure

The operator fails for any of the following reasons

■ The user provides invalid input; for example: empty inline script, empty remote host, negative port, and so on.

■ The user specifies an Inline Script Type other than '.bsh' and '.js'.

■ Unable to establish a connection to the remote host. Review the remote host and port provided to the operator.

■ The user provides an unknown host in the remote host.

■ The user provides a complex data type in the list of parameters. Complex value types (arrays, Value Maps, and so on) cannot be passed to the script. Use simple value types such as Double, Integer, Long, String, Date, and Boolean.

■ When executing a '.bsh' or '.js' script:

– Parse or syntax error while evaluating the script.

– The script threw an exception.

– An error occurred when executing the script.

– Error when retrieving a variable from the scope of the script (at the end of execution).

Custom Ports

If set by the user during the process design.

Example

The following procedure provides an example of how to use the Run Telnet Script operator.

1. The Run Telnet Script operator reads the login information you specify. The operator uses this login information to connect to the remote Telnet host and to create the "conn" object. The following Remote Login Information example shows typical entries:
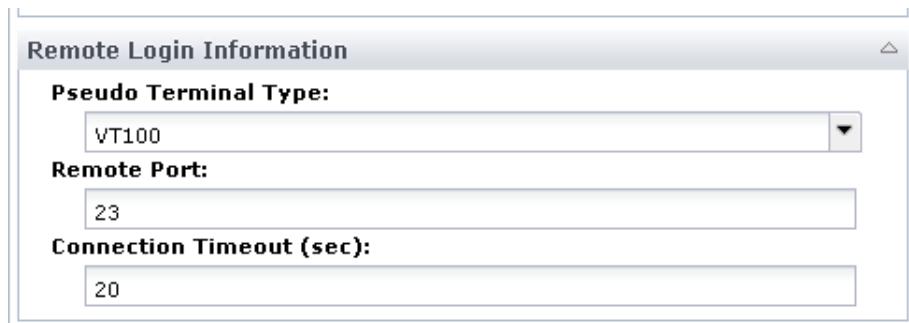
2. You specify the following in the Script palette:

    ■   The remote host name.

    ■   The parameters to pass to the script. In the following example, the user, the password, and a string whose value is the word: '"date".

    ■   The name of the output variables that you create in the script (bean shell or javascript) that you want saved to the operator's dataset at the end of execution. If you do not want a variable saved into the operator's dataset at the end of execution, then you do not need to specify it here.

        Note: The creation of dataset variables directly from the script is currently not supported. The *Output Variable Names* field plays the role of C2OSVD in this case.

    In this example, the following output variable names are saved to the dataset of the operator at the end of execution: 'svrDate', 'loginStr', 'pwdStr', and 'promptStr'.

3. In the Inline script, you can leverage the 'conn' object, built by CA Process Automation. How you can leverage the 'conn' object is shown in the following bean shell script example:
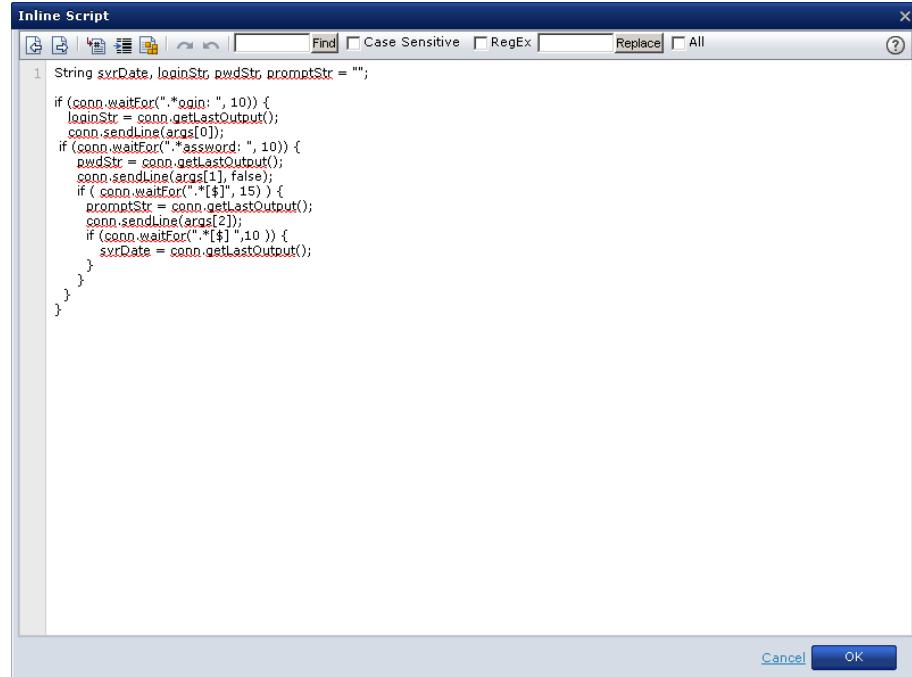
a. Create the svrDate, loginStr, pwdStr, and promptStr variables to be visible at the script scope, so they can be saved to the operator's dataset at the end of execution.

b. Use '*conn.waitFor()*' to wait for the login prompt, "*.*ogin: *" (Reg Ex) up to 10 seconds.

c. If the login prompt is found within 10 seconds, then use '*conn.getLastOutput()*' to save the login prompt that was matched in the loginStr variable.

d. Use '*conn.sendLine()*' to send the username passed as the first object in the list of parameters: *args[0]*.

e. Use '*conn.waitFor()*' to wait for the password prompt "*.*assword: *" (Reg Ex) up to 10 seconds

f. If the password prompt is found within 10 seconds, then use '*conn.getLastOutput()*' to save the password prompt we matched in the pwdStr variable.

g. Use '*conn.sendLine()*' to send the password passed as the second object in the list of parameters: *args[1]*.

h. Use '*conn.waitFor()*' to wait for the user prompt "*.*[$]* " (Reg Ex) up to 15 seconds.

i. If the user prompt is found within 15 seconds, then use '*conn.getLastOutput()*' to save the user prompt we matched in the promptStr variable.

j. Use '*conn.sendLine()*' to send the command passed as the third object in the list of parameters: *args[2]*.

k. Use '*conn.waitFor()*' to wait for the user prompt "*.*[$]* " (Reg Ex) up to 10 seconds.

l. If the user prompt is found within 10 seconds, then use '*conn.getLastOutput()*' to save the matched user prompt with the output of the command in the svrDate variable.

m. At the end of execution, the Run Telnet Script operator saves the output variables in the dataset of the operator as CA Process Automation objects.

The prompt that was matched appears in the output returned by *conn.getLastOutput()*.

Chapter 5: Databases

Databases operators support JDBC type 2 drivers to communicate a database. Database operators can perform different database operations such as queries on the database, but they do not support administrative operations such as stopping a database server, back up/recovery, and so on.

The connection string differs based on the server type.

The TNS name and thin driver type combination is supported only in Oracle 12.*x* versions and up. CA Process Automation does not validate any combination or the server credentials provided at the operator level.

Note: You can use the Databases operators with a different Relational Database Management System (RDBMS) than the one used by the CA Process Automation databases. For example, if CA Process Automation was installed using Microsoft SQL, you can use the Databases operators with Oracle. However, the appropriate database driver file must first be deployed correctly to your CA Process Automation installation. See the *Installation Guide* for details.

Oracle Parameters

Inherit Settings

If checked, values shown reflect the current values of the Domain. At runtime, the values are selected from the Environment (if defined).

Driver Type

Accepts one of the following options:

Thin

The thin driver is a pure Java implementation of Oracle's networking protocol (Net8). Being self-contained, it may be used on any machine with or without Oracle installed, or even distributed with application classes in an applet.

OCI

The "OCI" (type 2) driver consists of Java wrappers to the low-level Oracle call interface (OCI) libraries used by utilities like SQL*Plus to access the database server. The OCI driver offers potentially better performance that the thin driver. It however requires the OCI libraries to be installed on the local machine.

KPRB

The "KPRB" driver is used for Java stored procedures and database JSP's.

Driver

Specifies the Oracle JDBC driver.

Server Host

Specifies the host where the Oracle database is running.

UserName

Specifies the default Oracle database user.

Password

Specifies the password for the default Oracle user.

ServiceID

Specifies the Oracle service ID.

TNS Name

Oracle TNS Names translates a local database alias to all the connectivity information needed to connect to the database. This includes IP address, port, database Service ID or service name, and so on. This information is stored in a file called tnsnames.ora in the Oracle directory.

Maximum Rows

Specifies the maximum rows to retrieve.

Client Encryption

Oracle supports multiple data encryptions for the client (RC4_40, RC4_56, RC4_128, RC4_256, DES40C, DES56C, 3DES112, 3DES168, SSL, AES256, AES192, and AES128). The user should provide one of these values. These values will be set as properties as part of the connection. The encryption levels RC4_128 and RC4_256 are for domestic editions only.

Client Checksum

Specifies checksums supported by Oracle. Refer to your Oracle documentation.

MSSQL Server Parameters

Inherit Settings

If checked, values shown reflect the current values of the Domain. At runtime, the values are selected from the Environment (if defined).

Default Driver

Specifies the default MSSQL driver.

Default Server Host

Specifies the host where the MSSQL database is running.

Default Server Port

Specifies the default MSSQL database server port.

Default UserName

Specifies the default MSSQL database user.

Default Password

Specifies the password for the default MSSQL user.

Default Maximum Rows

Specifies the maximum number of rows to retrieve.

Default Database Name

Specifies the MSSQL database name.

Default Instance Name

Specifies the MSSQL instance name.

MySQL Parameters

Inherit Settings

If checked, values shown reflect the current values of the Domain. At runtime, the values are selected from the Environment (if defined).

Default Driver

Specifies the default MySQL driver.

Default Server Host

Specifies the host where the MySQL database is running.

Default Server Port

Specifies the default MySQL database server port.

Default UserName

Specifies the default MySQL database user.

Default Password

Specifies the password for the default MySQL user.

Default Maximum Rows

Specifies the maximum number of rows to retrieve.

Default Database Name

Specifies the default MySQL database name.

Sybase Parameters

Inherit Settings

If checked, values shown reflect the current values of the Domain. At runtime, the values are selected from the Environment (if defined).

Default Server Type

Specifies one of the following Sybase server types:

■　　Adaptive Server Anywhere (ASA) (the default value)

■　　Adaptive Server Enterprise (ASE)

Default Connection Protocol

Specifies the default connection protocol. The default value is Tds.

Default Driver

Specifies the default Sybase driver. The default value is com.sybase.jdbc2.jdbc.SybDriver.

Default Server Host

Specifies the host were Sybase is running.

Default Server Port

Specifies the default Sybase server port.

Default UserName

Specifies the default Sybase username.

Default Password

Specifies the password for the Sybase user.

Default Maximum Rows

Specifies the maximum number of rows to retrieve. If blank, the default value is 10 rows.

Default Cache Buffer Size

The Sybase Cache Buffer Size is the amount of memory used by the driver to cache insensitive result set data. Valid values are:

-1 = All data is cached.

0 = All data is cached up to 2GB.

X = Must be positive. This is the buffer size (must be a power of 2). This value is specified in KB.

After the limit is reached (if any), the result set data is written to disk.

Default Batch Performance Workaround

The Sybase batch performance workaround is one of the following:

■ True is the JDBC v3.0 compliant mechanism.

■ False is the native batch mechanism. False is the default.

Operator Level Properties

The following are connection parameters for the Database operators.

Database Server Login Parameters

The Database Server Login parameters configure settings that are required to log into the database server and communicate with the database server.

User Name

Specifies the database username.

Password

Specifies the password for the database user.

Notes:

■ If "Other" is selected as the Database Type, enter the User Name and Password port and other information to use to connect to the database (if necessary). The connection wizard constructs a URL that is populated under the operator properties.

■ If you want to use Windows Authentication, do *not* specify a User Name/Password when configuring a Databases operator. See the *Content Administrator Guide* for more information about configuring Windows Authentication for the Databases Operators.

Connection Wizard

A wizard that lets you specify connection properties. You can enter the that configure how the operator connects to the database.

Connection URL

Represents a Universal Resource Locator (URL) that specifies a particular type of database server (compatible with the local JDBC driver) and a particular host.

Notes:

■   If "Other" is selected as the Database Type (in the Connection Wizard), you must enter a JDBC URL in this field.

■   If you want to use Windows Authentication, append the following string to the Connection URL:

`;integratedSecurity=true`

Connection Wizard Properties

Database Type

Select the type of database from the drop-down list:

■   MySQL (default)

■   Oracle

■   SQLSERVER

■   Sybase

■   Informix

■   Hypersonic

■   Postgres

■   DB2

■   Interbase

■   Ingres

■   Other

Database Type, Server Host, Server Port, Database Name, Driver Name and Connection URL are always displayed in the Connection Wizard. The remaining fields are shown/hidden based on the selection of Database Type.

For example, if you select "Oracle" as a database type, then all the fields related to Oracle are shown and the remaining fields are hidden.

Note: If you select SQLSERVER as the Database Type and you want to use Windows Authentication (integrated security), leave all fields blank except the Connection URL field. In this field, enter something similar to the following example:

"jdbc:sqlserver//ms-db-host:1433;DatabaseName=dbname;integratedSecurity=true "

If you define Host, Port, Database Name or SQL Server Instance name, the operator creates the URL based on those values instead of using the one you configured in the Connection URL field with integratedSecurity set to true.)

Note: If Other is selected as the Database Type, enter the User Name and Password to be used to connect to the database (if required) in the Database Server Login parameters.

Other Database Type

If your database is not listed in the Database Type drop-down list, enter it here.

Server Host

Specifies the host where database is running.

Server Port

Specifies the database server port.

Database Name

Specifies the name of the database.

Driver Name

Specifies the database driver name (the Java class that interfaces with the database).

Connection URL

Specifies a database URL is a Universal Resource Locator (URL) that specifies a particular type of database server (compatible with the local JDBC driver) and a particular host.

This field is updated as information is entered into the Connection Wizard.

The following properties only display if they apply to specified Database Type.

Sybase Cache Buffer Size

Available when Sybase is selected as the database type. This field specifies the amount of memory used by the driver to cache insensitive result set data. Valid values are:

-1

All data is cached.

0

All data is cached, up to 2GB.

X

This is the buffer size; must be positive, and a power of 2. This value is specified in kilobytes.

Sybase Batch Performance Workaround

Available when Sybase is selected as the database type. Select from either True or False.

■   False is the default (native batch mechanism).

■   True is for JDBC v3.0 compliant mechanism.

Sybase Connection Protocol

Available when Sybase is selected as the database type. Specifies the connection protocol for Sybase. The default connection protocol is Tds. The connection string differs based on the server type.

Sybase Server Type

Available when Sybase is selected as the database type. Specifies the Sybase server types. Select one of the following from the drop-down list:

■   Adaptive Server Anywhere (ASA) (default)

■   Adaptive Server Enterprise (ASE)

Oracle Driver Type

Available when Oracle is selected as the database type. Specifies the driver type for Oracle. Select one of the following from the drop-down list:

thin

The thin driver is a pure Java implementation of the Oracle networking protocol (Net8). Being self-contained, it may be used on any machine with or without Oracle installed, or distributed with application classes in an applet.

OCI

The OCI (type 2) driver consists of java wrappers to the low-level Oracle call interface (OCI) libraries used by utilities such as SQL*Plus to access the database server. The OCI driver can potentially improve performance over the thin driver, however, it requires the OCI libraries to be installed on a local machine.

KBRP

The KPRB driver is used for Java stored procedures and database JSPs.

Oracle Service ID

Available when Oracle is selected as the database type. A support expression that specifies the Oracle service ID.

Oracle TNS Name

Available when Oracle is selected as the database type. Translates a local database alias to all the connectivity information needed to connect to the database. This includes IP address, port, database Service ID, or service name, and so on. This information is stored in a file called tnsnames.ora in the Oracle directory.

Oracle Client Encryption

Available when Oracle is selected as the database type. Oracle supports the following multiple data encryptions for the client:

RC4_40, RC4_56, RC4_128, RC4_256

DES40C, DES56C, 3DES112, 3DES168

SSL, AES256, AES192, AES128

Specify one of these values that will be set as properties as part of the connection. The encryption levels RC4_128 and RC4_256 are for domestic editions only.

Oracle Client Checksum

Available when Oracle is selected as the database type. Specifies the Oracle Client Checksum value (a number calculated by the database from all the bytes stored in a data or redo block). Oracle supports MD5 checksum. For more information, refer to the Oracle documentation.

SQLServer Instance Name

Available when SQLServer is selected as the database type. On a given server, you can run multiple SQLServer services, each with their own ports, logins, and databases. Each of these services is called an instance of SQL Server. This field specifies a particular instance name for SQLServer.

Hypersonic Database Type

Available when Hypersonic is selected as the database type. Select one of the following from the drop-down list:

■ Server

■ File

■ In-memory

Bulk Insert into Database Operator

The Bulk Insert into Database operator lets you quickly import a bulk number of rows into a database table or view that you specify.

Input Parameters

Data Source

The table name to supply in the SQL statement, as a String or Variable.

Insert map array

An array of ValueMaps; each one represents a row to insert into the database.

Important! Verify that single quotes encapsulate any string values.

The ValueMap parameter name that you want to insert into a table must be same as its associated column name.

Each of the variables within the value map must correspond to the columns in the table. For example, if you have a table with two columns: "Name" and "Number", then your ValueMap must be organized the same way.

Output Parameters

DataSource

JDBCInsertMapArray

UserName Password

DatabaseType

OtherDatabaseType

DriverName

DatabaseName

CacheBufferSize

BatchPerfWorkaround

ConnectionProtocol

ServerType

DriverType

ServiceID

TNSName

ClientEncryption

ClientChecksum

InstanceName

HypersonicDatabaseType

ServerHost

ServerPort

ConnectionString

Delete from Database Operator

Use the Delete from Database operator to delete rows in a table based on criteria that you specify.

Input Parameters

Input Source

Specifies that the user can choose to submit an SQL statement as an in-line expression (the default) or data variable. Select either Inline Text or Expression from the drop-down list.

Inline Text

Only available when Inline Text is selected as the Input Source. Specifies the generic SQL statement as inline text. Click the Inline Text field to open the Inline Text editor, where you can enter a literal SQL statement.

Expression

Only available when Expression is selected as the Input Source. Specifies the generic SQL statement as an expression. Use this field to provide a variable.

Input Parameters

An array of input values. If the generic SQL statement specified uses JDBC escape syntax and requires input parameters, they can be specified here.

Output Parameters

Query Results

Returns the number of rows deleted.

Reason

Specifies the reason if the Operator fails after execution.

Result

Specifies the result of the Operator execution.InputSource

InlineText

InputSourceExpression

JDBCInputParamArray

IsConstructSQLStatement

CompleteSQLStatement

DataSource

SelectionCriteria

UserName

Password

DatabaseType

OtherDatabaseType

DriverName

DatabaseName

CacheBufferSize

BatchPerfWorkaround

ConnectionProtocol

ServerType

DriverType

ServiceID

TNSName

ClientEncryption

ClientChecksum

InstanceName

HypersonicDatabaseType

ServerHost

ServerPort

ConnectionString

Get Database Schema Operator

Use the Get Database Schema operator to retrieve schema names from the database.

Input Parameters

The Database Server Login parameters (see page 147) are required for this operator.

Output Parameters

UserName

Password

DatabaseType

OtherDatabaseType

DriverName

DatabaseName

CacheBufferSize

BatchPerfWorkaround

ConnectionProtocol

ServerType

DriverType

ServiceID

TNSName

ClientEncryption

ClientChecksum

InstanceName

HypersonicDatabaseType

ServerHost

ServerPort

ConnectionString

Get Free Space Operator

Use the Get Free Space operator to return the free space (in MB) available in the database.

Input Parameters

Schema Name

Specifies the name of the schema for which free space must be calculated.

Output Parameters

Query Results

Returns free space (in MB).

Reason

Specifies the reason if the operator fails after execution.

Result

Specifies the result of the operator execution.

SchemaName

UserName

Password

DatabaseType

OtherDatabaseType

DriverName

DatabaseName

CacheBufferSize

BatchPerfWorkaround

ConnectionProtocol

ServerType

DriverType

ServiceID

TNSName

ClientEncryption

ClientChecksum

InstanceName

HypersonicDatabaseType

ServerHost

ServerPort

ConnectionString

When the required input parameters are provided, ConnectionString is automatically constructed.

Get Stored Procedure Operator

Use the Get Stored Procedure operator to return the stored procedure names available in the database.

Input Parameters

Catalog Name

Must match the same catalog name as it is stored in the database. The values ""
and null indicate that the catalog name should not be used to narrow the search.

Schema Pattern

Must match the same schema name as it is stored in the database. The values ""
and null indicate that the schema name should not be used to narrow the search.
The pattern should be a regular expression.

Procedure Name Pattern

Must match the same procedure name as it is stored in the database. The pattern
should be a database-supported regular expression.

Output Parameters

Query Results

Returns an array in which each row is a procedure name.

Reason

Specifies the reason if the operator fails after execution.

Result

Specifies the result of the executed operator.

CatalogName

SchemaPattern

TableNamePattern

UserName

Password

DatabaseType

OtherDatabaseType

DriverName

DatabaseName

CacheBufferSize

BatchPerfWorkaround

ConnectionProtocol

ServerTypeDriverType

ServiceID

TNSName

ClientEncryption

ClientChecksum

InstanceName

HypersonicDatabaseType

ServerHost

ServerPort

ConnectionString

Get Table Operator

Use the Get Table operator to return the list of tables from the database.

Input Parameters

Catalog Name

The catalog name must match the catalog name as it is stored in the database. The values "" and null indicate that the catalog name should not be used to narrow the search.

Schema Pattern

The schema pattern must match the schema name as it is stored in the database. The values "" and null indicate that the schema name should not be used to narrow the search. The pattern should be a regular expression.

Table Name Pattern

The table name pattern must match the table name as it is stored in the database. The pattern should be a database-supported regular expression.

Output Parameters

Query Results

Returns an array in which each row is a table name.

Reason

Specifies the reason if the operator fails after execution.

Result

Specifies the result of the executed operator.

CatalogName

SchemaPattern

TableNamePattern

UserName

Password

DatabaseType

OtherDatabaseType

DriverName

DatabaseName

CacheBufferSize

BatchPerfWorkaround

ConnectionProtocol

ServerType

DriverType

ServiceID

TNSName

ClientEncryption

ClientChecksum

InstanceName

HypersonicDatabaseType

ServerHost

ServerPort

ConnectionString

Get Used Space Operator

This Get Used Space operator returns the used space (in MB) in the database.

Input Parameters

Schema Name

Specifies the name of the schema for which the operation must return the used space.

Output Parameters

Query Results

Returns free space (in MB).

Reason

Specifies the reason if the operator fails after execution.

Result

Specifies the result of the operator execution.

SchemaName

UserName

Password

DatabaseType

OtherDatabaseType

DriverName

DatabaseName

CacheBufferSize

BatchPerfWorkaround

ConnectionProtocol

ServerType

DriverType

ServiceID

TNSName

ClientEncryption

ClientChecksum

InstanceName

HypersonicDatabaseType

ServerHost

ServerPort

ConnectionString

Get Version Operator

Use the Get Version operator to return the name and version number of the database.

Input Parameters

The Database Server Login parameters (see page 147) are required for this operator.

Output Parameters

UserName

Password

DatabaseType

OtherDatabaseType

DriverName

DatabaseName

CacheBufferSize

BatchPerfWorkaround

ConnectionProtocol

ServerType

DriverType

ServiceID

TNSName

ClientEncryption

ClientChecksum

InstanceName

HypersonicDatabaseType

ServerHost

ServerPort

ConnectionString

Get View Operator

Use the Get View operator to return a list of views from the database.

Input Parameters

Catalog Name

The catalog name must match the catalog name as it is stored in the database. The values "" and null indicate that the catalog name should not be used to narrow the search.

Schema Pattern

The schema pattern must match the schema name as it is stored in the database. The values "" and null indicate that the schema name should not be used to narrow the search. The pattern should be a regular expression.

View Name Pattern

The view name pattern must match the view name as it is stored in the database. The pattern should be a regular expression.

Output Parameters

Query Results

Returns an array in which each row is a view name.

Reason

Specifies the reason if the operator fails after execution.

Result

Specifies the result of the executed operator.

CatalogName

SchemaPattern

TableNamePattern

UserName

Password

DatabaseType

OtherDatabaseType

DriverName

DatabaseName

CacheBufferSize

BatchPerfWorkaround

ConnectionProtocol

ServerType

DriverType

ServiceID

TNSName

ClientEncryption

ClientChecksum

InstanceName

HypersonicDatabaseType

ServerHost

ServerPort

ConnectionString

Insert into Database Operator

Use the Insert into Database operator to insert a new row in a table.

Input Parameters

Input Source

Specifies that the user can choose to submit an SQL statement as an in-line expression (the default) or data variable. Select either Inline Text or Expression from the drop-down list.

Inline Text

Only available when Inline Text is selected as the input source. Specifies the generic SQL statement as inline text. Click the Inline Text field to open the Inline Text editor, where you can enter a literal SQL statement.

Expression

Only available when Expression is selected as the input source. Specifies the generic SQL statement as an expression. Use this field to provide a variable.

Input Parameters

An array of input values. If the generic SQL statement specified uses JDBC escape syntax and requires input parameters, they can be specified here.

Output Parameters

Query Results

Returns the number of rows inserted into a table.

Reason

Specifies the reason if the operator fails after execution.

Result

Specifies the result of the executed operator.

InputSource

InlineText

InputSourceExpression

JDBCInputParamArray

IsConstructSQLStatement

CompleteSQLStatement

DataSource

ColumnNameMode

ColumnNames

ColumnNameAsArray

ColumnValueMode

ColumnValues

ColumnValueAsArray

UserName

Password

DatabaseType

OtherDatabaseType

DriverName

DatabaseName

CacheBufferSize

BatchPerfWorkaround

ConnectionProtocol

ServerType

DriverType

ServiceID

TNSName

ClientEncryption

ClientChecksum

InstanceName

HypersonicDatabaseType

ServerHost

ServerPort

ConnectionString

Query Database Operator

Use the Query Database operator to issue a single SQL statement against the database. This operator supports the JDBC escape syntax. The Query Database operator uses CallableStatement and ParameterMetaData to gather information about the input and output parameters prior to and after statement execution. If the JDBC driver does not support this behavior, the Generic SQL operator may not be able to gather all of the results from the SQL statements. Additionally, the JDBC driver supplied by the database vendor may not support the use of all data types with the JDBC driver. For example, the SQL Server JDBC driver does not support the SQL Server sql_variant data type. Always refer to your JDBC driver documentation for more information on what JDBC features and data types are supported by the driver.

Input Parameters

Input Source

Specifies that the user can choose to submit SQL statement as an in-line expression (the default) or data variable. Select either Inline Text or Expression from the drop-down list.

Inline Text

Only available when Inline Text is selected as the Input Source. Specifies the generic SQL statement as inline text.

Click the Inline Text field to open the Inline Text editor, where you can enter a literal SQL statement.

Expression

Only available when Expression is selected as the Input Source. Specifies the generic SQL statement as an expression. Use this field to provide a variable.

Maximum rows to retrieve

Specifies the maximum number of rows to retrieve. If blank, the default value is 10 rows. The Generic SQL Operator retrieves a maximum of 512 rows. Additional rows will be truncated.

Input parameters

An array of input values. If the generic SQL statement specified uses JDBC escape syntax and requires input parameters, they can be specified here.

Display null values

If checked, the Generic SQL Operator results will contain the field NullFieldFlags. If the generic SQL statement returns results sets, this field can be used to distinguish null values from default values.

For example, querying an integer column that contains a null value returns 0. The NullFieldFlags value for this column would be true. However, if the table actually stored the value 0, then the NullFieldFlags value would be false.

Output Parameters

ResultsSets

An array of indexed ValueMaps where each item contains the results of a query. The size of ResultsSets matches the number of results sets returned by the SQL statement. Each ValueMap contains the following fields:

Rows

An array of ValueMaps representing the rows of the result set. Each ValueMap contains a field for each column and the value of the column in that particular row.

NullFieldFlags

An array of ValueMaps. The fields of each ValueMap correspond to the fields in rows. The value of each field is either true. The corresponding value in rows is null, or false otherwise. This output only displays if Display null values is selected.

UpdatedRowCounters

An array of integers representing the number of rows updated by the generic SQL statement. If the generic SQL statement performs multiple updates, then this value contains multiple values.

RowCount

(Deprecated) Returns either the row count for SQL Data Manipulation Language (DML) statements or the number of rows in the first result set in ResultSets. If the generic SQL statement performs no updates and returns no results sets, then this value will be set to -1. This field is included for backward compatibility.

OutputParam

The value of the output parameter of the generic SQL statement. If the generic SQL statement does not return any output parameters then this field will not be included in the operator results. The outputParam fields are numbered; for example: outputParam1, outputParam2.

isNullOutputParam

A Boolean value indicating if the corresponding OutputParam value is null. This field only displays if the generic SQL statement returns an output parameter and Display null values is selected.

Reason

Specifies the reason if the operator fails after execution.

Result

Specifies the result of the executed operator.

InputSource

InlineText

InputSourceExpression

MaximumRows

JDBCInputParamArray

JDBCReportNull

UserName

Password

DatabaseType

OtherDatabaseType

DriverName

DatabaseName

CacheBufferSize

BatchPerfWorkaround

ConnectionProtocol

ServerType

DriverType

ServiceID

TNSName

ClientEncryption

ClientChecksum

InstanceName

HypersonicDatabaseType

ServerHost

ServerPort

ConnectionString

Run a Stored Procedure

To run a stored procedure against a database, use the Query Database operator.

This example uses the SQL Server driver that CA Process Automation provides during installation. To run the process against another database, upload the corresponding JDBC driver from the Manage User Resources palette on the Configuration tab. CA Process Automation can access the driver when you restart the Orchestrator service. For more information, see the *Content Administrator Guide*.

You can duplicate this example in the SQL Server Management Studio in the PAMReporting database. PAMReporting is the database name that is provided for the CA Process Automation Reporting tables during the installation. You can use a different name. You can also use any database of your choice.

Follow these steps:

1. Create a stored procedure that contains the following body:

```
USE PAMReporting
GO
-- ==========================================
-- Template generated from Template Explorer using:
-- Create Procedure (New Menu).SQL
--
-- Use the Specify Values for Template Parameters
-- command (Ctrl-Shift-M) to fill in the parameter
-- values below.
--
-- This block of comments will not be included in
-- the definition of the procedure.
-- ==========================================
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE sp_getSOAPRows
        -- Add the parameters for the stored procedure here
AS
BEGIN
        -- SET NOCOUNT ON added to prevent extra result sets from
        -- interfering with SELECT statements.
        -- SET NOCOUNT ON;

    -- Insert statements for procedure here
            select count(*) from SOAPClientCall;
END
GO
```

2. The procedure returns the number of rows in a table that was called as a SOAPClientCall.

3. Click Parse in the SQL Server Management Studio.

4. Click Execute in the SQL Server Management Studio.

   CA Process Automation saves the new stored procedure in the PAMReporting database.

5. Create a process with the Query Database operator, then enter the following line in the Inline text for the operator:

   EXECUTE sp_getSOAPRows

6. In the Database Server Login (see page 147) parameters, enter relevant details that CA Process Automation requires to communicate with your database.

7. Save the process.

8. Run the process.

9. Open the operator dataset.

   The number of rows the procedure returns is displayed as a result of the process.

   Note: Your result depends on the number of rows in the SOAPClientCall table.

Select from Database Operator

Use the Select from Database operator to selectively retrieve data from one or more data sources with optional selection criteria.

Input Parameters

Input Source

Specifies that the user can choose to submit SQL statement as an in-line expression (the default) or data variable. Select either Inline Text or Expression from the drop-down list.

Inline text

Only available when Inline Text is selected as the Input Source. Specifies the generic SQL statement as inline text. Click the Inline Text field to open the Inline Text editor, where you can enter a literal SQL statement.

Expression

Only available when Expression is selected as the Input Source. Specifies the generic SQL statement as an expression. Use this field to provide a variable.

Maximum rows to retrieve

Specifies the maximum number of rows to be retrieved by the select statement. This parameter overrides the property set at the operator category level.

Input parameters

An array of input values. If the generic SQL statement specified uses JDBC escape syntax and requires input parameters, they can be specified here.

Display null values

If checked, the Generic SQL Operator results contain the field NullFieldFlags. If the generic SQL statement returns results sets, this field can be used to distinguish null values from default values.

For example, querying an integer column that contains a null value returns 0. The NullFieldFlags value for this column is true. However, if the table actually stored the value 0, then the NullFieldFlags value are false.

Output Parameters

QueryResults

An array of ValueMaps representing the rows of the result set. Each ValueMap contains a field for each column and the value of the column in that particular row.

NullFieldFlags

An array of ValueMaps. The fields of each ValueMap correspond to the fields in rows. The value of each field is either true (the corresponding value in rows is null) or false (the corresponding field is not null). This output only displays if Display null values is selected.

InputSource

InlineText

InputSourceExpression

MaximumRows

JDBCInputParamArray

JDBCReportNull

IsConstructSQLStatement

CompleteSQLStatement

ReturnValueMode ReturnValues

ReturnValuesAsArray

DataSourceMode

DataSources

DataSourcesAsArray

SelectionCriteria

SortCriteriaMode

SortCriteria

SortCriteriaAsArray

UserName Password

DatabaseType

OtherDatabaseType

DriverName

DatabaseName

CacheBufferSize

BatchPerfWorkaround

ConnectionProtocol

ServerType

DriverType

ServiceID

TNSName

ClientEncryption

ClientChecksum

InstanceName

HypersonicDatabaseType

ServerHost

ServerPort

ConnectionString

Update in Database Operator

Use the Update in Database operator to update records in a table.

Input Parameters

Input Source

Specifies that the user can choose to submit an SQL statement as an in-line expression (the default) or data variable. Select either Inline Text or Expression from the drop-down list.

Inline text

Only available when Inline Text is selected as the Input Source. Specifies the generic SQL statement as inline text. Click the Inline Text field to open the Inline Text editor, where you can enter a literal SQL statement.

Expression

Only available when Expression is selected as the Input Source. Specifies the generic SQL statement as an expression. Use this field to provide a variable.

Input parameters

An array of input values. If the generic SQL statement specified uses JDBC escape syntax and requires input parameters, they can be specified here.

Output Parameters

Query Results

Returns the number of rows updated.

Reason

Specifies the reason if the operator fails after execution.

Result

Specifies the results of the executed operator.

InputSource

InlineText

InputSourceExpression

JDBCInputParamArray

IsConstructSQLStatement

CompleteSQLStatement

FieldValueModeArray

FieldValueModeC2OValueMap

DataSource

FieldsValues

FieldsAsArray

ValuesAsArray

FieldValueMap

SelectionCriteria

UserName

Password

DatabaseType

OtherDatabaseType

DriverName

DatabaseName

CacheBufferSize

BatchPerfWorkaround

ConnectionProtocol

ServerType

DriverType

ServiceID

TNSName

ClientEncryption

ClientChecksum

InstanceName

HypersonicDatabaseType

ServerHost

ServerPort

ConnectionString

Chapter 6: Date-Time

The Date-Time operators execute time and calendar constraints in processes. All operators in this group can only be run on an Orchestrator.

Check Calendar Operator

The Check Calendar operator determines whether a date falls within a set of calendar rules.
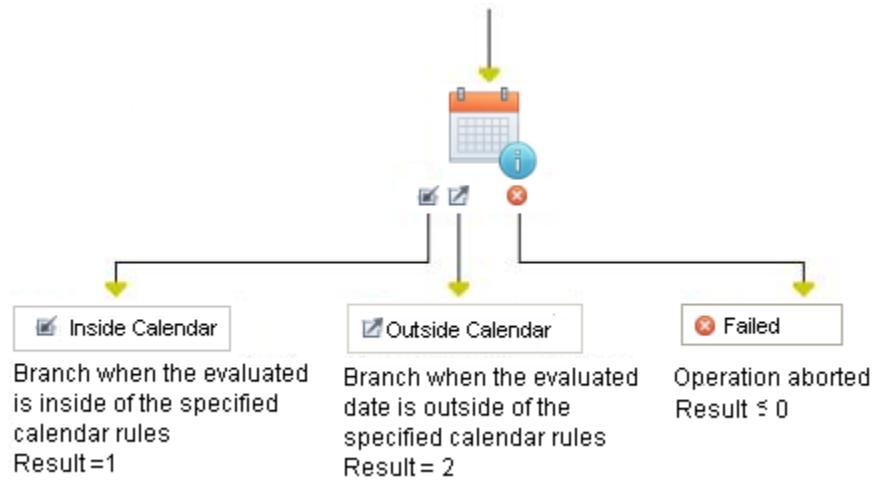
Input Parameters

Allow dates Calendar

Specifies the full path of the calendar. This expression defines the allowed dates for subsequent branches in the process to be processed.

Click ![search icon] to locate a calendar object. After you select a calendar, click Open to open the calendar object in the calendar designer.

Exclude dates Calendar

Specifies the full path of the calendar used to define excluded days. Excluded days are days when subsequent branches in the process may not be processed.

There are no excluded days when no calendar is specified.

Click ![search icon] to locate a calendar object. After you select a calendar, click Open to open the calendar object in the calendar designer.

Delta

Specifies the number of days an allowed date is shifted when it falls on an excluded date.

The shift depends on whether the value is positive, negative, or zero. A negative value shifts forward (earlier), and a positive value shifts backward (later). When this value is zero, the allowed date is skipped.

Open days only

Counts only open days when shifting an eligible date that falls on an excluded date.

Open days are those days not specified by a condition or rule that closes or excludes dates.

Maximum shifts

This option defines the maximum number of shifts that are allowed if subsequent shifts fall on a closed date.

This setting is only relevant if Open days only is selected.

Date

Specifies the date to test against the calendar rules.

Click the calendar icon to select a date. This option is unavailable if you select either the Use current date or the Use calculated date check box.

Use current date

Specifies to use the current date to test against the calendar rules.

Clear this check box to specify a particular date in the Date field. This check box is initially selected.

Use calculated date

Specifies to use the date from the Calculated date field.

Calculated date

If Use calculated date is selected, this parameter returns a date. Typical uses include calculating a future date based on the current date.

Output Parameters

CalenderDate

VacationsDate

Delta

OpenDays

MaxShifts

Date

UseCurrentDate

UseCalculatedDate

CalculatedDate

Check Date-Time Operator

The Check-Date Time operator conditionally executes branches in a process depending on whether a specified date and time has passed. The Check-Date Time operator places a date-time check condition in a process.

A date-time check condition allows processing to continue to subsequent branches in a process before and after the date specified in the operator properties. The operator can be used to place date and time conditions on different segments of processes that are run several times a day. With this setting, you can add extra links on the operator to specify branches that are to be processed before (<) or after (>) a date and time.

In contrast, when the Wait for specified date and time check box is initially selected, the operator creates a date-time wait condition. Then, the operator only processes after (>) extra links while ignoring any before (<) extra links.

Input Parameters

Date

Specifies the date when to determine eligibility for processing subsequent branches in the process.

Click the Calendar icon to open the calendar and select a date. This option is unavailable if you select either the Use current date or the Use calculated date check box.

Use current date

Specifies the date that the process is run to determine eligibility for processing subsequent branches in the process.

Clear this check box to specify a particular date in the Date field. This check box is initially selected.

Use calculated date

Returns a date. This expression lets you use the CA Process Automation date variables and functions to return a date.

Calculated date

If Use calculated date is selected, this parameter specifies an expression that returns a date. Typical uses include calculating a future date based on the current date.

Time

Specifies a time in a 12-hour HH:MM PM/AM format.

For example: 07:30 PM

Wait for specified date and time

Creates a date-time wait condition. This property delays processing of subsequent operators in a branch of the process until the specified time. Only exit links designated to occur after (>) the specified time are processed.

Clear this check box to divert processing to different branches before or after the specified date and time. The operator imposes the following conditions when the Wait for specified date and time check box is cleared:

If the specified date and time are in the future, only exit links specified to be processed *before* (<) are processed.

Output Parameters

Date

UseCurrentDate

UseCalculatedDate

CalculatedDate

Time

WaitForSpecifiedDate

Chapter 7: Directory Services

The Directory Services operators provide an interface to support *Lightweight Directory Access Protocol* (LDAP). You can automate the operations that are performed on LDAP servers. All of these operators work with different LDAP servers except for operators specific to the Active Directory. The Directory Services operators run on a CA Process Automation Orchestrator or agent with the same results, regardless of the operating system platform on which CA Process Automation is running.

LDAP Login Parameters

The default LDAP fields that are specified at the Directory Services category level can be overridden on the LDAP Login Parameters page. This page is part of the input for each Directory Services operator. If a field contains a value, it overrides the value that is specified for the same field at the category configuration level.

Remote LDAP Host

Specifies the LDAP Server URL or IP.

Remote LDAP Server Port

Specifies the LDAP Server Port.

LDAP User

Specifies that the LDAP User who has access to the LDAP server can log in. However, the operations that the user can perform are limited by the ACIs set on the LDAP entries.

LDAP Password for User

Specifies the password for the LDAP user.

Base DN

Specifies the base Distinguished Name (DN) to be used. This is the base DN where the LDAP User is located.

User Prefix

Specifies the user prefix to use; either userid (uid) or commonName (cn).

Important! If you access Active Directory through LDAP, set this field to cn. Otherwise the connection string uses uid, which causes an error.

For example, assume that you need to use the administrator credentials for this operator.

The LDAP connection string would be:

`cn=Administrator,cn=Users,dc=RBC,dc=com`

If you do *not* set the User Prefix to cn, the connection string stays as:

`uid=administrator,cn=users,dc=rbc,dc=com`

...and an error occurs.

Set the LDAP connection string as:

`User Prefix" = "LDAP User" + "," + "Base DN`

Where User Prefix, LDAP User, and Base DN are the values of the corresponding fields that are specified under LDAP Login Parameters.

Note: To get the User Prefix from Active Directory, follow these steps:

1.   Look up the user DN.

2.   Extract the values for the prefix, user, and the Base DN (as previously shown).

3.   Specify them in the three fields under LDAP Login Parameters .

Add Computer to Domain Operator

The Add Computer to Domain operator to create a new computer object in the Active Directory server. This operator applies to an Active Directory server only.

Input Parameters

Computer Path

Specifies the distinguished name of the object under which you want to create the new computer object.

Computer Name

Specifies the name of the new computer object.

Output Parameters

LDAPADComputerBaseDn

LDAPADComputerName

remoteLDAPHost

remoteLDAPPort

remoteLDAPUser

remoteLDAPPassword

LDAPBaseDN

LDAPUserPrefix

Example

Add the LDAP attribute to the name of the new computer object. Computer objects typically use the attribute "cn" as part of the relative distinguished name (RDN) of the Computer Name.

Operator Failure

This operator fails in the following cases:

- The name of the new computer object is already used.

- Some of the mandatory attributes necessary to create the new computer object is missing

- The path, under which computer object is to be created, is invalid.

- Unable to connect to the Active Directory server.

Add User to Group Operator

The Add User to Group operator adds an LDAP user to an LDAP group on the LDAP server.

Input Parameters

User DN

Specifies the distinguished name of the user that you want to add to the group.

Group DN

Specifies the distinguished name of the group to which you want to add the user.

Output Parameters

LDAPUserDn

LDAPGroupDn

remoteLDAPHost

remoteLDAPPort

remoteLDAPUser

remoteLDAPPassword

LDAPBaseDN

LDAPUserPrefix

Example

Operator Failure

This operator fails in the following cases:

- The user is already a member of the group.

- The group does not exist in the LDAP server.

- The Group DN points to an LDAP object that is not of type group, groupofnames or groupofuniquenames.

- Unable to connect to the LDAP server.

Create Group Operator

The Create Group operator creates an LDAP group object on the LDAP server. This group object can be of type: group, group of names, or group of unique names.

Input Parameters

Group Path

Specifies the distinguished name of the object under which you want to create the new group object.

Group Name

Specifies the name of the group you want to create. This is the CN attribute of the group.

LDAP Group Type

Specifies the type of LDAP group. Select either group, group of names, or group of unique names.

Use the specified array field for the Group Members

When checked, then the array of members is used for this request.

LDAP Group Members Array

Specifies the array of members of the group (required for Group of Names or Group of Unique Names in Active Directory). This field is enabled only when the Use the specified array field for the Group Members field is checked.

LDAP Group Members

Specifies the members of the group (required for Group of Names or Group of Unique Names in Active Directory). This field is enabled only when the Use the specified array field for the Group Members field is unchecked.

Creating an object of type 'Group' in Active Directory?

Check if we are creating an object of type 'Group', as specified in the LDAP Group Type, in an Active Directory server. For an object "Group" in Active Directory, we can set two extra attributes: Group Scope and Group Type.

Active Directory Group Scope

Specifies the scope of the group that is created in an Active Directory. Select either Domain Local, Global, or Universal. This field is enabled only when you select the Creating an object of type 'Group' in Active Directory? check box.

Active Directory Group Type

Specifies the type of the group that is created in an Active Directory. Select either Security or Distribution. This field is enabled only when the Creating an object of type 'Group' in Active Directory? field is checked.

Notes

- The Active Directory does not allow for a group of Universal scope to be of type Security. The operation fails in this case.

- The values that are specified in the Active Directory Group Scope and Active Directory Group Type fields are ignored when the LDAP Group Type field is not set to the value "Group".

- AD Group Scope and AD Group Type fields are not ignored when LDAP Group Type is not set to value "Group". Those fields are activated when you select the check box Creating an object of type Group in AD (because there is no way to perform multiple selections in CA Process Automation). The important thing is that the values of these fields are only relevant when you create a "Group" in Active Directory. In the UI, they might still be active but, in the back end, they are ignored in all other cases.

Output Parameters

LDAPGroupBaseDn

LDAPGroupName

LDAPGroupType

LDAPGroupMembersType

LDAPGroupMembersArray

LDAPGroupMembers

LDAPIsADGroup

LDAPADGroupScope

LDAPADGroupType

remoteLDAPHost

remoteLDAPPort

remoteLDAPUser

remoteLDAPPassword

LDAPBaseDN

LDAPUserPrefix

Example

Operator Failure

This operator fails in the following cases:

- The group exists already.

- Some of the mandatory attributes necessary to create the new group object are missing.

- The path, under which the new group object is to be created, is invalid.

- The LDAP server does not support the group type specified: group, groupofnames, or groupofuniquenames.

- The group is created in an Active Directory server with a 'Universal' scope and a 'Security' type.

- The object is created as a 'Group of Names' in Active Directory without any members set in the operation.

- The object is created as a 'Group of unique Names' in Active Directory without any members set in the operation.

- Unable to connect to the LDAP server.

Create Object Operator

The Create Object operator creates an LDAP object of any type on the LDAP server.

Input Parameters

Object Path

Specifies the distinguished name of the object under which you want to create the new LDAP object.

Object Name

Specifies the name of the new LDAP object.

Be sure to add the LDAP attribute to the name of the new LDAP object. The attribute could be "ou", "cn", "uid", and so on, and depends on the type of LDAP object being created.

Use the specified array field for the Object's "objectclass" Attributes Values

If checked, the "objectclass" Attribute Values Array will be used for this request.

Object's "objectclass" Attribute Values Array

Specifies the array containing the values of the "objectclass" Attribute. This dataset field must be defined as an array (indexed string). If Use the specified array field for Object's "objectclass" Attributes Values is checked, this field will be used.

Object's "objectclass" Attribute Values

Specifies the values of the "objectclass" Attribute. If the Use the specified array field for Object's "objectclass" Attributes Values is unchecked, this field will be used.

The "objectclass" is the LDAP attribute that defines the type of the new object.

Additional Object's LDAP Attributes Value Maps

This is an array of value maps containing additional LDAP attributes to be set for the new object. Each value map's Key must be of type string, Value must be of type string or array of strings (indexed string). The key must be named Keys and the value must be named Values.

The user can set the Values field to be of type string to create single-valued LDAP attributes for the new LDAP object being created. For example:

The object newObjAttributes is an indexed ValueMap whose key fields are called Keys and are of type string and value fields are called Values and are of type string.

Alternatively, the user can set the Values field to be of type array of strings (indexed string) to create multi-valued LDAP attributes for the new LDAP object being created.

The object newObjAttributes2 is an indexed ValueMap whose key fields are called Keys and are of type string and value fields are called Values and are of type indexed string. In this case the user can create both single-valued and multi-valued LDAP attributes for the new LDAP object being created.

For example:

Within the same newObjAttributes2 object, we have a multi-valued telephonenumber attribute and also a single-valued description attribute.

Note that if the same key appears multiple times within the indexed ValueMap, only the last value associated with the key will remain.

Note that the attribute names entered in the Additional Object's LDAP Attributes Value Maps must be the LDAP names of these attributes as specified in the LDAP server schema. For instance, to set the value of the attribute "Last name" you must use the LDAP name of this attribute: "sn", to set the value of the attribute "First Name", you must use the attribute "givenname", and so on. See Common LDAP Attribute Names (see page 221).

The LDAP names are different from the attributes display names.

Most LDAP servers differ in the display names of the LDAP attributes, but they all must support the LDAP names of these attributes, thus the reason why we require the usage of the LDAP names of the attributes instead of the display names.

Output Parameters

LDAPCreateObjectBaseDn

LDAPCreateObjectName

LDAPCreateObjObjectClassUseArray

LDAPCreateObjObjectClassArray

LDAPCreateObjObjectClass

LDAPCreateObjectAttributes

remoteLDAPHost

remoteLDAPPort

remoteLDAPUser

remoteLDAPPassword

LDAPBaseDN

LDAPUserPrefix

Example

In this example, we are creating an organizational unit called Testing Unit. The objectclass attribute defines the object to be of type 'top' and 'organizationalunit'. The 'top' type is the root of all LDAP types.

We are also adding additional attributes to the new organizational unit through the Process.newObjAttributes indexed value map.
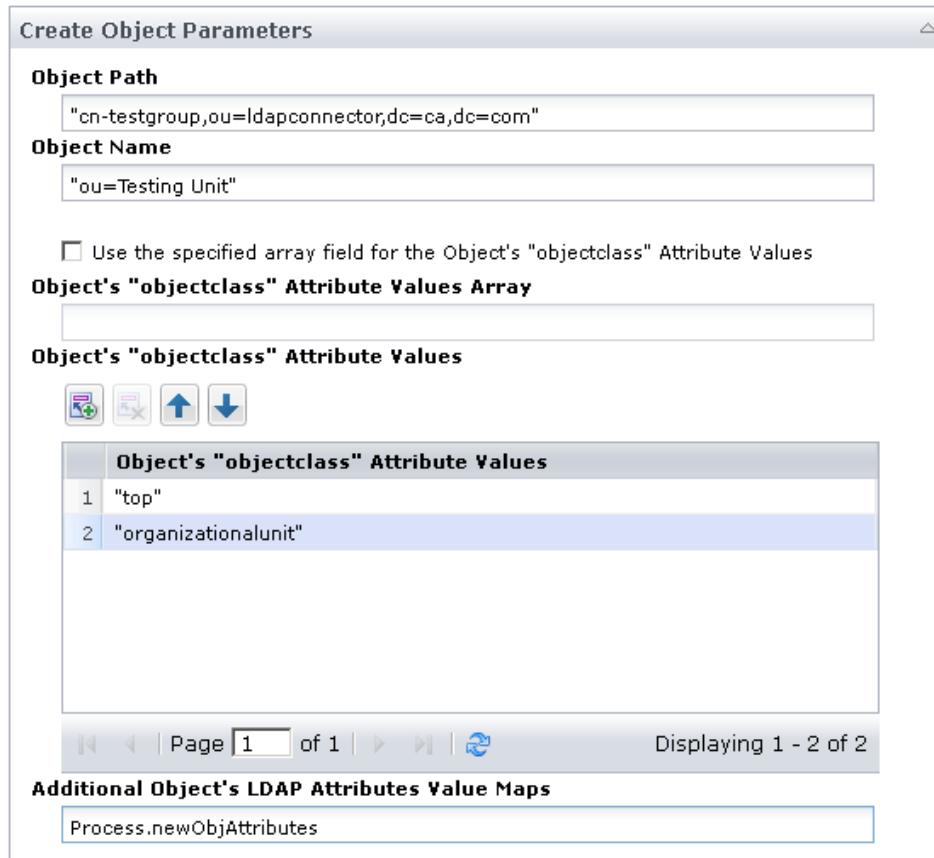
Operator Failure

This operator fails in the following cases:

- The name of the new LDAP object is already used.

- Some of the mandatory attributes necessary to create the new LDAP object are missing.

- The "objectclass" of the new LDAP object is missing or incorrect.

- Some of the attributes being created for the object contain invalid values.

- Some of the attributes being created for the object do not apply to this type of object, for instance, you cannot add a 'mail' attribute to an LDAP object of type Organizational Unit.

- The path, under which the LDAP object is to be created, is invalid.

■ The user checked that an array of attributes is used for the "objectclass" attribute, but the CA Process Automation object entered in the array field is actually not of type array (indexed strings).

■ Unable to connect to the LDAP server.

Create Organizational Unit Operator

The Create Organizational Unit operator allows a user to create an LDAP object of type Organizational Unit on the LDAP server.

Input Parameters

Organizational Unit Path

Specifies the distinguished name of the object under which to create the new organizational unit object.

Organizational Unit Name

Specifies the name of the new organizational unit object.

Output Parameters

LDAPOrgUnitBaseDn

LDAPOrgUnitName

remoteLDAPHost

remoteLDAPPort

remoteLDAPUser

remoteLDAPPassword

LDAPBaseDN

LDAPUserPrefix

Example

Example

Important! Make sure to add the LDAP attribute to the name of the new organizational unit. Organizational units usually use the attribute "ou" as part of the name's RDN (*relative distinguished name*).

Operator Failure

This operator fails in the following cases:

- The name of the new organizational unit is already used.

- Some of the mandatory attributes necessary to create the new organizational unit is missing.

- The path, under which the organizational unit is to be created, is invalid.

- Unable to connect to the LDAP server.

Create User Operator

The Create User operator creates an LDAP object of type user account on the LDAP server.

Input Parameters

User Account Path

Specifies the distinguished name of the object under which you want to create the new user account.

First Name

Specifies the first name of the user.

Middle initials

Specifies the middle initials of the user. Note that Active Directory does not allow middle initials to be over six characters long.

Last name

Specifies the last name of the user.

UserID

Specifies the user ID for the user.

Password

Make sure you specify a password that conforms to the Password Policy Requirements set in your LDAP server, especially for an Active Directory Server.

Active Directory?

Check if you are creating the new user account in an Active Directory server.

Create UserID as User Logon Name

Specifies whether you want the UserID to also be the user's logon name. In this case, create a User Logon Name of the form "UserID@domain" where domain represents the Active Directory's Domain. This field is enabled only when the Active Directory? field is checked.

Enable user?

Select yes to make the new user active or no to make the new user inactive. This field is enabled only when the Active Directory? field is checked.

Password Expires for User

Select whether the user password expires as per the Domain Policy or Never expires. This field is enabled only when the Active Directory? field is checked.

When the user password is set to never expire, you are not be forced to change a password at the first logon.

Password Change at First Login?

Forces the User to change password at first logon, this option is applicable only when the password is chosen to expire. Note that the user could set the combination Password to *not* expire and Password change required at first login. In this case, CA Process Automation sets the password to *not* expire and ignores the password change at first login request. This field is enabled only when the Active Directory? field is checked.

Note that Active Directory will not allow for a user's password to be set unless CA Process Automation is connected to the Active Directory server through SSL. If CA Process Automation is not connected through SSL, the user account will be created without a password and without the specified account controls (account enabled/disabled, password expiration, password change at logon) and the operation will fail in CA Process Automation.

See the topic Add an SSL Certificate to CA Process Automation (see page 246) to find out how to import an Active Directory certificate to CA Process Automation. Once the certificate is imported, you can change the Directory Services category's properties to establish an SSL connection with the Active Directory server.

Output Parameters

LDAPUserBaseDn

LDAPUserFirstName

LDAPUserMiddleInits

LDAPUserLastName

LDAPUserId

LDAPUserPwd

LDAPIsAD

LDAPAsUserLogon

LDAPEnableUser

LDAPPwdExpire

LDAPForcePwdChg

remoteLDAPHost

remoteLDAPPort

remoteLDAPUser

remoteLDAPPassword

LDAPBaseDN

LDAPUserPrefix

Example


```
Create User Account Parameters                                    △

User Account Path
    "cn=Users,ou=ldapconnector,dc=ca,dc=com"
First Name
    "John"
Middle Initials
    "Michael"
Last Name
    "Smith"
UserID
    "jsmith"
Password
    "jsmithpwd"


    ☑ Active Directory?
Create UserID as User Logon Name
    Use Specified Domain Controller                              ▼
Enable User?
    Yes                                                          ▼
Password Expires for User
    Never                                                        ▼
Password Change at First Login?
    No                                                           ▼
```

Operator Failure

This operator fails in the following cases:

- The user exists already.

- Some of the mandatory attributes necessary to create the new user account are missing.

- The value of one of the attributes used to create the user account is invalid.

- The path, under which the new user account is to be created, is invalid.

- The new user account is created in an Active Directory server and not connected through SSL.

- Unable to connect to the LDAP server.

- You specified a user password that does not conform to the Password Policy Requirements set in your LDAP server, especially if it is for an Active Directory. In this case, Active Directory returns a generic error message: WILL_NOT_PERFORM to indicate that it cannot perform the operation.

Delete Object Operator

The Delete Object operator deletes a single LDAP object or multiple LDAP objects from the LDAP server.

Input Parameters

Use the specified array field for the Objects Distinguished Names

If this is checked, then the Array of Distinguished Names will be used for this request.

Objects Distinguished Names Array

Specifies the array of DNs of the objects you want to delete. This field is enabled only when the Use the specified array field for the Objects Distinguished Names field is checked.

Objects Distinguished Names

Specifies the DNs of the objects you want to delete. This window is enabled only when the Use the specified array field for the Objects Distinguished Names field is unchecked.

Objects Deletion Scope

Select one of the following:

Delete object (return an error if children exist)

To attempt to delete each object (from the list of objects in the Objects Distinguished Names Array or Objects Distinguished Names fields) as if it does not have a subtree under it in the LDAP tree. If a subtree exists for an object in the list, then CA Process Automation will fail the operation; but will also continue to delete all the other objects in the list of objects to be deleted.

Delete object and subtree (if exists)

To attempt to delete each object (from the list of objects in the Objects Distinguished Names Array or Objects Distinguished Names fields) and the entire subtree under it if such a subtree exists.

Output Parameters

NumberOfObjectsToDelete

Specifies the number of objects found to be deleted.

■   If the Objects Deletion Scope is set to Delete object and subtree then this variable will return the number of all the objects found in the subtrees as well.

■   If the Objects Deletion Scope is set to Delete object then this variable will return the number of objects set in the operation.

NumberOfDeletedObjects

Contains the number of objects actually deleted.

DeletionFailures

Specifies an array of value maps created only when the operation fails. In this case, this array of value maps will contain the DNs of the objects that were not deleted along with error messages indicating why each object was not deleted.

Note that the delete operation will succeed when trying to delete an object that does not exist in the LDAP server.

LDAPDeleteObjsUseArray

LDAPDeleteObjsArray

LDAPDeleteObjs

LDAPDeleteObjectsScope

remoteLDAPHost

remoteLDAPPort

remoteLDAPUser

remoteLDAPPassword

LDAPBaseDN

LDAPUserPrefix

Examples

Examples of both a successful and failed deletion are provided here.

Operator Failure

This operator fails in the following cases:

- The operator was unable to delete any of the objects entered in the operation.
- Unable to connect to the LDAP server.

Failed Deletion

Delete a single object and set the operation to fail if any child objects (subtree) exist under this object.
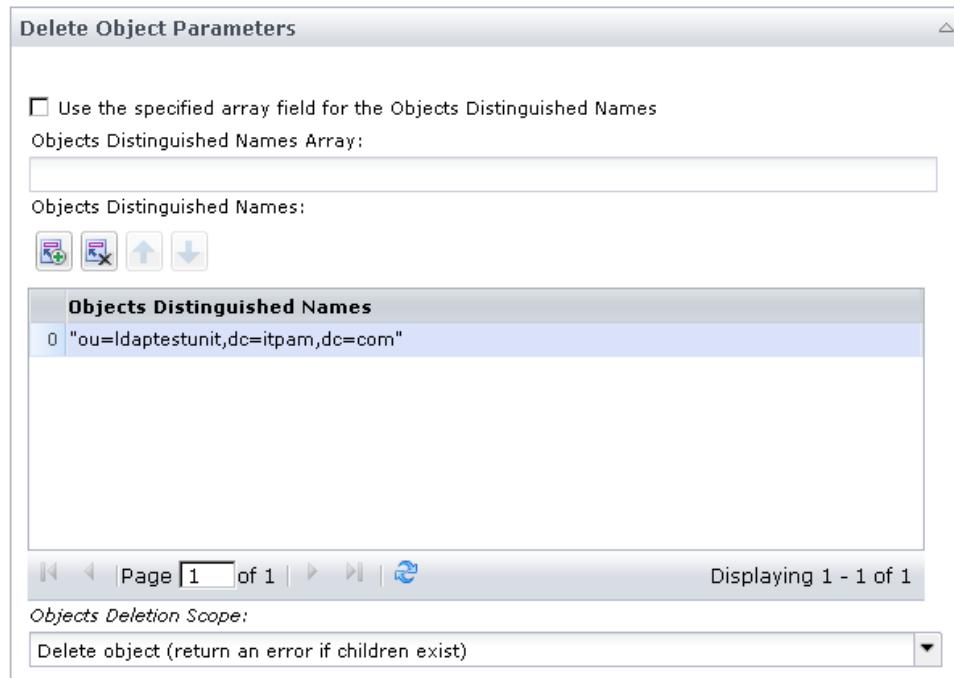
This is the operator dataset of a runtime instance:

DeletionFailures	[1]
▷ Element Type	
[0]	
Row0	
DN	ou=ldaptestunit,dc=itpam,dc=com
Reason	[LDAP: error code 66 - Not Allowed On Non-leaf]

After the operator runs, it fails and following data displays on the Operation Results tab.

NumberOfObjectsToDelete

1 (as we only attempted to delete one object).

NumberOfDeletedObjects

0 (as we were unable to delete the object).

DeletionFailures

Specifies an array of ValueMaps with a single object that contains the DN of the object you tried to delete, and the LDAP error message stating why it was not deleted.

DeletionFailures	[1]
▷ Element Type	
[0]	
Row0	
DN	ou=ldaptestunit,dc=itpam,dc=com
Reason	[LDAP: error code 66 - Not Allowed On Non-leaf]

The LDAP message indicates that this object has a subtree under it (it is a non-leaf).

Another failed deletion

Suppose you want to delete three objects within the same operation: two objects do not have subtrees under them in the LDAP tree, and one object has a subtree under it.
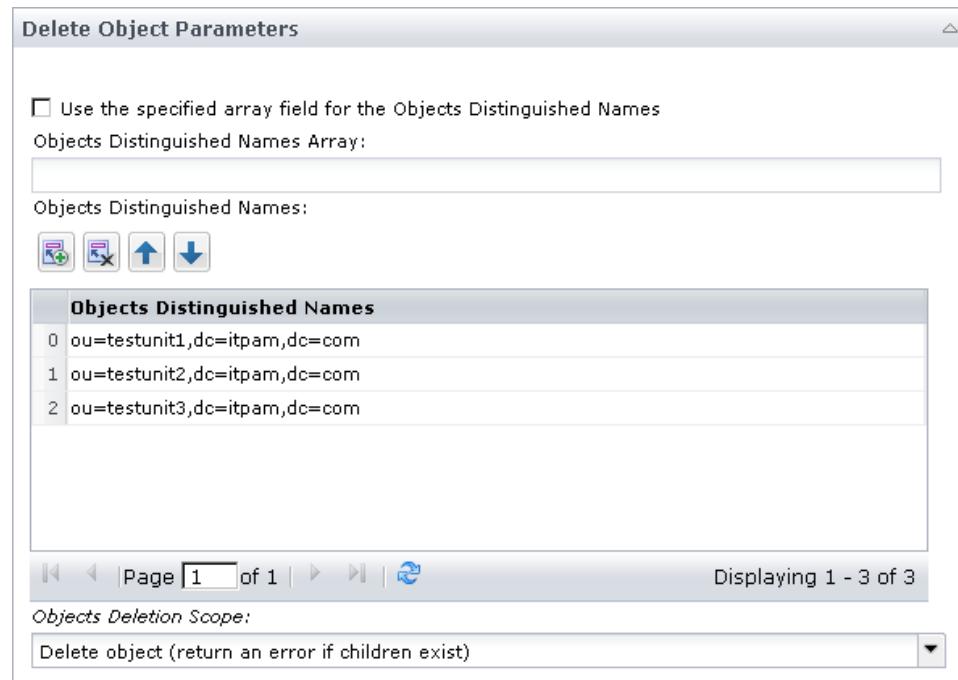
In this example, the first and second objects do not have any subtrees under them. The third object in the list has a subtree under it.

Note that you specified to delete the object (and return an error if children exist).

After the operation ran, it failed and the Operation Results page contains the following data:

NumberOfObjectsToDelete

> 3 (as we attempted to delete 3 objects)

NumberOfDeletedObjects

> 2 (as we were able to delete only two objects)

DeletionFailures

Array of ValueMaps with a single object that contains:

■   The DN of the object that we were unable to delete

■   The LDAP error message stating why it was not deleted

The LDAP message indicates that this object has a subtree under it (it is a non-leaf).

Keep in mind that the delete operation searches the entire list of objects to be deleted. If an object fails to be deleted, the operation continues deleting all other objects in the list, but the operation will also fail when it is over.

Operator Success

Attempt to delete the same object and all its children (subtree under it):
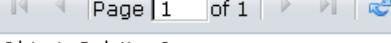
After the operation ran, the Operation Results show the following data:

NumberOfObjectsToDelete: 25 (as the object had 24 child objects in the subtree under it).

NumberOfDeletedObjects: 25 (as we were able to delete the object and all the child objects in the subtree under it).
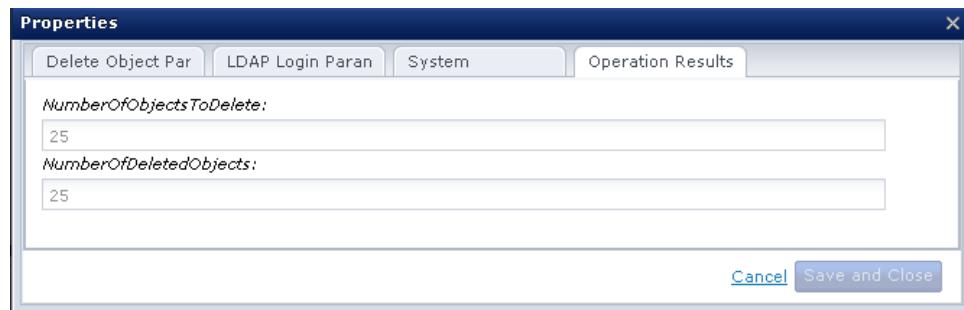
Notice the DeletionFailures variable was not created.

If you had attempted to delete more than one object in the previous examples, NumberOfObjectsToDelete and NumberOfDeletedObjects will represent the sum of all 'objects deleted' and 'objects to be deleted' for all objects entered in the operation (including all their subtrees, if applicable).

Get Domain Controller Operator

The Get Domain Controller operator retrieves all domain controllers from the Active Directory server. This operator applies to Active Directory only.

Input Parameters

The Get Domain Controller operator does not include any input parameters. The operator simply retrieves the Active Directory server information from the LDAP Login Parameters page associated with the operation or from the default LDAP Login information set at the Directory Services category level.

Output Parameters

DomainControllers

Specifies an array of strings (indexed string) that contains all the domain controllers retrieved from the Active Directory server. This variable is created if the operation succeeds.

remoteLDAPHost

remoteLDAPPort

remoteLDAPUser

remoteLDAPPassword

LDAPBaseDN

LDAPUserPrefix

Example

This example shows a successful instance of the operator retrieving the DomainControllers from an Active Directory server. The operator succeeds and the DomainControllers variable is created with the following data:

▲ 🗁 *Operation Results*	
▲ 🗁 *DomainControllers*	[1]
[0]	SSA-ITC-AD.ssaitc.com

Properties ✕

LDAP Login Paran	System	Operation Results

DomainControllers:

DomainControllers

0 SSA-ITC-AD.ssaitc.com

◀◀ ◀ Page 1 of 1 ▶ ▶▶ 🔁   Displaying 1 - 1 of 1

Cancel Save and Close
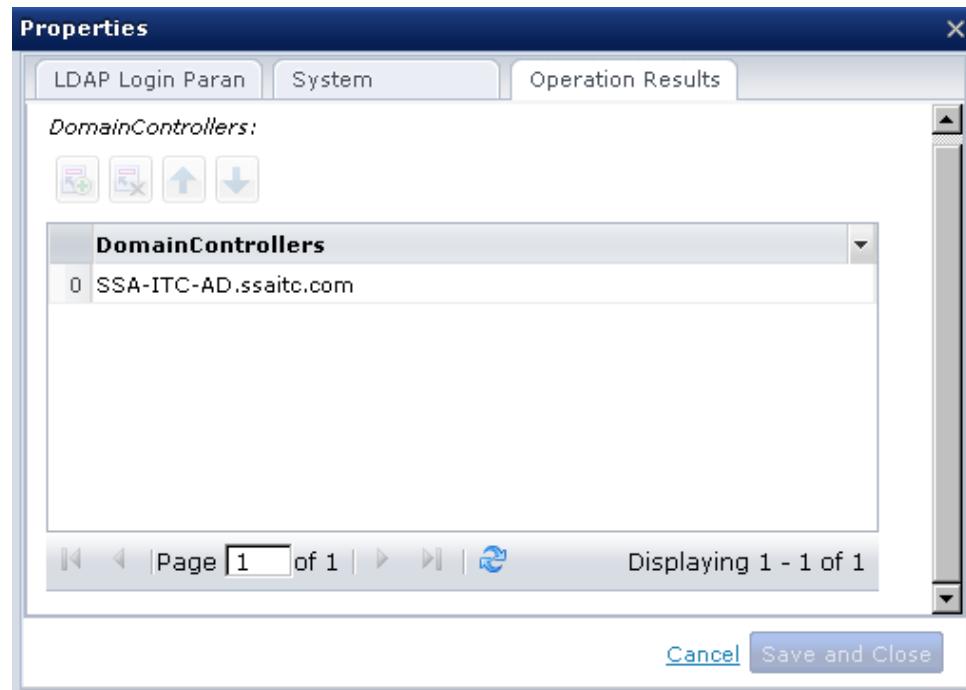
In this example, we now have a single domain controller in this Domain.

Operator Failure

This operator fails in the following cases:

- CA Process Automation is unable to retrieve the configurationNamingContext from the Active Directory server.

- No objects of type TDSDSA exist in the Active Directory server.

- Unable to connect to the LDAP server.

Get Dormant Account Operator

Use the Get Dormant Account operator to retrieve all dormant accounts from the Active Directory. You can specify a date and number of days as a dormant range, any user account whose last logon date falls earlier than this range is considered dormant. This operator applies to Active Directory only.

Input Parameters

Days Dormant

Specifies the number of dormant days

Date

Specifies the designated date for dormant accounts. This field is enabled when the Use Calculated Date? is unchecked. Click the calendar icon to select a date.

Use Calculated Date?

Specifies that the user can supply a dataset variable that contains a date value in the Calculated Date field.

Calculated Date

Specifies the calculated date for dormant accounts. This field is enabled when the Use Calculated Date? is checked.

Last Logon Attribute

LastLogonTimeStamp

Select this field when retrieving dormant accounts from Active Directory 2003/2008 (NOT Active Directory 2000).

The LastLogonTimeStamp attribute contains the last logon date of a user; but it is replicated across all Domain controllers only after a period of time defined in the msDS-LogonTimeSyncInterval attribute of the Active Directory.

If the dormant date falls before "today's date - msDS-LogonTimeSyncInterval", then using the lastLogonTimestamp retrieved from a single Domain controller retrieves all the dormant accounts.

The msDS-LogonTimeSyncInterval attribute specifies the frequency (in days) with which the last logon time for a user/computer, recorded in the lastLogonTimestamp attribute, is replicated to all Domain Controllers in a Domain.

When using LastLogonTimeStamp, CA Process Automation retrieves the LastLogonTimeStamp information for each user from only one Domain controller and uses it to determine if the user is dormant or not.

LastLogon

Select this field when retrieving dormant accounts from Active Directory 2000, 2003, and 2008.

The LastLogon attribute contains the last logon date of a user but it is NOT replicated across all domain controllers.

In this case, CA Process Automation begins to retrieve the list of all domain controllers, then review them to save the most recent lastLogon attribute value for each user. Finally, the most recent lastLogon value for each user is used to determine the dormant accounts.

Output Parameters

DormantAccounts

Specifies an array of value maps, where each value map represents a dormant account. Each value map contains the following keys/values:

DN

Specifies the distinguished name of the dormant user account.

Last Logon Date

Specifies the last logon date of the dormant user account.

Dormant Days

Specifies the number of days between the dormant user account's last logon date and the dormant date set in the operation. Note that this number is rounded up, for instance if a user's last logon date is May 15, 2009 12:25:49 PM and we have a dormant date of: May 18, 2009, then the Dormant Days will be 4 (not 3).

The Last Logon Date and Dormant Days will both be set to -1 for each dormant user account that's never logged in before.

NoLogonParameterAccounts

Specifies an array of strings (indexed string) that contains the DNs of the user accounts that do not have the LastLogonTimeStamp or LastLogon attribute set in the Active Directory (depending on which attribute was selected by the user for the search). If all user accounts do have the selected attribute set then this variable will be empty.

LDAPADDormantDays

LDAPADDormantDate

LDAPADUseCalculatedDate

LDAPADCalculatedDate

LDAPADLastLogonAttr

remoteLDAPHost

remoteLDAPPort

remoteLDAPUser

remoteLDAPPassword

LDAPBaseDN

LDAPUserPrefix

Get Object Operator

The Get Object operator retrieves any type of LDAP objects from the LDAP server. You can specify the search path, the search filter (see page 220), the search scope, the attributes (see page 221) to retrieve with each object, and the sort criteria.

Input Parameters

Get Criteria

Start Search Path

Specifies the start location for the search, such as Like
CN=Users,dc=domainpart,dc=company-name,dc=top-level-domain-name

Retrieve Scope

Select one of the following options:

Subtree Scope

Search the entire subtree (including the object at the search path).

One Level Scope

Search the objects directly under the object at the search path.

Object Scope

Search the object at the search path only.

Results Limit Count

Maximum number of entries to return.

- If you enter 0 or nothing in this field, CA Process Automation uses the value set in the Max Number of Search Results field (from the Directory Services category configuration).

- If you enter a value in this field, CA Process Automation uses the smaller value between this field's value and the value set in the Max Number of Search Results field (from the Directory Services category configuration).

Time Limit for Retrieve

Time in seconds to wait before timing out of this search. If this limit is 0 (or nothing is entered), there is no time limit set on the search.

Retrieve Object Type

Select one of the following options:

- User
- Computer
- Group
- Organizational Unit
- Role
- Other

This field controls the filter value displayed in the Retrieve Filter field.

Retrieve Filter

Choose the search filter to use for this search.

You can:

Use a generic filter by selecting User, Group, Organizational Unit, Role, or Computer in the Retrieve Object Type field, which displays the associated filter value in the Retrieve Filter field and makes this field read-only.

or

Enter your own filter by selecting Other in the Retrieve Object Type field, which displays "objectclass=" in the Retrieve Filter field and makes this field writable so you can fill out your own filter value. The expression must be syntactically correct otherwise the search can fail. See LDAP Search Filter Basics (see page 220) for a primer on LDAP search filters syntax.

Note that you can use different filters for User, Group, Organizational Unit, Role, or Computer by selecting Other in the Retrieve Object Type field and entering his/her filter value in the Retrieve Filter field. The provided generic search filters may not work with some LDAP servers, especially if the LDAP server does not support some of the object classes listed in the filters.

Retrieve Attributes specified as an Array Variable?

If checked, you can supply a dataset variable that contains an array of attributes to retrieve.

Retrieve Attributes Array Variable

The dataset variable that supplies an array of attributes to retrieve. This field is enabled when the Retrieve Attributes specified as an Array Variable? field is checked.

Retrieve Attributes List

Specifies a list of attributes to retrieve for this search filter. This list is enabled when the Retrieve Attributes specified as an Array Variable? field is unchecked.

Sort Criteria

Sort Fields specified as an Array Variable?

If checked, you can supply a dataset variable that contains an array of attributes used for sorting the retrieved data.

Sort List Fields Array Variable

The dataset variable that supplies an array of attributes to use for the sort order. This field is enabled when Sort Fields specified as an Array Variable? is checked.

Sort List Order

List of attributes to use for the sort order. This list is enabled when the Sort Fields specified as an Array Variable?: field is unchecked.

If nothing is entered in the Sort Criteria section, the retrieved objects are not sorted.

Note that some LDAP servers (for instance OpenLdap) do not support "Sorting" of data. In this case, the operator might fail with the following reason: [LDAP: error code 12 - critical extension is not recognized]. Do not provide any sorting criteria in this case. This is a limitation in the LDAP server, not CA Process Automation.

LDAP Search Filter Basics

The LDAP search filter syntax is a logical expression in prefix notation, where the logical operator appears before the associated arguments.

For example: (&(givenname=John)(sn=Green))

In the filter above & is the And operator and it appears before its arguments. In this example, we are searching for LDAP objects with John as the givenname (givenname is the LDAP attribute for first name), and sn as Green (sn is the LDAP attribute for last name).

Each item in the filter is composed using an LDAP attribute identifier and either an attribute value or symbols that denote the attribute value. Each item must also be enclosed within a set of parentheses, as in "(sn=Green)".

Items within a filter are combined together using logical operators to create logical expressions. Each logical expression can be further combined with other items that themselves are logical expressions, as in some of the filters used in CA Process Automation:

(&(|(objectclass=user)(objectclass=person))(!(objectclass=computer)))

In this filter, we are searching for all objects where the objectclass is either user OR person:

(|(objectclass=user)(objectclass=person))

AND the objectclass is *not* computer

(!(objectclass=computer))

Note the & at the beginning of the filter that combines these two segments together in a logical AND.

Note that the LDAP attribute objectclass stores the type(s) of an LDAP object in the LDAP directory.

Some of the logical operators used for creating filters are listed in the following table:

Symbol	Description
=	Equality Example: (givenname=John) Search for objects with John as first name.

&	Logical AND
	Example: (&(givenname=John)(sn=Green))
	Search for objects with John as first name and Green as last name
|	Logical OR
	Example: (|(givenname=John)(givenname=Michael))
	Search for objects with either John or Michael as first name
!	Logical NOT
	Example: (&(givenname=John)(!(sn=Green)))
	Search for objects with John as first name and Green is not the last name
>=	Greater than
	Example: (numsubordinates>=2)
	Search for objects with 2 or more child nodes in the LDAP tree.
<=	Less than
	Example: (numsubordinates<=2)
	Search for objects with 2 or less child nodes in the LDAP tree.
=*	Presence
	The object must have the attribute but its value is irrelevant.
	Example: (givenname=*)
	Search for objects with the givenname attribute.
*	Wildcard
	Example: (givenname=Joh*)
	Search for objects whose givenname starts with Joh

Common LDAP Attribute Names

Some common LDAP attributes are listed below. The complete list of LDAP object classes and attributes used in the LDAP server schema is located on the LDAP server.

LDAP Attribute Name	Description
cn	Common Name attribute, which contains the name of the object
dc	Domain Component attribute
objectClass	Object Class attribute, which contains the LDAP type(s) of the object

LDAP Attribute Name	Description
distinguishedName	Distinguished Name attribute in Active Directory This is the attribute that uniquely identifies the object in the Active Directory.
entrydn	Distinguished Name attribute in LDAP servers (other than Active Directory) This is the attribute that uniquely identifies the object in an LDAP server.
o	Organization Name attribute which contains the name of the organization
ou	Organizational Unit Name attribute which contains the name of the organizational unit
sn	Surname attribute which contains the family name of an individual
givenName	First name attribute which contains the first name of an individual
personalTitle	Personal Title attribute which contains the personal title of a person Examples of personal titles are "Mr", "Dr", "Prof" and "Rev".
initials	Initials attribute which contains the initials of some or all of an individual's names, but not the surname(s)
uid	User ID attribute
userPassword	Password attribute which contains a user's password Passwords are stored using an Octet String syntax and are not encrypted.
title	Title attribute which specifies the designated position or function of the object within the organization
mail	Mail attribute which contains a user's email address
company	Company or organization name attribute
department	Department Name attribute
manager	Boss, manager attribute
mobile	Mobile Phone number attribute
homephone	Home Phone number attribute
telephoneNumber	Telephone Number attribute
facsimileTelephoneNumber	Fax Number attribute

LDAP Attribute Name	Description
postalAddress	Postal Address attribute, which contains information required for the physical delivery of postal messages
postalCode	Postal Code attribute If this attribute value is present it will be part of the object's postal address.
c	Country Name attribute which contains a two-letter ISO 3166 country code
l	Locality Name attribute which contains the name of a locality, such as a city, county or other geographic region
st	State Or Province Name attribute
street	Street attribute which contains the physical address of the object, such as an address for package delivery
owner	Owner attribute which specifies the name of some object which has some responsibility for the associated object The value is a Distinguished Name
description	Description attribute which contains a human-readable description of the object
seeAlso	See Also attribute.
serialNumber	Serial Number attribute which stores the serial number of a device
member	The member attribute is used in entries defining groups It has Distinguished Name syntax, so each value is effectively a pointer to another entry in the directory. Note that the standard groupOfNames object class makes the member attribute mandatory. As attributes cannot have empty values, this effectively requires all groups to have at least one member at all times.
uniqueMember	The uniqueMember attribute is similar to the Member attribute stated above, and it is used to store the unique members in a groupOfUniqueNames object
sAMAccountName	Old NT 4.0 logon name attribute (Active Directory only), which must be unique in an Active Directory domain

LDAP Attribute Name	Description
LastLogonTimeStamp	Last Logon Time Stamp attribute (Active Directory 2003/2008 only), which contains the last logon date of a user; but it is replicated across all domain controllers only after a period of time defined in the msDS-LogonTimeSyncInterval attribute of the Active Directory
LastLogon	Last Logon attribute (Active Directory only), which contains the last logon date of a user but it is NOT replicated across all domain controllers

Output Parameters

RetrievedObjects

> An array of value maps, where each value map contains the attributes retrieved for each object. This variable is created only when the operation succeeds.

LDAPSearchPath

LDAPSearchScope

ResultsLimit

LDAPGetTimeLimit

LDAPSearchType

LDAPGetFilter

LDAPGetAttributesType

LDAPGetAttributesArray

LDAPGetAttributes

LDAPGetSortAttributesType

LDAPGetSortAttributesArray

LDAPGetSortAttributes

remoteLDAPHost

remoteLDAPPort

remoteLDAPUser

remoteLDAPPassword

LDAPBaseDN

LDAPUserPrefix

Notes:

■ The attribute names entered in the sort and retrieve sections must be the LDAP names of these attributes as specified in the LDAP server schema. For example, to retrieve the attribute "Last name" you must use the LDAP name of this attribute: "sn", to retrieve the attribute "First Name", you must use the attribute "givenname", and so on. See Common LDAP Attribute Names (see page 221).

■ The LDAP names are different from the attributes display names.

■ Most LDAP servers differ in the display names of the LDAP attributes, but they all must support the LDAP names of these attributes, thus the reason why we require the usage of the LDAP names of the attributes instead of the display names.

■ You must provide the names of the attributes to be retrieved; otherwise CA Process Automation will not return any data in the RetrievedObjects variable.

■ If no object was found under the specified search path, the search operation will succeed and the RetrievedObjects variable will be empty. The search operation will not fail in this case.

Examples

Example - Use a generic filter
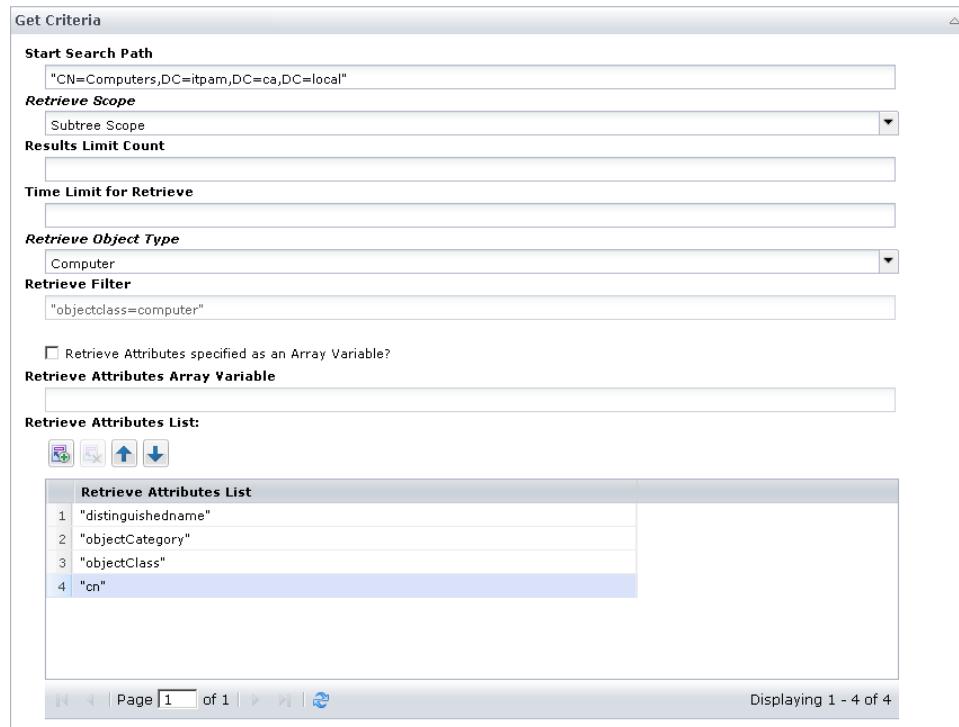
In this example, we are trying to retrieve all computer accounts under the "CN=Computers,DC=itpam,DC=ca,DC=local" path. We are specifically asking for the "cn", "distinguishedname", "objectcategory", and "objectclass" attributes of these accounts.

We are also sorting the returned user accounts by "cn".

After the operation ran, it ended successfully and the RetrievedObjects variable was created as follows:

For each value map in the RetrievedObjects variable, we now have the values retrieved for each computer account attribute. In this example, the RetrievedObjects variable contains eight objects.

Also note that the values of multi-valued attributes (objectclass in this example) are returned with a "|" between the multiple values:

Example – Use your own filter
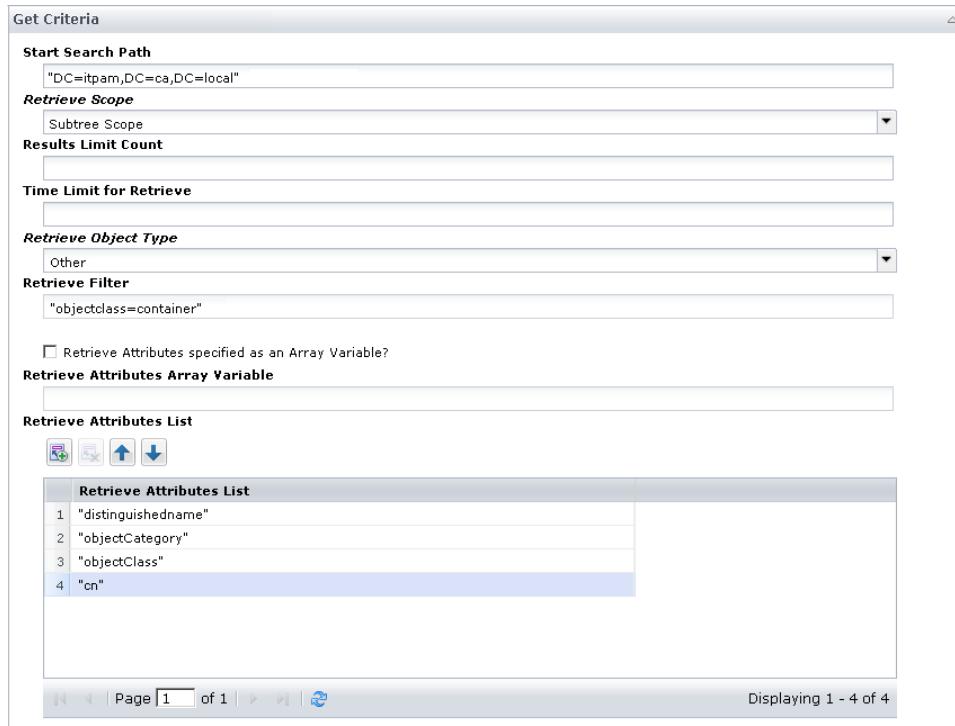
In this example, we are using our own filter (note retrieve Object Type is set to Other) to retrieve all container accounts under the "DC=itpam,DC=ca,DC=local" path. We are specifically asking for the "cn", "distinguishedname", "objectcategory", and "objectclass" attributes of these accounts.

We are also sorting the returned user accounts by "cn".

After the operation ran, it ended successfully and the RetrievedObjects variable was created as follows:

In each value map in the RetrievedObjects variable, we now have the values retrieved for each container account attribute. The RetrievedObjects variable contains 86 objects in this example.

Please also note that the values of multi-valued attributes (objectclass in this example) are returned with a "|" between the multiple values:

| objectClass | top\|container\|groupPolicyContainer |

Operator Failure

This operator fails in the following cases:

- The search path does not exist on the LDAP server.

- The search time limit was exceeded.

- Unable to connect to the LDAP server.

This operation *may* fail in the following cases, depending on the LDAP server as some LDAP servers consider these as errors while others do not:

- The search filter is invalid.

- The returning attributes are invalid.

- The sorting attributes are invalid.

Get User Operator

The Get User operator retrieves LDAP objects of type user account from the LDAP server. You can specify the search path, the search filter, the search scope, the attributes to retrieve with each object, and the sort criteria.

Input Parameters

Input parameters for the Get User operator include get criteria and sort criteria.

Get Criteria

Start Search Path

Specifies the start location for the search, Like
CN=Users,dc=domainpart,dc=company-name,dc=top-level-domain-name.

Retrieve Scope

Select one of the following options:

Subtree Scope

Searches the entire subtree (including the object at the search path).

One Level Scope

Searches the objects directly under the object at the search path.

Object Scope

Searches the object at the search path only.

Results Limit Count

Maximum number of entries to return.

- Enter 0 or nothing in this field to use the value set in the Max Number of Search Results field (from the Directory Services category configuration).

- Enter a value in this field to use the smaller value between this field's value and the value set in the Max Number of Search Results field (from the Directory Services category configuration).

Time Limit for Retrieve

Specifies time in seconds to wait before timing out of this search.

If 0 or nothing is entered, there is no time limit set on the search.

Retrieve Filter

Specifies the search filter to use for this search.

- The generic search filter searches for user accounts in LDAP, and you can tweak this filter as necessary. This field is writable (can be modified). See LDAP Search Filter Basics (see page 220) for a primer on LDAP search filters syntax.

- The generic search filter may not work with some LDAP servers, especially if the LDAP server does not support some of the objectclasses listed in the filter:

"(&(|(objectclass=user)(objectclass=person))(!(objectclass=computer)))"

■   The expression must be syntactically correct, otherwise the search can fail.

■   Consult with your LDAP administrator regarding the supported LDAP objectclasses.

Retrieve Attributes specified as an Array Variable?

Check this box to supply a dataset variable that contains an array of attributes to retrieve.

Retrieve Attributes Array Variable

Specifies the dataset variable that supplies an array of attributes to retrieve. This field is enabled when the Retrieve Attributes specified as an Array Variable? check box is enabled.

Retrieve Attributes List

Specifies a list of attributes to retrieve for this search filter. This list is enabled when the Retrieve Attributes specified as an Array Variable? field is unchecked.

Sort Criteria

Sort Fields specified as an Array Variable?

Check this box to supply a dataset variable that contains an array of attributes used for sorting the retrieved data.

Sort List Fields Array Variable

Specifies the dataset variable that supplies an array of attributes to be used for the sort order. This field is enabled when the Sort Fields specified as an Array Variable? check box is selected.

Sort List Order

List of attributes to be used for the sort order. This field is enabled when the Sort Fields specified as an Array Variable? check box is unchecked.

If nothing is entered in the Sort Criteria section, the retrieved objects will not be sorted.

Note: Some LDAP servers (such as OpenLdap) do not support "Sorting" of data. In this case, the operation might fail with the following reason: [LDAP: error code 12 - critical extension is not recognized]. Do not provide any sorting criteria in this case. This is a limitation in the LDAP server, not CA Process Automation.

Output Parameters

UserAccounts

This is an array of value maps, where each value map contains the attributes retrieved for the user. This variable is created only when the operation succeeds.

LDAPSearchPath

LDAPSearchScope

ResultsLimit

LDAPGetTimeLimit

LDAPGetFilter

LDAPGetAttributesType

LDAPGetAttributesArray

LDAPGetAttributes

LDAPGetSortAttributesType

LDAPGetSortAttributesArray

LDAPGetSortAttributes

remoteLDAPHost

remoteLDAPPort

remoteLDAPUser

remoteLDAPPassword

LDAPBaseDN

LDAPUserPrefix

Attribute names entered in the sort and retrieve sections must be the LDAP names of these attributes as specified in the LDAP server schema. For instance, to retrieve the attribute "Last name" you must use the LDAP name of this attribute: "sn", to retrieve the attribute "First Name", you must use the attribute "givenname", and so on. For more information, see the topic "Common LDAP Attribute Names (see page 221)".

LDAP names are different from the attributes display names.

Most LDAP servers differ in the display names of the LDAP attributes, but they all must support the LDAP names of these attributes, thus the reason why we require the usage of the LDAP names of the attributes instead of the display names.

You must provide the names of the attributes to be retrieved; otherwise CA Process Automation will not return any data in the UserAccounts variable.

If no user accounts were found under the specified search path, the search operation will succeed and the UserAccounts variable will be empty. The search operation will not fail in this case.

Examples

In this example, we are trying to retrieve all user accounts under the "ou=ldapconnector,dc=ca,dc=com" path. We specifically asked for the "entrydn", "uid", and "objectclass" attributes of these accounts.
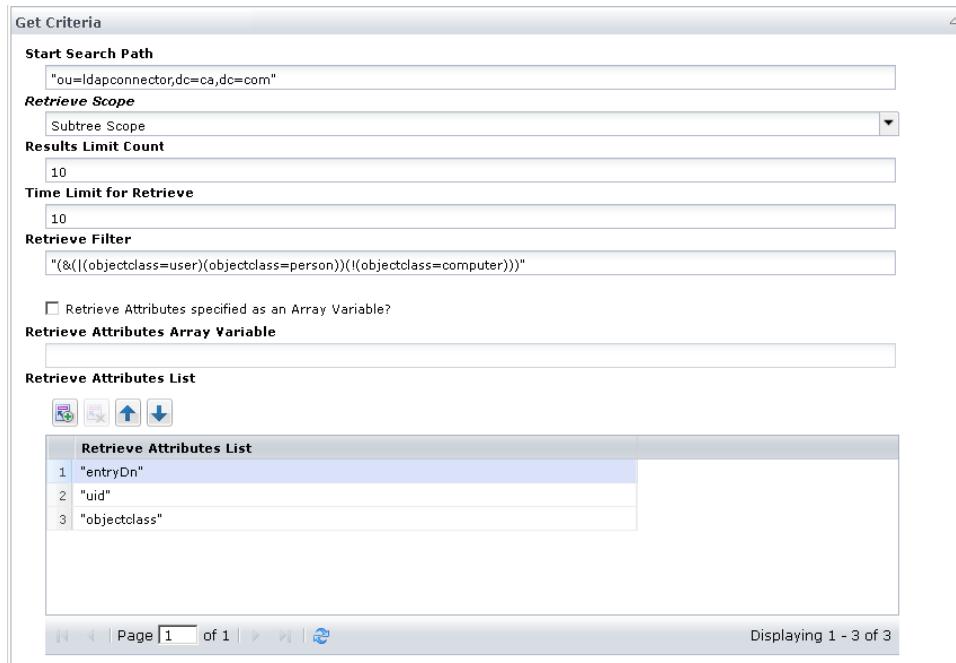
We are also sorting the returned user accounts by "entryDn" then "uid".
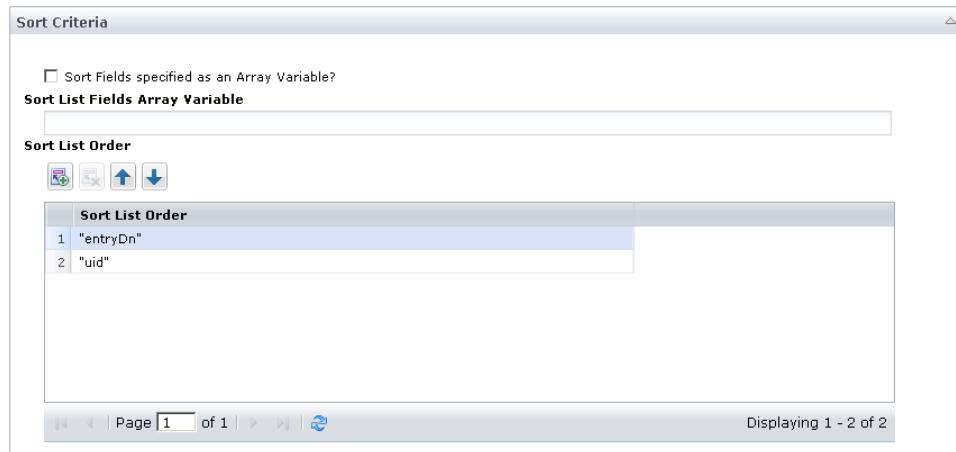
After the operator ran, it ended successfully and the UserAccounts variable was created as follows:

Each ValueMap in the UserAccounts variable now shows the values retrieved for each user account attribute. The operator returned only ten user accounts because we specified our search results limit to be ten. The values of multi-valued attributes (objectclass in this example) are also returned with a "|" between the multiple values:

Operator Failure

This operator fails in the following cases:

- The search path does not exist on the LDAP server.

- The search time limit was exceeded.

- Unable to connect to the LDAP server.

This operation *may* fail in the following cases, depending on the LDAP server as some LDAP servers consider these as errors while others do not:

- The search filter is invalid.

- The returning attributes are invalid.

- The sorting attributes are invalid.

Move Object Operator

The Move Object operator moves an LDAP object from one location to another within the LDAP server.

Input Parameters

Object Old DN

Specifies the distinguished name of the object that you want to move.

Object New DN

Specifies the destination for the distinguished name where you want the object to be moved.

Output Parameters

LDAPMoveObjectOldDn

LDAPMoveObjectNewDn

remoteLDAPHost

remoteLDAPPort

remoteLDAPUser

remoteLDAPPassword

LDAPBaseDN

LDAPUserPrefix

Example

Operator Failure

This operator fails in the following cases:

- The old DN of the object does not exist in the LDAP server.

- The new DN of the object exists already.

- The LDAP server is setup to prevent a 'Move' Operator from occurring programmatically.

- Unable to connect to the LDAP server.

Remove User from Group Operator

The Remove User from Group operator lets you remove an LDAP user from an LDAP group on the LDAP server.

Input Parameters

User DN

Specifies the distinguished name of the user that you want to remove from the group.

Group DN

Specifies the distinguished name of the group from which you want to remove the user.

Output Parameters

LDAPUserDn

LDAPGroupDn

remoteLDAPHost

remoteLDAPPort

remoteLDAPUser

remoteLDAPPassword

LDAPBaseDN

LDAPUserPrefix

Example

Remove User from Group Parameters

User DN:

"cn=JSmith,ou=ldapconnector,dc=ca,dc=com"

Group DN:

"cn=Users,dc=ca,dc=com"

Operator Failure

This operator fails in the following cases:

■ The user is not a member of the group.

■ The group does not exist in the LDAP server.

■ The Group DN points to an LDAP object that is not of type group, groupofnames or groupofuniquenames.

■ Unable to connect to the LDAP server.

Update Object Attributes Operator

The Update Object Attributes operator performs all of the following tasks simultaneously:

- Add new attributes to an existing LDAP object.

- Replace the values of attributes of an existing LDAP object.

- Remove attributes from an existing LDAP object.

You can perform all three operations on the same object at the same time, or you can choose to ignore any of the operations by not entering anything in the operator's parameters.

Note: The Active Directory will not allow for a user's password to be modified unless CA Process Automation is connected to the Active Directory server through SSL. If CA Process Automation is not connected through SSL, a replace operator on a user password will fail.

For more information about how to import an Active Directory's certificate to CA Process Automation, see Add an SSL Certificate to CA Process Automation (see page 246). Once the certificate is imported, you can change the Directory Services category's properties to establish an SSL connection with the Active Directory server.

Input Parameters

Input parameters for the Update Object Attributes operator are as follows.

Objects Parameters

Object Distinguished Name

Specifies the distinguished name of the LDAP object whose attributes you want to modify.

Add Attributes Parameters Page

Use the Add Attributes Parameters page to enter all the attributes that you want to create for the LDAP object.

Use specified array fields for the LDAP attributes to be added

Check this box to use the Attributes and Attributes Values arrays for this request.

LDAP Attributes Array

The array containing the LDAP names of the attributes to be added to the object. This Dataset field must be defined as an array (indexed string). If Use specified array fields for the LDAP attributes to be added is checked, this field will be used.

LDAP Attributes Values Array

The array containing the values of the attributes to be added to the object. This Dataset field must be defined as an array (indexed string). If Use specified array fields for the LDAP attributes to be added is checked, this field will be used.

LDAP Attributes

The LDAP names of the attributes to be added to the object. If Use specified array fields for the LDAP attributes to be added is unchecked, this field will be used.

LDAP Attributes Values

The values of the attributes to be added to the object. If Use specified array fields for the LDAP attributes to be added is unchecked, this field will be used.

LDAP Attributes Value Maps

This is an array of value maps containing additional LDAP attributes to be added to the object. Each value map's Key and Value must be of type string; moreover, the key must be named Keys and the value must be named Values.

Example

In this example, the object addAttrHashMapArray is an indexed ValueMap whose key fields are called *Keys* and value fields are called *Values*.

Note that the user can use the LDAP Attributes Value Maps alone or as an addition to any attributes (and associated attribute values) entered in the other fields of the page.

Replace Attributes Parameters Page

Use the Replace Attributes Parameters page to enter all the attributes whose values you want to replace in the LDAP object.

Use specified array fields for the LDAP attributes to be added

If this is checked, then the Attributes and Attributes Values arrays will be used for this request.

LDAP Attributes Array

Specifies the array containing the LDAP names of the attributes whose values are to be replaced in the object. This Dataset field must be defined as an array (indexed string). If Use specified array fields for the LDAP attributes to be added is checked, this field will be used.

LDAP Attributes Values Array

Specifies the array containing the new values of the attributes to be replaced in the object. This Dataset field must be defined as an array (indexed string). If Use specified array fields for the LDAP attributes to be added is checked, this field will be used.

LDAP Attributes

Specifies the LDAP names of the attributes whose values are to be replaced in the object. If Use specified array fields for the LDAP attributes to be added is unchecked, this field will be used.

LDAP Attributes Values

Specifies the new values of the attributes to be replaced in the object. If Use specified array fields for the LDAP attributes to be added is unchecked, this field will be used.

LDAP Attributes Value Maps

Specifies an array of value maps containing the LDAP names and new values of the attributes whose values are to be replaced in the object. Each value maps Key and Value must be of type string; moreover, the key must be named Keys and the value must be named Values.

Remove Attributes Parameters Page

The Remove Attributes Parameters page is used to enter all the attributes that you want to remove from the LDAP object.

Use specified array field for the LDAP attributes to be removed

Check this box to use Attributes arrays for this request.

LDAP Attributes Array

Specifies the array containing the LDAP names of the attributes to be removed from the object. This Dataset field must be defined as an array (indexed string). If Use specified array field for the LDAP attributes to be added is checked, this field will be used.

LDAP Attributes

Specifies the LDAP names of the attributes to be removed from the object. If Use specified array field for the LDAP attributes to be added is unchecked, this field will be used.

Output Parameters

LDAPModifyObjAttrsDN

LDAPAddAttributesType

LDAPAddAttributesKeyArray

LDAPAddAttributesValueArray

LDAPAddAttributesKeys

LDAPAddAttributesValues

LDAPAddAttributesMap

LDAPRemoveAttributesType

LDAPRemoveAttributesKeyArray

LDAPRemoveAttributesKeys

LDAPReplaceAttributesType

LDAPReplaceAttributesKeyArray

LDAPReplaceAttributesValueArray

LDAPReplaceAttributesKeys

LDAPReplaceAttributesValues

LDAPReplaceAttributesMap

remoteLDAPHost

remoteLDAPPort

remoteLDAPUser

remoteLDAPPassword

LDAPBaseDN

LDAPUserPrefix

Operator Failure

This operator fails in the following cases:

- The LDAP object specified does not exist.

- One of the LDAP attribute you want to add, replace, or remove does not exist

- The list of attributes and attribute values of the 'Add' or 'Modify' pages do not match in length.

- The user checked that an array of attributes and an array of attribute values are used for the 'Add', 'Replace', or 'Remove' pages, but the CA Process Automation object entered in the array field is actually not of type array (indexed strings).

- The LDAP server is setup to prevent any modifications of LDAP objects done programmatically.

- Unable to connect to the LDAP server.

Update User Home Directory Operator

The Update User Home Directory operator sets up a share for a user in an Active Directory server. The share includes a home drive, home directory, and log in script.

Input Parameters

User DN

Specifies the distinguished name of the user object for which you want to set the home directory, home drive, and logon script.

Home Directory

Specifies the new home directory of the user.

Home Drive

Specifies the new home drive of the user.

Logon Script

Specifies the new logon script of the user.

Output Parameters

LDAPUserDn

LDAPADHomeDirectory

LDAPADHomeDrive

LDAPADLogonScript

remoteLDAPHost

remoteLDAPPort

remoteLDAPUser

remoteLDAPPassword

LDAPBaseDN

LDAPUserPrefix

Example

Operator Failure

This operator fails in the following cases:

- The user object does not exist.

- The Active Directory is setup to prevent any modifications of LDAP objects to occur programmatically.

Add an SSL Certificate to CA Process Automation

To add an SSL certificate to CA Process Automation

1. Retrieve the certificate file from the Active Directory server.

   For instance, to establish an SSL connection between CA Process Automation and an Active Directory server, retrieve the certificate. Log in to the http://*i.p.*/certsrv where *i.p.* is the IP address of the Active Directory server, then download the certificate.

2. Copy the certificate file to the computer where the CA Process Automation Directory Services operators are running.

3. Import the certificate using the keytool command:

   `keytool -import -alias PAM -file` *certnew.cer* `-keystore "C:\\Program Files\\Java\\jdk1.6.0_03\\jre\\lib\\security\\cacerts"`

   Where *certnew.cer* is the path to the certificate file retrieved in step 1.

   "C:\\Program Files\\Java\\jdk1.6.0_03\\jre\\lib\\security\\cacerts" is the path to the cacerts file within the Java JRE or JDK.

   - The keytool program is part of the Java installation.

   - Keytool prompts for a password. The password is 'changeit' by default.

   - Keytool prompts whether to 'Trust this certificate?[no]'. Enter yes.

4. Add the following lines in the CA Process Automation file:

   PAM\server\c2o\bin\c2osvcw.conf

    (or in the case of an upgrade): I

   PAM_DIR%\server\c2o\bin\c2osvcw.conf:

   `wrapper.java.additional.11=-Djavax.net.ssl.trustStore="C:\Program Files\Java\jdk1.6.0_03\jre\lib\security\cacerts"`
   `wrapper.java.additional.12=-Djavax.net.ssl.trustStorePassword="changeit"`

   The numbers might be different for you. Start with the next available number. If wrapper.java.additional.11 is already defined, use 12 and 13.

   The program folder is different for your JDK installation.

   The password is "changeit".

5. Restart the CA Process Automation Touchpoint that contains the Directory Services operators.

Set Up the Active Directory Server

To establish an SSL connection between the CA Process Automation-Directory Services operators and an Active Directory server, verify that the Active Directory server is set up:

1. The Certificate Services are installed on your Active Directory server (consult your Active Directory admin for this task).

2. The Automatic Certificate Request is configured for Domain Controllers (consult your Active Directory admin for this task).

Note: When you create a new user account or modify the password of an existing user account in Active Directory, the Active Directory does not allow you to create or modify a user password unless CA Process Automation is connected to the Active Directory server through SSL.

Chapter 8: Email

The Email operators can automate tasks that are performed on messages and folders in an email server. Email operators communicate with your mail server remotely using one of the following protocols:

Post Office Protocol version 3 (POP3)

POP3, suitable for single-user access to a mailbox, allows you to download email messages to your local computer. By default, POP3 servers listen on TCP Port 110.

POP-SSL

By default, POP-SSL servers listen on TCP Port 995.

Internet Message Access Protocol (IMAP)

IMAP, suitable for multi-user access to a mailbox, allows simultaneous access by multiple clients. By default, IMAP servers listen on TCP Port 143.

IMAP-SSL

By default, IMAP-SSL servers listen on TCP Port 993.

The Email operators can communicate with your mail server only if it supports either IMAP or POP3. The Email operators perform actions such as get email counts. Both protocols support this type of action. Some actions, such as delete folder, are supported only by the IMAP protocol.

Common Email Operator Parameters

The following properties apply for various Email operators:

- Message Filter Criteria
- Mail Server Login Parameters

Message Filter Criteria

Message ID

Specifies the unique ID of the email to filter. You can also pass a substring of the Message ID. If the IDs for the desired emails are not known, you can retrieve Message IDs for messages using the Get Email List operator (see page 263). Action is taken on all the emails that match the subject substring.

Note: This parameter is not available for all Email operators.

Message Number

Specifies the message number of the email to filter. This parameter is *not* a static number for a message. If emails are deleted and moved to different folders, the message number changes. If there are parallel actions taking place on the same mailbox folder, we recommend using Message ID (rather than Message Number) to specify messages.

Note: This parameter is not available for all Email operators.

Message Subject

Specifies the subject of the email that you want to match. This parameter can be a substring or regular expression. Action is taken on all the emails that match the subject substring.

Message Sender

Specifies the email sender that you want to match. This parameter can be a substring or regular expression. Action is taken on all the emails that match the subject substring.

Message Subject and Sender values are regular expressions

Select this check box to specify that Message Subject and Message Sender values as regular expressions (rather than a simple string) when filtering email.

Important! To parse emails for the Email operators using regular expressions, *all* emails must be retrieved from the mailbox and parsed at the client side. That is, on the touchpoint where the operator runs. We recommend that you do not select this field and use regular expressions only if it is required.

Message Body

Specifies as a substring of the email body that you want to match.

Earliest Message Sent Time

Match emails that are sent after the time specified. This parameter identifies the earliest time the email that you want to match was sent, specified in an CA Process Automation variable.

Latest Message Sent Time

Match messages that are sent before the time specified. This parameter identifies the latest time the message that you want to match was sent, specified in an CA Process Automation variable.

Note: Earliest Message Sent Time and Latest Message Sent Time fields are CA Process Automation date type variables. System functions like now() (see page 553) or today() (see page 560) generate date type variables. A system function named parseDate (see page 553) (stringDate, simpleDateFormat) creates the date properly. stringDate is the date in string format and simpleDateFormat is the format to use to parse the date. This function can be used to parse a string to a CA Process Automation date type variable. For example:

```
parseDate("2010/07/28 13:00:01", "yyyy/MM/dd HH:mm:ss")
```

IMAP Message Flag

IMAP uses message flags to monitor the state of an email. These flags are stored on the server. Different clients accessing the same mailbox at different times can detect the changes that other clients make.

The following flags are valid and can be set programmatic by setting the values to name of flags. The name of the flags is case-sensitive and is passed with all upper cases.

Select the flag that you want to set for your message. These flags work in conjunction with the IMAP Message Flag is set to true check box.

ANSWERED

When you select the IMAP Message Flag is set to true check box, only the emails that have been answered return (those emails on to which the replies were sent).

DELETED

When you select the IMAP Message Flag is set to true check box, any emails that were not marked as deleted from the server return.

DRAFT

When you select the IMAP Message Flag is set to true check box, all emails that are currently not in a draft state return.

FLAGGED

When you select the IMAP Message Flag is set to true check box, any emails that are not flagged return.

RECENT

When you select the IMAP Message Flag is set to true check box, any emails that were never opened (clicked) return.

SEEN

When you select the IMAP Message Flag is set to true check box, any email which is marked as READ returns. When you do *not* select the IMAP Message Flag is set to true check box, any email which is marked as UNREAD returns.

IMAP Message Flag is set to true

The IMAP flag name is set in the IMAP Message Flag drop-down list. Its Boolean value can be selected from this check box. For example, if you want to select Seen as true, select Seen from the drop-down and then select this check box.

Mail Server Login Parameters

Protocol for Connection

Select the email protocol to use to connect to the server:

- IMAP
- IMAP-SSL
- POP3
- POP3-SSL

Mail Server Host

Specifies the hostname/IP address of the mail server.

Mail Server Port

Specifies the port of the mail server.

Username

Specifies the username of the user to access the mail server.

Password

Specifies the password for the user to access the mail server.

Create Folder Operator

The Create Folder operator creates a folder on the mail server. Folders are created recursively using the IMAP protocol.

If the folder exists, an exception is thrown that cites "Folder already exists".

Input Parameters

Mailbox Folder Name

Name of the folder to create in the mail server.

Mail Server Login Parameters **(see page 252)**

Output Parameters

FolderCreated

Returns true when the folder gets created successfully and false otherwise.

FolderName

Protocol

ServerHost

ServerPort

UserName

Password

Delete Email Operator

The Delete Email operator deletes messages from the mailbox and returns the number of messages deleted. This operator uses the IMAP protocol.

Note: If all fields are left blank, this operator deletes all messages from the mailbox (the specified mailbox folder name).

Input Parameters

Mailbox Folder Name

Specifies the name of the folder that contains messages to delete in the mail server.

Message Filter Criteria (see page 250)

Mail Server Login Parameters (see page 252)

Output Parameters

DeletedCount

Returns the number of messages deleted.

FolderName

MessageID

MessageNumber

Subject

From

IsRegExp

Body

SentFromDate

SentToDate

FlagField

FlagValue

Protocol

ServerHost

ServerPort

UserName

Password

Delete Folder Operator

The Delete Folder operator deletes a folder on the server using the IMAP protocol. The folder is deleted even if it contains subfolders.

Input Parameters

Mailbox Folder Name

Specifies the name of the folder to delete on the mail server.

Mail Server Login Parameters (see page 252)

Output Parameters

FolderCreated

Returns true when the folder is deleted successfully and false otherwise.

FolderName

Protocol

ServerHost

ServerPort

UserName

Password

Example

Important! The following scenario uses a hmail mail server. This scenario does not work when a Dominos server is used.

1. Create hierarchy of folders, such as test1\test2\test3\test4 (parallel to "Inbox").

2. Create another hierarchy, test2\test5\test6 (parallel to "Inbox").

3. Delete "test2" folder using the Delete Folder operator.

Expected result

The hierarchy "test2\test5\test6" is deleted.

Actual result

The process runs successfully and returns false. Ideally it returns true. The folder still exists on the mail server.

Get Email Content Operator

The Get Email Content operator retrieves the email body and attachments. If the message number field is not blank or null, then the operator retrieves a single email (based on the message number) and the operator returns the details. Otherwise, the operator returns the content of all the emails within the folder.

Note: If all fields are left blank, this operator retrieves the content of all the emails from the specified mailbox folder.

Input Parameters

Mailbox Folder Name

Specifies the name of the folder that contains emails to process.

Set messages retrieved as seen

When selected, sets the retrieved messages as seen.

Start index of mail content to get in Dataset variable

If you enter an index in this field, content starting from that index displays in the Dataset variable. Leave blank to start from the beginning.

Length of mail content to get in Dataset variable

If you enter an index in this field, content until that index displays in the Dataset variable. Leave this blank to get the maximum mail content possible.

Process attachments of mail

When selected, attachments are also processed. The default is unchecked.

Attachment Operation

Specifies one of the following operations that can be performed on the attachment. This option is enabled when the user selects the Process attachments of mail check box.

Save attachment to a file

Saves the attachment in the destination folder.

Get attachment content in Dataset variable

Writes the attachment contents to a dataset variable.

The property to retrieve mail email content is exposed as a configurable property. The default size is 64K Add "mail.maximum.content.size=64" in the OasisConfig.properties to configure the size. The unit of size is K.

Both

Performs both the save and Write Contents to Dataset variable operations.

Destination Folder Name

Specifies the destination folder where the attachment is saved.

Generate unique filenames to save attachment

When selected, provides the option to generate unique file names when saving attachments.

Start index of attachment content to get in Dataset variable

If you enter an index in this field, attachment content starting from that index displays in the Dataset variable. Leave blank to start from the beginning.

Length of attachment content to get in Dataset variable

If you enter an index in this field, attachment content until that index displays in the Dataset variable. Leave this blank to get the maximum mail content possible.

Message Filter Criteria (see page 250)

Mail Server Login Parameters (see page 252)

Output Parameters

MessageContent

(ValueMap) Returns the message contents and attachment contents.

Contains:

ResultRow

AttachmentContents

MailContents

AttachmentFiles

FolderName

MarkMessagesAsSeen

MessageContentStartIndex

MessageContentLength

IsProcessAttachement

ProcessAttachmentType

DestinationFolderName

GenerateUniqueName

MessageAttachmentStartIndex

MessageAttachmentLength

MessageID

MessageNumber

Subject

From

IsRegExp

Body

SentFromDate

SentToDate

FlagField

FlagValue

Protocol

ServerHost

ServerPort

UserName

Password

Get Email Count Operator

The Get Email Count operator returns the number of emails in the folder.

You can connect to the mail server either through the POP3 protocol or the IMAP Protocol and, based on the protocol, the user must provide the appropriate port number.

- The default port for POP3 is 110.

- The default port for IMAP is 143.

- The default port for POP-SSL is 995.

- The default port for IMAP-SSL is 993.

Input Parameters

Mailbox Folder Name

Specifies the name of the folder that contains emails to process.

Message Filter Criteria (see page 250)

Mail Server Login Parameters (see page 252)

Output Parameters

MessageCount

Returns the number of emails in the folder.

FolderName

MessageID

MessageNumber

Subject

From

IsRegExp

Body

SentFromDate

SentToDate

FlagField

FlagValue

Protocol

ServerHost

ServerPort

UserName

Password

Get Email Envelope Operator

The Get Email Envelope operator retrieves the email envelopes from the specified filter criteria. If the message number is not blank or null (specified in the Message Filter Criteria (see page 250)), then this operator retrieves a single email (based on the message number) and the operator returns the details. Otherwise, this operator returns envelopes of all the emails within the folder.

Note: If all fields are left blank, this operator retrieves the content of all the emails from the specified mailbox folder.

Input Parameters

Mailbox Folder Name

Specifies the name of the folder that contains emails to process.

Set messages retrieved as seen

When selected, sets retrieved emails as seen.

Message Filter Criteria (see page 250)

Mail Server Login Parameters (see page 252)

Output Parameters

MessageEnvelope

(ValueMap) Returns the envelope of messages in the folder.

Contains:

ResultRow

SentDate

Subject

To

Bcc

Cc

From

FolderName

MarkMessagesAsSeen

MessageID

MessageNumber

Subject

From

IsRegExp

Body

SentFromDate

SentToDate

FlagField

FlagValue

Protocol

ServerHost

ServerPort

UserName

Password

Get Email List Operator

The Get Email List operator retrieves a list of emails that match certain filter criteria. You can configure fields described in the Message Filter Criteria (see page 250) to filter only the emails you want to retrieve envelopes from. This operator can use both IMAP and POP3 protocol. Use the Get Email List operator specifically to retrieve basic information of emails. This information includes Message ID and Message Number that can be used in other operators.

Note: The Get Email List operator can retrieve envelopes of a maximum of 512 emails in one iteration.

Input Parameters

Mailbox Folder Name

Specifies the name of the target mailbox folder name that contains emails to be processed. This field cannot be left blank.

Message Filter Criteria (see page 250)

Mail Server Login Parameters (see page 252)

Output Parameters

MessageList

An array of ValueMaps. Each index of the array is a CA Process Automation ValueMap data type. The ValueMap contains the following fields that hold the following information in a single message:

MessageID

UniqueID of the message.

MessageNumber

This message number can vary for the same email if emails are moved from the folder to anther folder or deleted and expunged. We recommended using MessageID to specify emails uniquely.

Subject

Subject of the email.

SenderAddress

Address of the sender of the email.

SentDate

A CA Process Automation date type variable with date when the email was sent.

ReceivedDate

A CA Process Automation date type variable with the date that the server received the email. This value is only populated when using IMAP protocol to connect to the server.

NumOfAttachments

An integer type variable to give number of attachments present in the email.

FolderName

Subject

From

IsRegExp

Body

SentFromDate

SentToDate

FlagField

FlagValue

Protocol

ServerHost

ServerPort

UserName

Password

Move Email Operator

The Move Email operator moves the emails from one folder to another.

Input Parameters

Mailbox Source Folder Name

Name of the source folder that contains the emails to move.

Mailbox Destination Folder Name

Specifies the name of the destination folder where the emails are copied.

Notes:

- If the source folder does not exist, CA Process Automation throws an exception saying "Source folder does not exist".

- If the destination folder does not exist, CA Process Automation creates the destination folder and then moves the emails from the source folder to the destination folder.

Message Filter Criteria (see page 250)

Mail Server Login Parameters (see page 252)

Output Parameters

MovedCount

SourceFolderName

DestinationFolderName

MessageID

MessageNumber

Subject

From

IsRegExp

Body

SentFromDate

SentToDate

FlagField

FlagValue

Protocol

ServerHost

ServerPort

UserName

Password

Purge Folder Operator

The Purge Folder operator expunges (permanently removes) folders marked DELETED and returns the number of emails expunged. This operator uses the IMAP protocol.

Input Parameters

Mailbox Folder Name

Specifies the name of the folder on the mail server that contains emails to delete permanently.

Mail Server Login Parameters (see page 252)

Output Parameters

ExpungedCount

Returns the number of emails deleted.

FolderName

Protocol

ServerHost

ServerPort

UserName

Password

Rename Folder Operator

The Rename Folder operator renames the folder in the mail server. The operator uses the IMAP protocol.

Input Parameters

Current Mailbox Folder Name

Specifies the old name of the folder to be renamed.

New Mailbox Folder Name

Specifies the new name for the folder.

Mail Server Login Parameters (see page 252)

Output Parameters

FolderRenamed

Returns true when the folder is renamed successfully and false otherwise.

OldFolderName

NewFolderName

Protocol

ServerHost

ServerPort

UserName

Password

Send Email Operator

Use the Send Email operator to deliver email notifications to specified recipients.

The CA Process Automation email implementation supports sending mail through an SMTP server. You can specify any valid email address that is supported by the SMTP server. These addresses can include aliases, a mailing list, a fax address, or a digital pager. The SMTP server, rather than CA Process Automation, handles the actual delivery of a message.
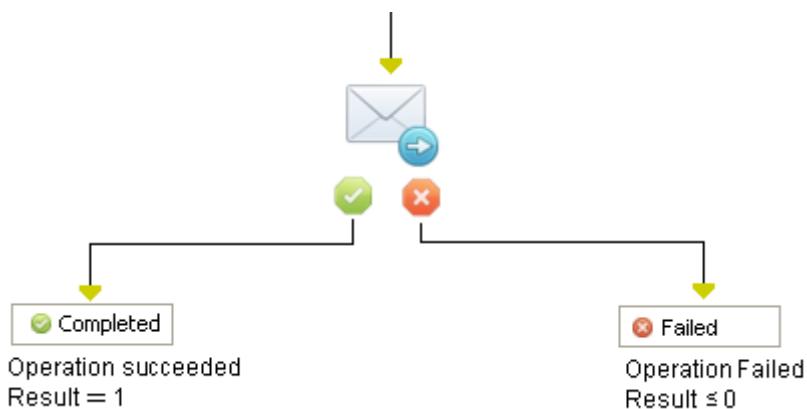
The Send Email operator can include attached files. You can specify attachments to send dynamically updated files rather than static information specified when you added a Send Email operator to a process. This feature is useful for sending attachments such as log and exception files.

The Send Email operator also supports HTTP/HTTPS URLs as an attachment path. The Send Email operator fails if a path name is not valid at run time.

Tip: Specify that locations for attachments are relative to the touchpoint running the Email operators.

Input Parameters

User Name

Specifies a valid user name or profile for sending mail on the SMTP server. For example, Process.Email.*Username.*

Password

Specifies the password for the user name. For example, Process.Email.*Password*.

From

The email address to appear in the sender field of the outgoing email.

To

Specifies the addresses of email recipients. Separate multiple email addresses with either commas or semi-colons. For example, support@PAM.com, sales@PAM.com.

Cc

Specifies the addresses of recipients who receive copies of this email alert. Separate multiple email addresses with either commas or semi-colons. For example, system_administrator@PAM.com; support@PAM.com.

Subject

Specifies a brief description to appear in the subject of the email message. For example, CA Process Automation Alert.

Message

Specifies the message that the email delivered. For example,

```
Notice
Backup problems on a Touchpoint: + Process.TouchpointName.
```

Send in HTML Format

When selected, CA Process Automation uses HTML text in the Message field. If this option is not selected, CA Process Automation uses plain text.

For example, use the following text to display "Notice" in red font:

```
<font size="5" color="red">Notice</font>
```

Attachment

Specifies the full paths for files to attach to the email. Separate multiple path names by commas or semi-colons. You can also specify HTTP/HTTPS URLs for local and remote locations.

This parameter must specify locations that are valid for the Email operators at runtime, and on the touchpoint where the Email operators are running. For example, C:\\CA Process Automation\\Data\\Log\\Global.log.

Receipt

Select this check box to request a delivery receipt for the message. The receipt is typically a service that the recipient mail client provides. CA Process Automation cannot guarantee the receipt.

Encoding

Specifies the encoding scheme that the reader receives the text in (UTF-8, UTF-16, US-ASCII, Windows-1250, Windows-1252, Shift_JIS).

Output Parameters

user

password

from

to

cc

subject

text

isContentHtml

attachment

recpt

Chapter 9: File Management

The File Management operators monitor directories, files, and their contents. These operators can be run either locally or on a remote system. The File Management operators also support operations on a proxy node. The process takes either a success or a failure path, based on the results of the operation.

Use the File Management operators to create, delete, rename, compress and uncompress local files. You can also use the File Management operators to watch files on the touchpoint where the operators are running. All File Management operators run under the same user name that is running the touchpoint, such as administrator on a Windows touchpoint or root on a UNIX touchpoint.

The File Management operators can run on proxy agents. Properties in a normal and proxy service are the same, but some behavior is different. For example, if the Process Proxy Service is running with the log output option checked, then the log file is created on the file system where the agent is running. The log file is not created on the machine where the proxy agent resides.

Important! The following conditions apply to all operators in this category when running them on a remote Windows host through a proxy touchpoint::

- Use UNIX-style paths for fields which are path-related (forward slashes and no drive letter).

- Each SSH server can have a different location for its "root" directory. The permitted commands that are relative to that root directory can vary too.

Compress File Operator

The Compress File operator compresses a file or directory. In a Windows environment, it is compressed using the WINZIP Command Line Utility. In UNIX environments, it is compressed using the gzip utility.

Prerequisites

- The WZZIP command line utility must be installed on the target host if it is a Windows host. The WZZIP command line utility is a free add-on for users of WinZip 12 standard or pro with a valid license.

- The gzip utility is required for UNIX environments.

Input Parameters

Source File/Directory Name

Specifies the name of file or directory to be compressed.

Notes: On the Windows host, the compressed file extension is ".zip".

On a UNIX host, if the source is a directory, then every file under that directory is compressed and is replaced one with the extension ".gz".

Working Directory

Specifies the working directory to execute this operation.

Notes: If the working directory is not specified, the user's home directory will be the working directory.

The file path can be absolute or relative to the working directory.

User ID

Specifies the user account to be used while executing the operator on the host. Overrides the user specified in the operator category level properties.

Password

Specifies the password for the user.

Output Parameters

fileName

workingDir

userID

password

Copy File Operator

The Copy File operator copies source to destination. The source and destination can be a file or a directory.

Input Parameters

Source File/Directory Name

Specifies the file or directory to copy.

Destination File/Directory

Specifies the file or directory to copy.

Working Directory

Specifies the working directory to execute this operation.

Notes: If the working directory is not specified, the user home directory becomes the working directory.

The file path can be absolute or relative to the working directory.

User ID

Specifies the user account to use while executing the operator on the host. Overrides the user specified at the operator category level.

Password

Specifies the password for the user.

Notes:

- If the destination does not exist and the source is a file, then the destination is assumed to be a file.

- If the destination does not exist and the source is a directory, then the destination is assumed to be a directory.

- On a Windows host, a "cannot perform cyclic copy" error is thrown if the source directory contains the destination directory.

Output Parameters

fileName

destinationFileName

workingDir

userID

password

Create Folder Operator

The Create Folder operator provides the functionality to create a directory. The operator does not throw an error if the directory already exists and the operator creates directories recursively as needed.

Input Parameters

Directory Name

Specifies the directory to create.

Default Shell

Permission Modes (for UNIX only)

Sets permission mode. This input is valid for a UNIX host only.

Working Directory

Specifies the working directory to execute this operation.

Notes: If the working directory is not specified, the user home directory becomes the working directory.

The file path can be absolute or relative to the working directory.

User ID

Specifies the user account to use while executing the operator on the host. Overrides user specified in the operator category level properties.

Password

Specifies the password for the user.

Output Parameters

fileName

defaultShell

permission

workingDir

userID

password

Decompress File Operator

The Decompress File operator extracts a compressed file/directory.

Prerequisites

■ On a Windows host, the target computer must have the WZZIP command line utility installed. The WZZIP command line utility is a free add-on for users of WinZip 12 standard or pro with a valid license.

■ On UNIX environments, this operator uses the gzip utility.

Input Parameters

File/Directory Name to be Uncompressed

Specifies the file/directory to uncompress.

Working Directory

Specifies the working directory to execute this operation.

Notes: If the working directory is not specified, the user home directory is the working directory.

The file path can be absolute or relative to the working directory.

User ID

Specifies the user account to use while executing the operator on the host. Overrides user specified in the operator category level properties.

Password

Specifies the password for the user.

Output Parameters

compressFileName

workingDir

userID

password

Delete File Operator

The Delete File operator removes (deletes) a file or directory.

Input Parameters

Source File/Directory Name

Specifies the file or directory to delete.

Working Directory

Specifies the working directory to execute this operation.

Notes: If the working directory is not specified, the home directory of the user is the working directory.

The file path can be absolute or relative to the working directory.

User ID

Specifies the user account to use while executing the operator on the host. Overrides the user specified in the operator category--level properties.

Password

Specifies the password for the user.

Notes:

- An attempt to delete a non-existent file or directory results in an error.

- An attempt to delete a file or directory with insufficient permission results in an error.

Output Parameters

fileName

workingDir

userID

password

Get Directory Content Operator

The Get Directory Content operator builds a list of paths for all folders or files within a specified directory that match a search condition.
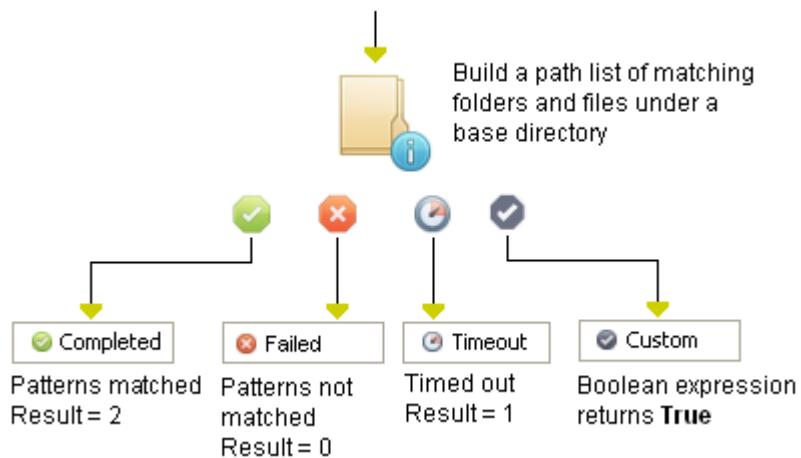
Input Parameters

Base directory

Specifies the path for the directory in which to start the search.

File path/name mask

Specifies the pattern that the operator detected.

Case sensitive pattern matching

When checked, matches upper-case and lower-case characters when searching for a pattern. If unchecked, the letters in a pattern match both upper and lower-case characters.

Sort items by last modification time

When checked, sorts the folders or files within a specified directory by the last modification time.

Match pattern on file/directory name

When checked, matches only file or directory names, instead of anywhere in a path.

Directories included in results

Select from one of the following options:

- All directories under the base directory

- Directories that include matching files

- Directories with matching path or name

Recursion level

The number of directory levels to go down to when matching files or directories.

State timer (secs)

The minimum interval for which the condition must be maintained.

Output Parameters

DirectoryCount

DirectoryList

FilesCount

FilesList

fileName

Pattern

caseSensitive

timeSort

matchPath

dirFilterOptions

recursionLevel

stateTimer

timeOut

Get File Attributes Operator

The Get File Attributes operator reads the attributes for a specified file and saves them as variables (such as FileExists, FileSize, IsFile, IsLink) in its operator dataset.
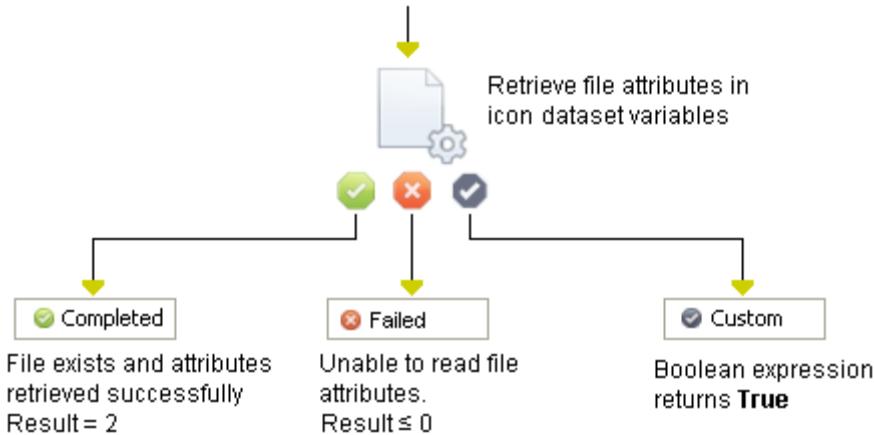
To access an attribute in a CA Process Automation expression, use the following syntax:

Process.Operator_name.field_name

*Process a*ccesses the process dataset.

Operator_name specifies the operator dataset.

field_name specifies the attribute in the operator dataset.

Input Parameters

File/directory name

Specifies the full path for a file or directory. The location must be accessible to the File Management operators on the touchpoint where they are running at run time.

For example:

/tmp/IT PAM/scripts/backup_ora1.log

If you specify a file or directory without specifying the full path, the File Management operators use <*install_dir*>\server\c2o as the relative path for the specified file or directory. In most cases, you can use the slash mark (/) character in a path.

Output Parameters

@FileName

The full path for the file as calculated from the "File Name" expression.

FileExists

1 if a file exists, 0 if it does not exist.

FileName

FileTime

Specifies the file time.

FileDate

Specifies a file date.

IsDirectory

1 if describing a directory, 0 if not describing a directory.

IsFile

1 if describing a file, 0 if not describing a file.

IsLink

1 if the item is a symbolic link in UNIX, 0 otherwise.

FileOtherRead

FileOtherWrite

FilePermission

FileSize

The size of the file in bytes.

FileSizeKB

The size of the file in kilobytes (KB = 1,024 bytes). A fraction of a kilobyte is counted as one kilobyte.

FileSizeMB

The size of the file in megabytes (MB = 1,024 KB). A fraction of a megabyte is counted as one megabyte.

FileGroup

FileOwner

FileGroupExec

> 1 if the group can execute, 0 if the group cannot execute.

FileGroupRead

> 1 if the group can read, 0 if the group cannot read.

FileGroupWrite

> 1 if the group can write, 0 if the group cannot write.

FileOtherExec

> 1 if others can execute, 0 if other cannot execute.

FileOwnerExec

> 1 if the owner can execute, 0 if the owner cannot execute.

FileOwnerRead

> 1 if the owner can read, 0 if the owner cannot read.

FileOwnerWrite

> 1 if the owner can write, 0 if the owner cannot write.

IsSpecial

> 1 if a special system file, 0 if not a special system file.

Note: The definition of a special system file is platform-dependent.

Example

This operator can correspond to a socket or fifo on UNIX platforms.

Others (for example, FileOtherExec) refer to users that are not the owner or not in the defined group for the specified permission.

Monitor File Operator

The Monitor File operator defines a wait for a condition on a file or directory. This operator lets you delay processing in a process for the existence or absence of a file or directory.
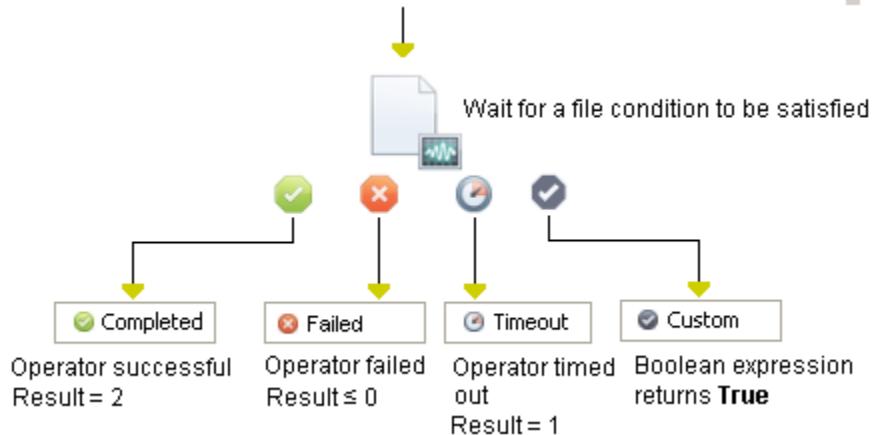
The available conditions are:

- The existence of a file, and optionally, a minimum size (in bytes).

- The absence of a file.

- The presence of strings matching a pattern in a file.

A stability timer specifies the minimum delay during which the condition must continuously hold before returning that the operator is successfully completed. For example, this operator can specify the minimum size of a file that an FTP transfer sends.

Tests on a file are discrete. This operator does not indicate that a condition is continuously present. Rather, the Watch File operator indicates that the condition is met at every test interval that the operator performs.

Input Parameters

File/directory name

Specifies the full path for a file or directory to watch. The location must be accessible to the File Management operators on the touchpoint where they are running at run time.

If you specify a file or directory without specifying the full path, the File Management operators use the CA Process Automation Bin directory as the relative path for the specified file or directory. Typically, you can use the slash mark (/) character in a path.

State timer (secs)

Specifies the minimum delay in seconds during which the specified condition must continuously hold before CA Process Automation executes branches for the Completed state.

Condition

Specifies the condition to execute branches for the Completed state:

■ Presence requires that the file exists and the file size is greater than the "Minimum file size".

■ Absence requires that the file does not exist.

■ Pattern Matching specifies that a pattern of characters occurs in the contents of a specified file or in the names of files in a specified directory.

Minimum file size

If "Presence" is specified for Condition, this option specifies the minimum file size in bytes for this operator to execute branches for the Completed condition.

Pattern

If Pattern matching is specified for Condition, this option specifies a regular expression that returns the pattern searched for by the operator (see Using Masks to Specify Patterns in Strings in the *Content Designer Guide*).

To match any number of multiple lines, you can use the \n escape in the Pattern field. The following example matches lines starting with "Log", followed by any number of intervening new lines and a string of text ending in "Error=89":

```
"Log.*\n.*Error=89"
```

If you are accustomed to using escape characters in programming languages, this \n escape matches any number of new lines on either Windows or UNIX. This escape does not match a single-line feed character.

Separator

Specifies the character that delimits the zone in Pattern to save to the variables that Variable names specify.

Start from end of file

Starts searching from the end of a specified file to find the last occurrence of a pattern in a file. This option lets you match the newest messages in a file.

Case sensitive pattern matching

Considers upper-case and lower-case characters when searching for a pattern. If you do not select this check box, the letters in a pattern match both upper and lower-case characters.

File Search Offset

Specifies a starting position for a search.

To perform a looped pattern match, you can use the MatchPos and MatchEntry variables from the operator dataset to start where the previous match left off:

`Process.Operator_name. MatchedPos+ Len(Process.Operator_name.MatchedEntry)`

Variable names

Specifies the variable names in which to save text that matches the delimited zones in the pattern. Operator dataset variables are accessed through the process dataset, using the keyword Process. For example, specifying the variable names Level and Code would assign extracted substring values to the operator dataset variables Process.Operator_name.Level and Process.Operator_name.Code. You can add, remove, and order the variables that are used to store matched strings using the toolbar.

Output Parameters

LastRead POs

MatchedEnd

MatchedEntry

MatchedPos

fileName

stateTimer

condition

minFileSize

Pattern

Separator

startFromEnd

caseSensitive

fileSearchOffset

variableNames

Example

This operator can wait for an outbound operator to spool and to delete a file. A problem would be indicated if a file exists in the spool directory longer than a specified duration. The process could then execute an alert to notify an operator of the problem.

Read from File Operator

The Read from File operator reads the content of the file into a dataset variable. The user can also read specific lines.

The dataset variable can be a string or a string array. If the dataset variable is an array, the maximum length can be 1024. Lines exceeding this limit are ignored and new dataset variable field "warnings" are created after execution. This new warnings dataset contains the warning message.

Input Parameters

Source File Name

Specifies the file to read.

Return file contents in a string array

Specifies if the dataset variable is a string array. If checked, the dataset variable is assumed to be a string.

From Line Number

Specifies the line number in the file from which the file content must be read. If this field is left blank, this operator reads from line number 1. This field must contain positive non-zero values only.

To Line Number

Specifies the line number in the file until where the content must be read. This field must contain positive non-zero values only. If it is left blank, this operator reads to the end of file.

Dataset Variable Name

Specifies the name for the dataset variable.

This field length can be a maximum of 1024. Lines exceeding this limit are ignored and a new dataset variable field named "warnings" is created after execution (which contains the warning message).

Working Directory

Specifies the working directory to execute this operation.

Notes: If the working directory is not specified, the user home directory becomes the working directory.

The file path can be absolute or relative to the working directory.

User ID

Specifies the user account to use while executing the operator on the host. Overrides the user who is specified in the operator category configuration.

Password

Specifies the password for the user ID.

File Encoding

Specifies the encoding scheme that is used to read from the file (UTF-8, UTF-16, US-ASCII, Windows-1250, Windows-1252, Shift_JIS).

Output Parameters

DatasetVariable

The dataset variable length can be a maximum of 1024 for an array. Lines exceeding this limit will be ignored and a new dataset variable field "warnings" will be created after execution, which contains the warning message.

fileName

isReturnAsStringArray

fromLineNumber

toLineNumber

datasetVariableName

workingDir

userID

password

fileencoding

Example (Read from File Operator)

This example lets you read from a file named ReadFile.txt. The file content is as follows:
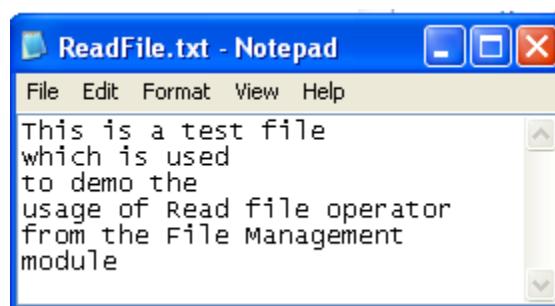
This example considers the following scenarios:

- Read from the file

- Read from the file and save the file content as an array

- Read specific lines from the file

- Read from the file and save the file content in a dataset

Follow these steps:

1. Design a process with the *Read from File* operator as shown in the following illustration:

2. Double-click the *Read from File* operator to open the Read from File properties, and select the *Read file into Dataset Variable* panel.

   To read from the file

3. Enter the following file path in the Source File Name.

   (Windows) C:\\ReadFile.txt

   (UNIX) /root/readfromfile.txt

4. Run the process.

5. Open the Operation Results to view the DatasetVariable value as shown in the following illustration:

To read from file and save the file content as an array

6. Select the *Return file contents in a string arra*y check box.

    The file content is available as an array as shown in the following illustration:

To read specific lines in the file

7. Enter 1 in the From Line Number file to read form the line one from the ReadFile.txt file.

8. Enter 4 in the To Line Number to read until the fourth line in the ReadFile.txtfile.

    The file content is read and saved in the dataset form line 1 through 4 as shown in the following illustration:

To read the file and save the file content in a dataset variable

9. Enter DataRead as Dataset Variable Name.

    The file content is saved in the DataRead dataset and not in the system dataset as shown in the following illustration:

DatasetVariable	[0]
◢ 🖿 DataRead	[6]
[0]	This is a test file
[1]	which is used
[2]	to demo the
[3]	usage of Read file operator
[4]	from the File Management
[5]	module

You can also specify the User ID and password of the user account to grant execute permission on a process. When you provide User ID and password values at the operator level you override the values that are defined in the Require user credentials field. You define the Require user credentials field in File Management properties.

Note: Ensure that you grant *Read* permission to the user and *Read and Execute* permissions to the PAM installation directory to run the ReadFile service operation. For more information about how to configure file management, see the Configure File Management section in the *Content Administrator Guide.*

Rename File Operator

The Rename File operator provides functionality to rename a file or a directory.

Input Parameters

Source File/Directory Name

Specifies the file/directory to rename.

New File/Directory Name

Specifies the new name for the file/directory.

Working Directory

Specifies the working directory to execute the operator.

Notes: If the working directory is not specified, the user home directory becomes the working directory.

The file path can be absolute or relative to the working directory.

User ID

Specifies the user account to use while executing the operator on the host. Overrides user specified in the operator category level parameters.

Password

Specifies the password for the user ID.

Note: For UNIX, if the destination location is other than the source, the file moves to the destination location.

Output Parameters

fileName

newFileName

workingDir

userID

password

Search File Content Operator

The Search File Content operator extracts information from fairly small files. This operator searches the content of a text file or directory for a string that matches a specified pattern. The pattern can indicate substrings to extract into operator dataset variables. Delimiters in the pattern can indicate the zones to extract into the variables.

■ For a file, this operator searches the contents of the file for the specified pattern.

■ A group search is recommended to search large files that contain gigabytes of information efficiently.

Scan file for matching string pattern and extract specified substrings into icon dataset variables.

Completed — String pattern found Result = 2

Failed — String pattern not found Result ≤ 0

Custom — Boolean expression returns **True**
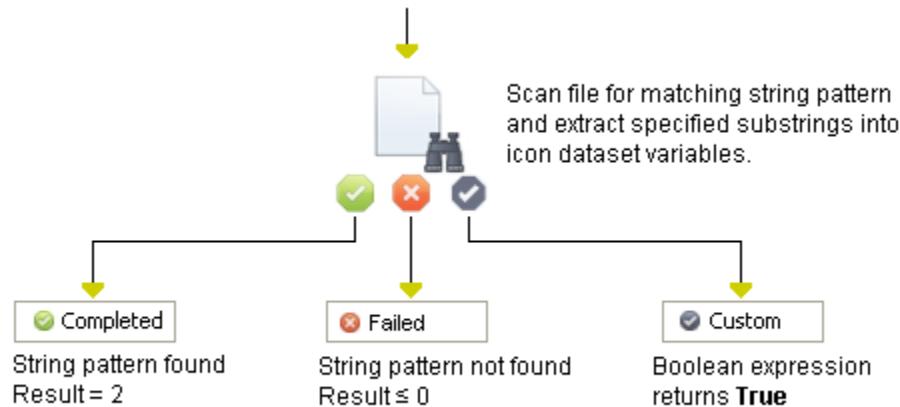
Input Parameters

File/directory name

Specifies the path for the file or directory to scan for pattern matches. The path must be accessible to the File Management operators on the touchpoint where they are running. For a file, the operator searches the contents of the file for the specified pattern.

If you specify a file or directory without specifying the full path, the File Management operators use <*install_dir*>\server\c2o as the relative path for the specified file or directory. In most cases, you can use the slash mark (/) character in a path.

Pattern

Specifies the pattern searched for by the operator (see Using Masks to Specify Patterns in Strings in the *Content Designer Guide*).

The pattern uses the number symbol (#) used as the separator to return values for the variables Level and Code:

```
"BACKUP LEVEL #.*# - CODE #.*#"
```

From the string "BACKUP LEVEL A400 - CODE FSC137.0359", this pattern would assign the substrings "A400" to the variable Level and "FSC731.0359" to the variable Code. The assignment is made in the same order as the variables are defined under Variable names.

To match any number of multiple lines, you can use the \n escape in the Pattern field. The following example matches lines starting with "Log", followed by any number of intervening new lines and a string of text ending in "Error=89":

```
"Log.*\n.*Error=89"
```

If you are accustomed to using escape characters in programming languages, this \n escape matches any number of new lines on either Windows or UNIX. This escape does not match a single line feed character.

Separator

Specifies a character that is used to delimit the zone to save to the variables that the variable names specify, such as the # symbol.

Start from end of file

Select this check box to start searching from the end of a specified file. This option is used to find the last occurrence of pattern in a file. This option lets you match the newest messages in a file.

Case sensitive pattern matching

Select this check box to take into account upper-case and lower-case characters when searching for a pattern. If you do not select this check box, the letters in a pattern match both upper and lower case characters.

File Search Offset

Specifies a starting position for a search. The value represents the number of characters from the start of the file unless you select the Start from end of file check box. In that case, the value represents the number of characters from the end of the file.

To perform a looped pattern match, you can use the MatchPos and MatchEntry variables from the operator dataset of an earlier Search File Content operator to start where the previous match left off:

```
Process.Operator_name.MatchPos + Len(Process.Operator_name.MatchEntry)
```

Variable names

Specifies the variable names in which to save text that matches the delimited zones in the pattern. Delimited zones are saved to the listed variables in the order defined in the variable list. Operator dataset variables are accessed through the process dataset, using the keyword process.

For example, specifying the variable names Level and Code to assign extracted substring values to the operator dataset variables Process.Operator_name.Level and Process.Operator_name.Code.

You can add, remove, and order the variables that are used to store matched strings using the toolbar.

Output Parameters

fileName

Pattern

Separator

startFromEnd

caseSensitive

fileSearchOffset

variableNames

LastReadPos

MatchedEnd

MatchedEntry

MatchedPos

Update File Ownership Operator

The Update File Ownership operator changes the user and/or group ownership of each given file. Only a super-user can change the owner and group to which a file belongs. This operator is supported on a UNIX host only.

Input Parameters

User Name

Specifies the owner of the file.

Group Name

Specifies the group to which the file belongs.

Note: The user must provide an input for at least one of the User name or Group name fields.

Source File/Directory Name

Specifies the name of the file or directory whose ownership is changing.

Recursive

When checked, specifies to change file ownership recursively.

Working Directory

Specifies the working directory to execute this operation.

Notes: If the working directory is not specified, the user home directory becomes the working directory.

The file path can be absolute or relative to the working directory.

User ID

Specifies the user account to use while executing the operator on the host. This field overrides the user specified in the operator category- level properties.

Password

Specifies the password for the user ID.

Output Parameters

userName

groupName

fileName

isRecursive

workingDir

userID

password

Update File Permission Operator

The Update File Permission operator changes the permissions of each given file according to mode, which can be either an octal number representing the bit pattern for the new permissions or a symbolic representation of changes to make, (+-= rwxXstugoa). This operator is supported on a UNIX host only.

Input Parameters

Source File/Directory Name

Specifies the name of file or directory whose permission is changing.

Permission (Modes)

Specifies permission or mode for the file, which can be either an octal number representing the bit pattern for the new permissions or a symbolic representation of changes to make, (+-= rwxXstugoa).

Recursive

When checked, specifies to change files and directories recursively.

Working Directory

Specifies the working directory to execute this operator.

Notes: If the working directory is not specified, the user home directory becomes the working directory.

The file path can be absolute or relative to the working directory.

User ID

Specifies the user account to use while executing the operator on the host. Overrides the user specified in the operator category level properties.

Password

Specifies the password for the user ID.

Output Parameters

fileName

permission

isRecursive

workingDir

userID

password

Update File Timestamp Operator

The Update File Timestamp operator changes file timestamps, such as, update the access and modification times of each file to the current time or user- specified timestamp. This operator is supported on a UNIX host only.

Input Parameters

Source File/Directory Name

Specifies the name of file or directory whose timestamp is changing.

Timestamp ([[CC]YY]MMDDhhmm[.ss]):

Use this field in ([[CC]YY]MMDDhhmm[.ss]) format instead of the current time.

Where each pair of letters represents the following information:

CC

Specifies the first two digits of the year (the century).

YY

Specifies the second two digits of the year. If "YY" is specified, but "CC" is not, a value for "YY" between 69 and 99 results in a "CC" value of 19. Otherwise, a "CC" value of 20 is used.

If the "CC" and "YY" letter pairs are not specified, the values default to the current year.

MM

Specifies the month of the year, from 1 to 12.

DD

Specifies the day of the month, from 1 to 31.

hh

Specifies the hour of the day, from 0 to 23.

mm

Specifies the minute of the hour, from 0 to 59.

ss

The second of the minute, from 0 to 61.

If the "ss" letter pair is not specified, the value defaults to 0.

Change Access Time

Specifies to change the access time. The default is checked.

Change Modification Time

Specifies to change the modification time. The default is checked.

Working Directory

Specifies the working directory to use to execute this operator.

Notes: If the working directory is not specified, the home directory of the user is the working directory.

The file path can be absolute or relative to the working directory.

User ID

Specifies the user account to use while executing the operator on the host. Overrides user specified in the operator category- level properties.

Password

Specifies the password for the user ID.

Output Parameters

fileName

timeStamp

isChangeAccessTime

isChangeModificationTime

workingDir

userID

password

Write File Operator

The Write File operator writes the content of the dataset variable to a file. The dataset variable can be a string or a string array. This operator also provides an option to either overwrite or append contents to an existing file.

Input Parameters

File Contents as Array

When checked, specifies that the dataset variable is a string array.

File Contents

If you do not select the File Contents As Array check box, write the contents in this field to the specified file.

File Contents as Array

If you select the File Contents As Array check box, write the contents in this field to the specified file, with each index as a new line in the file.

You can add, remove, and order the variables using the toolbar.

Destination File Name

Specifies the name of the destination file.

Note: If the destination file does not exist, the file is created before writing data to the file.

Append (if file exists)

Specifies that the content of the dataset variable is appended to a file if it already exists. The default is checked.

Working Directory

Specifies the working directory to execute this operator.

Notes: If the working directory is not specified, the home directory of the user becomes the working directory.

The file path can be absolute or relative to the working directory.

User ID

Specifies the user account to use while executing the operator on the host. Overrides the user-specified in the operator category level parameters.

Password

Specifies the password for the user ID.

File Encoding

Specifies the encoding scheme that is used to write to the file (UTF-8, UTF-16, US-ASCII, Windows-1250, Windows-1252, Shift_JIS).

Output Parameters

fileContentsMode

fileContents

fileContentsAsArray

fileName

isFileAppend

workingDir

userID

password

fileencoding

Chapter 10: File Transfer

The File Transfer operators provide file transfer operators (FTP/SFTP).

Use the File Transfer operators to manage directories and files such as FTP or an SFTP client. These operators connect to standard FTP servers on target computers. The remote host for all File Transfer operators must have a configured FTP server.

Important! With the exception of the TFTP Download File operator and the TFTP Upload File operator, the following conditions apply to all operators in this category when running them on a remote Windows host through a proxy touchpoint:

- Use UNIX-style paths for fields which are path-related (forward slashes and no drive letter).

- Each SSH server can have a different location for its "root" directory. The permitted commands that are relative to that root directory can vary too.

Create Directory Operator

The Create Directory operator creates a directory on the remote file system.

To allow the operator to create a directory, the specified user credentials must have the appropriate change directory and write permissions on the remote host. The remote host must have a configured FTP server.
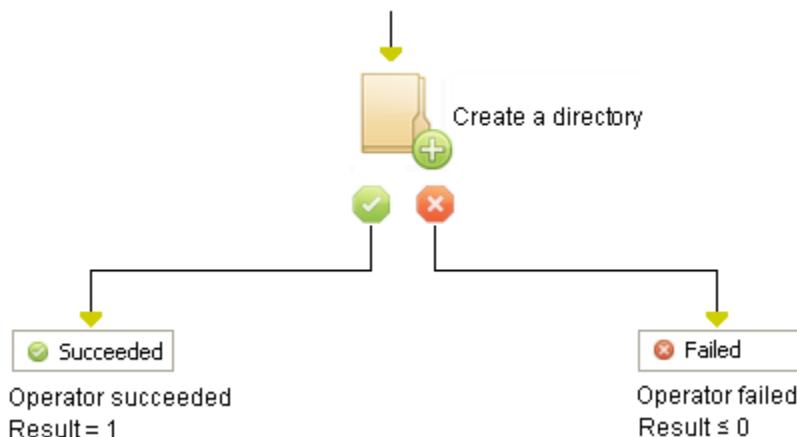
Input Parameters

Remote path

Specifies the path for the directory to create on the remote host. For example:

`/temp/IT PAM/scripts`

The parent directory (/temp/IT PAM in the example) must exist for the File Transfer operators to complete this operator successfully. The relative path can also be specified in this field and the path is relative to the home directory of the FTP user.

Remote host

Specifies the IP address or FTP URL for the remote host. For an FTP site on your company intranet, you can specify the server name (//servername) for the FTP site.

By default, FTP sites use port 21. However, you can assign a private port to an FTP site. Private ports range from 49152 to 65535. To specify a private FTP port, add a colon (:) and then a private port number to the end of the address. For example, the following specifies port 50021 on a remote FTP server:

`172.24.36.107:50021`

Remote user ID

Specifies a user ID to access the remote FTP host.

Remote user password

Specifies the password to access the remote FTP host.

Use secure FTP (SFTP)

Opens a secure FTP (SFTP) session. SFTP is similar to FTP, but unlike FTP, the entire session is encrypted. No passwords are sent in clear text form, so they are much less vulnerable to third-party interception.

Specify an optional SITE command

Specifies to use the SITE command to invoke services that are specific to the host system. Then use the Site parameters field to specify an expression that returns parameters for the SITE command.

Site parameters

Enter a CA Process Automation expression that returns parameters for the SITE command.

Output Parameters

remotedir

remoteHost

remoteUserId

remoteUserPassword

secureFtp

siteCommand

siteParameters

Delete Directory Operator

The Delete Directory operator deletes a specified directory from the remote file system. The specified directory must be empty before the process runs this operator.

Deleting a directory requires that the specified user credentials have the appropriate change directory and write permissions on the remote host. The remote host must have a configured FTP server.
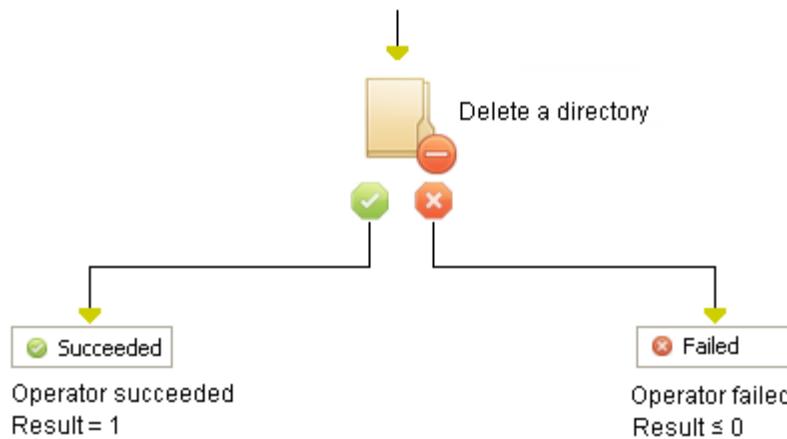
Delete Remote Directory Properties

Remote path

Specifies the path for the directory to delete on the remote host. For example: /temp/IT PAM/scripts. The specified directory must be empty for the File Transfer operators to complete this operator.

Remote host

Specifies the IP address or FTP URL for the remote host. For an FTP site on your company intranet, you can specify the server name (//servername) for the FTP site.

By default, FTP sites use port 21. However, you can assign a private port to an FTP site. Private ports range from 49152 to 65535. To specify a private FTP port, add a colon (:) and then a private port number to the end of the address. For example, the following specifies port 50021 on a remote FTP server:

```
172.24.36.107:50021
```

Remote user ID

Specifies a user ID to access the remote FTP host.

Remote user password

Specifies the password to access the remote FTP host.

Use Secure FTP (SFTP)

Opens a secure FTP (SFTP) session. SFTP is similar to FTP, but unlike FTP, the entire session is encrypted. No passwords are sent in clear text form, so they are much less vulnerable to third-party interception.

Specify an optional SITE command

Specifies to use the SITE command to invoke services that are specific to the host system. Then use the Site parameters field to specify a CA Process Automation expression that returns parameters for the SITE command.

Remove all files/subdirectories under the target directory

This option is used to delete a directory that is not empty. If this check box is selected, all the directories under the specified directory are deleted including the specified directory. If this check box is unchecked, any attempt to delete a directory that is not empty results in service operator failure.

Site Parameters

Enter a CA Process Automation expression that returns parameters for the SITE command.

Output Parameters

remotedir

remoteHost

remoteUserId

remoteUserPassword

secureFtp

siteCommand

forceDelete

siteParameters

Delete File Operator

The Delete File operator removes a specified file from a remote location. The remote host must have a configured FTP server. To allow the operator to delete a file, the specified user credentials must have the appropriate write permissions on the remote host.
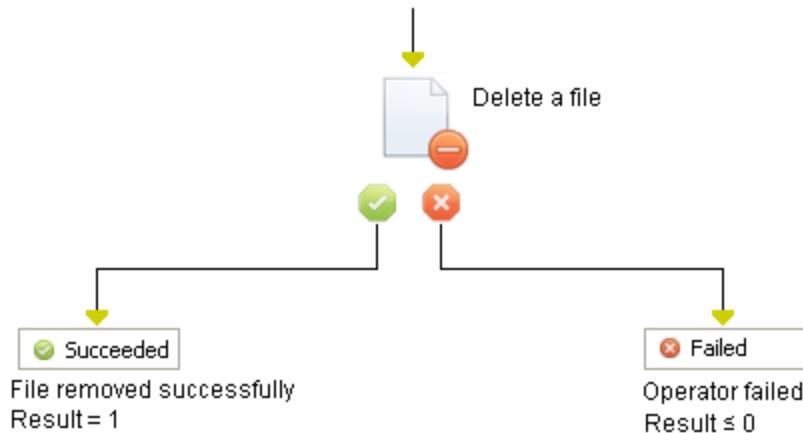
Input Parameters

Remote path

Specifies the path for the file to delete on the remote FTP host. For example: /tmp/IT PAM/scripts/script_ora1.sh.

Remote host

Specifies the IP address or FTP URL for the remote host. For an FTP site on your company intranet, you can specify the server name (//servername) for the FTP site.

By default, FTP sites use port 21. However, you can assign a private port to an FTP site. Private ports range from 49152 to 65535. To specify a private FTP port, add a colon (:) and then a private port number to the end of the address. For example, the following specifies port 50021 on a remote FTP server:

`172.24.36.107:50021`

Remote user ID

Specifies a user ID to access the remote FTP host.

Remote user password

Specifies the password to access the remote FTP host.

Use Secure FTP (SFTP)

Opens a secure FTP (SFTP) session. SFTP is similar to FTP, but unlike FTP, the entire session is encrypted. No passwords are sent in clear text form, so they are much less vulnerable to third-party interception.

Specify an optional SITE command

Specifies to use the SITE command to invoke services that are specific to the host system. Then use the Site parameters field to specify an expression that returns parameters for the SITE command.

Site parameters

Enter a CA Process Automation expression that returns parameters for the SITE command.

Output Parameters

remoteFile

remoteHost

remoteUserId

remoteUserPassword

secureFtp

siteCommand

siteParameters

Download File Operator

The Download File operator copies a file from a remote location. This operator corresponds to the FTP get command.

The remote host must have a configured FTP server. To get a remote file, the specified user credentials must have the appropriate change directory and read rights on the file.
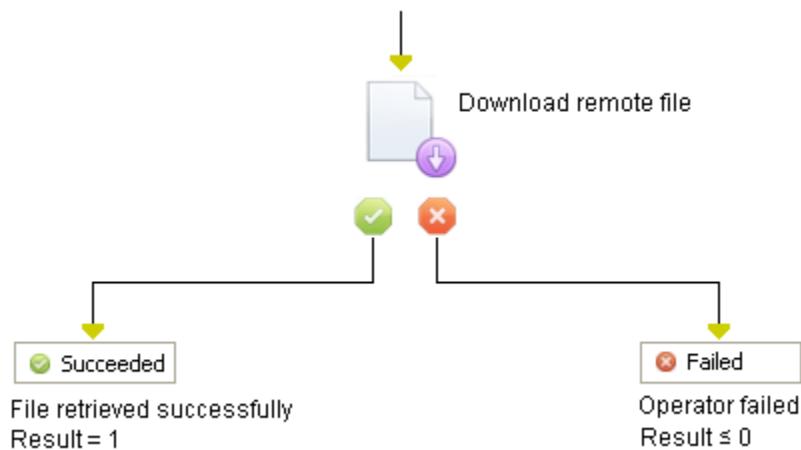
Get Remote File Properties

Local file

Specifies the location to save the file locally. For example: /IT PAM/import/script_ora1.sh.

The location must be valid at run time on the touchpoint running the File Transfer operators.

When you use Windows file nomenclature, backslashes must be escaped as follows:

C:\\IT PAM\\import\\script_ora1.bat

We recommend using "normalized" file names, with slash marks (/), even when specifying a path on a Windows host. For example: C:/IT PAM/import/script_ora1.bat.

Remote file

Specifies the full path for the file on the remote FTP host. For example: /tmp/IT PAM/scripts/script_ora1.sh.

Remote host

Specifies the IP address or FTP URL for the remote host. For an FTP site on your company intranet, you can specify the server name (//servername) for the FTP site.

By default, FTP sites use port 21. However, you can assign a private port to an FTP site. Private ports range from 49152 to 65535. To specify a private FTP port, add a colon (:) and then a private port number to the end of the address. For example, the following specifies port 50021 on a remote FTP server:

`172.24.36.107:50021`

Remote user ID

Specifies the user ID to access the remote FTP host.

Remote user password

Specifies the password to access the remote FTP host.

Binary transfer

Specifies to use the FTP binary mode for transferring binary files. For example:

Select a check box with the following types of files:

- Executable files

- SPSS System files

- SAS Transport files

- Stata datasets

- Graphics files

Convert from ASCII to EBCDIC

Specifies to convert ASCII character code to EBCDIC before the file transfer. EBCDIC is used in a z/OS environment, where the file needs to be readable in a z/OS host.

Use Secure FTP (SFTP)

Opens a secure FTP (SFTP) session. SFTP is similar to FTP, but unlike FTP, the entire session is encrypted. No passwords are sent in clear text form, so they are much less vulnerable to third-party interception.

CA Process Automation uses the SSH2/SFTP protocol with user name/password authentication. The SSH2/SFTP protocol only supports binary transfers.

Specify an optional SITE command

Uses the SITE command to invoke services that are specific to the host system. Then use the SITE parameters field to specify a CA Process Automation expression that returns parameters for the SITE command. This option is used, for example, to dimension files on a target MVS system.

Site parameters

Enter a CA Process Automation expression that returns parameters for the SITE command.

Output Parameters

localFile

remoteFile

remoteHost

remoteUserId

remoteUserPassword

transferMode

secureFtp

siteCommand

siteParameters

Get File Information Operator

The Get File Information operator gets file attributes for a remote file and saves them to variables (such as Permissions, Size, and Group) in its operator dataset.

The list of meaningful attributes is file system-dependent. To view the attributes for a specified file or folder, view the operator's dataset.

The remote host must have a configured FTP server. The specified user credentials must have the appropriate read permissions on the remote host.

Input Parameters

Remote file

Specifies the path for the file on the remote FTP host. For example: "/tmp/IT PAM/scripts/script_ora1.sh"

Remote host

Specifies the IP address or FTP URL for the remote host. For an FTP site on your company intranet, you can specify the server name (//servername) for the FTP site.

By default, FTP sites use port 21. However, you can assign a private port to an FTP site. Private ports range from 49152 to 65535. To specify a private FTP port, add a colon (:) and then a private port number to the end of the address. For example, the following specifies port 50021 on a remote FTP server:

`172.24.36.107:50021`

Remote user ID

Specifies a user ID to access the remote FTP host.

Remote user password

Specifies the password to access the remote FTP host.

Use secure FTP (SFTP)

Opens a secure FTP (SFTP) session. SFTP is similar to FTP, but unlike FTP, the entire session is encrypted. No passwords are sent in clear text form, so they are much less vulnerable to third-party interception.

Specify an optional SITE command

Specifies to use the SITE command to invoke services that are specific to the host system. Then use the Site parameters field to specify a CA Process Automation expression that returns parameters for the SITE command.

Site parameters

Enter a CA Process Automation expression that returns parameters for the SITE command.

Output Parameters

remoteFile

remoteHost

remoteUserId

remoteUserPassword

secureFtp

siteCommand

siteParameters

FileExists

FileName

FileTime

IsDirectory

IsFile

IsSpecial

FileGroup

FileGroupExec

FileGroupRead

FileGroupWrite

FileOtherExec

FileOtherRead

FileOtherWrite

FileOwner

FileOwnerExec

FileOwnerRead

FileOwnerWrite

FilePermission

FileSize

FileSizeKB

FileSizeMB

Move File Operator

The Move File operator moves a file from one remote location to another remote location on the same server. You can use it to rename a file by specifying the same paths for both the old name and the new name.

The remote host must have a configured FTP server. To move a file, the specified user credentials must have the appropriate change directory and read rights on the file.
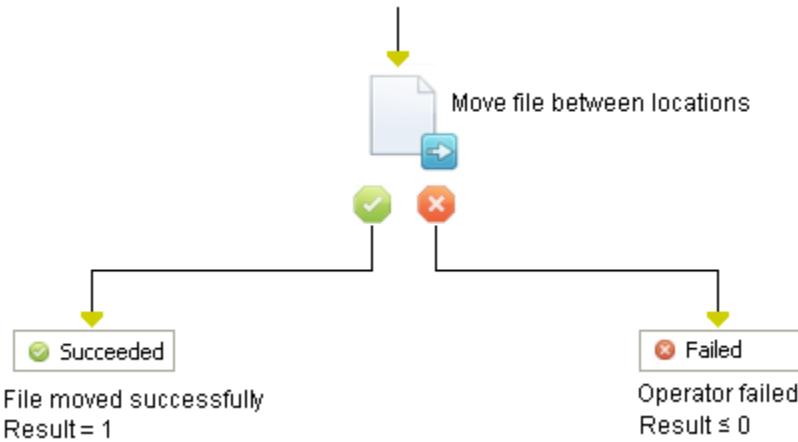
Input Parameters

New name

Specifies the path and new name for the file on the remote FTP server.

"/tmp/IT PAM/scripts/archived/IT PAM.new"

Current name

Specifies the existing path and name for the file on the remote FTP host. For example: /tmp/IT PAM/scripts/IT PAM.old.

Remote host

Specifies the IP address or FTP URL for the remote host. For an FTP site on your company intranet, you can specify the server name (//servername) for the FTP site.

By default, FTP sites use port 21. However, you can assign a private port to an FTP site. Private ports range from 49152 to 65535. To specify a private FTP port, add a colon (:) and then a private port number to the end of the address. For example, the following specifies port 50021 on a remote FTP server:

`172.24.36.107:50021`

Remote user ID

Specifies a user ID to access the remote FTP host.

Remote user password

Specifies the password to access the remote FTP host.

Use secure FTP (SFTP)

Opens a secure FTP (SFTP) session. SFTP is similar to FTP, but unlike FTP, the entire session is encrypted. No passwords are sent in clear text form, so they are much less vulnerable to third-party interception.

Specify an optional SITE command

Specifies to use the SITE command to invoke services that are specific to the host system. Then use the Site parameters field to specify a CA Process Automation expression that returns parameters for the SITE command.

Site parameters

Enter a CA Process Automation expression that returns parameters for the SITE command.

Output Parameters

newName

currentName

remoteHost

remoteUserId

remoteUserPassword

secureFtp

siteCommand

siteParameters

TFTP Download File Operator

The TFTP Download File operator receives a file from a host on a network through the TFTP protocol. The host must have a running TFTP server.

Details on the TFTP protocol follow:

- Uses UDP to transfer the data.

- Issues sends and waits for ACKs.

■ Typically initiates data transfer on port 69.

■ Typically sends data in a block size of 512b or smaller.

■ Uses no authentication or authorization.

Important! Test TFTP functionality outside CA Process Automation before incorporating it into CA Process Automation process flows. Firewalls or routing can block UDP to port 69 by default. Typically, the TFTP service is also either not installed or disabled.

Input Parameters

Remote Hostname

The host name or the IP address of the remote host.

Remote UDP Port for TFTP

The UDP port of the TFTP service on the remote host. If this parameter is specified, this value overrides the value of the operator category field: Default UDP Port for Trivial FTP. If none of these fields are specified, the operator uses the default value of 69.

Remote File name

The name of the remote file to obtain from the remote host. This parameter is not the path of the remote file on the remote host. Depending on its setup, the TFTP server retrieves the file from its base directory in the remote host.

Data Transfer Type

BIN for binary or ASCII for ASCII (text). If this type is not specified, the operator uses the default value: ASCII. If any other value is specified (other than ASCII or BIN), the operator uses the default value of ASCII.

Local File to Download

The fully qualified path of the local file (where you save the file retrieved through TFTP).

Local Port Number (0 for anonymous port)

The local port number to use when retrieving data from the remote host/port. If 0, an anonymous port is used. If the port is specified and the port is unavailable, the operation could fail. If this number is not specified, the operator uses the default value: 0.

Timeout (sec)

The timeout value to use when opening the connection to the TFTP server. If this number is not specified, the value defaults to 20.

Maximum Retries after TFTP Timeout

The maximum number of times to retry the download file operation (not the entire CA Process Automation operator) after a TFTP timeout. If this number is not specified, the operator uses the default value: 5.

Output Parameters

Result:

■ 1: If the operator finished successfully.

■ -1: If the operator failed.

Reason:

■ Completed: If the operator finished successfully,

■ An error message if the operator failed.

remoteSSHHost

PORT

RemoteURL

TransferType

LocalFileName

LocalPort

Timeout

MaxRetries

Operator Ports

Success

The operator completes successfully.

Failure

The operator fails for any of the following reasons:

■ Invalid input parameter from the user. The reason field contains an error message specifying the problem.

■ The local file exists but cannot be written to.

- The local file is a directory.

- The local file does not exist. Its parent directory cannot be written to due to the current privileges and restrictions preventing CA Process Automation from writing to the directory.

- The current privileges and restrictions prevent CA Process Automation from writing to the local file.

- Unknown host specified.

- IO error when receiving the remote file.

- Timeout error if CA Process Automation is unable to connect to the remote host at the specified remote port. In such a case, the operator does not time out, as the TFTP client reports this issue as an IO error (not a timeout error).

- Others (specified in the reason field).

Custom Ports

If set by the user during the process design.

TFTP Upload File Operator

The TFTP Upload File operator sends a file to a host on a network through the TFTP protocol. The host must have a running TFTP server.

Details on the TFTP protocol follow:

- Uses UDP to transfer the data.

- Issues sends and waits for ACKs.

- Typically initiates data transfer on port 69.

- Typically sends data in a block size of 512b or smaller.

- Uses no authentication or authorization.

Important! Test TFTP functionality outside CA Process Automation before incorporating it into CA Process Automation process flows. Firewalls or routing can block UDP to port 69 by default. Typically, the TFTP service is either not installed or disabled.

Input Parameters

Remote Hostname

The host name or the IP address of the remote host.

Remote UDP Port for TFTP

The UDP port of the TFTP service on the remote host. If this port is specified, this value overrides the value of the operator category field: Default UDP Port for Trivial FTP. If none of these fields are specified, the operator uses the default value of 69.

Remote File name

The name to use when creating the file (being sent) on the remote host. This parameter is not the path of the remote file on the remote host. Depending on its setup, the TFTP server saves the file in its base directory in the remote host.

Data Transfer Type

BIN for binary or ASCII for ASCII (text). If not specified, the operator uses the default value: ASCII. If any other value is specified (other than ASCII or BIN), the operator uses the default value of ASCII.

Local File to Upload

The fully qualified path of the local file to send through TFTP.

Local Port Number (0 for anonymous port)

The local port number to use when sending data to the remote host/port. If 0, an anonymous port is used. If the port is specified and the port is unavailable, the operation could fail. If this number is not specified, the operator uses the default value: 0.

Timeout (sec)

The timeout value to use when opening the connection to the TFTP server. If this number is not specified, the value defaults to 20.

Maximum Retries after TFTP Timeout

The maximum number of times to retry the upload file operation (not the entire CA Process Automation operator) after a TFTP timeout. If this number is not specified, the operator uses the default value: 5.

Output Parameters

Result:

- 1: If the operator finished successfully.

- -1: If the operator failed.

Reason:

- Completed: If the operator finished successfully,

- An error message if the operator failed.

remoteSSHHost

PORT

RemoteURL

TransferType

LocalFileName

LocalPort

Timeout

MaxRetries

Operator Ports

Success

The operator completes successfully.

Failure

The operator fails for any of the following reasons:

- Invalid input parameter from the user. The reason field contains an error message specifying the problem.

- The local file is either non-existent, invalid, or cannot be read.

- The current privileges and restrictions prevent CA Process Automation from reading the local file.

- IO error when sending the local file.

- Unknown host specified.

- Timeout error if CA Process Automation is unable to connect to the remote host at the specified remote port. In such a case, the operator does not time out, as the TFTP client reports this issue as an IO error (not a timeout error).

- Others (specified in the reason field).

Custom Ports

If set by the user during the process design.

Upload File Operator

The Upload File operator copies a file from a local location to a remote location. This action corresponds to the FTP put command.

Uploading a remote file requires that the specified user credentials have the appropriate change directory and write permissions on the remote host. The remote host must have a configured FTP server.
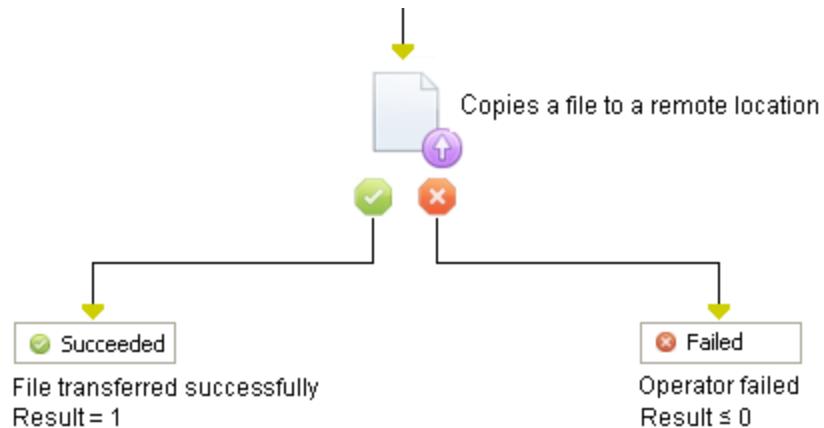
Input Parameters

Local file

Specifies the full path for the local file to transfer. For example: C:\\IT PAM\\import\\script_ora1.sh. This option is unavailable if you select the Is inline data check box.

Is inline data

When selected, transfers text stored with the operator in the library. Click the Inline data field to enter data.

Inline data

Click this field to open the Inline data editor where you can enter the data to transfer.

Remote file

Specifies the path for the file on the remote FTP host. For example: /tmp/IT PAM/scripts/script_ora1.sh. If you do not specify a file name, the operator saves the file using the same name as the copied file. The location must be valid at run time on the touchpoint running the File Transfer operators.

Remote host

Specifies the IP address or FTP URL for the remote host. For an FTP site on your company intranet, you can specify the server name (//servername) for the FTP site.

By default, FTP sites use port 21. However, you can assign a private port to an FTP site. Private ports range from 49152 to 65535. To specify a private FTP port, add a colon (:) and then a private port number to the end of the address. For example, the following specifies port 50021 on a remote FTP server:

```
172.24.36.107:50021
```

Remote user ID

Specifies a user ID to access the remote FTP host.

Remote user password

Specifies the password to access the remote FTP host.

Binary transfer

Uses the FTP binary mode for transferring binary files. For example, if selected, uses the following types of files:

- Executable files

- SPSS System files

- SAS Transport files

- Stata datasets

- Graphics files

Use Secure FTP (SFTP)

Select this check box to open a secure FTP (SFTP) session. SFTP is similar to FTP but, unlike FTP, the entire session is encrypted. No passwords are sent in clear text form, and are much less vulnerable to third-party interception.

Specify an optional SITE command

Select this check box to use the SITE command to invoke services that are specific to the host system. Then use the Site parameters field to specify a CA Process Automation expression that returns parameters for the SITE command.

Site parameters

Enter a CA Process Automation expression that returns parameters for the SITE command.

Output Parameters

localFile

isInline

Inline Data

remoteFile

remoteHost

remoteUserId

remoteUserPassword

transferMode

secureFtp

siteCommand

siteParameters

Chapter 11: Java Management

The Java Management operators provide a management interface to external systems that support JMX.

JMX Login Parameters

The JMX Login parameters display for each of the Java Management operators. These parameters configure the settings that are required to log in to and communicate with the JMX server.

Use user-specified JMX Service URL

Select this check box to specify a JMX Service URL instead of specifying a server.

When you select this check box, the following fields are enabled:

- JMX Service URL
- Remote JMX User
- Remote JMX Password

When you select this check box, the following fields are disabled:

- Remote JMX Host
- Remote Registry Port
- Remote JMX Server

JMX Service URL

Specifies a JMX Service URL.

URL Pattern:
service:jmx:rmi:///jndi/rmi://*<TARGET_MACHINE>*:*<RMI_REGISTRY_PORT>*/JMXRMIServer

For example:

TARGET_MACHINE: PA-w2k3-3

RMI_REGISTRY_PORT: 9999

JMX RMI Server: server

The URL looks like:

service:jmx:rmi:///jndi/rmi://PA-w2k3-2:9999/server

Remote JMX Host

Specifies the host machine name for the JMX agent. You can specify the full machine name or an IP address on your network.

When the JMX agent is running on the same machine as CA Process Automation, localhost is the default.

RMI Registry Port

Specifies the connection port for the JMX agent.

The default port is 12345. However, you can run the JMX agent on an RMI Registry port that you define.

Remote JMX Server

Specifies the name of the JMX server.

Remote JMX User

Specifies the user name to connect to the JMX agent on the MBean server. If security is disabled for the JMX agent, leave this option blank.

Remote JMX Password

Specifies the password to connect to the JMX agent on the MBean server. If security is disabled for the JMX agent, leave this option blank.

Get MBean Attributes Operator

The Get MBean Attributes operator gets the value of an MBean through JMX on a node. The template operator requires the name of the MBean and contact information and credentials for the JMX server. The retrieved values are placed in the process dataset. The operator dataset variable is assigned the same name as the attribute and its value is the retrieved value.

Input Parameters

Managed Bean Name

Specifies the name of the management Bean to access.

From the drop-down list, select the name of the management Bean to access or you can type in the user-defined Mbean name manually.

Managed Bean Attribute

Select or type the name of the attribute to fetch from the list.

JMX Login Parameters (see page 323)

Output Parameters

ManagedBeanName

ManagedBeanAttribute

State

UserSpecifiedURL

RemoteJMXURL

RemoteHost

RemoteRMIRegistryPort

RemoteJMXServer

RemoteUser

RemotePassword

Example

This example shows a user-defined Managed Bean Name and Managed Bean Attribute.
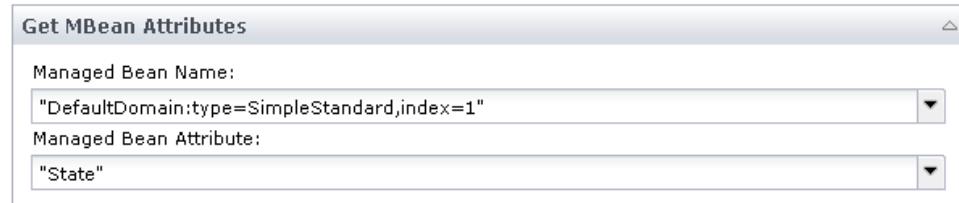
In the JMX Login Parameters, a remote JMX Service URL is provided. The Remote JMX Host, RMI Registry Port, and Remote JMX Server fields are disabled as a result.

The Remote JMX User and Remote JMX Password fields are blank, as the server does not have security enabled.

After the operator successfully executes, the State parameter returns as initial state from the MBean, as shown in the output dataset of the operator from the Dataset panel:

This information also appears in the operator output variable properties from the log panel:

Invoke MBean Method Operator

This Invoke MBean Method operator invokes a method on a JMX server.

Input Parameters

Managed Bean Name

Specifies the name of the management Bean to access.

From the drop-down list, select the name of the management Bean to access or you can type in the user-defined MBean name manually.

Managed Bean Method

Specifies the method to invoke.

Method Parameters

Specifies parameters for the method.

Use the buttons on this field to add, remove, or reorder parameters.

JMX Login Parameters (see page 323)

Output Parameters

ManagedBeanName

ManagedBeanMethod

ManagedBeanMethodParams

InvokeResults

UserSpecifiedURL

RemoteJMXURL

RemoteHost

RemoteRMIRegistryPort

RemoteJMXServer

RemoteUser

RemotePassword

Example

This example invokes the MBean method "sayHello" without any parameters from the user-defined MBean.

After the operator successfully executes, the MBean method returns the value to the InvokeResults parameter, as shown in the output dataset of the operator from the Dataset panel:

This information also appears in the operator output variable properties from the log panel:

Update MBean Attributes Operator

The Update MBean Attributes operator sets the MBean attribute value to the MBean attribute.

Input Parameters

Managed Bean name

Specifies the name of the management Bean to access.

From the drop-down list, select the name of the management Bean to access or you can type in the user-defined Mbean name manually.

Managed Bean Attribute

Specifies the name of a JMX MBean attribute to update.

Attribute Value

Specifies a value which is set as the value of the JMX attribute.

JMX Login Parameters (see page 323)

Output Parameters

ManagedBeanName

ManagedBeanAttribute

ManagedBeanAttributeValue

UserSpecifiedURL

RemoteJMXURL

RemoteHost

RemoteRMIRegistryPort

RemoteJMXServer

RemoteUser

RemotePassword

Example

This example illustrates:

■ Updating the MBean attribute value from initial to intermediate using the Update MBean Attribute operator.

■ Obtaining the updated value of the MBean attribute state using the Get MBean Attributes operator.
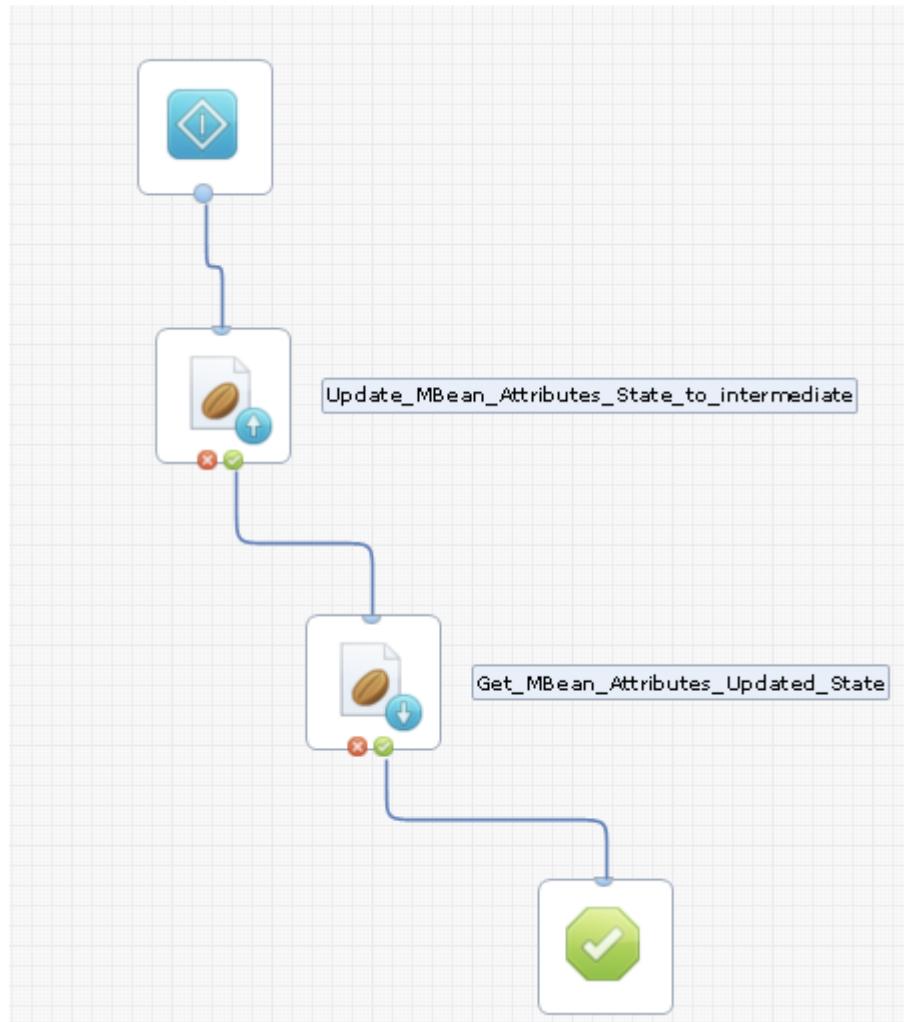
These parameters are the Update MBean Attributes operator input parameters:

After the operator successfully executes, the MBean attribute State value is set to intermediate, as shown in the output dataset of the operator from the Dataset panel:

This information also appears in the operator output variable properties from the log panel:

These variables are the operator output dataset variables from Dataset Panel of the Get MBean Attributes. The State Value has been updated to intermediate.

◢ 📁 *Page.JMXGetParameters*	
ManagedBeanName	DefaultDomain:type=SimpleStandard,index=1
ManagedBeanAttribute	State
State	intermediate

Chapter 12: Network Utilities

The Network Utilities operators provide various standard network protocol utilities to the automated business processes made possible by CA Process Automation. These operators provide general utilities that validate various network interfaces and devices. These operators also operate on remote services and servers and moves data as part of the general automation process, thus reducing manual validation and verification.

The Network Utilities operators have the option to create custom exit ports. You can set custom ports (nonautomatic exit ports) on the operator when you are creating the process. These ports are in addition to the automatic exit ports that all operators have (success and failure ports).

Use the Network Utilities operators to use native network interface utilities (rather than use host- based scripting languages and other host-based utilities).

Get Local Network Interfaces Operator

The Get Local Network Interfaces operator lists all the network interfaces in the local host. The local host is the host on which the touchpoint for the operator is running.

For each network interface, the operator lists the following information:

- Interface name
- Mac address
- Display name
- List of inetaddresses that are associated with the interface.

    Note: Each InetAddress consists of its canonical name, host name, and IP address.

Input Parameters

No operator input is required.

Output Parameters

StartTime

StartDate

NetworkInterfaces

Specifies an array of value maps containing the network interfaces associated with the local host.

Each value map in NetworkInterfaces contains the following information:

Name

Specifies the name of the network interface.

Display

Specifies the display name of the network interface.

MacAddress

Specifies the hardware address, usually MAC address, of the network interface. A blank in this field indicates that the operator is unable to retrieve the MAC address from the network interface. The retrieve failure can be due to lack of privileges or can be due to the lack of a MAC address for this network interface.

InetAddresses

Specifies an array of value maps containing the InetAddresses associated with the network interface. Each value map in InetAddresses contains the following information:

IpAddress

Specifies the IP address.

Host

Specifies the host name. The IP address is returned if the operator is unable to perform a reverse lookup due to network setup.

CanonicalName

Specifies the canonical name. If the operator is unable to retrieve the FQDN, an IP address is returned.

Result

One of the following items:

1

Indicates that the operator finished successfully.

-1

Indicates that the operator failed.

Reason

One of the following items:

Completed

Indicates that the operator finished successfully.

<error message>

Specifies why the operator failed in an error message.

Operator Ports

Success

The operator finished successfully.

Failure

The operator failed for one of the following reasons:

- Unable to retrieve the list of local network interfaces of the local host.

- Other reasons specified in error messages.

Custom Ports

If set by the user during the process design.

Example

Example ValueMap: Network Interfaces and InetAddresses

Get Network Service Status Operator

The Get Network Service Status operator lets you communicate with a local or remote service, over TCP or UDP. You can use this operator to send data and then receive a reply. This operator can validate the reply against a predetermined pattern to determine if the network service is up. This type of validation enables the operator to report the status of the service and the computer that is hosting that service.

Note: You cannot send a binary message using this operator, because only strings (text) are supported.

TCP is a connection-oriented protocol. The operator lets you connect to the service, send data, then receive a reply that can be matched against a pattern. Another service can listen on the same port. Therefore, a successful connection does not necessarily mean that a given service is running.

UDP is a connection-less protocol. The operator does not connect to the service to get its status through UDP. To get the status, the operator sends a UDP message, reads the reply, and verifies that the reply data matches a pattern.

Input Parameters

Remote Hostname

Specifies the hostname or FQDN of the computer that hosts the service.

Default

Blank - Indicates that the operator assumes that the service is running on the touchpoint host.

Remote Port

Specifies the host port on which the service is listening. Some well-known and registered ports include the following:

- 21: FTP - File Transfer

- 22: SSH - The Secure Shell (SSH) Protocol

- 23: Telnet

- 2483: Oracle TTC

- 25: SMTP - Simple Mail Transfer

- 3306: MySQL

- 69: TFTP - Trivial File Transfer

- 80: HTTP

- 1433: Microsoft SQL Server

You can specify any valid port in this field.

Local Port Number (0 for anonymous port)

Specifies the local port that the Get Network Service Status operator uses on the touchpoint host to connect to the remote port.

Values

- 0 - Indicates that an anonymous port is used.

- Blank - Same as 0.

Note: If the specified port is unavailable, the operator fails.

Protocol to Use

Specifies the protocol to use when verifying the status of the service and sending data. If UDP is specified, the Connection Timeout (sec) field is disabled because UDP is a connectionless protocol.

Values

This value can be one of the following:

- TCP

- UDP

Default

TCP (if left blank).

Connection Timeout (sec)

Specifies the maximum amount of time that the operator waits for a connection to the service before timing out. This field is applicable to TCP protocol only.

Values

This value can be one of the following:

- 0 (zero) - Indicates no timeout.

- Any positive integer (in seconds)

Default

20 seconds (if blank)

Data to send

(Optional) Specifies data to send to the service. Most services do not expect any data.

Read Data from Service?

Specifies whether the operator reads data from the service after contacting it.

Values

This value can be one of the following:

- Selected - Indicates that the operator reads data from the service after contacting it. Select this option to enable Time to Read Data (sec), Max Data to Read (bytes), and Reply Pattern to match fields.

- Cleared - Indicates that the operator does not read data from the service after contacting it. This value is appropriate for services that do not return data.

Notes:

■ With UDP, all the data (if any) is read at once. If there is no data, the operator waits until the Time to Read Data is up. All the Max Data to Read is read at once in a string of length Max Data to Read.

■ With TCP, the operator reads the data in chunks until it reaches one of the following thresholds:

  – Time to Read Data

  – Max Data to Read

Time to Read Data (sec)

Specifies the amount of time to spend waiting for reply data from the service. This field is specified because the data from the service does not have an EOF at the end.

Values

This value is a positive integer. Zero (0) is *not* allowed. Otherwise, the operator would wait for a long time until the service closes the socket.

Default

20 seconds (if blank).

Max Data to Read (bytes)

Specifies the maximum amount of data to read from the service.

Default

4096 bytes

Reply Pattern to match

(Optional) Specifies a pattern to use to match the data returned from the service. If specified, pattern matching determines whether the operator succeeds or fails.

The operator matches the pattern against the data read during the time period specified in Time to Read Data up to the number of bytes specified in Max Data to Read.

The operator matches the pattern as a substring of the reply data.

■ To match reply data that starts with a specific sequence, use ^ at the beginning of the pattern.

■ To match reply data that ends with a specific sequence, use $ at the end of the pattern.

■ To match a new line terminator, use dot (.). Dot is used to match multiline reply data.

Output Parameters

ReplyMessage

Contains the data received from the service. This field is empty if the service did not send data or if the operator does not read data from the service.

LocalAddressInfo

Information regarding the local address used to connect to the service, in the form of:

`localhost/IP:port`

This information lets you identify the actual local port number used when the Local Port Number (see page 339) is set to 0 for anonymous.

RemoteAddressInfo

Information regarding the remote address that the operator connected to, in the form of:

`Hostname/IP:port`

Note: UDP is a connection-less protocol; this field is empty when UDP is selected in Protocol to Use (see page 339).

Result

■   1: The operator succeeded.

■   -1: The operator failed.

Reason

■   Completed, if the operator finished successfully.

■   An error message returns if the operator fails.

remoteSSHHost

PORT

LocalPort

SendProtocol

ConnTimeout

CommandData

IsReadData

ReadDataPeriod

ReadDataBytes

ReplyPattern

StartTime

StartDate

Operator Ports

Success

- The operator finished successfully.

- If the operator is configured to read data from the service, then the operator succeeds if:

  - TCP:

    a. It binds to the local port (if specified)

    b. Connects to the remote host at the specified remote port

    c. Writes data to the service (if specified)

    d. Reads the reply data from the service

    e. Verifies that the reply data matches the pattern (if specified).

  - UDP:

    a. It binds to the local port (if specified)

    b. Sends a UDP message to the remote host at the specified remote port

    c. Reads the reply from the service

    d. Verifies that the reply data matches the pattern (if specified).

- If the operator is configured so it does not read data from the service, the operator succeeds if:

  - TCP:

    a. It binds to the local port (if specified)

    b. Connects to the remote host at the specified remote port

    c. Writes data to the service (if specified).

  - UDP:

    a. It binds to the local port (if specified)

    b. Sends a UDP message to the remote host at the specified remote port.

    This process does not mean that the UDP service is operational. We recommend configuring the operator to read the response from the service and match it to a pattern. Then, you can be sure that the UDP service is up and running. You cannot be sure that a service is verified properly through UDP when that service does not return any data.

Timeout

Timeout occurs when attempting to open a connection to the service through TCP. The timeout value is specified in Connection Timeout (sec) (see page 339).

Failure

Reasons include:

■ The operator is set to read data from the service, but the data read did not match the pattern specified by the user.

■ The user specifies invalid data. For instance: negative remote/local ports, negative Connection Timeout (for TCP only), 0 or negative Time to Read Data (sec).

■ The remote host is not known.

■ Cannot connect to the remote host at the remote port (or connection refused) through TCP; the service could be down in this case.

■ Cannot bind to the specified local port.

■ Error when sending or receiving data to/from the service.

Custom Ports

Returned if set by the user during the process design.

Get SNMP Variable Operator

The Get SNMP Variable operator reads the value of a remote management information base (MIB) variable. The object IDs (OIDs) and semantics of SNMP variables are documented in the MIB of the remote SNMP Agent.

SNMP is a connectionless, unreliable protocol. A timeout option specifies the time that is allowed for the request to reach the destination address. A retry option specifies the number of times that a request is sent in case of failures. This operator may fail for various error conditions, such as the SNMP variable is not found, read permission is denied, or a device is unavailable.
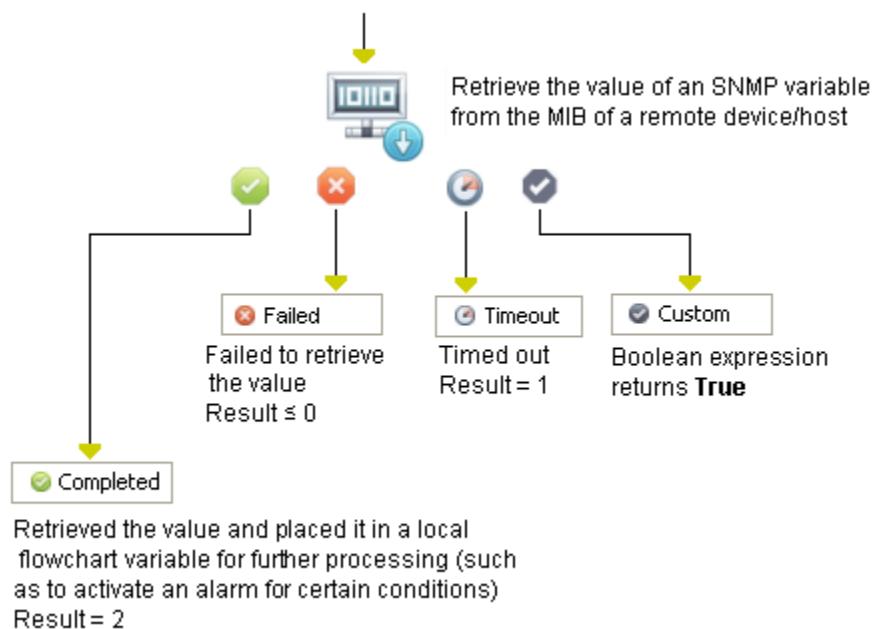
Retrieve the value of an SNMP variable
from the MIB of a remote device/host

Failed
Failed to retrieve
the value
Result ≤ 0

Timeout
Timed out
Result = 1

Custom
Boolean expression
returns **True**

Completed
Retrieved the value and placed it in a local
flowchart variable for further processing (such
as to activate an alarm for certain conditions)
Result = 2

Input Parameters

Agent Host

Defines the IP address or fully qualified domain name for the agent host. For example: 192.168.1.254.

To specify a port with the host name, use either of the following formats:

- host:port

- host/port

For example: comet.hq.company.com:10162

Community

Defines the community under which to access the variable (for example, public).

Object ID (OID)

Defines the object ID (OID) for the variable. The management information base (MIB) associated with a remote agent documents the OIDs. For example: 1.3.6.1.2.1.1.1.0.

Retry Count

Defines how many times the product retries a failed request.

Time-Out Interval (secs)

Defines the interval in seconds until the operator times out.

SNMP Version

Specifies the SNMP agent version number. Select **Version 1** or **Version 2** from the list.

Output Parameters

Object_ID

Retry_Count

Timeout

SNMP_Version

Community

Agent_Host

Port

OIDValue

Agent_Host

Community

RequestId

ErrorIndex

ErrorStatus

Monitor SNMP Variable Operator

The Monitor SNMP Variable operator waits until an SNMP variable has a value that satisfied specified conditions. These conditions are defined by a pattern or a range of values. The operator can extract substrings from a matched pattern.

This operator is implemented with an iterative Get SNMP Variable operator until the specified condition is satisfied.
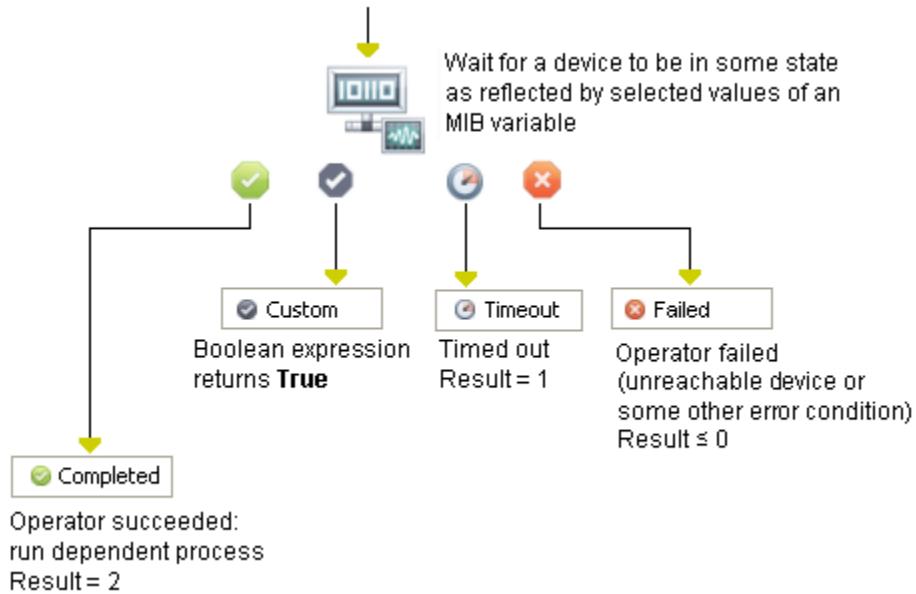
Input Parameters

Agent Host

Defines the IP address or fully qualified domain name for the agent host. For example: 192.#68.1.254

To specify a port with the host name, use either of the following formats:

■ host:port

■ host/port

For example: comet.hq.company.com:10162

Community

Defines the community under which to access the variable (for example, public).

Object ID (OID)

Defines the object ID (OID) for the variable. The management information base (MIB) associated with a remote agent documents the OIDs. For example: 1.3.6.1.2.1.1.1.0

Retry Count

Defines how many times the product retries a failed request.

Time-Out Interval (secs)

Defines the interval in seconds the product waits for a Get SNMP variable before the operator times out.

Variable Type

Specifies a data type for the variable. To configure this parameter, select a value from the drop-down list.

Sleep Time (secs)

Defines the maximum interval in seconds between attempts to check the value of the watched variable.

SNMP Version

Specifies the SNMP agent version number. Select **Version 1** or **Version 2** from the list.

Low Value

Defines the minimum expected numerical value.

High Value

Defines the maximum expected numerical value.

Mask

Defines the pattern for which the operator searches (for more information, see the *Content Designer Guide*).

Separators (#) in the pattern delimit the text to save to operator dataset variables.

Separator

Defines the character that delimits the zone to save to the variables that variable names specify.

Variable Names

Defines the variable names in which to save text that matches the delimited zones in the pattern. The product saves delimited zones to the listed variables in order. The product uses the keyword process to access operator dataset variables through the process dataset.

For example, specifying the variable names V1 and V2 assigns extracted substring values to the operator dataset variables Process.Operator_name.V1 and Process.Operator_name.V2. To add, remove, and order the variables to store the matched strings, use the toolbar.

Case Sensitive Pattern Matching

Specifies whether to use case-sensitive pattern matching.

Selected: Match upper-case and lower-case letters in a pattern only with letters of the same case.

Cleared: Ignore case when matching the pattern.

Output Parameters

Object_OID

Retry_Count

Timeout

Variable_Type

Sleep Time

SNMP_Version

Low_Value

High_Value

Mask

Seperator

Variable_Names

IsCaseSensitivePatternMatching

Watch Expiration

Community

Agent_Host

Port

MatchedEntry

LastReadPos

RequestId

ErrorIndex

ErrorStatsu

Port

OIDValue

Ping Host Operator

The Ping Host operator lets you evaluate access to a given host or IP Address. You can specify the number of requests to make to the remote host, and the timeout and TTL values. You can also specify the local IP address of the computer on which the operator is running. In this case, the operator uses the local network interface that is associated with the local IP address to initiate the ping operation.

The output variable: "isHostReachable" indicates whether the host is reachable.

- If *any* of the ping requests indicates that the host is reachable, then "isHostReachable" is set to True.

- If *all* ping requests indicate that the host is not reachable, then "isHostReachable" is set to False and the operator fails.

The Ping Host operator fails when an error occurs or when *all* ping requests to a host fail.

To ping the destination host, the Ping Operator uses ICMP echo requests if the privilege can be obtained, otherwise it attempts to establish a TCP connection on port 7 (Echo) of the destination host. Typically, port 7 is blocked and a TCP connection cannot be established.

On Windows, the Ping Operator may not use ICMP protocol (a java limitation), hence, run this operator on a *nix Agent or Orchestrator. Alternatively, you can use the Run Script operator to call the OS Ping command.

Firewalls and server configuration may block a ping request, resulting in an unreachable status.

Input Parameters

Remote Hostname

Specifies the host name or the IP address to ping. For IPv6 address, use either the form defined in RFC 2732 or the literal IPv6 address format defined in RFC 2373. If not specified, the default is used.

Default

The loopback address of the host associated with the touchpoint.

Local IP Address

Specifies the local IP address of the host with the agent associated with the touchpoint, whose network interface initiates the ping. If not specified, the Ping Host operator uses the default.

Default

Blank - Indicates any interface.

Number of Requests

Specifies the number of times to run the operation that determines whether the remote host is reachable. The Ping Host operator deems the remote host to be unreachable when all of these requests return that the host is unreachable. If not specified, the operator uses the default.

■ If *any* of the ping requests indicates that the host is reachable, then '"sHostReachable'" is set to True.

■ If *all* ping requests indicate that the host is not reachable, then "isHostReachasble" is set to False and the Ping Host operator fails.

Default

1

Time to Live

Specifies the maximum time to live value for each request in the specified number of requests. For pings (ICMP requests), it specifies the maximum number of hops the packets should go through before giving up and deeming the remote host unreachable. If not specified, the operator uses the default.

Default

30

Timeout (secs)

Specifies the time out in seconds, where the value applies to each request in the specified number of requests. If a request times out before getting an answer, that request deems the remote host to be unreachable. If not specified, the operator uses the default.

Default

5

Output Parameters

isHostReachable

isHostReachable is set to one of the following:

True

Indicates that at least one of the ping requests reached the host.

False

Indicates that none of the ping requests reached the host. The operator fails.

A request deems the host to be unreachable if:

- The host is not reachable
- TTL expires
- Timeout expires

HostIpAddress

The IP address of the remote host.

HostCanonicalName

The canonical name of the remote host. The operator retrieves either the FQDN or the IP address depending on the underlying system configuration.

LocalNetworkInterface

One of the following:

The name of the local network interface used to send the requests to the remote host.

Indicates the user specified a local IP Address.

Blank

Indicates that no local IP Address was specified in the operator input.

TotalRequests

The number of requests issued by the operator.

If no error occurs when running the operator, the TotalRequests should be equal to the value of the Number of Requests.

SuccessfulRequests

The number of requests that reached the remote host.

FailedRequests

The number of requests that found the remote host to be unreachable.

FailurePercentage

FailedRequests * 100 / TotalRequests.

Result

1

Indicates that the operator finished successfully.

-1

Indicates that the operator failed.

Reason

One of the following:

Completed

Indicates that the operator finished successfully.

<error message>

Specifies why the operator failed in an error message.

remoteSSHHost

localIp

NUMBER_OF_REQUESTS

TTL

Timeout

Operator Ports

Success

The operator finished successfully.

Failure

The operator failed for one of the following reasons:

- isHostReachable is false.

- A firewall or a network issue prevents CA Process Automation from looking up or connecting to the machine.

- Unknown remote hostname.

- Invalid local IP address.

- Unable to retrieve the local Network Interface associated with the local IP address.

- The local Network Interface associated with the local IP address is not up.

- Timeout, number of requests, or time to live is less than or equal to 0 (zero).

- Another reason, specified in the reason field.

Custom Ports

If set by the user during the process design.

Send SNMP Trap Operator

The Send SNMP Trap operator generates SNMP traps. A *trap* is an unsolicited message that an SNMP agent sends to an SNMP management system. The agent sends a trap when it detects that a specific type of event has occurred locally on the managed host. For example, the agent can send a trap message on a system restart event. SNMP traps are typically used to trigger alarms and notifications or to cause predefined actions by remote devices (such as a device reboot or reset).

The precise semantics of specific traps are defined in the SNMP agent MIB (Management Information Base) documentation. For custom traps, see the destination agent documentation.
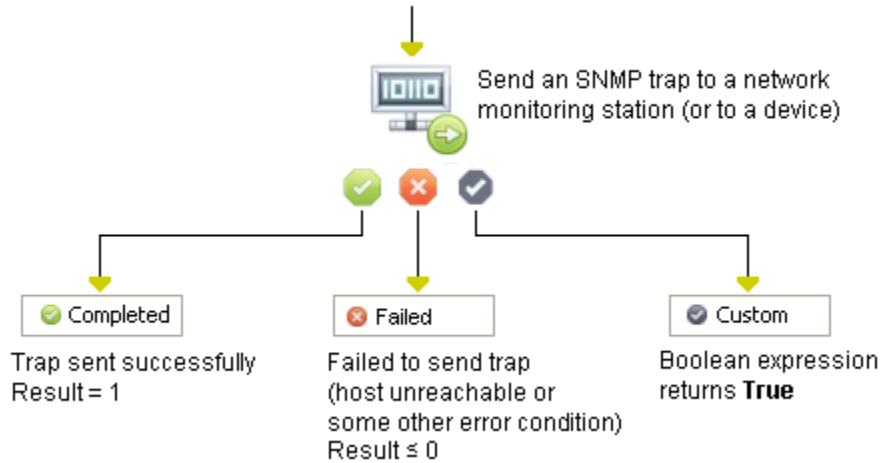
Input Parameters

Agent Host

Defines the IP address or fully qualified domain name for the agent host.

To specify a port with the host name, use either of the following formats:

■ host:port

■ host/port

For example: comet.hq.company.com: #####

Community

Defines the SNMP trap community name. For example: public.

SNMP Version

Specifies the SNMP agent version number. Select Version 2 from the list.

Trap ID

Specifies one of the standard Trap IDs:

- Cold Start

- An enterprise-specific Custom Trap ID

- Egp Neighbor Loss

- Link Down

- Link Up

- Warm Start

- Authentication Failure

Custom Trap ID

When you set the Trap ID parameter to Custom, this parameter defines the enterprise-specific Custom Trap ID. For more information, see the destination agent documentation.

Payload Trap ID

Defines the payload Trap ID.

Payload Trap Type

Specifies the data type for the SNMP Trap message payload.

Payload Trap Value

Specifies the SNMP Trap payload value.

Output Parameters

SNMP_Version

Trap_ID

Custom_Trap_ID

Payload_Trap_ID

Payload_Trap_Type

Payload_Trap_Value

Community

Agent_Host

Port

RequestId

Update SNMP Variable Operator

The Update SNMP Variable operator sets the value of an SNMP variable that a remote SNMP agent manages. In general, SNMP variables control the behavior of IP devices. The precise semantics of SNMP variables are defined in the MIB associated with a device.

To set the variable, the user account executing the Update SNMP Variable operator must have write permission on the SNMP server to change the value of the OID.
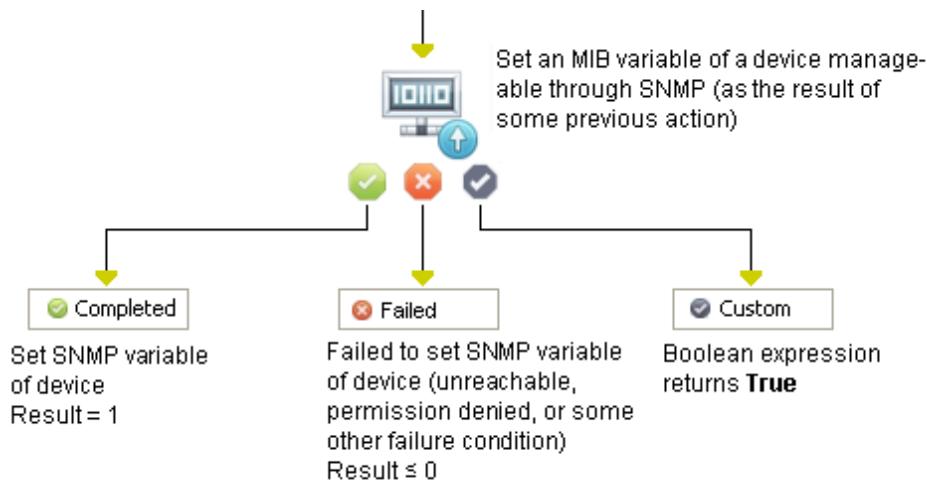
Input Parameters

Agent host

Specifies the IP address or fully qualified domain name for the agent host. For example: 192.#68.1.254.

You can specify a port along with the host name using either of the following formats:

- host:port
- host/port

Community

Specifies the community under which the variable is accessed (for example, public).

Object ID (OID)

Specifies the object ID for the variable. Object IDs (OIDs) are documented in the management information base (MIB) associated with a remote agent.

Variable type

> Specifies a data type for the variable. Select one of the list values to configure this operator parameter.

Variable value

> Specifies the value for the variable.

SNMP version

> Specifies the version number for the remote SNMP Agent. Select Version 1 or Version 2 from the list.

Output Parameters

Object_ID

Variable_Type

Variable_Value

SNMP_Version

Community

Agent_Host

Port

RequestId

ErrorIndex

ErrorStatus

Chapter 13: Process Control

The Process Control operators run, monitor, and control CA Process Automation processes. The Process Control operators start system processes from within a process. The Assign User Task operator prompts users for information during execution of a process.

All of the Process Control operators run only on Orchestrators, not on agents.

Assign User Task Operator

The Assign User Task operator displays a CA Process Automation interaction request form to prompt a user and waits for input. The last page of an interaction request form dialog prompts the user to approve or reject the interaction request form. You can specify a time-out or can wait indefinitely. The user input values are saved to variables in the Assign User Task operator dataset.

You can also use the Assign User Task operator to notify a user to respond to a specific task through an embedded URL.

Prompt user for input
and start flowchart

| Completed | Failed | Timeout | Custom |
| Operator succeeded Result =1 | Operator failed Result = -1 | Operator timed out Result = -1 | Boolean expression returns **True** |
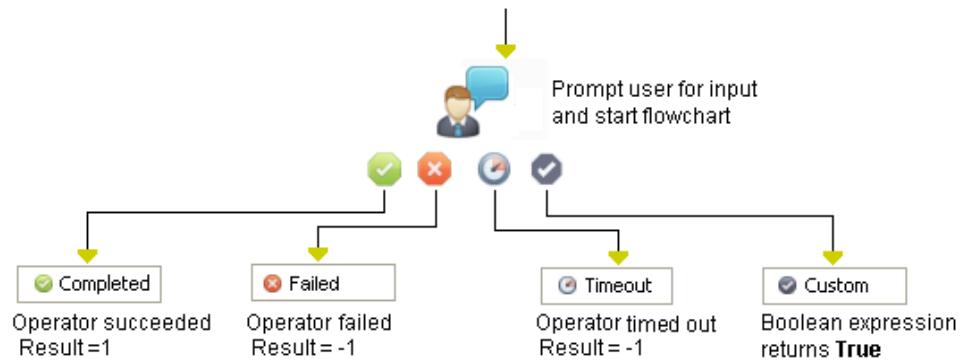
The Assign User Task operator has four standard (noncustomized) exit links:

Completed

Processed when a user does the following things:

- Successfully completes the interaction request form.

- Approves the user prompt on the last page of the interaction request form.

The parameter values are assigned to operator dataset variables. The Result variable is set to "1" and the Reason variable is set to "COMPLETED." The task displays in the Task List on the Operations tab with a state of "Completed."

Canceled

Processed when:

- A process has an Assign User Task operator running, and the task shows up in the Task List and is waiting for a user to respond. Before anyone responds, the process encounters a Stop Failure operator (typically because another path in the process is still running while the Assign User Task is waiting for user response).

- The Assign User Task operator is aborted (manually or because the process ended while the operator was still running).

- A process is suspended and the Assign User Task operator aborted the task.

- A process is suspended and the Assign User Task operator is reset (manually or automatically).

- A process execution bypasses the selection made in the user prompt (in the Form Disposition field).

The Result variable is set to "-1" and the Reason variable is set to CANCELED. In each case, the task displays in the Task List on the Operations tab with a state of "Canceled."

Timeout

Processed if the user does not complete the interaction request form within an optionally specified time-out interval. Any parameter values are assigned to operator dataset variables. The Result variable is set to "-1" and the Reason variable is set to "TIMEOUT." The task displays in the Task List on the Operations tab with a state of "Timeout."

Rejected

Processed if the user selects Reject at the user prompt (in the Form Disposition field). The Result variable is set to "_" and the Reason variable is set to "REJECTED." The task displays in the Task List on the Operations tab with a state of "Rejected."

Input Parameters

The Assign User Task operator includes the following input parameters.

Assignees Parameters

The Assignees parameters specify authorized CA Process Automation users or groups to approve or reject the user prompt. The Assign User Task operator only verifies user credentials when a user or group is specified.

Users

Specifies the names of authorized CA Process Automation users who can approve or reject the user prompt. Delimit multiple users with the colon (:) character. For example: malcolm:samirab:sam:seren.

To open the Users dialog to select users, click . Select individuals from the Available Users list to move to the Selected Users list using the arrow buttons (or the other way around). You can also enter a user name to search for in the text box.

Groups

Specifies the names of authorized CA Process Automation groups who can approve or reject the user prompt. Delimit multiple groups with the colon (:) character. For example: domainadmin:pamuser:envconfigadmin (or in the case of an upgrade: domainadmin:pamuser:envconfigadmin).

To open the Groups dialog to select groups, click . Select individuals from the Available Groups list to move to the Selected Groups list using the arrow buttons (or the other way around). You can also enter a group name to search for in the text box.

Transfer/Delegates Parameters

The Transfer/Delegates parameters specify those individuals that are authorized by CA Process Automation users or groups to approve or reject the user prompt. The Assign User Task operator only verifies user credentials when a user or group is specified.

Users

Specifies the names of authorized CA Process Automation users who can approve or reject the user prompt. Delimit multiple users with the colon (:) character. For example: malcolm:samirab:sam:seren.

To open the Users dialog to select users, click . Select individuals from the Available Users list to move to the Selected Users list using the arrow buttons (or the other way around). You can also enter a group name to search for in the text box.

Groups

Specifies the names of authorized CA Process Automation groups who can approve or reject the user prompt. Delimit multiple groups with the colon (:) character. For example: domainadmin:pamuser:envconfigadmin (or in the case of an upgrade: domainadmin:pamuser:envconfigadmin).

To open the Groups dialog to select groups, click . Select individuals from the Available Groups list to move to the Selected Groups list using the arrow buttons (or the other way around). You can also enter a group name to search for in the text box.

User Task Parameters

Title

Specifies a title for the user task (optional). This string describes the title of the form to present to the user.

Description

Provides an optional description for the user task.

Interaction Request Form

Specifies the path in the CA Process Automation library for the interaction request form object that prompts the user. The interaction request form must be in the same library as the process. For example: /Backups/Forms/Input.

To locate an interaction request form in the CA Process Automation Library, click .

To view the interaction request form in the Form Browser once one has been selected, click Open.

Form data initialization code

Allows you to add code that dynamically initializes form fields at run time. This allows you to display information in read-only fields or change default values for editable fields.

To add CA Process Automation expressions to change field values, open a code editor window.

In the editor, use the Form keyword to access operator dataset variables in the following format:

Form.*fieldname*

Where:

fieldname represents the name of the field in the user interaction form object definition. For example:

```
Form.DateTomorrow = System.tomorrow;
```

For the current operator.

Note: For more information about creating calculated expressions in CA Process Automation, see the <u>input parameters</u> (see page 406) for the Run JavaScript operator.

You can also use this field to dynamically initialize form fields at run time. The initialization is not the same for simple and complex types.

■ For simple types, if the interaction request form has a simple element (text field) Var_0, this element can be initialized by providing:

```
Form.Var_0='text'
```

■ For a ValueMap, if the interaction request form has a ValueMap that contains ID value_map, along with a text field with ID text_field, the text_field must be initialized in the following way:

```
Form.value_map= newValueMap();
Form.valuemap.text_field="test";
```

■ If the interaction request form has a nested ValueMap - specifically, if there is a ValueMap inside a ValueMap with ID value_map_nested, and a variable inside the same nested ValueMap with ID text_field_nested, the initialization must be:

```
Form.value_map.value_map_nested= newValueMap();
Form.value_map.value_map_nested.text_field_nested="test";
```

Show approval page

Lets you approve or reject the task. If selected, the included form is presented with an approval/rejected page at the end when replying, to decide the final outcome of the form.

Output Parameters

Title

Description

inputForm

showAcceptanceScreen

Userinfo

InitialAssignedUsers

initialAssignedGroups

assignedUsersFilter

assignedGroupsFilter

Approve

RepliedBy

Task ID

initCode

dueDateTime

Example

This example explains how you can send a notification to a user to reply to a task using an embedded URL in an email. You can include the Task ID output parameter of the Assign User Task operator in the embedded URL to access a direct task. You can include the embedded URL in the Send Email operator to notify the user through email to reply to a task.

The Assign User Task remains in a waiting state until the user responds. You can run a notification process in parallel to notify the user with the direct URL to reply to the task triggered from the Assign User Task operator, as shown here:
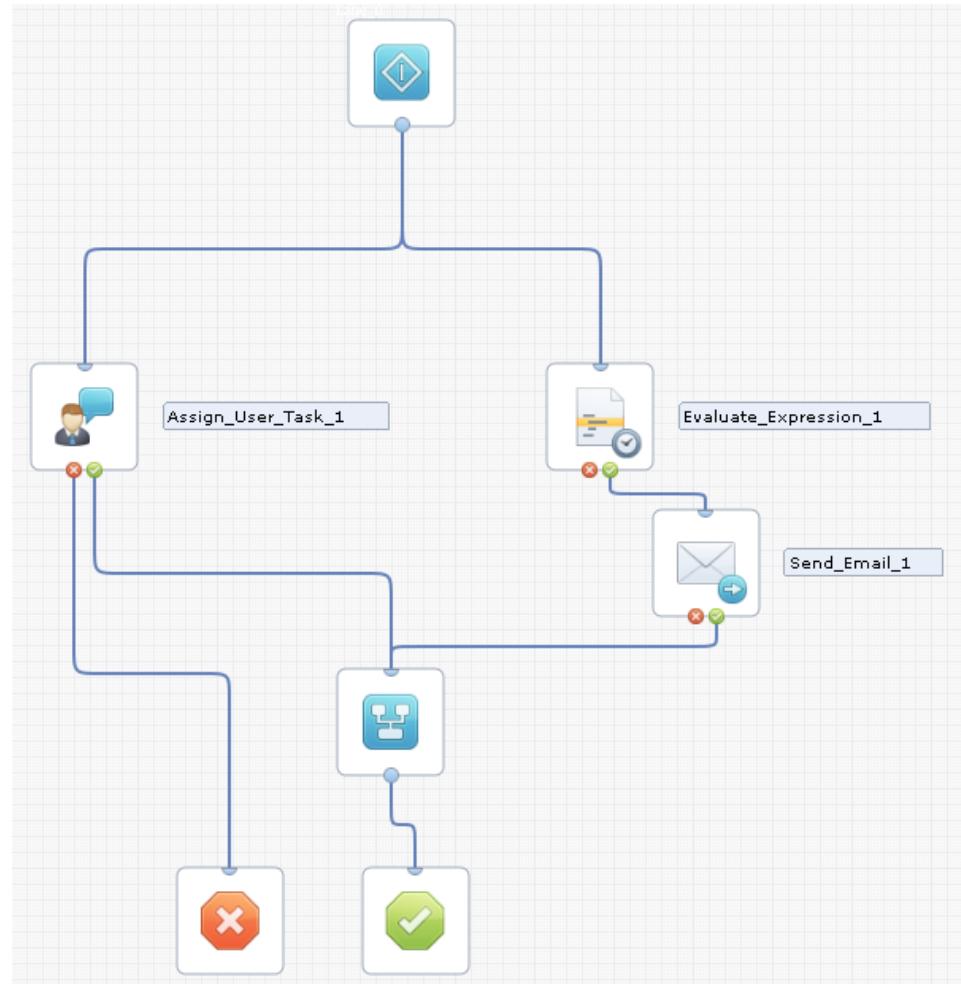
You can use the Evaluate Expression operator to wait until the Assign User Task expression evaluates to true while refreshing and evaluating the expression every five seconds.

The Evaluate Expression parameters can be:

Ensure that you select the No Timeout check box in the Assign User Task Timeout parameters to avoid a timeout for this operator until the expression evaluates to true.

Next, once the Task ID is calculated and is no longer zero, the expression evaluates to true and the Send Email operator sends an email with the following embedded URL as part of the message:

```
getOrchestratorURL() + "itpam" + "?ROID=" + Process.Assign_User_Task_1.TaskID +
"&page=replytask"
```

The And operator synchronizes the two branches of the process into a single one.

When the user receives the email, the task is presented in a URL that the user clicks to continue to the Login page of CA Process Automation. After authentication, the user is taken directly to the form attached to the task, and the user can then directly reply to the specific task.

Evaluate Expression Operator

The Evaluate Expression operator delays processing on the branch of a process until a condition that is represented by a Boolean expression evaluates to true. This operator provides a mechanism to pause a process while waiting for a condition to change. The operator is often used to synchronize interdependent processes or to control the use of shared resources that are represented by variables.
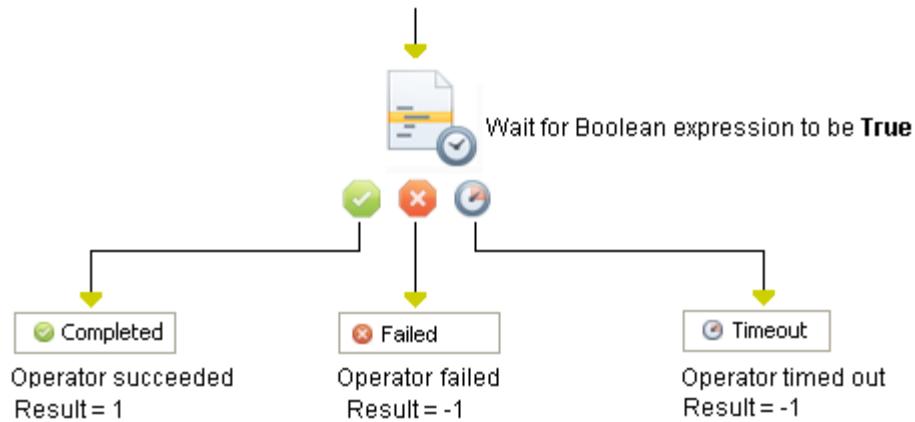
The condition is evaluated periodically according to a specified rate. The rate must be long enough to increase CPU usage within acceptable limits. When there is a condition for some minimum known amount of time, the load can be further reduced by putting a before the Evaluate Expression operator. An example of this situation occurs when another process uses a resource and the process does not release the resource before a certain time of the day.

Input Parameters

Expression

A Boolean expression that specifies a True condition when some condition is satisfied. Here are two examples:

`(Datasets["/exploit/variables/set_1"].var1 == 1)`

`System.Time >= Process.FinishTime`

Note: This field includes dataset assistance when using of any of the keywords.

Refresh rate (secs)

The interval in seconds at which to evaluate the condition for a True condition.

Output Parameters

Expression

RefreshRate

TimeOutSec

TimeOut

Manage Resources Operator

The Manage Resources operator executes actions on CA Process Automation resources. These actions include taking resource units, freeing resource units, and locking and unlocking resources.

The Manage Resources operator provides a way to validate and wait for particular resources and to affect the state of such resources. The operator can make processing of any branch of a process contingent on resource availability. Within an environment, resource operators can be used to regulate and coordinate the processing of multiple processes. The operators assure that individual processes have exclusive access to external resources.
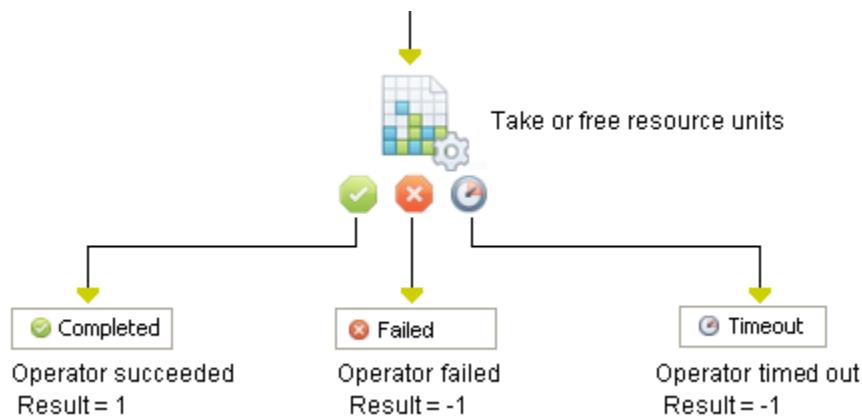
The Manage Resources operator has three possible exit links:

- Successful when actions are executed successfully before any specified timeout.

- Failed when resources do not exist or in the event of some other error condition.

- Timeout when the specified time-out expires before the required resources become valid.

Resources are typically taken from a resource quota before processing other operators and then replaced when the operators are completed:

Conditions on a resource must be evaluated periodically for possible changes. Be careful to avoid specifying a refresh interval that is too short. Use a Check Date-Time operator (see page 185) to add a delay before executing the Manage Resources operator if the required resources are not available before:

■   A specified interval of time has passed.

    Or

■   Before a specific date or time.

Input Parameters

Action

Lists the actions to execute. The Add, Delete, and Edit buttons add, remove, or modify actions in this list. Each action specifies:

■   A resources object

■   The name of the resource in the object to use

■   The action to perform on the resource (take resources units, free units, lock, or free a resource)

■   How many units of the resource to take or free

Action Properties

This dialog defines an action to perform on a resource. Click either the Add or the Edit button next to the Action list box. This dialog opens the resources properties for a Resources operator.

ResourcePath

Specifies a resources object. Enter the full path to the resource in the CA Process Automation library or click to locate the object. Double quotation marks must enclose a literal string. You can use a dataset variable or an expression to specify the resources object. To open the object in the resources editor, click the Open button.

ResourceName

Specifies the resource in the resources object on which to perform the action. Type the name exactly as it is defined in the resources object. Double quotation marks must enclose a literal string. You can use a dataset variable or an expression to specify the resource.

Action

Select the action to perform on the resource:

TakeUnits

Takes the number of resource units specified in the Amount field.

FreeUnits

Returns the number of resource units specified in the Amount field.

LockResource

Locks the resource so other Resources operators cannot take resource units or cannot lock the resource. This action effectively takes all unused resource units for a resource. Actions can still free resource units that were taken before a resource was locked. However, the freed units are only available when the resource is unlocked.

UnlockResource

Unlocks a locked resource.

Amount

For the TakeUnits or FreeUnits actions, this value specifies the number of resource units to take or free. Amount is disabled for the rest of the two options, such as UnlockResource and LockResource.

All resources must be available

If selected, all of the resources that are required by the actions that are listed under Actions must become available within the constraints that the Timeout options impose. The operator succeeds only if all of the resources become available within the time-out constraints of the Timeout setting.

If unchecked, the operator completes successfully when the resources that are required by at least one of the actions that are listed under Actions becomes available within the constraints of the Timeout option. If resources for any of the listed actions are available, the Process Control operator category processes the Successful exit link for this operator.

Execute actions

Determines whether the actions listed under Actions are executed. To only verify whether resources are available without executing actions, clear this check box. The operator then executes the Successful exit link. This link *only* executes if resources are available within the constraints of the "Timeout" and "All resources must be available" settings without executing any action.

This setting can be used with a resource that is set to enable or disable a whole set of processes. Those processes verify that there is no lock on the resource before starting their tasks. This lock check is done by attempting to take a single resource unit from the resource. Depending on the outcome of the test, some other mechanism can lock or unlock the resource, such as:

■ Schedule tasks (where enabling or disabling of the processes is based on time constraints)

■ Manually started tasks (using a start request form)

■ A process that an external monitoring application starts (using the CA Process Automation Web services daemon)

■ A process that monitors an internal or external condition in a loop.

Output Parameters

ActionProperties

ResourcePath

ResourceName

Action

Amount

All resources must be available

Execute actions

TimeOut

Event Operators

CA Process Automation provides event management through two operators:

- Monitor Event

- Send Event

Other processes can post events. In addition, the Web services that are exposed by CA Process Automation can also post events.

Note: These two operators run only on Orchestrators, not on agents.

Monitor Event Operator

The Monitor Event operator is used in a process to wait for certain events before continuing down a path of execution. For example, a process can wait for an event that signals that a ticket has been approved, instead of periodically querying the ticket and checking the approval status.

The Monitor Event operator consumes the available/matching events as its default behavior.

Note: Monitor Event cannot be scheduled (that is, it cannot be used in schedules). However, a user can design a process with Monitor Event and then schedule the process from the schedule editor.

Input Parameters

Event name

Specifies the name of the event. This expression is matched against Name of the Event. This name can be a regular expression, a partial match that is based on user choice, or both.

Event type

Specifies the type of event (optional). This expression is matched against Type of the Event. This type can be a regular expression, a partial match that is based on user choice, or both.

Event source

Specifies the source of the event. This expression is matched against Source of the Event. This source can be a regular expression, a partial match that is based on user choice, or both.

Event destination

Specifies the name of the Event destination (optional). This expression is matched against Destination of the event. This destination can be a regular expression, a partial match that is based on user choice, or both.

Expression

Specifies a CA Process Automation Boolean expression for additional event parameters (optional). This expression is matched against Event Parameters field of Event. Event Parameters can be accessed using a "payload" keyword (for example, payload.ticketId=="1443132").

Note: This field includes dataset assistance when using of any of the keywords.

Retrieve all matching events

When selected, the Monitor Event operator receives all events, instead of the first one that matches. Once these events are delivered, they are never sent to you again. Any event that is delivered to you and is also marked as 'deliver to single subscriber' is invalidated and is not delivered to anybody else.

Enable pattern matching

Enables pattern matching against the respective event attributes like Name, Type, Source and Destination.

Allow partial match

Allows a partial match against the respective event attributes like Name, Type, Source and Destination.

Output Parameters

eventId

eventName

eventType

eventSource

eventDestination

toSingleSubsriber

payload

creationTime

expirationTime

user

Send Event Operator

The Send Event operator is used to publish an event to the CA Process Automation Orchestrator. The event manager running on the CA Process Automation Orchestrator (which holds all the subscribers) receives an event. The event is checked against any interested subscribers by matching the event parameters. All subscribers who are waiting for this type of event are then notified. As a result, the Monitor Event operator is completed and the process continues down the path of execution.

Send Event cannot be scheduled (that is, it cannot be used in schedules). However, a user can design a process with Send Event and then schedule the process from the schedule designer.

Note: The same event never gets delivered twice to the same operator, in the same process instance.

Input Parameters

Event name

Specifies the name of the event (mandatory).

Event type

Specifies the type of event (optional).

Event source

Specifies the Event Source (optional).

Event destination

Specifies the Event Destination (optional).

Subscribers of the event match a regular expression against these fields to decide if they are interested in this event.

Deliver to single subscriber

When set to true, indicates that the events are not delivered to more than one waiting process. The event is "consumed" by the first event handler that is "consuming" events.

Event parameters

Specifies additional event parameters that can be a CA Process Automation data type (optional).

The Expression parameter in the Monitor Event operator is evaluated against Event Parameters. These parameters can be accessed using a payload Keyword (for example, payload.ticketId=="1443132").

Expire after (sec)

Specifies the number of seconds that an event can take to match with any subscribers.

Output Parameters

expirationDuration

eventId

eventName

eventType

eventSource

eventDestination

toSingleSubsriber

payload

creationTime

expirationTime

user

Usage Patterns for Events

The following two usage patterns are available for events:

Queue pattern

Every event is delivered to a single consumer. You must mark the event accordingly on the sending side (deliver to single subscriber). Events of this type are cleared as soon as they are delivered or expire.

Note: This pattern affects triggers; see the *Content Administrator Guide* for more information.

Notification Mechanism

The event is intended to signal a state to an arbitrary number of interested parties. For instance, a notification signifies that something has changed, a system is shutting down, and so on. Such an event is delivered once to all subscribers, until the event times out.

Start Process Operator

Use the Start Process operator to start a process from within another process. The Start Process operator creates an instance of a process on a touchpoint and queues a start request with the appropriate engine. You can reference the child process dataset by the operator name in the process dataset for the parent process. Use the following syntax:

Process.*OperatorName*.*FieldName*

OperatorName represents the name of the Start Process operator in the parent process.

FieldName is the dataset variable that you want to access in the child process.

Input Parameters

Process name

Specifies the path for the process in the CA Process Automation Library. The process must be in the CA Process Automation Library of the touchpoint on which the operator is configured to be executed.

For example: "/Doc/NT_Charts/Alert"

To select a process from the library, click .

Open

Opens the process that is specified by process name for editing. This button is available only after you enter the path to a process in the adjacent box.

Process Dataset initialization code

Specifies statements that initialize dataset variables in the process that is being started. For example:

- Process.WorkDir = "C:\temp";

- Process.User= Caller.User;

- Process.DatabaseServer=Caller.DatabaseServer;

In this box, the keyword *Process* refers to the dataset in the new instance of the process that is specified by Process name. The keyword Caller refers to the dataset of the process containing the Start Process operator.

The *Process* or *Caller* keyword is mandatory for referencing or creating variables in the parent or child process dataset. Without either keyword, the dataset initialization script always creates or attempts to reference a calculation variable.

Mode

Select from one of the following options:

Attached

Runs the child process as a separate process.

The Start Process operator does not complete until after the new instance finishes processing. The process executing the operator is the parent process.

Detached

Runs a process in detached mode.

An instance of a process started in detached mode has no parent relationship to the process that started it and is the root process in any call sequence originating from that process.

Inline

Runs a child process as a part of parent process itself (that is, it is expanded into the parent process).

Inherit Lane Change Handler from parent process

When selected, the child process inherits the lane change handler from the parent process (if not already defined in the child process).

Start date

Specifies the date on which to start a detached instance of the process. The default value is the date on which the operator is executed (System.Date). This option is only available when Detached is selected as the process mode.

Start time

Specifies the time at which to start a detached instance of the process. The default value is the time at which the operator is executed (System.Time). This option is only available when Detached is selected as the process mode. Combined with the Start date option, Start time allows a process to schedule the execution of another process.

Output Parameters

32WorkflowName

Local (Process Dataset Initialization Code)

processMode (Attached, Detached, or Inline)

inheritLaneChangeFromParentProcess

Chapter 14: Utilities

The Utilities operators can be used for utility purposes in processes.

Apply Xpath Operator

The Apply Xpath operator parses and retrieves data from an XML document. This operator supports the following functions:

■ Parses an XML document and retrieves specified data from the document.

■ Stores the results into CA Process Automation datasets that subsequent operators in a process can access.
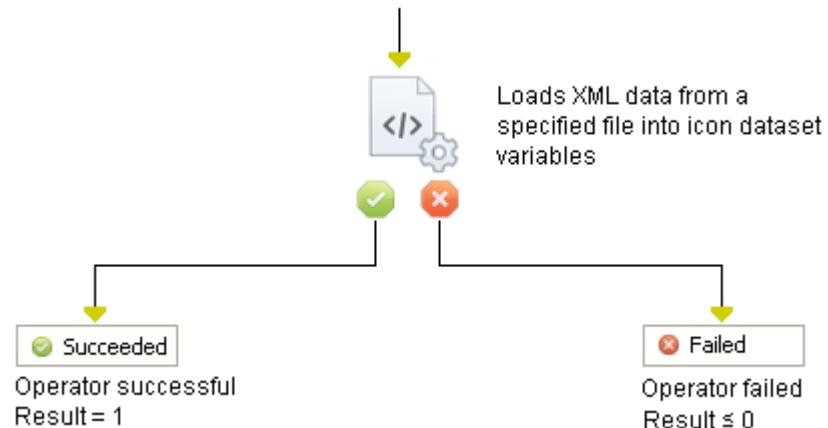
Input Parameters

Input Source

Select the source for the SOAP service input request: Expression or Input File Name.

Expression

Specify the expression to load XML content. For example:

`Process.xmlContent`

or

`Datasets["xmlData"].xmlContent`

XML input file

Specifies the XML document from which to extract data. Enter an expression that returns the path of the XML file for a valid XML document.

Strip Namespace in XML Structure

CA Process Automation provides an option to strip XML namespaces from a response so that a user can provide simpler XPath expressions to look for a value of specific element. This option is available in all the SOAP operators.

The following javascript functions are provided:

- applyXPath(xmldata,xpath_query,namespaceAware)

- applyXPathToUrl(urls,xpath_query,namespaceAware)

Note: The default value of namespaceAware is true. The value of namespaceAware is false if you want stripping of Namespace in XML Structure (and true otherwise).

```
Process.x="<getMatchingEventsResponse
xmlns='http://www.ca.com/itpam'><events> <event
><eventName>test</eventName></event></events></getMatchingEventsResponse>";
Process.s=applyXPath(Process.x,"//eventName",true);
Process.aa1=applyXPathToUrl("file:C:/test.xml","//message",true);
Process.aa2=applyXPathToUrl("file:C:/test.xml","//message",false);
Process.s1=applyXPath(Process.x,"//eventName",false);
Process.s2=applyXPath(Process.x,"//eventName");
```

Additional extracted data

Specifies XPath expressions to extract data from the XML document. For each expression specified here, specify a dataset variable to which to store the extracted data and a data type.

Use the Add, Edit, and Delete buttons to add, edit, or delete expressions from the list box. The Add and Edit buttons open the Additional Extracted Data dialog.

Specify values for the following options:

Xpath expression

Specifies the XPath expression selected under Additional extracted data.

Dataset variable

Specifies the name of an operator dataset variable in which to save values extracted based on the selected XPath expression.

Type

Specifies the type of element being extracted from the response. Select one of the following currently supported types:

■ Integer

■ Integer Array

■ String

■ String Array

■ XML Fragment

■ XML Fragment Array

Output Parameters

inputSource

ExtractedVarInfo (ValueMap)

xPathQuery

dataSetVa

type

expressionVal

xmlInputFileName

isStripXMLNamespaces

Apply XSLT Operator

XSLT applies a predefined style sheet to transform an XML source document to another presentation-oriented format such as HTML, XHTML, or SVG.

Input Parameters

Input XML Source

Defines the source XML document to transform to one of the following formats:

Expression

Defines a pattern for identifying a string of values. For example, you can define the expression *Datasets["/VER2_Dataset"].srcXML*

XML File Path

Defines the path of a file where an operator is executed. The file path can be a shared location or a URL.

Consider the following examples:

- File path: c:\sourcefiles\books.xml

- Shared location: \\fileserver\sourcefiles\books.xml

- URL: http://fileserver:8080/sourcefiles/books.xml

Inline XML

Specifies the XML data that acts as an input.

Input XSLT Source

Specifies the source XSLT information in one of the following formats:

Expression

Defines a pattern for identifying a string of values. For example, you can define the expression *Datasets["/VER2_Dataset"].srcXML*

XSL File Path

Defines the path of a file where an operator is executed. The file path can be a shared location or a URL.

Consider the following examples:

- File path: c:\sourcefiles\books.*xsl*

- Shared location: \\fileserver\sourcefiles\books.*xsl*

- URL: http://fileserver:8080/sourcefiles/books.*xsl*

Inline XSL

Specifies the XSL data that acts as an input.

XSLT Version

Specifies one of the following options to determine the XSLT version that is used to transform the source XML:

- Version 1

- Version 2

■   Specified in XSLT

Input Parameters

Specifies the input parameters as key-value pairs in XSLT Operator. You can dynamically assign values to the input parameters (key) that are defined in XSL.

You can define a key (top-author) in XSL and can assign a value (Robert Kisosk) from an XSLT Operator as in the following example:

<xsl:param name="top-author">Jasper Forde</xsl:param>

Key: top-author Value="Robert Kisosk"

Output Parameters

Displays the predefined output parameters as key-value pairs. The output parameters are XSLT version-specific and based on the XSLT standards. For more information about output parameters, see http://www.w3.org/TR/xslt#output.

XML Output

You can view the XML output in the Dataset Variable or in a file placed in the Output File Path.

You can assign a value (yes) to the key (include-content-type) from an XSLT Operator as in the following example:

include-content-type="yes"

Delay Operator

The Delay operator delays processing subsequent branches of a process until a specified interval of days, hours, minutes and seconds has passed. The delay can be relative to either when processing starts for the Delay operator or when processing starts for the process.
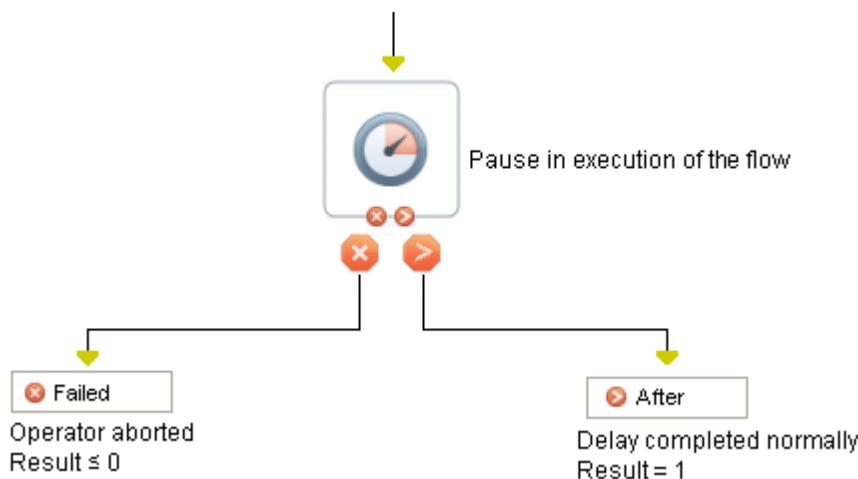
Input Parameters

Days

Specifies the number of days to delay processing subsequent branches of the process.

Hours

Specifies any additional hours to delay processing subsequent branches of the process.

To specify the portion of a day in hours, enter 0 to 23.

Note: The number of hours is an expression so there are no validations.

Minutes

Specifies any additional minutes to delay processing subsequent branches of the process.

To specify the portion of an hour in minutes, enter 0 to 59.

Note: The number of minutes is an expression so there are no validations.

Seconds

Specifies any additional seconds to delay processing subsequent branches of the process.

To specify the portion of a minute in seconds, enter 0 to 59.

Note: The number of seconds is an expression so there are no validations.

Relative to Process start time

Makes the delay relative to starting the process.

When this check box is cleared, the delay is after the process starts processing the operator. For example, this option can be used to trigger an alarm if the process does not end (reach a Stop operator) within a specified period of time.

Output Parameters

flowchart_start_time

Days

Hours

Minutes

Seconds

Relative_to_Flowchart

TargetTime

targetDate

Invoke Java Operator

The Invoke Java operator leverages the functionality that is contained in external JAR files (or .class files) in CA Process Automation. You identify the JAR files or .class files by specifying their location in the operator input parameters.

Once the JAR file is located, you can write Java code that references classes in the JAR file. You can pass variables to this code by using the input parameters of the Invoke Java operator. The code that you write goes in the main method of the operator.

You can specify to save a Java object in the operator dataset after execution of the code by the operator. The Java object is saved in CA Process Automation under data type: JavaObject. You can make this JavaObject data type available to subsequent Invoke Java operators.

For example, say that you want to use the Invoke Java operator in a process. You include it in a process in the process editor and name it Java Operator 1. Once the operator runs, the Java object is saved to the operator dataset and displays as a JavaObject data type. Now you have another Invoke Java operator later in your process. You can use that same JavaObject from Java Operator 1 in your new Java Operator 2. You pass the saved object from Java Operator 1 to Java Operator 2 as a JavaObject data type.

The Invoke Java operator cannot be executed on an Orchestrator. This operator only runs on an agent.

Input Parameters

Input parameters for the Invoke Java operator are as follows.

Code

List External Jar Paths?

Select to provide a list of paths to the required external JARS in the External Jar Paths as Expression field.

Clear to provide the paths to the required external JARs as an expression, that is, as a dataset variable in the External Jar Paths as Expression field.

This check box is selected by default.

External Jar Paths

Specifies the list of paths to the external JARs that are required by the main method code of the operator, if any.

The Invoke Java operator loads the JARs listed in this field. Any JAR entered in this list is available to the Java code executed by the operator. The classes defined in the operator-level JARs override the same classes specified in the JARs at the operator category level.

For each path, you can:

■ Enter the full path to a JAR file that resides on the host where the CA Process Automation agent (mapped to the touchpoint) is running. The full path is specified as follows:

– Starts with: /

– Starts with: \\

– Of the form: ^.:.* (a regular expression that starts with one character followed by a colon - : - and then the rest of the string.)

■ Enter the path to a JAR file that is downloadable over HTTP. Verify that the HTTP path does not require authentication and is not through an HTTP proxy. The path to the JAR must start with http:// or https://.

■ Any other path is assumed to be a relative path to a JAR file that was uploaded in the CA Process Automation user resources. CA Process Automation appends the JAR file path to the path of the "CA Process Automation User Resources" directory of the agent (that is mapped to the touchpoint) running the operator.

Do not start the JAR file relative path with:

– /

– \\

Otherwise, CA Process Automation assumes that the JAR file path is a full path.

Resources within CA Process Automation, including user resources, are mirrored within the mirroring interval of the agent. Verify that the JARs uploaded in the user resources are already mirrored before using them in the Invoke Java operator.

Class Files

In addition to external JARS, you could load .class files as follows:

- For .class files in an unnamed package, enter a path that ends with the directory that contains the .class files.

  For example, if MyAccount.java does not belong to a package, and MyAccount.class is located at:

  C:\java\tests\MyAccount.class

  ...then set the operator to use the following path:

  C:\\java\\tests

- For .class files in a named package, enter a path that ends with the directory that contains the "root" package. This package is also known as the first package in the full package name.

  For example, if MyAccount.java belongs to package com.ca.tech, and MyAccount .class is located at:
  C:\java\othertests\com\ca\tech\MyAccount.class

  ...then set the operator to use the following path:

  C:\\java\\othertests

If you specify the path to a directory (to load .class files), enter it as a full path. You can also enter it as a relative path to CA Process Automation User Resources. Do not enter the path as an HTTP path.

Specify the path to a directory to load .class files, not JAR files. Unlike .class files, each JAR file requires a separate path that ends with the JAR file (not the directory where it resides).

External Jar Paths as Expression

Specifies the indexed string dataset variable that contains the list of paths to the external JARs required by the main method code of the operator, if any.

Enter Required Main Method?

Select to provide the Java code in the Required Main Method field.

Clear to provide the Java code as a dataset variable in the Required Main Method as Expression field.

This check box is selected by default.

Required Main Method

Specifies the Java code text of the main method. You can browse to locate any file that contains this code. The main method consists of normal Java statements and expressions. You must initialize and use objects defined in the Java SDK or the external JARs.

Note: CA Process Automation parses the code and checks for its structural validation when you click OK. An error message displays if an error is found in the structure of the code.

See Java Code in the Invoke Java Operator (see page 393).

Required Main Method as Expression

Specifies the dataset variable that contains the content of the main method. The main method consists of normal Java statements and expressions. You must initialize and use objects defined in the Java SDK or the external JARs. No structural validation is performed.

See Java Code in the Invoke Java Operator (see page 393).

Set Context Class Loader?

Set this field (to something different than default) if your main method/external JARs rely on the Java context class loader to load classes. Set the Java context class loader to either the operator class loader or the module class loader to avoid a ClassCastException.

The Invoke Java operator uses a chain of class loaders to load classes while running the Java code. This chain was designed as follows (among other class loaders):

1. Operator Class Loader: class loader that loads the classes provided at the operator level

2. Module Class Loader: class loader that loads the classes provided at the module level

3. Context Class Loader

4. Regular java Class.ForName

The operator consults each class loader before moving to the next (if the class is not found).

This chain works as long as the code that you execute does not explicitly use its own class loader to load a class. In this case, you see in the logs a 'ClassCastException'.

Example:

Consider class MyChildXMLParser extends class MyParentXMLParser. The following code fragment listed creates a MyParentXMLParser by using a Java factory. This factory actually loads and creates a MyChildXMLParser, which is then cast into a MyParentXMLParser object:

```
public MyParentXMLParser() {
        super((MyParentXMLParser)ObjectFactory.createObject("co
m.ca.parser.MyChildXMLParser"));
          ....
      }
```

In this example, consider:

■   The ObjectFactory.createObject() method actually calls its own class loader to load the MyChildXMLParser class. This behavior is typical of Java factories, where they either use the system class loader, or (if it exists) a context class loader to load the class (instead of using the class loader that is used by the executing program).

■   The MyParentXMLParser class is loaded by the Invoke Java operator's class loader (as selected using the previous chain.)

■   The cast: ((MyParentXMLParser)ObjectFactory.createObject) throws a ClassCastException. Although MyChildXMLParser extends MyParentXMLParser, the two classes were loaded by different class loaders. As a result, they are completely different from each other.

To resolve this issue, set the field 'Set Context Class Loader' to either:

■   "1 : Operator class loader": If the jar that contains 'MyChildXMLParser' and 'MyParentXMLParser' classes is provided in the operator properties.

■   "2 : Module class loader": If the jar that contains 'MyChildXMLParser' and 'MyParentXMLParser' classes is provided in the module properties.

Note: "0 : Default" is used in all other cases where your Java code does not explicitly load classes using its own class loader. This value is the default value of this field.

By setting the context class loader to the class loader of the Invoke Java operator, the Java factories that are called by the user's code are forced to use the Invoke Java operator class loader. This action removes ClassCastException.

Java Code in the Invoke Java Operator

When using the Invoke Java operator, use the following guidelines for implementing Java code:

■ CA Process Automation executes the Java code in a BeanShell interpreter. Use this operator with Java code syntax or BeanShell scripting syntax *without* use of BeanShell commands, in the following cases:

   – BeanShell commands do not work under Strict Java Mode (set at the module level).

   – BeanShell commands that modify the classpath are not recommended. They can affect the way CA Process Automation saves Java Object instances from the running Java code into the dataset of the operator.

   – BeanShell commands that modify the classpath can affect the way CA Process Automation loads Java Object instances from a CA Process Automation dataset into the code.

   For more information about BeanShell syntax and commands, see the following site: http://www.beanshell.org/

■ You can use the standard Java variable modifiers on typed variables:

   – private / protected / public

   – transient

   – volatile

   – static

   – final

   The BeanShell interpreter only implements "final" (and ignores the others).

■ You can use the standard Java modifiers on methods:

   – private / protected / public

   – final

   – native

   – abstract

   – static

   – synchronized

   Only "synchronized' is currently implemented. The BeanShell interpreter ignores the others.

- Complete all class definitions in external JARS and use them in the Main Method code of the operator.

- The java rt.jar, which contains all the core java libraries, is automatically placed in the classpath of the operator at runtime.

- The JAR files used by CA Process Automation are in the classpath of the operator at runtime. Your code can work, even without listing all the needed JARS in the operator/category, if you happen to use classes already used by CA Process Automation.

- Common Java core packages and some extensions are automatically imported into your Java code at runtime. You do not need to import them in your code. Packages are as follows:

  - javax.swing.event

  - javax.swing

  - java.awt.event

  - java.awt

  - java.net

  - java.util

  - java.io

  - java.lang

  - bsh.EvalError

  - bsh.Interpreter

- The Java code can consist of normal Java statements and expressions. You can also define your own methods and use them inside the code. An example is as follows:

```
// Import the classes that you want to use
import ca.tech.pam.MyAccount;
// Note: no need to import StringBuffer and Date because they are part of the
// automatically imported packages
// import java.lang.StringBuffer;
// import java.util.Date;
// Note: the jar that contains the ca.tech.pam.MyAccount class
// must be in the list of External Jars of the operator or the module;
// but java lang and java util are in rt.jar, which is automatically put in the classpath
MyAccount acct = new MyAccount(newDate(),100);
// Use the public methods of the MyAccount object
acct.addFunds(34);
acct.subFunds(10);
// Define your own method
String getStatement(MyAccount acc) {
    StringBuffer strBuff = new StringBuffer("Account Balance: " + acc.getBalance());
    Date dt = new Date(System.currentTimeMillis());
    strBuff.append(" on date: " + dt);
    return strBuff.toString();
}
// Use the method you defined
// also print the statement using the 'logger' object that you
// setup in the 'Logger' page of the operator
logger.info(getStatement(acct));
```

Note: To execute this statement, set the logger to true and provide the log file name. Otherwise an error occurs during execution.

At the end of execution, the log message contains:

Account Balance: 124. on date: Wed Jul 13 12:53:37 EDT 2011

(The message includes the correct date and time of execution.)

Input/Output

Parameters

The CA Process Automation parameters to pass to the main method. Enter the parameters in the order that they are to be passed to the main method. Leave this field blank if no parameters are required.

Only simple CA Process Automation parameter types can be passed to the main method as follows:

- PAM Boolean is passed as a Boolean object.

- PAM Date is passed as a Date object.

- PAM Double is passed as a Double object

- PAM Integer is passed as an Integer object.

- PAM Long is passed as a Long object.

- PAM String is passed as a String object.

- PAM Object Reference is passed as a String object

- PAM JavaObject is deserialized and loaded into a Java object instance of its original class type, and then passed to the Java code.

Note: The operator (or operator category) must contain the path to the JAR file that contains the class definition of this object. Otherwise, the operator fails with the reason:

Class Not Found Error when deserializing object. Make sure the class jar is in the operator or module list of jars.

Complex CA Process Automation parameters types (indexed types, ValueMaps, and so on) *cannot* be passed to the Java code.

The main method can access the passed parameters through the args array of objects:

- args[0] corresponds to the first parameter in the list.

- args[1] corresponds to the second parameter in the list, and so on.

Output Variable Names

The names of the variables that are saved in the operator dataset at the end of execution of the main method. These variables must be defined in the scope of the main method. Leave this field blank if no output variables are to be saved in the operator dataset.

The output variables are saved as follows:

- Boolean object is saved as a PAM Boolean

- Date object is saved as a PAM Date

- Integer object is saved as a PAM Integer

- Number object is saved as PAM Long or Double object

- String object is saved as PAM string

- Character object is saved as PAM string

- An array of any of these listed objects is saved as an indexed CA Process Automation type. The type of the first object in the array of objects defines the CA Process Automation type.

- Undefined is saved as a CA Process Automation string with 'undefined' as its value.

- Any other Java object not listed here is serialized and saved as a CA Process Automation JavaObject.

  Note: The Java object must be serializable (implements java.io.Serializable) to save it as a CA Process Automation JavaObject. Otherwise, the operator fails with the reason:

  `Error when serializing object of class: x. Object is not serializable.`

  Where *x* is the class name of the object.

Logger

Use Logger?

Set this field to true to use an instance of an org.apache.log4j.Logger object to log data to the specified log file. The logger handles opening and closing the file. The logger is available in the context of the main method and can be used as 'logger.debug()', 'logger.info()', and so on.

True

Prompts the operator to use true. The operator uses an instance of 'logger'.

False

Prompts the operator to use false. The operator does not use an instance of 'logger'.

Blank

Prompts the operator to use the value set in the Use Default Logger field of the operator category. If this value is blank at the operator category level, Use logger? is set to false by default.

Any other value prompts the operator to use false, and the operator does not use an instance of 'logger'.

If an instance of 'logger' is used, then it is available in the context of the main method of the operator. 'logger' is used as follows:

- logger.debug("my log message")

- logger.info("my log message")

- logger.warn("my log message")

- logger.error("my log message")

- logger.fatal("my log message")

If an instance of 'logger' is not used, the 'logger' object does not exist in the context of the main method of the operator.

Log File Path

The path to the log file used by the logger. This path must point to a file that resides on the CA Process Automation agent host. If this field is empty, the operator inherits the value set in the Default Log File Path field of the operator category.

Log Level

Specify the log level of the logger.

0

Prompts the operator to use DEBUG, which causes the logger to write Debug, Info, Warn, Error, and Fatal log messages.

1

Prompts the operator to use INFO, which causes the logger to write Info, Warn, Error, and Fatal log messages.

2

Prompts the operator to use WARN, which causes the logger to write Warn, Error, and Fatal log messages.

3

Prompts the operator to use ERROR, which causes the logger to write Error and Fatal log messages.

4

Prompts the operator to use FATAL, which causes the logger to write Fatal log messages.

Blank

Prompts the operator to inherit the value set in the Default Log Level of the operator category. If this value is blank at the operator category level, Log Level is set to Debug by default.

Any other integer value

Prompts the operator to use DEBUG.

Note: You can overwrite the log level at run time in the main method of the operator. This example sets the log level to Fatal:

```
import org.apache.log4j.Level;
logger.setLevel((Level) Level.FATAL);
```

Append to Log File?

Set this field to true to append any data from this operator to the log file (if it exists).

True

Prompts the operator to use true. The operator appends to the log file.

False

Prompts the operator to use false. The operator deletes the content of the existing log file before writing the new data from the operator.

Blank

Prompts the operator to use the value set in the Append to Default Log File? field of the operator category. The operator can append to the log file depending on what the value is set at. If this value is blank at the operator category level also, Append to Log File? is set to false by default.

Any other value prompts the operator to use false, and the operator does not append to the log file.

Log Data Without Logging Info?

Set to true to let the logger write the data with no additional logging information. Only the log message is written.

Set to false to write additional logging information in the following format:

`Day Month Year Hours:Minutes:Secs Log_level [UUID of the Invoke_Java operator that logged this message]: log message`

True

Prompts the operator to use true. The logger writes data with no additional logging information.

False

Prompts the operator to use false. The logger writes data with additional logging information.

Blank

Prompts the operator to use the value set in the Default Log Data Without Logging Info? field of the operator category. If this value is blank at the operator category level, Log Data Without Logging Info? is set to false by default.

Any other value prompts the operator to use false. The logger writes data with additional logging information.

Output Parameters

ErrorLineNumber

If an error occurs due to the executing main method, this variable contains the number of the line of code that caused an error (if available). This field is empty if no error occurs due to the executing Java code.

ErrorMessage

If an error occurs due to the executing main method, this variable contains the error message. This field is empty if no error occurs due to the executing Main method.

ErrorRoot

If an error occurs due to the executing main method, this variable contains the line of code that caused the error (if available). This field is empty if no error occurs due to the executing main method.

ErrorException

If the executing main method throws an exception, this variable contains the exception that was thrown. This field is empty if no error occurs due to the executing main method.

Result

> **1**
>
>> The operator finishes successfully.
>
> **-1**
>
>> The operator fails.

Reason

> **Completed**
>
>> The operator finishes successfully.
>
> **Error Message**
>
>> The operator fails. If an error occurs due to the executing main method, this variable contains the ErrorMessage, the ErrorLineNumber, the ErrorRoot, and the ErrorException (when applicable).

ResponseMessage

ResponseCode

externalOpJarsType

externalOpJars

externalOpJarsExpression

inlineScriptType

inLineScript

scriptExpression

parameters

outputVariables

useLogger

logFile

logLevel

appendToLogFile

useSimpleLoggerLayout

Java Example

The following example is located in the Examples section of the Required Main Method field.

```
/*
The Main Method is used to invoke objects and methods defined in the Java SDK or in external Jars.
The Main Method consists of normal Java statements and expressions.
You can also:
- Define your own methods and use them inside the Main Method
- Pass input parameters to the Main Method
- Save output variables in the Operator's dataset at the end of execution of the Main Method
- Use a logger object in the Main Method

Typically, you would complete all class definitions in external Jars and list them in the External
Jar Paths of the operator, then initialize and use these objects in the Main Method.
*/

/*
Below is an example on how to initialize and use a MyAccount object, which is defined in an external
Jar file.
Operator Configuration:
1. Specify the path:
"Invoke_Java_Op_Example_Jars/MyAccount.jar" in the operator's list of External Jar Paths.
MyAccount.jar contains the ca.tech.pam.MyAccount class, which is used in the code below. During the
installation of CA Process Automation, MyAccount.jar is uploaded as a User Resource.
2. Specify a dataset variable of type Date as the first object in the list of Input Parameters of the
Operator.
    This Parameter can be accessed in the Main Method as args[0].
3. Specify a dataset variable of type Integer and value 100 as the second object in the list of Input
Parameters of the Operator.
    This Parameter can be accessed in the Main Method as args[1].
4. Specify the variable name acct (without quotes) as the first object in the list of Output Variable
Names of the Operator.
    acct is created in the Main Method as a MyAccount object, hence at the end of execution of the
operator, acct will be saved in the operator's dataset as a variable of type JavaObject.
5. Configure the operator to use a logger, set 'Log File Path' to a local file path, 'Log Level' to
Info, 'Append to Log File?' to false, and 'Log Data Without Logging Info?' to true.
*/

// Import the classes that you want to use
import ca.tech.pam.MyAccount;

// Note: no need to import StringBuffer and Date (used below) because they are part of the
// automatically imported packages (full list of these packages is provided in the documentation)
// import java.lang.StringBuffer;
// import java.util.Date;

// Initialize MyAccount object
// Note that the MyAccount constructor is defined in the external jar as:
```

```
// public MyAccount(Date date, int balance)
MyAccount acct = new MyAccount(args[0], args[1]);

// Use the public methods of the MyAccount object
// Note that addFunds is defined in the external jar as:
// public int addFunds (int amnt)
acct.addFunds(34);

// Note that subFunds is defined in the external jar as:
// public int subFunds (int amnt)
acct.subFunds(10);

// Define your own method
String getStatement(MyAccount acc) {
    StringBuffer strBuff = new StringBuffer("Account Balance: " + acc.getBalance());
    Date dt = new Date(System.currentTimeMillis());
    strBuff.append(" on date: " + dt);
    return strBuff.toString();
}

// Use the method you defined and also print the statement using the 'logger' object that you
// setup in the 'Logger' page of the operator
logger.info(getStatement(acct));

// At the end of execution of the operator:
// acct will be saved in the operator's dataset as a variable of type JavaObject.
// The logger's log file will contain the message:
//      Account Balance: 124 on date: Thu Aug 22 11:27:29 EDT 2013
//      (The message includes the correct date and time of execution).
```

Resource for Running Invoke Java Operator Example

The installation process adds one resource to the User Resource folder under Repository in the Manage User Resources palette on the Configuration tab. The JAR file, MyAccount.jar, is located in the Invoke_Java_Op_Example_jars folder. You can use the MyAccount.jar file to run the Java example that is provided in the Required Main Method field of the Invoke Java operator.
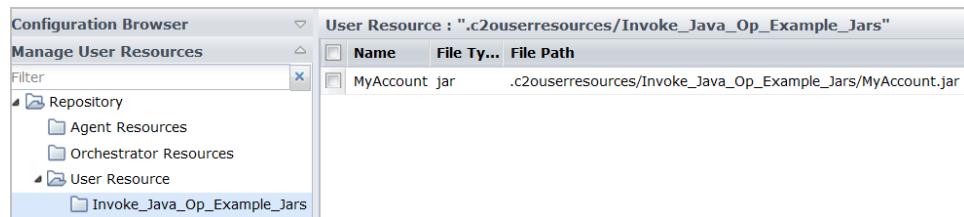
Operator Ports

Success

The operator finishes successfully.

Failure

The operator fails due to any of the following reasons:

■   You use a BeanShell command in the operator code when the operator is set to run in Strict Java Mode. A BeanShell command may not be supported in Strict Java Mode.

■   You use untyped variable declarations in the operator code when the operator is set to run in Strict Java Mode.

■   An error occurs due to the executing code. For example: calling the wrong method on a Java object.

■   The operator code threw an exception while it was executing.

■   You try to use the logger in the code of the operator while the operator is set not to use the logger. The logger is not defined in the operator code context, and the operator cannot resolve any of methods of the logger.

■   The logger is configured to use a read-only file.

■   The logger is configured to use a log file that is actually a directory.

■   You enter a bad path in the external JAR files of the operator category.

■   The Java object that CA Process Automation attempts to save in the operator at the end of execution is not serializable. The operator fails with an error message.

■   You pass a dataset variable of type: JavaObject to the operator. However, you do not specify the JAR file where the class definition of the Java object resides. An error indicates that the operator failed to read the class descriptor when deserializing the object.

- You pass an empty dataset variable of type: JavaObject (value: '[JavaObject]') to the operator. The operator fails with an error indicating that it failed when deserializing object Null.

- The list of JARS for the operator contains a JAR file that does not exist.

- The list of JARS for the operator category contains a JAR file that does not exist.

- The Invoke Java operator cannot run on an Orchestrator. If the Target field entry resolves to an Orchestrator, the operator fails with the following output variables:

  - **ResponseMessage**=The Invoke Java operator cannot be executed on an Orchestrator.

  - **ResponseCode**=SYSTEM_ERROR

- You did not specify Java code in the main method of the operator. The main method consists of normal Java statements and expressions. You must initialize and use objects defined in the Java SDK or the external jars.

  The Invoke Java operator is typically used in a way where you specify external jar paths in the operator or module. You must also enter java code in the main method of the operator. This code uses the objects that you defined in the external jar files.

Custom Ports

Available if set by the user during the process design.

Run JavaScript Operator

The Run JavaScript operator executes calculations and performs dataset variable assignments. The operator performs the following actions:

- Interprets JavaScript statements in its source code.

- Allows computations to set values for variables. These values can then be used for parameter settings in subsequent operators in the same process or in other processes.
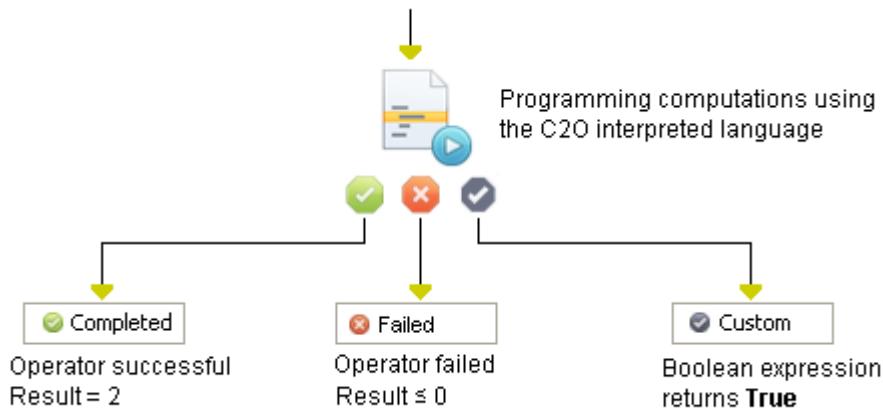
Input Parameters

SourceCode

Opens the code editor.

Use the code editor to specify one or more JavaScript statements. Each statement ends with a semicolon (;). For example, the following statements set the day, month, and year variables in a named dataset:

```
Datasets["/exploit/variables/date"].day = "31";
Datasets["/exploit/variables/date"].month = "July";
Datasets["/exploit/variables/date"].year = "2013";
```

Note: See the *Content Designer Guide* for more information about using the CA Process Automation Code Editor.

Output Parameters

SourceCode

Chapter 15: Web Services

The Web Services operators support calls to remote services using SOAP or XML. These operators also retrieve responses and save information from the response for use by other operators in a process.

The Web Services operators also provide data management facilities over a network using standard and widely available protocols like HTTP. Support for RESTful services is also provided through the HTTP operators.

HTTP Operators: Common Input Parameters

The input parameters that apply to all HTTP operators fall into separate categories as follows:

- HTTP URL Information (see page 408)

- HTTP Proxy Information (see page 411)

- HTTP Headers Information (see page 415)

- HTTP Cookies Information (see page 415)

- HTTP Response Content Information (see page 416)

- HTTP Configuration Information (see page 418)

The properties of HTTP operators require the following input parameters:

- HTTP Delete - Common only

- HTTP Get - Common and Get Information (see page 430)

- HTTP Head - Common and Head Information (see page 434)

- HTTP Options - Common only

- HTTP Post - Common and Post information (see page 442)

- HTTP Post Form - Common and Post Form information (see page 448)

- HTTP Put - Common and Put information (see page 455)

- HTTP Trace - Common only

Note: Unless otherwise specified, field entries override corresponding field values that are inherited from the operator category level configuration.

HTTP URL Information

HTTP URL Information includes input parameters that apply to the following operators:

- HTTP Delete

- HTTP Get

- HTTP Head

- HTTP Options

- HTTP Post

- HTTP Post to Form

- HTTP Put

- HTTP Trace

URL

Specifies the URL of the HTTP request. The URL starts with http:// or https://.

Valid SSL Certificate?

Specifies whether a valid SSL certificate is found. This field is relevant when querying an HTTPS URL.

Values

- True - Validates the SSL certificate and fail the operator if the certificate is invalid.

- False - Accepts the SSL certificate even if it is invalid and continue to make the HTTP call.

- Empty - Uses the value set for "Default Validate SSL Certificate?" at the operator category level.

Authentication Type

Select one of the following options.

HTTP Authentication

Specifies whether the HTTP server, at the specified URL, requires authentication. The HTTP operators support Basic HTTP authentication, HTTP Preemptive Authentication, or NTLM authentication. If either type of authentication is needed, set this value to true.

Values

- True - Specifies that the HTTP server requires authentication.

- False - Specifies that the HTTP server does not require authentication.

- Blank - Specifies to use the value set at the operator category level.

- Any other value - Same as false.

Default

Blank - Specifies to use the value set at the operator category level.

Notes

- If HTTP authentication is set to false in the operator, then NTLM Authentication, User name, Password, and Domain name are all disabled.

- If HTTP authentication is not set to false in the operator (true, dataset variable or any other value), then the following are enabled:

  - NTLM authentication

  - User name

  - Password

  - Domain name

HTTP Preemptive Authentication

During the interaction with a Web Service, a CA Process Automation HTTP operator typically acts like a web browser where it first negotiates with the Web service on the authentication scheme to use. This negotiation is done before the operator issues its request to the Web service.

Normally, for Basic HTTP authentication, the Web service specifies during the authentication negotiation, with the CA Process Automation operator, that it requires Basic Authentication. If configured for HTTP Authentication, the CA Process Automation operator then agrees with the Web service to use Basic Authentication.

However, some Web services do not specify during the negotiation that they require Basic Authentication. Instead, they assume that the operator will preemptively send the Basic Authentication. This is typically done to reduce the connection overhead. In this case, the CA Process Automation HTTP operator should be configured to use HTTP Preemptive Authentication (instead of HTTP Authentication), which prompts the operator to send Basic HTTP Authentication to the Web service without negotiation.

NTLM Authentication

Specifies whether the HTTP server at the specified URL requires NTLM authentication. CA Process Automation uses basic HTTP authentication if NTLM authentication is not selected.

Values

■   True - Specifies that the HTTP server requires NTLM authentication.

■   False - Specifies that the HTTP server does not require NTLM authentication. The server uses basic HTTP authentication

■   Blank - Specifies to use the value set at the operator category level.

■   Any other value - Same as false.

User name

Specifies the username to use when authenticating against the specified URL.

Password

Specifies the password for the specified User name.

Domain name

Specifies the name of the domain to use when authenticating against the specified URL.

Use the following guidelines:

■   Enter the Domain name (required) if the operator uses NTLM authentication.

■   Leave blank if the Domain name is not required for authentication. A Domain name may not be required if the operator uses basic HTTP authentication.

Usage Notes for Domain name, User name, and NTLM authentication

A blank Domain name field does not automatically prompt the operator to inherit the Domain name value from the operator category. The Domain name field is tied to the User name field as follows:

- If the Domain name for the operator is specified, the operator uses it.

- If the Domain name of the operator is blank and the user name of the operator is specified (not blank), the operator uses a blank Domain name.

The operator uses the default Domain name from the operator category if the following are blank (not specified):

- The Domain name of the operator.

- The User name for the operator.

A specified Domain name is used as follows:

- If the operator uses NTLM authentication, the Domain name is used as provided without being appended to the user name.

- If the operator uses Basic HTTP authentication, the Domain name is appended to the user name as: User name = user name@domain name

HTTP Proxy Information

HTTP Proxy Information includes input parameters that apply to the following operators:

- HTTP Delete

- HTTP Get

- HTTP Head

- HTTP Options

- HTTP Post

- HTTP Post to Form

- HTTP Put

- HTTP Trace

Note: Unless otherwise specified, field entries override corresponding field values inherited from the operator category level configuration.

Use Proxy?

Specifies whether the HTTP calls go through a proxy server. This field overrides the module field. If left blank, the operator uses the default value set at the module level.

Values

One of the following:

- True - Indicates to route HTTP calls through a proxy server.

- False - Indicates that HTTP calls do not go through a proxy server.

- Blank - Indicates to use the value set at the module level.

- Any other value - Same as False.

Notes:

- Setting this field to False disables the remaining fields in HTTP Proxy Information.

- Not setting this field to False in the operator (true, dataset variable or any other value) enables the remaining fields in HTTP Proxy Information.

Proxy Host

Specifies either the URL (with http or https) of the proxy server or the FQDN of the proxy server.

Note: If the FQDN is entered, the HTTP scheme is used to contact the Proxy server, that is, http://<FQDN of proxy>:<port>.

Proxy Port

Specifies the port of the specified Proxy Host.

Values

One of the following:

- Blank - Inherits the Default Proxy Port value set at the module level, if present. Otherwise, port 80.

- The specified port number.

Proxy Authentication?

Specifies whether the proxy server, at the specified proxy URL, requires authentication. Proxy authentication can be either Basic HTTP authentication or NTLM authentication. If either type of authentication is needed, set this value to true.

Values

One of the following:

- True - Indicates the proxy server requires authentication.

- False - Indicates the proxy server does not require authentication.

- Blank - Indicates to use the default value set at the module level.

- Any other value - Same as False.

Proxy NTLM Authentication?

Indicates whether the specified Proxy Host requires NTLM authentication.

Values

One of the following:

- True - Indicates that the specified Proxy Host requires NTLM authentication.

- False - Indicates that the specified Proxy Host does not require NTLM authentication. The proxy host uses basic HTTP authentication.

- Blank - Specifies to use the value set at the module level.

- Any other value - Same as False.

Note

If Proxy Authentication is set to false in the operator then the following are disabled:

- Proxy NTLM authentication

- Proxy User name

- Proxy Password

- Proxy Domain name

If Proxy Authentication is not set to false in the operator (true, dataset variable or any other value) then the following are enabled:

- Proxy NTLM Authentication

- Proxy User name

- Proxy Password

- Proxy Domain name

Proxy User name

Specifies the username to use when authenticating with the proxy.

Proxy Password

Specifies that password associated with the Proxy User name.

Proxy Domain name

Specifies the name of the domain to use when authenticating against the specified Proxy server.

Use the following guidelines:

- Enter the proxy domain name (required) if the operator uses NTLM authentication against the Proxy server.

- Leave blank if the Proxy Domain name is not required for authentication. If the operator uses basic HTTP authentication against the Proxy server, a domain name is typically not required.

Usage Notes for Proxy Domain name, Proxy User name, and Proxy NTLM authentication

- A blank Proxy Domain name field does not automatically prompt the operator to inherit the proxy domain name value from the module.

- The Proxy Domain Name field of the operator is tied to the Proxy User name field of the operator as follows:

  - If the Proxy Domain name of the operator is specified, the operator uses this name.

  - If the Proxy Domain name is blank and the Proxy User name is specified (not blank), the operator uses a blank Proxy Domain name.

  - If the Proxy Domain name is blank and the Proxy User name is not specified (blank), the operator uses the inherited Default Proxy Domain name.

- A specified Proxy Domain name is used as follows:

  - If the operator uses NTLM authentication against the proxy server, the specified Proxy Domain name is used as provided. The Proxy Domain name is not appended to the Proxy User name.

  - If the operator uses Basic HTTP authentication against the proxy server, the specified Proxy Domain name is appended to the Proxy User name as:

  User name = user name@domain name

HTTP Headers Information

Use Indexed Value Map for HTTP Headers?

Specifies whether to use an indexed value map for HTTP request headers.

Values:

- Selected - Indicates to enter HTTP request headers as an indexed value map in the HTTP Headers Indexed Value Map field.

- Cleared - Indicates to enter the HTTP request headers in the HTTP Headers field.

HTTP Headers

Specifies the names of the HTTP headers in the Key column and the values of the HTTP headers in the Value column. Headers must be in US-ASCII format.

Use the buttons to add, remove, or reorder headers.

Note: The operator ignores any header where the Key is blank, that is, where no header name is specified.

HTTP Headers Indexed Value Map

Specifies the name of an indexed ValueMap that contains the HTTP header names and corresponding values. The indexed ValueMap must be of the same format as the one listed in the HTTP Headers field. The indexed value map must have both the Key and Value parameters.

Note: The operator ignores any header where the Key is blank, that is, where no header name is specified.

HTTP Cookies Information

HTTP Cookies Store Indexed Value Map

Type an indexed value map that contains the HTTP cookies to set in this operator.

This field enables HTTP state management by allowing users to pass the HTTPCookiesStore from one operator to another who is targeting the same cookie domain. The indexed value map must be of the same format as the ones returned in the HTTPCookiesStore output variable of other HTTP operators. Typically, this field has the following format:

```
PreviousHttpOperator.HTTPCookiesStore
```

By getting the HTTPCookiesStore of another operator, this operator can send any applicable cookies. Applicable cookies include those set in the HTTP request or HTTP response of the previous operator. This operator sends only the unexpired cookies (from the HTTPCookiesStore) whose attributes are applicable to the URL of this operator. Example attributes include domain, path, and isSecure.

HTTP Response Content Information

Save HTTP Response Content to File?

Specifies whether to save the body of the HTTP response message to a file. Select this field to enable the HTTP Response Content File Path field and the If Text Response, Save it Using Encoding field.

Values

- Selected - Saves the body of the HTTP response message to a file.

- Cleared - Does not save the body of the HTTP response message to a file.

HTTP Response Content File Path

Specifies where to save the HTTP response message body. Type the path of the local file on the host where the touchpoint is running.

If Response File exists?

Specifies the action to take if the response file exists. Available actions are to create a file or overwrite the content of the existing file.

The path of the file is listed in the HTTPResponseContentFilePath operator output variable.

Values

This value can be:

- createFile - Indicates to create a file.

- overwriteFile - Indicates to overwrite the content of the response file with the new HTTP response message body.

- Blank - Same as createFile.

- Any other integer - Same as createFile.

If Text Response, Save it Using Encoding

Specify this encoding if you are expecting a text response. The content-type of the response is of the format:

`text/XXXX`

This encoding is used to write the response in the Response File.

If the response received is not of type text/XXXX, this field is ignored.

Values

This value can be:

- 0 : Specified in HTTP Response Header - Enter 0 to use the encoding specified in the HTTP response header.

- 1 : PAM's Default System Encoding - Enter 1 to use CA Process Automation's default system encoding.

- 2 : Specify an Encoding in 'User Specified Text Response Encoding' - Enter 2 to specify the encoding (to be used) in the User Specified Text Response Encoding field.

- Blank - Prompts the operator to use 0 (use the encoding specified in the HTTP Response Header).

- Any other integer - Prompts the operator to use 0 (use the encoding specified in the HTTP Response Header).

If this field is set to 0 or 1, then field User Specified Text Response Encoding is disabled.

User Specified Text Response Encoding

Specify an encoding to use when writing the text response in the Response File.

Save HTTP Response Content to Dataset Variable?

Specifies whether to save the body of the HTTP response message to the HTTPResponseContent variable in the dataset of the operator. When saving is selected, the HTTP Response Dataset Field Size Limit field is enabled.

Values

- Selected - Saves the body of the HTTP response content to the HTTPResponseContent variable in the dataset of the operator.

- Cleared - Does not save the HTTP response content.

HTTP Response Dataset Variable Size Limit (bytes)

Specifies the maximum number of bytes (of the HTTP response message body) to save in the HTTPResponseContent dataset variable of the operator.

Value

A numerical value.

Default

4096 bytes (if left blank)

HTTP Configuration Information

HTTP Version

Specifies the HTTP protocol version.

Values:

One of the following:

■ 1.0 - Indicates that the operator is to use HTTP protocol version 1.0.

■ 1.1 - Indicates that the operator is to use HTTP protocol version 1.1

■ Blank - Indicates the operator is to use the value set at the operator category level, where blank or any value other than 1.1 or 1.0 at the category level prompts the operator to use 1.1.

■ Any value other than 1.0 or 1.1 - use HTTP protocol 1.1

Default:

Blank

Connection Timeout (sec)

Specifies the maximum amount of time to wait for an HTTP connection to establish before the operator times out.

Values:

One of the following:

■ A numeric value indicating the connection timeout in seconds.

■ 0 indicates no timeout, that is, zero seconds.

■ Blank indicates the Default Connection Timeout set at the operator category level, if available, otherwise 0 seconds.

Default:

Blank.

Socket Timeout (sec)

Specifies the maximum amount of time to wait between two consecutive HTTP response data packets.

Values:

One of the following:

- A numeric value indicating the socket timeout in seconds.

- 0 indicates no timeout, that is, zero seconds.

- Blank - Indicates the Default Socket Timeout set at the operator category level, if available, otherwise 0 seconds.

Default:

Blank.

Handle Redirects?

Indicates whether to handle redirects automatically.

Values:

One of the following:

- True - Handle redirects automatically.

- False - Do not handle redirects automatically.

- Blank - Use the Default Handle Redirects? value set at the operator category level.

- Any other value - Indicates false.

Default:

Blank.

Maximum Number of Redirects

Specifies the maximum number of redirects to follow, when Handle Redirects? is set to True.

Values:

One of the following:

- A numeric value indicating the maximum number of redirects to allow.

- Blank - The Default Maximum Number of Redirects, if set. Otherwise, 100.

Default:

Blank.

HTTP Operators: Common Output Parameters

Output variables do not contain data when the operator does not receive an HTTP response due to an error such as the following:

- Input contains an unknown URL.

- The HTTP connection times out.

- The socket times out.

HTTPRequestUrl

Specifies the HTTP request URL, including any URL parameters.

HTTPResponseStatusLine

Specifies the status line of the HTTP response. The status line is the first line of the HTTP response message. The status line consists of the protocol version, the status code, and the associated reason phrase.

HTTPResponseStatusCode

Specifies the status code of the HTTP response. The operator fails or succeeds depending on this status code.

- The operator fails if the status code is greater than or equal to 300.

- The operator succeeds if the status code is less than 300.

HTTPResponseReasonPhrase

Specifies the reason phrase of the HTTP response.

HTTPResponseProtocolVersion

Specifies the protocol version of the HTTP response.

HTTPResponseContentType

Specifies the content-type header of the HTTP response content.

HTTPResponseContentCharset

Specifies the character encoding of the HTTP response content. This character encoding is part of the content-type header, and appears in the following form:

```
"content-type= xxxxx; charset=xxxx"
```

This charset is only set with an all character content-type such as text/xxx.

HTTPResponseContentLength

Specifies the number of bytes of the HTTP response content. A negative number means that the content length is not known.

HTTPResponseContentEncoding

Specifies the content-encoding header of the HTTP response content. Blank indicates that the content-encoding is unknown.

HTTPResponseContentIsChunked

True indicates that the HTTP response content was received with chunked encoding. False is returned if the True condition is not met.

HTTPResponseContentFilePath

Specifies the path to the file where the HTTP response content was saved. Blank indicates that the operator is not set up to save the HTTP response content (message body) to a file.

■ If the input for If Response File exists? was 0 and the file path that was specified as input in HTTP Response Content File Path exists, then the HTTPResponseContentFilePath field contains the path to the new file where the HTTP response content was saved.

■ If the input for If Response File exists? was 1 and the file path that was provided as input in HTTP Response Content File Path exists, then the HTTPResponseContentFilePath field contains the path provided in HTTP Response Content File Path.

HTTPResponseContent

Specifies the HTTP response content, up to the number of bytes entered in HTTP Response Dataset Variable Size Limit (bytes) field. Blank can indicate that the operator is not set up to save the HTTP response content (message body) to its dataset. Blank also can indicate that the HTTP response content is empty.

HTTPResponseHtmlContent

Specifies the HTTP response content rendered as HTML in the dataset of the operator. The content-type header starting with "text/html" indicates that the HTTP response content is HTML. When CA Process Automation detects that the HTTP response content is HTML, the HTTP response content is rendered as HTML in the dataset of the operator. The raw data remains accessible for javascript code in HTTPResponseContent. Blank can indicate that the operator is not set up to save the HTTP response content (message body) to a dataset. Blank can also mean that CA Process Automation detects that the HTTP response content is not HTML or that the HTTP response content is empty.

Note: CA Process Automation renders only basic HTML pages. CA Process Automation does not render complex HTML pages.

HTTPResponseHeaders

Specifies the HTTP headers of the HTTP response. The headers are returned as an indexed ValueMap where each ValueMap contains a single header and the following two parameters:

Key

Specifies the name of the HTTP header.

Value

Specifies the value of the HTTP header.

HTTPRequestHeaders

Specifies the HTTP headers of the HTTP request that was sent. This field contains the HTTP headers that were provided as input in the HTTP Headers or HTTP Headers ValueMap fields of the operator. This field also contains the HTTP headers for authentication, proxy, and others that the operator added before sending the request.

The headers are returned as an indexed ValueMap where each ValueMap contains a single header and the following parameters:

Key

Specifies the name of the HTTP header.

Value

Specifies the value of the HTTP header.

HTTPRequestLine

Specifies the request line of the HTTP request that was sent. The HTTP request line contains the HTTP method, the URL, and the HTTP version.

HTTPCookiesStore

Specifies the parsed version of the HTTP cookies sent in the request and the HTTP cookies embedded in the response headers. The cookies are returned as an indexed ValueMap where each ValueMap contains a single cookie that was defined with the following parameters:

Name

Specifies the name of this HTTP cookie.

Value

Specifies the value of this HTTP cookie.

Version

Specifies the version of the cookie specification that this HTTP cookie conforms to.

Domain

Specifies the domain of this HTTP cookie. The HTTP cookie is valid in this Domain.

Path

Specifies the path of this HTTP cookie. This value specifies the subset of URLs, for which this HTTP cookie applies, on the original HTTP server.

ExpirationDate

Specifies the expiration date of this HTTP cookie. Some cookies return an expiration date, while others return a maximum age. The expiration date is returned in the following format:

"yyyy.MM.dd 'at' HH:mm:ss z"

MaxAge

Specifies the maximum age of this HTTP cookie. Some cookies return a maximum age, while others return an expiration date.

Comment

Specifies the purpose of this HTTP cookie.

Ports

Specifies the ports of this HTTP cookie. The ports are returned as a string of comma-separated values. This value specifies the ports on which this HTTP cookie can be sent back in a request header.

IsSecure

One of the following options:

■   True - Indicates that this HTTP cookie can be sent only on a secure connection.

■   False - Indicates that a secure connection is not necessary for sending this cookie.

ResponseHeaderName

Specifies the name of the response header that contains this HTTP cookie. This value can be "Set-Cookie" or "Set-Cookie2".

Result

This value is one of the following options:

1

Indicates that the operator finished successfully.

-1

Indicates that the operator failed.

Reason

This value is one of the following options:

Completed

This reason is associated with the result of 1, successful completion.

<error message>

An explanation of why the error occurred; associated with the result of -1, where the operator failed.

HTTP Operators: Common Output Ports

Success

The operator finished successfully.

Timeout

A connection timeout or a socket timeout occurred.

Failure

The HTTP response has a status code greater than or equal to 300.

The HTTP Response Content can contain the HTTP status code and the reason for operator failure. The HTTPResponseReasonPhrase can contain a generic reason of failure. A generic reason for failure is returned in the HTTP response Status Line. Examine the HTTPResponseContent for details.

Descriptions for status codes 401 and 407 and other failure reasons follow:

401

Status code 401 indicates one of the following conditions:

■   Incorrect URL authentication credentials.

■   Incorrect URL authentication scheme (Basic vs NTLM)

■   No authentication credentials are provided when the HTTP URL requires authentication.

■   URL authentication failure.

With a 401 error code, the HTTP server typically returns the *WWW-Authenticate* response header. This response header contains the authentication scheme that the HTTP server is using. Use this information to determine which authentication scheme to use against the URL. Basic HTTP authentication and NTLM authentication are the two schemes that HTTP operators support.

407

Status code 407 indicates one of the following conditions:

■   Incorrect proxy authentication credentials.

■   Incorrect proxy authentication scheme (Basic vs NTLM).

■   No authentication credentials are provided when the proxy requires authentication.

■   Proxy authentication failure.

With a 407 error code, the HTTP proxy typically returns the *Proxy-Authenticate* response header. This response header contains the authentication scheme that the proxy server is using. Use this information to determine which authentication scheme to use against the proxy. Basic HTTP authentication and NTLM authentication are the two schemes that HTTP operators support.

■ The URL or Proxy Host that was specified as input is unknown.

■ The HTTP call goes through a proxy but the input did not include specification of a proxy. In this case, the operator can specify that the connection to the HTTP URL is refused.

■ Input included an invalid proxy port. In this case, the operator can specify that the connection to the 'ProxyHost:ProxyPort' is refused.

■ Invalid input, such as the following information, was detected:

- Negative connection or socket time outs.

- Negative maximum number of redirects.

- Negative response data set field size limit.

- Save response to file with no file path provided.

Custom Ports

If set by the user during the process design.

HTTP Delete Operator

The HTTP Delete operator sends an HTTP Delete to a URL. The HTTP Delete operation causes the HTTP server to delete the resource that is located at the requested URL.

The HTTP Delete operator can be used for RESTful services.

Important! Use the HTTP Options operator to determine whether the HTTP Delete method is supported. Typically, the HTTP Delete method is disabled on public HTTP servers to prevent deletion of files on the HTTP servers.

Input Parameters

See the following sections for descriptions of the input parameters for the HTTP Delete operator:

HTTP URL Information

See HTTP URL Information (see page 408) for descriptions of input parameters.

HTTP Proxy Information

See HTTP Proxy Information (see page 411) for descriptions of input parameters.

HTTP Headers Information

See HTTP Headers Information (see page 415) for descriptions of input parameters.

HTTP Cookies Information

See HTTP Cookies Information (see page 415) for descriptions of input parameters.

HTTP Response Content Information

See HTTP Response Content Information (see page 416) for descriptions of input parameters.

HTTP Configuration Information

See HTTP Configuration Information (see page 418) for descriptions of input parameters.

Output Parameters

HttpRemoteURL

HttpValidateSSLCert

HttpSvrAuth

HttpSvrNtlmAuth

HttpAuthUser

HttpAuthPwd

HttpAuthDomain

HttpProxy

HttpProxyHost

HttpProxyPort

HttpProxyAuth

HttpProxyNtlmAuth

HttpProxyUser

HttpProxyPwd

HttpProxyDomain

HttpHeaderFieldsType

HttpHeaderFieldsValueMap

HttpHeaderFieldsVarValueMap

HttpCookieFieldsVarValueMap

HttpRespSaveToFile

HttpRespLocalFile

HttpRespLocalFileExists

HttpRespSaveToDataset

HttpRespDatasetVarLimit

HttpVersion

HttpConnectionTimeout

HttpSocketTimeout

HttpHandleRedirects

HttpMaxRedirects

HTTPRequestUrl

HTTPResponseStatusLine

HTTPResponseStatusCode

HTTPResponseReasonPhrase

HTTPResponseProtocolVersion

HTTPResponseContentType

HTTPResponseContentCharset

HTTPResponseContentLength

HTTPResponseContentEncoding

HTTPResponseContentFilePath

HTTPResponseContent

HTTPResponseHtmlContent

HTTPResponseHeaders

HTTPResponseContentIsChunked

HTTPRequestHeaders

HTTPRequestLine

HTTPCookiesStore

See <u>HTTP Operators: Common Output Parameters</u> (see page 420) for more information.

Output Ports

Output ports for the HTTP Delete operator consist of only the <u>common output port</u> (see page 424) for HTTP operators.

HTTP Get Operator

The HTTP Get operator sends an HTTP Get request to a URL.

If the specified URL points to a resource, then the HTTP Get operator retrieves the resource. Use the HTTP Get operator to download a file from an HTTP server by specifying the URL of the file.

If the specified URL points to a process that produces data, then the HTTP Get operator retrieves the data produced by the process. The HTTP Get operator does not retrieve data from the process source.

The HTTP Get operator can be used for RESTful services.

Input Parameters

HTTP URL Information

See <u>HTTP URL Information</u> (see page 408) for descriptions of input parameters.

HTTP Get Information

URL Parameters Encoding

Specifies the character encoding to use when encoding the URL parameters. ASCII is the recommended encoding for URL parameters in an HTTP Get operation. The URL parameters are transferred in the URL. Encodings other than ASCII, UTF-8, and ISO-8859-1 typically do not work.

Values

One of the following:

- ASCII

- UTF-8

- ISO-8859-1

Default

Blank - Same as ASCII.

Use Indexed Value Map for URL Parameters?

Indicates which of the following fields to use for entry of URL parameters: URL Parameters Indexed ValueMap or URL Parameters.

Values

Selected - Enter the URL parameters as an indexed Value Map in the URL Parameters Indexed ValueMap field.

Cleared - Enter the URL parameters in the URL Parameters field.

URL Parameters

Specifies the names of the URL parameters in the Key column, and the values of the URL parameters in the Value column.

Use the buttons to add, remove, or reorder parameters.

Note: The operator ignores any URL parameter where the Key is blank, that is, one where no URL parameter name specified.

URL Parameters Indexed Value Map

Specifies the name of an indexed ValueMap. This name is a dataset variable of type indexed value map with keys and values, where the keys are URL parameter names. The indexed ValueMap must consist of Key and Value parameters. The indexed ValueMap must be in the same format as the one listed in the URL Parameters field. The operator ignores any URL parameter with a blank Key.

HTTP Proxy Information

See <u>HTTP Proxy Information</u> (see page 411) for descriptions of input parameters.

HTTP Headers Information

See <u>HTTP Headers Information</u> (see page 415) for descriptions of input parameters.

HTTP Cookies Information

See <u>HTTP Cookies Information</u> (see page 415) for descriptions of input parameters.

HTTP Response Content Information

See <u>HTTP Response Content Information</u> (see page 416) for descriptions of input parameters.

HTTP Configuration Information

See <u>HTTP Configuration Information</u> (see page 418) for descriptions of input parameters.

Output Parameters

HttpRemoteURL

HttpValidateSSLCert

HttpSvrAuth

HttpSvrNtlmAuth

HttpAuthUser

HttpAuthPwd

HttpAuthDomain

HttpReqUrlParamsEncoding

HttpReqUrlParamsType

HttpReqUrlParamsValueMap

HttpReqUrlParamsVarValueMap

HttpProxy

HttpProxyHost

HttpProxyPort

HttpProxyAuth

HttpProxyNtlmAuth

HttpProxyUser

HttpProxyPwd

HttpProxyDomain

HttpHeaderFieldsType

HttpHeaderFieldsValueMap

HttpHeaderFieldsVarValueMap

HttpCookieFieldsVarValueMap

HttpRespSaveToFile

HttpRespLocalFile

HttpRespLocalFileExists

HttpRespSaveToDataset

HttpRespDatasetVarLimit

HttpVersion

HttpConnectionTimeout

HttpSocketTimeout

HttpHandleRedirects

HttpMaxRedirects

HTTPRequestUrl

HTTPResponseStatusLine

HTTPResponseStatusCode

HTTPResponseReasonPhrase

HTTPResponseProtocolVersion

HTTPResponseContentType

HTTPResponseContentCharset

HTTPResponseContentLength

HTTPResponseContentEncoding

HTTPResponseContentFilePath

HTTPResponseContent

HTTPResponseHtmlContent

HTTPResponseHeaders

TTPResponseContentIsChunked

HTTPRequestHeaders

HTTPRequestLine

HTTPCookiesStore

See HTTP Operators: Common Output Parameters (see page 420) for more information.

Output Ports

Output ports for the HTTP Get operator consist of the common output ports for HTTP operators plus an additional failure case.

Operator Failure

The HTTP Get operator can fail for the following reasons:

■ Failures that are common to output ports for all HTTP operators.

   Note: See HTTP Operators: Common Output Ports (see page 424) for descriptions.

■ You specified an invalid encoding in the URL Parameters Encoding field.

HTTP Head Operator

The HTTP Head operator sends an HTTP Head request to a URL. The HTTP Head method is similar to the HTTP Get method. The difference between the two methods is that with HTTP Head, the HTTP server does not return the resource located at the URL. The HTTP headers of the response are the same for the Head method and the Get method.

The HTTP Head method is typically used to obtain information about a resource without actually getting (transferring) it.

Input Parameters

HTTP URL Information

See HTTP URL Information (see page 408) for descriptions of input parameters.

HTTP Head Information

URL Parameters Encoding

Specifies the character encoding to use when encoding the URL parameters. ASCII is the recommended encoding for URL parameters in an HTTP Head operation. The URL parameters are transferred in the URL. Encodings other than ASCII, UTF-8, and ISO-8859-1 typically do not work.

Values

One of the following:

- ASCII
- UTF-8
- ISO-8859-1

Default

Blank - Same as ASCII.

Use Indexed Value Map for URL Parameters?

Indicates which of the following fields to use for entry of URL parameters. Specifically, indicates whether to specify URL parameters as an indexed ValueMap in the URL Parameters Indexed ValueMap field or to enter the URL parameters in the URL Parameters field.

Values

Selected - Enter the URL parameters as an indexed ValueMap in the URL Parameters Indexed ValueMap field.

Cleared - Enter the URL parameters in the URL Parameters field.

URL Parameters

Specifies the names of the URL parameters in the Key column, and the values of the URL parameters in the Value column.

Use the buttons to add, remove, or reorder parameters.

Note: The operator ignores any URL parameter where the Key is blank, that is, one where no URL parameter name is specified.

URL Parameters Indexed Value Map

Specifies the name of an indexed ValueMap. This name is a dataset variable of type indexed ValueMap with keys and values, where the keys are URL parameter names. The indexed ValueMap must consist of Key and Value parameters. The indexed ValueMap must be in the same format as the one listed in the URL Parameters field. The operator ignores any URL parameter with a blank Key.

HTTP Proxy Information

See HTTP Proxy Information (see page 411) for descriptions of input parameters.

HTTP Headers Information

See HTTP Headers Information (see page 415) for descriptions of input parameters.

HTTP Cookies Information

See HTTP Cookies Information (see page 415) for descriptions of input parameters.

HTTP Response Content Information

See HTTP Response Content Information (see page 416) for descriptions of input parameters.

HTTP Configuration Information

See HTTP Configuration Information (see page 418) for descriptions of input parameters.

Output Parameters

HttpRemoteURL

HttpValidateSSLCert

HttpSvrAuth

HttpSvrNtlmAuth

HttpAuthUser

HttpAuthPwd

HttpAuthDomain

HttpReqUrlParamsEncoding

HttpReqUrlParamsType

HttpReqUrlParamsValueMap

HttpReqUrlParamsVarValueMap

HttpProxy

HttpProxyHost

HttpProxyPort

HttpProxyAuth

HttpProxyNtlmAuth

HttpProxyUser

HttpProxyPwd

HttpProxyDomain

HttpHeaderFieldsType

HttpHeaderFieldsValueMap

HttpHeaderFieldsVarValueMap

HttpCookieFieldsVarValueMap

HttpRespSaveToFile

HttpRespLocalFile

HttpRespLocalFileExists

HttpRespSaveToDataset

HttpRespDatasetVarLimit

HttpVersion

HttpConnectionTimeout

HttpSocketTimeout

HttpHandleRedirects

HttpMaxRedirects

HTTPRequestUrl

HTTPResponseStatusLine

HTTPResponseStatusCode

HTTPResponseReasonPhrase

HTTPResponseProtocolVersion

HTTPResponseContentType

HTTPResponseContentCharset

HTTPResponseContentLength

HTTPResponseContentEncoding

HTTPResponseContentFilePath

HTTPResponseContent

HTTPResponseHtmlContent

HTTPResponseHeaders

HTTPResponseContentIsChunked

HTTPRequestHeaders

HTTPRequestLine

HTTPCookiesStore

See HTTP Operators: Common Output Parameters (see page 420) for more information.

Note: The HTTP Head operator does not transfer the resource located at the specified URL. Therefore, output variables such as HTTPResponseContentType and HTTPResponseContentLength are blank. To find the information about the resource at the specified URL, view the HTTPResponseHeaders. The HTTPResponseHeaders contain information on headers such as content-type and content-length. This is information returned by the HTTP server on this resource.

Output Ports

Output ports for the HTTP Head operator consist of the common output ports for HTTP operators plus an additional failure case.

Operator Failure

The HTTP Head operator can fail for the following reasons:

- Failures that are common to output ports for all HTTP operators.

  Note: See HTTP Operators: Common Output Ports (see page 424) for descriptions.

- The user specified an invalid encoding in the URL Parameters Encoding field.

HTTP Options Operator

The HTTP Options operator sends an HTTP Options request to a URL. The HTTP Options lets you determine what HTTP methods the HTTP server supports.

The supported HTTP methods are listed in the HTTPAllowedMethods output variable at the end of the execution of the operator.

Input Parameters

See the following sections for descriptions of the input parameters for the HTTP Options operator:

HTTP URL Information

See HTTP URL Information (see page 408) for descriptions of input parameters.

HTTP Proxy Information

See HTTP Proxy Information (see page 411) for descriptions of input parameters.

HTTP Headers Information

See HTTP Headers Information (see page 415) for descriptions of input parameters.

HTTP Cookies Information

See HTTP Cookies Information (see page 415) for descriptions of input parameters.

HTTP Response Content Information

See HTTP Response Content Information (see page 416) for descriptions of input parameters.

HTTP Configuration Information

See HTTP Configuration Information (see page 418) for descriptions of input parameters.

Output Parameters

HttpRemoteURL

HttpValidateSSLCert

HttpSvrAuth

HttpSvrNtlmAuth

HttpAuthUser

HttpAuthPwd

HttpAuthDomain

HttpProxy

HttpProxyHost

HttpProxyPort

HttpProxyAuth

HttpProxyNtlmAuth

HttpProxyUser

HttpProxyPwd

HttpProxyDomain

HttpHeaderFieldsType

HttpHeaderFieldsValueMap

HttpHeaderFieldsVarValueMap

HttpCookieFieldsVarValueMap

HttpRespSaveToFile

HttpRespLocalFile

HttpRespLocalFileExists

HttpRespSaveToDataset

HttpRespDatasetVarLimit

HttpVersion

HttpConnectionTimeout

HttpSocketTimeout

HttpHandleRedirects

HttpMaxRedirects

HTTPRequestUrl

HTTPResponseStatusLine

HTTPResponseStatusCode

HTTPResponseReasonPhrase

HTTPResponseProtocolVersion

HTTPResponseContentType

HTTPResponseContentCharset

HTTPResponseContentLength

HTTPResponseContentEncoding

HTTPResponseContentFilePath

HTTPResponseContent

HTTPResponseHtmlContent

HTTPResponseHeaders

HTTPResponseContentIsChunked

HTTPRequestHeaders

HTTPRequestLine

HTTPCookiesStore

HTTPAllowedMethods

Specifies an indexed string containing the names of the methods that are supported by the resource or server located at the specified URL.

See HTTP Operators: Common Output Parameters (see page 420) for more information.

Output Ports

Output ports for the HTTP Options operator consist of only the common output ports for HTTP operators.

See HTTP Operators: Common Output Ports (see page 424) for descriptions.

HTTP Post Operator

The HTTP Post operator sends an HTTP Post request to a URL. The HTTP Post operator typically requests that the HTTP server store the resource that is enclosed as the HTTP request content. The HTTP server process at the specified URL then processes the resource.

Note: Unlike the HTTP Put operator, the URL of an HTTP Post operator points to the process that can handle the enclosed resource.

The HTTP Post operator can be used for RESTful services.

Input Parameters

HTTP URL Information

See HTTP URL Information (see page 408) for descriptions of input parameters.

HTTP Post Information

HTTP Post Information specifies the body of the HTTP request.

Is Chunked?

Specifies whether to send the HTTP request chunked.

When chunk coding is set, the HTTP request does not contain the "content-length" header.

Note: HTTP 1.0 does not support chunk coding. The HTTP Post operator fails with an 'HTTP client protocol error' if chunk coding is set and the HTTP version is 1.0.

Values

One of the following:

■ True - Indicates to send the HTTP request chunked.

■ False - Indicates that the HTTP request is not to be sent chunked.

■ Any other value - Same as False.

Default

Blank - Same as False.

Content Type

Specifies the type of the content that composes the HTTP request body, which is set as a header (content-type) in the HTTP request.

Values

This value is one of the following:

- A media type selected from the drop-down list.

- Blank, where the content is retrieved from a file specified in 'Content File Path.'

  CA Process Automation sets the value to *application/octet-stream*. The HTTP server is responsible for interpreting this generic content-type.

- Blank, where content is retrieved from the 'Content' field.

  CA Process Automation does not set the content-type. The HTTP server is responsible for interpreting the no content-type header.

- A valid media type that you type into the field.

  For valid media types, see the IANA website pages on assignments/media-types.

Note: Make sure to set the right content-type, especially when the content is not retrieved from a file.

Content Character Encoding

Specifies the character encoding of the content of the HTTP request body. Set this field only if the content type is all characters, for example: 'text/XXX'.

Values

This value is one of the following:

- A character set selected from the drop-down list.

- A valid character set (encoding) that you type into the field.

  For valid encodings, see the IANA website pages on assignments/character-sets.

Note: Make sure to set the right character encoding, especially when the content is not retrieved from a file.

Retrieve Content From File?

Specifies whether to retrieve the HTTP request body from a local file on the host where the touchpoint is running.

Values

This value is one of the following:

- Selected - Indicates to retrieve the HTTP request body from a local file on the host where the touchpoint is running.

- Cleared - Indicates to retrieve the HTTP request body from the Content field.

Content File Path

Specifies the path to a local file on the host where the touchpoint is running. The local file contains the HTTP request body.

Content

Specifies the HTTP request body.

HTTP Proxy Information

See HTTP Proxy Information (see page 411) for descriptions of input parameters.

HTTP Headers Information

See HTTP Headers Information (see page 415) for descriptions of input parameters.

HTTP Cookies Information

See HTTP Cookies Information (see page 415) for descriptions of input parameters.

HTTP Response Content Information

See HTTP Response Content Information (see page 416) for descriptions of input parameters.

HTTP Configuration Information

See HTTP Configuration Information (see page 418) for descriptions of input parameters.

Output Parameters

HttpRemoteURL

HttpValidateSSLCert

HttpSvrAuth

HttpSvrNtlmAuth

HttpAuthUser

HttpAuthPwd

HttpAuthDomain

HttpReqIsChunked

HttpReqContentType

HttpReqContentCharset

HttpReqContentFromFile

HttpReqContentFilePath

HttpReqContent

HttpProxy

HttpProxyHost

HttpProxyPort

HttpProxyAuth

HttpProxyNtlmAuth

HttpProxyUser

HttpProxyPwd

HttpProxyDomain

HttpHeaderFieldsType

HttpHeaderFieldsValueMap

HttpHeaderFieldsVarValueMap

HttpCookieFieldsVarValueMap

HttpRespSaveToFile

HttpRespLocalFile

HttpRespLocalFileExists

HttpRespSaveToDataset

HttpRespDatasetVarLimit

HttpVersion

HttpConnectionTimeout

HttpSocketTimeout

HttpHandleRedirects

HttpMaxRedirects

HTTPRequestUrl

HTTPResponseStatusLine

HTTPResponseStatusCode

HTTPResponseReasonPhrase

HTTPResponseProtocolVersion

HTTPResponseContentType

HTTPResponseContentCharset

HTTPResponseContentLength

HTTPResponseContentEncoding

HTTPResponseContentFilePath

HTTPResponseContent

HTTPResponseHtmlContent

HTTPResponseHeaders

HTTPResponseContentIsChunked

HTTPRequestHeaders

HTTPRequestLine

HTTPCookiesStore

See HTTP Operators: Common Output Parameters (see page 420) for more information.

Output Ports

Output ports for the HTTP Post operator consist of the common output ports for HTTP operators plus an additional failure case.

Operator Failure

The HTTP Post operator can fail for the following reasons:

- Failures that are common to output ports for all HTTP operators.

    Note: See HTTP Operators: Common Output Ports (see page 424) for descriptions.

- 'HTTP client protocol error' can occur if chunk coding is set and the HTTP version is 1.0. This can cause a ClientProtocolException.

- The input includes invalid encoding in the Content Character Encoding field.

    Note: CA Process Automation does not verify that the Content Type value specified in the input is valid when the content is retrieved from a file. The receiving HTTP server is responsible for returning an error code if it detects an invalid Content Type.

HTTP Post Form Operator

The HTTP Post Form operator posts data to an HTTP form. The data is of type name-value pairs that can include values that are files.

Unlike the HTTP Post operator, the HTTP Post Form operator sends the HTTP post request to an HTTP form. The URL of the operator is the action element of the form. For example:
```
<form name="myForm" method="post" action ="XXXXXXXX">
```

Express the destination URL as the full path to the script or process on the HTTP server that the action element points to. Do not use a relative path for the destination URL.

The script or process where the action element points must be *publicly* available for the HTTP Post Form operator to call it.

Input Parameters

HTTP URL Information

See <u>HTTP URL Information</u> (see page 408) for descriptions of input parameters.

HTTP Post Form Information

Form Fields Encoding

Specifies the character encoding to use when encoding the form fields parameters. The data in the HTTP Post Form operator goes in the body of the request.

Values

This field accepts the following values:

- A value from the drop-down list.

- A value entered by the user.

- Blank - ISO-8859-1

Default

ISO-8859-1

Use Indexed Value Map for Form Fields?

Specifies whether to specify form fields in the Form Fields Indexed Value Map field or in the Form Fields field. The choice enables entry to the corresponding field.

Values

- Selected - Indicates that the form fields are specified as an indexed Value Map in the Form Fields Indexed Value Map field.

- Cleared - Indicates that the form fields are specified in the Form Fields field.

Form Fields

Specifies the combination of key and value for each form field. The Key column contains the names of the form fields and the Value column contains the values of the form fields.

Use the buttons to add, remove, or reorder parameters.

Note: The HTTP Post Form operator ignores any form field with a blank Key.

Form Fields Indexed Value Map

Specifies the name of an indexed ValueMap.

This Value Map consists of Key and Value. A Key entry is the name of a form field; the corresponding Value entry is the value of that form field. The indexed ValueMap is in the same format as the one listed in the Form Fields field.

Note: The HTTP Post Form operator ignores any form field with a blank Key.

File Fields in Form?

Indicates whether the form contains fields that allow for files to be uploaded to the form. Selecting this check box lets you enter related data in the following fields: Use Indexed Value Map for Form Files?, Form Files, and Form Files Indexed Value Map.

Note: The HTTP request header content-type depends on how the File Fields in Form? field is set.

Values

This field is set in one of the following ways:

■   Selected - The content type of the HTTP request follows, where XXXXXXX is the boundary string that separates the different parts of the HTTP request. Each part in the HTTP request body can have its own content-type.

   `content-type=multipart/form-data;boundary=XXXXXXX`

   –   The *form fields* are encoded using the encoding specified in Form Fields Encoding. Each form field is placed as a separate part in the request body.

   –   Each form field part has a content-type=text/plain; charset='encoding' where 'encoding' is the value (or default value) of Form Fields Encoding.

   –   The *form files* are encoded using the encoding specified in the ContentType and ContentCharacterEncoding (if applicable) columns associated with each form file. Each form file is placed as a separate part in the request body.

   –   The content-type of each form file part is specified in the ContentType column of the Form Files or Form Files Value Map. If applicable, this specification is combined with ContentCharacterEncoding.

■   Cleared - The content type of the HTTP request follows, where 'encoding' is the type of encoding specified in Form Fields Encoding.

   `content-type=application/x-www-form-urlencoded;charset='enc oding'`

   –   The form fields are URL encoded (using the 'encoding').

   –   The form fields are placed in the body of the request.

Use Indexed Value Map for Form Files?

Indicates whether to specify the form file fields in the Form Files field or in the Form Files Indexed Value Map field.

■   Selected - Indicates that the form file fields are specified as an indexed Value Map in the Form Files Indexed Value Map field.

■   Cleared - Indicates that the form file fields are specified in the Form Files field.

Form Files

Specifies each form file with four fields: Key, File Path, Content Type, and Content Character Encoding. Descriptions of each field of a form file follow:

Key

Specifies the names of the form file fields.

FilePath

Specifies the path to the file to upload. (This field is required.) The operator ignores any form file field with a blank FilePath.

ContentType

Specifies the content type of the file to upload. Valid content-types are listed on the IANA website under assignments/media-types. If left blank, the content-type of the corresponding file's part in the HTTP request body is set to application/octet-stream. It is then the responsibility of the HTTP server to interpret this generic content-type.

ContentCharacterEncoding

Specifies the character set of the content of the files to upload, if the corresponding content type is all characters such as 'text/XXX'. Leave blank for other kinds of content types.

Note: For a list of different character sets (encodings), see the IANA website pages under assignments/character-sets.

Form Files Indexed Value Map

Specifies the name of an indexed ValueMap that contains the form file fields names and corresponding values. The indexed ValueMap must be of the same format as the one listed in the Form Files field. That is, it must consist of Key, FilePath, ContentType, and ContentCharacterEncoding.

HTTP Proxy Information

See HTTP Proxy Information for descriptions of input parameters.

HTTP Headers Information

See HTTP Headers Information (see page 415) for descriptions of input parameters.

HTTP Cookies Information

See HTTP Cookies Information (see page 415) for descriptions of input parameters.

HTTP Response Content Information

See HTTP Response Content Information (see page 416) for descriptions of input parameters.

HTTP Configuration Information

See HTTP Configuration Information (see page 418) for descriptions of input parameters.

Output Parameters

HttpRemoteURL

HttpValidateSSLCert

HttpSvrAuth

HttpSvrNtlmAuth

HttpAuthUser

HttpAuthPwd

HttpAuthDomain

HttpFormFieldsEncoding

HttpFormFieldsType

HttpFormFieldsValueMap

HttpFormFieldsVarValueMap

HttpFormMultipartPost

HttpFormFilesToMultipartType

HttpFormFilesToMultipartValueMap

HttpFormFilesToMultipartVarValueMap

HttpProxy

HttpProxyHost

HttpProxyPort

HttpProxyAuth

HttpProxyNtlmAuth

HttpProxyUser

HttpProxyPwd

HttpProxyDomain

HttpHeaderFieldsType

HttpHeaderFieldsValueMap

HttpHeaderFieldsVarValueMap

HttpCookieFieldsVarValueMap

HttpRespSaveToFile

HttpRespLocalFile

HttpRespLocalFileExists

HttpRespSaveToDataset

HttpRespDatasetVarLimit

HttpVersion

HttpConnectionTimeout

HttpSocketTimeout

HttpHandleRedirects

HttpMaxRedirects

HTTPRequestUrl

HTTPResponseStatusLine

HTTPResponseStatusCode

HTTPResponseReasonPhrase

HTTPResponseProtocolVersion

HTTPResponseContentType

HTTPResponseContentCharset

HTTPResponseContentLength

HTTPResponseContentEncoding

HTTPResponseContentFilePath

HTTPResponseContent

HTTPResponseHtmlContent

HTTPResponseHeaders

HTTPResponseContentIsChunked

HTTPRequestHeaders

HTTPRequestLine

HTTPCookiesStore

See HTTP Operators: Common Output Parameters (see page 420) for more information.

Output Ports

Output ports for the HTTP Post Operator consist of the common output ports for HTTP Operators plus an additional failure case.

Operator Failure

The HTTP Post Form operator can fail for the following reasons:

- Failures that are common to output ports for all HTTP Operators.

  Note: See HTTP Operators: Common Output Ports (see page 424) for descriptions.

- The input includes invalid encoding in the Form Fields Encoding or the ContentCharacterEncoding fields.

  Note: CA Process Automation does not verify that the ContentType input is valid when the content is retrieved from a file. The receiving HTTP server is responsible for returning an error code if it detects an invalid content type.

HTTP Put Operator

The HTTP Put operator sends an HTTP Put request to a URL. The HTTP Put operator requests that the resource, enclosed as the HTTP request content, be stored at the specified URL on the HTTP server. The URL must allow CA Process Automation to create a resource or replace an existing one.

If the URL points to an existing resource, the HTTP server handles the enclosed resource as a modified version of the existing resource.

If the URL does not point to an existing resource, then the HTTP server creates a resource with the HTTP request content. The HTTP server then saves the new resource at the specified URL.

Note: Unlike the HTTP Post operator, the URL of an HTTP Put operator identifies the resource enclosed in the HTTP request content.

The HTTP Put operator can be used for RESTful services.

Important! Use the HTTP Options operator to determine whether you can use the HTTP Put operator. The HTTP Put method is typically disabled on public HTTP servers.

Input Parameters

HTTP URL Information

See HTTP URL Information (see page 408) for descriptions of input parameters.

HTTP Put Information

HTTP Put Information specifies the body of the HTTP request

Is Chunked?

Specifies whether to send the HTTP request chunked.

When chunk coding is set, the HTTP request does not contain the "content-length" header.

Note: HTTP 1.0 does not support chunk coding. The HTTP Put operator fails with an 'HTTP client protocol error' if chunk coding is set and the HTTP version is 1.0.

Values

- True - Indicates to send the HTTP request chunked.
- False - Indicates that the HTTP request is not to be sent chunked.
- Any other value - Same as False.

Default

Blank - Same as False.

Content Type

Specifies the type of the content that composes the HTTP request body, which is set as a header (content-type) in the HTTP request.

Values

This value is one of the following:

- A value selected from the drop-down list of different media types.
- Blank, where the content is retrieved from a file specified in 'Content File Path.'

  CA Process Automation sets the value to *application/octet-stream*. The HTTP server is then responsible for interpreting this generic content-type.
- Blank, where content is retrieved from the 'Content' field.

  CA Process Automation does not set the content-type. The HTTP server is then responsible for interpreting the no content-type header.

■ A valid media type that you manually enter into the field.

For valid media types, see the IANA website pages on assignments/media-types.

Note: Ensure that you set the right content-type, especially when the content is not retrieved from a file.

Content Character Encoding

Specifies the character encoding of the content of the HTTP request body. Set this field only if the content type is all characters, for example: 'text/XXX'.

Values

This value is one of the following:

■ A value selected from the drop-down list of different character sets (encodings).

■ A valid character set (encoding) that you manually enter into the field.

You can find valid encodings on the IANA website pages on assignments/character-sets.

Note: Ensure that you set the right character encoding, especially when the content is not retrieved from a file.

Retrieve Content From File?

Specifies whether to retrieve the HTTP request body from a local file on the host where the touchpoint is running.

Values

■ Selected - Indicates to retrieve the HTTP request body from a local file on the host where the touchpoint is running.

■ Cleared - Indicates to retrieve the HTTP request body from the Content field.

Content File Path

Specifies the path to a local file on the host where the touchpoint is running. The local file contains the HTTP request body.

Content

Specifies the HTTP request body.

HTTP Proxy Information

See HTTP Proxy Information for descriptions of input parameters.

HTTP Headers Information

See for descriptions of input parameters.

HTTP Cookies Information

See HTTP Cookies Information (see page 415) for descriptions of input parameters.

HTTP Response Content Information

See HTTP Response Content Information (see page 416) for descriptions of input parameters.

HTTP Configuration Information

See HTTP Configuration Information (see page 418) for descriptions of input parameters.

Output Parameters

HttpRemoteURL

HttpValidateSSLCert

HttpSvrAuth

HttpSvrNtlmAuth

HttpAuthUser

HttpAuthPwd

HttpAuthDomain

HttpReqIsChunked

HttpReqContentType

HttpReqContentCharset

HttpReqContentFromFile

HttpReqContentFilePath

HttpReqContent

HttpProxy

HttpProxyHost

HttpProxyPort

HttpProxyAuth

HttpProxyNtlmAuth

HttpProxyUser

HttpProxyPwd

HttpProxyDomain

HttpHeaderFieldsType

HttpHeaderFieldsValueMap

HttpHeaderFieldsVarValueMap

HttpCookieFieldsVarValueMap

HttpRespSaveToFile

HttpRespLocalFile

HttpRespLocalFileExists

HttpRespSaveToDataset

HttpRespDatasetVarLimit

HttpVersion

HttpConnectionTimeout

HttpSocketTimeout

HttpHandleRedirects

HttpMaxRedirects

HTTPRequestUrl

HTTPResponseStatusLine

HTTPResponseStatusCode

HTTPResponseReasonPhrase

HTTPResponseProtocolVersion

HTTPResponseContentType

HTTPResponseContentCharset

HTTPResponseContentLength

HTTPResponseContentEncoding

HTTPResponseContentFilePath

HTTPResponseContent

HTTPResponseHtmlContent

HTTPResponseHeaders

HTTPResponseContentIsChunked

HTTPRequestHeaders

HTTPRequestLine

HTTPCookiesStore

See HTTP Operators: Common Output Parameters (see page 420) for more information.

Output Ports

Output ports for the HTTP Post operator consist of the common output ports for HTTP operators plus an additional failure case.

HTTP Trace Operator

The HTTP Trace operator sends an HTTP Trace to a URL. The trace method requests that the HTTP server sends back the request that it received. This process can be beneficial for testing purposes and for identifying changes that were made to the request by proxies. The request is sent back as the Response content.

Use the HTTP Options operator to verify if the HTTP Trace is enabled.

Input Parameters

See the following sections for descriptions of the input parameters for the HTTP Options operator:

HTTP URL Information

See HTTP URL Information (see page 408) for descriptions of input parameters.

HTTP Proxy Information

See HTTP Proxy Information (see page 411) for descriptions of input parameters.

HTTP Headers Information

See HTTP Headers Information (see page 415) for descriptions of input parameters.

HTTP Cookies Information

See HTTP Cookies Information (see page 415) for descriptions of input parameters.

HTTP Response Content Information

See HTTP Response Content Information (see page 416) for descriptions of input parameters.

HTTP Configuration Information

See HTTP Configuration Information (see page 418) for descriptions of input parameters.

Output Parameters

HttpRemoteURL

HttpValidateSSLCert

HttpSvrAuth

HttpSvrNtlmAuth

HttpAuthUser

HttpAuthPwd

HttpAuthDomain

HttpProxy

HttpProxyHost

HttpProxyPort

HttpProxyAuth

HttpProxyNtlmAuth

HttpProxyUser

HttpProxyPwd

HttpProxyDomain

HttpHeaderFieldsType

HttpHeaderFieldsValueMap

HttpHeaderFieldsVarValueMap

HttpCookieFieldsVarValueMap

HttpRespSaveToFile

HttpRespLocalFile

HttpRespLocalFileExists

HttpRespSaveToDataset

HttpRespDatasetVarLimit

HttpVersion

HttpConnectionTimeout

HttpSocketTimeout

HttpHandleRedirects

HttpMaxRedirects

HTTPRequestUrl

HTTPResponseStatusLine

HTTPResponseStatusCode

HTTPResponseReasonPhrase

HTTPResponseProtocolVersion

HTTPResponseContentType

HTTPResponseContentCharset

HTTPResponseContentLength

HTTPResponseContentEncoding

HTTPResponseContentFilePath

HTTPResponseContent

HTTPResponseHtmlContent

HTTPResponseHeaders

HTTPResponseContentIsChunked

HTTPRequestHeaders

HTTPRequestLine

HTTPCookiesStore

See HTTP Operators: Common Output Parameters (see page 420) for more information.

Invoke SOAP Method Operator

The Invoke SOAP Method operator invokes SOAP-based Web service methods, either to trigger an action or to retrieve information. A SOAP request can also be generated using a WSDL.
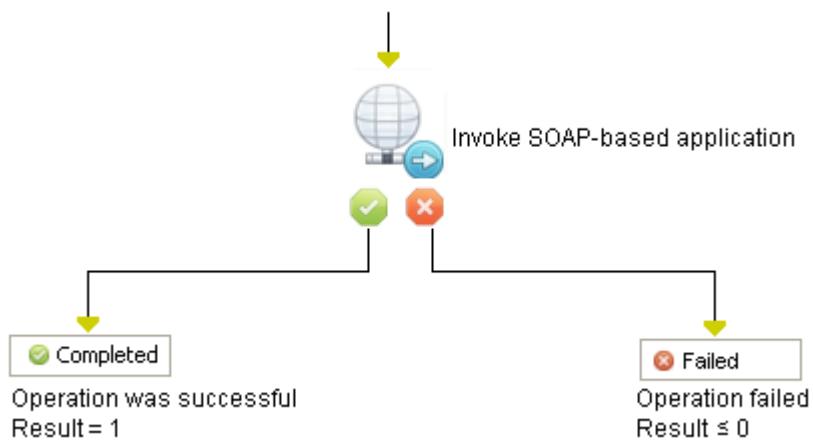
Input Parameters

Input parameters for the Invoke SOAP Method operator are as follows.

WSDL Explorer

WSDL Wizard

A wizard that lets you select SOAP methods. Click to open the WSDL Explorer window, where you can enter:

WSDL URL

- Enter a WSDL URL, then click Load. Examples:

  `http://`*server*`:8080/itpam/soap?wsdl`

  `https://`*server*`:8443/itpam/soap?wsdl`

- Select a recently entered WSDL URL from the drop-down list, then click Load.

  Note: The WSDL URL drop-down list contains the last ten applied entries.

The wizard populates the remaining fields. Revise as needed by selecting other entries from the drop down lists and then click Apply.

WSDL services

WSDL ports

WSDL operations

- Use the default entry that was loaded

- Select an entry from the drop-down list.

  For details on the listed Web Services methods, see the *Web Services API Reference*.

Response area, for example:

```
<tns:AsyncSoapResponse xmlns:tns="http://www.ca.com/itpam">
<tns:MessageID>MessageID__</tns:MessageID>
<!--xsd:Any Type Here-->
</tns:AsyncSoapResponse>
```

SOAP Call Data Parameters

Service URL

Defines the URL for the SOAP service. The URL is typically accessed over HTTP or HTTPS. The URL is typically an entry point for one or more methods.

Method Name

Defines the method or function to run on the SOAP server. The product passes this method to the SOAP service as a MIME SOAPAction header.

Authentication Type

Specifies how to authenticate the call on the SOAP server.

This parameter can have one of the following values:

No Authentication

Do not authenticate the SOAP call.

Basic SOAP

Use the authorized user name and password for basic SOAP authentication.

HTTP Authentication

Use the authorized user name and password for HTTP authentication. If this feature is enabled and the authorized user name and password are provided, then the product uses these credentials for basic HTTP authentication. WS Security applies if the user provides it in the WS Security input page (see page 469).

NTLM Authentication

Use the authorized user name, password, and Domain name that are associated with the SOAP Web Service server to connect to the SOAP server with NTLM authentication.

SOAP Version

Specifies the SOAP server version on which the call is made. This option provides a hint to the underlying logic, which uses the appropriate SOAP MessageFactory when making the SOAP request.

This parameter can have one of the following values:

■　SOAP 1.1

■　SOAP 1.2

Input Source

Defines the source for the SOAP service input request.

The input request can contain macros and XPath assignments that dynamically modify the SOAP request at runtime. If necessary, these XPath assignments let the SOAP request be updated with values obtained at run time.

The following methods include a complete, properly formatted XML message that can include a SOAP envelope:

Inline Text

This option lets you use the Invoke SOAP Method operator to specify the input request. To specify the formatted SOAP input message, use the Inline Text parameter.

Preformatted SOAP File

This option lets you specify the input request in a preformatted SOAP file. To specify the path to a file that contains a message in a valid XML document, use the File Name parameter.

Expression

Select this option if a CA Process Automation expression contains the SOAP request. Define the CA Process Automation expression in the Expression parameter.

Inline Text

Take one of the following actions if you set the Input Source parameter to Inline Text:

■ To use a formatted SOAP input message, click (...) to open the Inline Text dialog, and then type the message.

■ To read a SOAP message from a text file, click (...), Import from File, and then select the file from a local or network drive.

File Name

Defines the fully qualified name of an appropriate file. To use this parameter, set the Input source to Preformatted SOAP File. The product qualifies the file name relative to the touchpoint that runs the Web Services operators.

Expression

Defines a CA Process Automation expression from which to extract the SOAP request.

Saved Call File

Defines the full path to the file to which to write the final outbound SOAP request. Use this file to validate that the data was sent to the Web Services operators. If the call rejects the request because of incorrect values, use the file to debug the request before retransmitting it to the Web Services operators.

Dynamic Parameters

The Dynamic Parameters provide update values in a SOAP request.

Parameter Style

Specifies the method with which to update values in a SOAP request. The Web Services operators support the following methods at run time:

XPath Assignments

Specifies that XPath expressions in the Parameters List parameter update values in a SOAP request.

Macro Expansions

Specifies that macros in the Parameters List parameter update values in a SOAP request.

Parameters List

Specifies the unique data entries in the SOAP request.

- Click Add to add a parameter.

- Click Edit to modify the currently selected parameter.

- Click Delete to delete the currently elected parameter.

A best practice is to add the parameters in the same order in which the WSDL specifies them for the SOAP call.

Each parameter contains the following values:

Macro Name/XPath Query

Defines the name of the macro or the XPath query. If the value is the name of a macro, the value substitutes the macro name. If the value is an XPath query, the value updates the node that the query returns.

Value

Specifies a runtime value for the parameter.

Type

Specifies one of the following data types for the parameter:

- Integer Value

- String Value

- ValueMap

- XML Fragment

Call Results Parameters

The following Call Results parameters determine how to save the results of a SOAP call:

Response Save File

Defines the fully qualified path for the file that restores the response to the SOAP request. Any existing file is overwritten by a new response.

Extract SOAP Response Body First-Level Elements to Individual Dataset Variables

Saves the first-level element in the body of the SOAP response to a separate dataset variable if the SOAP response exceeds 12 KB.

Extract SOAP Response Body to Dataset Variable

Saves the body of the response to a dataset variable.

Extract SOAP Header to Dataset Variable

Saves the header of the response to a dataset variable.

Extract SOAP Header First-Level Elements into Individual Dataset Variables

Saves the first-level headers of the response into an individual dataset variable.

Strip XML Namespaces from Response

Strips the namespaces from a response so that a user can provide simpler XPath expressions to find the value of a specific element. This option is available in all the SOAP operators.

Additional Extracted Data (from Entire Response)

Specifies XPath expressions that extract data from the body of the SOAP response. For each expression specified, define a dataset variable to which to store the extracted data and a data type. Click Add to add an expression, Edit to modify a selected expression, or Delete to delete a selected expression. The ordering of the expressions has no significance to CA Process Automation.

Each parameter contains the following values:

XPath expression

Specifies the XPath expression.

Dataset Variable

Specifies the name of an operator dataset variable in which to save values extracted based on the selected XPath expression.

Type

Specifies the type of element being extracted from the response. Select one of the following currently supported types:

- Integer
- String
- Integer Array
- String Array
- XML Fragment
- XML Fragment Array

MIME Attachments

If the content that you want to send is already in a dataset variable, use the Expression field.

Is an Expression?

If selected, an expression must resolve the attachment.

Expression

Defines the MIME (Multipurpose Internet Mail Extensions) expression that extracts the attachment from the body of the SOAP response. For each expression defined, specify a dataset variable to which to store the extracted data and a data type. The ordering of the expressions has no significance to CA Process Automation. Click Add to add an attachment, Edit to modify a selected attachment, or Delete to delete a selected attachment.

Content Type

Defines the content type of the MIME attachment (for example, text).

Content ID

Defines the unique identifier for the MIME attachment.

File URL

Defines the URL of the MIME attachment.

WS Security

Web services (WS) security enables CA Process Automation to conduct secure SOAP message exchanges with a Web service that requires more security.

WS security features:

- Timestamps
- UsernameTokens
- Signatures
- Encryption

Note: The WS security parameters can only be set in the operators. No operator category parameters are available for WS security.

After they are defined, the SOAP request header <wsse:Security> tag includes the following parameters (or portions of the parameters). The Web service then:

- Reviews the parameters for authentication.
- Verifies that the SOAP request was not modified while in transit between the client and the server.

Common WS Security Parameters

Actor

Sets the actor attribute of the SOAP request <wsse:Security> header. If you set any of the following attributes, the product sets the actor attribute:

■ Add Encryption

■ Add Signature

■ Add Timestamp

■ Add Username Token

Note: Leave this parameter blank if no actor is specified or if you use SOAP 1.2.

Must Understand

If you set any of the following attributes, this parameter sets the mustUnderstand attribute of the <wsse:Security> header to "true":

■ Add Encryption

■ Add Signature

■ Add Timestamp

■ Add Username Token

Note: The mustUnderstand attribute specifies whether the <wsse:Security> header entry is mandatory or optional for the recipient to process.

Timestamp

Add Timestamp

Adds a timestamp to the <wsse:Security> header and enables all of the Timestamp Parameters.

Timestamp Parameters

Includes the following values:

Time to Live (sec)

Defines the time difference between when the SOAP request was created and when it expires (in seconds). If you leave this field blank, it defaults to 0 and the product does not set the Expires time. If the timestamp expires, the web service rejects the SOAP request.

Set Timestamp Precision to Milliseconds

If selected, the product sets the timestamp precision to milliseconds.

Username Token

Add Username Token

Adds a Username Token to the <wsse:Security> header and enables all of the Username Token parameters.

Username Token Parameters

Includes the following values:

User Name

Defines the value of the Username token.

Password

Defines the password that is associated with the Username Token.

Password Type

Specifies the type of password.

Values:

■   **0:** The product delivers the password in clear text.

■   **1:** The product delivers the password in digest form (the password is encrypted and not delivered in clear text).

■   **2:** The product delivers no password.

Add Nonce?

If selected, the product adds a nonce element (such as a hash value) to the Username Token. The web service may not require this element.

Add Created?

If selected, the product adds a Created element that indicates when the Username Token was created. The web service may not require this element.

Note: If you set the Password Type to Digest, the product automatically adds the Nonce and the Created elements to the Username Token.

Keystore Parameters

When you sign or encrypt a SOAP request, the product requires many keys to handle various responsibilities. The product uses keys in WS Security to sign or encrypt the SOAP request and to validate the SOAP response signature or decryption (if applicable).

CA Process Automation uses a *keystore* (a repository of security certificates) to maintain the keys that are used in WS Security. Keystores provide organization and consolidation for keys, and prevent other users from accessing the unique private keys. The product requires that you create a keystore (see page 472) or use an existing one.

WS Security includes the following keystore options:

■ Sign the SOAP request (or parts of it). The keystore contains the following items:

   – The private key to use to sign the request.

   – The associated public key with which the receiver validates the signature. The product adds a reference to this public key to the signed SOAP request.

■ Encrypt the SOAP request (or parts of it). The keystore contains the public key with which to encrypt the symmetric key. The product uses the symmetric key to encrypt the request.

■ Validate the SOAP response signature. The keystore contains the public key with which to validate the signature (if applicable).

■ Decrypt the SOAP response. The keystore contains the private key with which to decrypt the response.

The following parameters define the keystore with which to sign or encrypt a SOAP request:

Signature/Encryption Keystore Path

Defines the path to the keystore.

Signature/Encryption Keystore Password

Defines the clear text password with which to access the keystore.

Note: This parameter is not the password to access a private key in the keystore, but rather the password to access the keystore itself.

Signature/Encryption Keystore Type

Specifies the type of keystore.

Values:

■ JKS

■ PKCS12 (which typically has a .p12 extension)

■ Blank (the product uses the default value)

Default: JKS

Create a Keystore

You can use a third-party tool to create and build your keystore, or import new certificates/private keys to an existing keystore. One keystore management tool is Keytool, which comes with the Java JRE or JDK. Some keytool commands can be found here:

- http://download.oracle.com/javase/1.5.0/docs/tooldocs/solaris/keytool.html

- http://download.oracle.com/javase/1.5.0/docs/tooldocs/windows/keytool.html

You can also execute:

`keytool —help`

Signature

Add Signature

If selected, the product signs the SOAP request and then it adds a signature to the <wsse:Security> header. A private key in the keystore signs the SOAP request content. The product also enables all of the Signature Parameters fields.

Signature Parameters

The following parameters define the signature:

Private Key Alias

Defines the key alias in the keystore that the product uses for signing.

Private Key Password

Defines the key password in the keystore.

Canonicalization Algorithm

Defines the canonicalization method with which to serialize the data (the SOAP request body or the parts to be signed) before applying the signature. Leave this field blank to use the implementation default exclusive XML canonicalization algorithm xml-exc-c14n#.

Signature Algorithm

Defines the algorithm with which to define the signature. Leave this field blank if you want the product to try to detect and use a signature algorithm that matches the data in the key.

Public Key Identifier Type

Specifies the key identifier that sets up the certificate (public key) identification elements in the signature. The receiver uses this value to identify the signature certificate (public key) that validates the SOAP request signature.

Values:

■ **1 (Binary Security Token):** The product adds <wsse:SecurityTokenReference> to the Signature element. The Signature element uses a URI fragment in a <wsse:Reference> element to reference the signature certificate (public key). The URI fragment references the signature public key. The product includes the signature public key as binary data in the <wsse:Security> header <wsse:BinarySecurityToken> element.

■ **2 (Issuer Name and Serial Number):** The product adds <wsse:SecurityTokenReference> to the Signature element. The Signature element uses a <ds:X509Data><ds:X509:IssuerSerial> element to reference the signature certificate (public key). This element uniquely identifies a certificate by its X.509 issuer name and serial number.

■ **3 (X509 Certificate Identifier):** The product adds <wsse:SecurityTokenReference> to the Signature element. The Signature element uses a <wsse:KeyIdentifier ValueType="oasis-200401-wss-x509-token-profile-1.0#X509v3"> element to reference the signature certificate (public key).

■ **4 (Subject Key Identifier):** The product adds <wsse:SecurityTokenReference> to the Signature element. The Signature element uses a <wsse:KeyIdentifier ValueType="#oasis-200401-wss-x509-token-profile-1.0#X509SubjectKeyIdentifier"> element to reference the signature certificate (public key).

Default: 0. The operator uses the default key identifier (the Issuer Name and Serial Number) from the implementation.

Parts to Sign

Specifies which parts of the SOAP request to sign. Click Add Parameter to enter either a security ID or a Name/Namespace combination of the element to sign.

Values:

- **WSU ID:** Defines the wsu:id attribute of the element to sign. You can add wsu:id as an attribute of an element in the SOAP request and you can specify your own value. For example:

  ```
  <token wsu:id="123"> </token>
  ```

  The following statement shows the definition of the WSU namespace:

  ```
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200
  401-wss-wssecurity-utility-1.0.xsd"
  ```

- **Name:** Defines the name of the element to sign.

- **Namespace:** Defines the namespace URI (not the local name of the namespace) of the element to sign. For example:

  ```
  "http://www.ca.com/itpam"
  ```

Note: Leave this parameter blank to sign the body of the SOAP request. If you specify the WSU ID, the product ignores the Name and Namespace values.

Encryption

Add Encryption

Encrypts the SOAP request and adds a new encrypted symmetric key to the SOAP request <wsse:Security> header. CA Process Automation uses a symmetric key to encrypt the content of the SOAP request. The certificate (public key), provided in the keystore, encrypts the symmetric key itself and includes it in the <wsse:Security> header. If this field is selected, then all the fields in the Encryption Parameters are enabled.

Encryption Parameters

The following parameters define the encryption:

Public Key Alias

Defines the certificate (public key) alias with which to encrypt the symmetric key in the keystore.

Canonicalization Algorithm

Defines the canonicalization method with which to serialize the data before applying the encryption. Leave this field blank to use a standard serialization.

Symmetric Encryption Algorithm

Specifies the type of symmetric algorithm with which to encrypt the data.

Values:

- **Tripledes-cbc:** Use triple DES. This method uses a key that is 8 bytes - 24 bits long.

- **aes128-cbc:** Use AES with a 128-bit key.

- **aes192-cbc:** Use AES with a 192-bit key.

- **aes256-cbc:** Use AES with 256-bit key.

Default: AES128

Note: If you set this parameter to aes192-cbc or aes256-cbc, the following error can occur:

```
Illegal key size or default parameters.
```

If this error occurs, download the following Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files from the Oracle web site:

- US_export_policy.jar

- local_policy.jar

Overwrite the existing jars of the same name at C:*path_to_JRE_used_by_PAM*\lib\security with the new ones.

Encrypt the symmetric key?

Encrypts the symmetric key with which the data was encrypted. The product then includes the key in the <wsse:security><xenc:EncryptedKey> header.

Symmetric Key Encryption Algorithm

Defines the algorithm with which to encrypt the symmetric key. This parameter is only applicable if the key is to be encrypted.

Default: RSA15

Public Key Identifier Type

Specifies the key identifier that sets up the certificate (public key) identification elements in the <xenc:EncryptedKey> element. The receiver uses the private key that corresponds to this certificate (public key) to decrypt the symmetric key. The product then uses the symmetric key to decrypt the SOAP request.

Values:

- **1 (Binary Security Token):** The product adds <wsse:SecurityTokenReference> to the <xenc:EncryptedKey> element. The <xenc:EncryptedKey> element uses a URI fragment in a <wsse:Reference> element to reference the certificate (public key). The URI fragment references the public key. The product includes the public key as binary data in the <wsse:Security> header <wsse:BinarySecurityToken> element.

- **2 (Issuer Name and Serial Number):** The product adds <wsse:SecurityTokenReference> to the <xenc:EncryptedKey> element. The <xenc:EncryptedKey> element uses a <ds:X509Data><ds:X509:IssuerSerial> element to reference the certificate (public key). This element uniquely identifies a certificate by its X.509 issuer name and serial number.

- **3 (X509 Certificate Identifier):** The product adds <wsse:SecurityTokenReference> to the <xenc:EncryptedKey> element. The <xenc:EncryptedKey> element uses a <wsse:KeyIdentifier ValueType="oasis-200401-wss-x509-token-profile-1.0#X509v3"> element to reference the certificate (public key).

- **4 (Subject Key Identifier):** The product adds <wsse:SecurityTokenReference> to the <xenc:EncryptedKey> element. The <xenc:EncryptedKey> element uses a <wsse:KeyIdentifier ValueType="#oasis-200401-wss-x509-token-profile-1.0#X509SubjectKeyIdentifier"> element to reference the certificate (public key).

- **8 (Thumbprint SHA1 Identifier):** The product adds <wsse:SecurityTokenReference> to the <xenc:EncryptedKey> element. The <xenc:EncryptedKey> element uses a <wsse:KeyIdentifier ValueType="#oasis-wss-soap-message-security-1.1#ThumbprintSHA1"> element to references the certificate (public key).

Default: 0. The operator uses the default key identifier (the Issuer Name and Serial Number) from the implementation.

Parts to Encrypt

Specifies which the parts of the SOAP request to encrypt. Click Add Parameter to enter either a security ID (WSU ID) or a Name/Namespace combination of the element to encrypt.

Values:

- **WSU ID:** Defines the wsu:id attribute of the element to encrypt. You can add wsu:id as an attribute of an element in the SOAP request and you can specify your own value. For example:

  ```
  <token wsu:id="123"> </token>
  ```

  The following statement shows the definition of the WSU namespace:

  ```
  xmlns:wsu=http://docs.oasis-open.org/wss/2004/01/oasis-2004
  01-wss-wssecurity-utility-1.0.xsd
  ```

- **Name:** Defines the name of the element to encrypt.

- **Namespace:** Defines the namespace URI (not the local name of the namespace) of the element to encrypt. For example:

  ```
  http://www.ca.com/pam
  ```

- **Encode:** Select Content to encrypt the content of the element, or Element to encrypt the entire element.

  Note: Leave this field blank to encrypt the body content of the SOAP request. If you specify WSU ID, the product ignores the Name and Namespace values.

Signature First?

Specifies whether to apply the signature before encrypting the data. This parameter is useful if the product encrypts and signs the same data in the SOAP request.

Decrypt and Validate Signature of SOAP Response

Specifies whether to decrypt the SOAP response content and (if applicable) validates the signature. When you select this option, the product enables the Decryption Private Key Password parameter.

Decryption Private Key Password

Defines the password with which to access the decryption private key in the keystore. Use this password to access the private key with which to decrypt an encrypted SOAP response.

Encryption/Signature Process for the SOAP Request

The Invoke SOAP Method operator and the Invoke SOAP Method Async operator sign or encrypt all or part of the SOAP request body.

Encryption

1. CA Process Automation uses the Canonicalization Algorithm to serialize the data to encrypt. This data includes either the entire request body or the parts that you specify in Parts to Encrypt.

2. CA Process Automation uses a symmetric key (depending on the Symmetric Encryption Algorithm) to encrypt the data. This data includes either the entire request body or the parts that you specify in Parts to Encrypt.

3. CA Process Automation uses the Public Key Alias to retrieve the public key from the keystore. The public key then uses the algorithm that you specify in the Symmetric Key Encryption Algorithm to encrypt the symmetric key.

4. CA Process Automation adds the encrypted symmetric key to the SOAP request in the <xenc:EncryptedKey> element.

5. Depending on the reference type you specified in the Public Key Identifier Type, CA Process Automation adds to the SOAP request a reference to the public key used to encrypt the symmetric key.

6. The receiver decrypts the symmetric key and then uses the decrypted symmetric key to decrypt the SOAP request.

Signature

1. CA Process Automation uses the Private Key Alias and the Private Key Password to retrieve the private key from the keystore.

2. CA Process Automation uses the Canonicalization Algorithm to serialize the data to sign. This data includes either the entire request body or the parts that you specify in Parts to Sign.

3. CA Process Automation uses the private key from the keystore to sign the SOAP request content using the Signature Algorithm that the user specifies. The product signs either the entire request body or the parts that you specify in Parts to Sign.

4. CA Process Automation adds a reference to the certificate/public key that is associated with the private key to the SOAP request. The user specifies the type of reference in the Public Key Identifier Type.

5. The receiver uses the public key to validate the signature in the SOAP request.

Troubleshooting WS Security

The Invoke SOAP Method and Invoke SOAP Method Async operators can fail when applying WS Security in the following cases.

Unable to Build a Crypto to Manage the Keystore

An error displays if the operator is unable to build a crypto to manage the keystore:

```
SOAP invocation failed: Unable to build a Crypto for the keystore containing the
Signature/Encryption keys…
```

This error can be due to:

- Bad keystore password

- Bad keystore type

- The keystore path points to a file that is not a keystore.

Monitor the CA Process Automation logs, in the error stack, to gain an indication of the nature of the problem. For instance:

- Bad keystore password: Keystore was tampered with, or password was incorrect.

- Bad keystore type: java.security.KeyStoreException: *x* not found (where *x* is the type provided)

- Keystore path points to a file that is not a keystore: Invalid keystore format.

Signature Errors

When signing the SOAP request, problems can occur when:

- The keystore path does not exist on the computer that contains the touchpoint.

- The keystore path is missing.

- The private key alias is missing.

- You attempt to sign a non-existent part of the SOAP request. The reason field contains a message:

    ```
    Element to encrypt/sign not found…
    ```

- The private key alias does not exist in the keystore. The reason field contains a message:

    ```
    No certificates for user x were found for signature…
    ```

    Where *x* is the private key alias provided.

- The private key password is required but not provided, or the provided password is wrong. The reason contains a message:

    ```
    Cannot recover key…
    ```

- You provide a bad canonicalization algorithm. The reason field contains a message:

    ```
    Unknown canonicalizer. No handler installed for URI x…
    ```

    Where *x* is the name of the canonicalization algorithm provided.

- You provide a bad signature algorithm. The reason field contains a message:

    ```
    The requested algorithm x does not exist...
    ```

    Where *x* is the name of the signature algorithm provided.

- You provide a bad public key identifier type. The reason field contains a message:

  `Unsupported key identification…`

Encryption Errors

When encrypting the SOAP request, problems can occur when:

- The keystore path does not exist on the computer that contains the touchpoint.

- The keystore path is missing.

- You attempt to encrypt a non-existent part of the SOAP request. The reason field contains a message:

  `Element to encrypt/sign not found…`

- You attempt to use symmetric encryption algorithm aes192-cbc or aes256-cbc without upgrading to the unlimited strength jurisdiction policy jars. The reason field contains a message:

  `Illegal key size or default parameters.`

- The public key alias does not exist in the keystore. The reason field contains a message:

  `No certificates for user` *x* `were found for encryption…`

  Where *x* is the public key alias provided.

- You provide a bad encryption algorithm. The reason field contains a message:

  `SOAP invocation failed: Unable to encrypt the SOAP message.null.`

  The WSS4J library throws a null pointer error in this case.

- You provide a bad symmetric key encryption algorithm. The reason field contains a message:

  `unsupported key transport encryption algorithm:` *x*

  Where *x* is the symmetric key encryption algorithm provided.

- You provide a bad public key identifier type. The reason field contains a message:

  `Unsupported key identification…`

Note: If you are encrypting a part of the SOAP request, then signing it, be sure to encrypt it as content. This action ensures its wsu:id (or name and namespace) remains in the SOAP request *after* encryption and *before* signing. Otherwise, that part is not found when attempting to sign it.

Decryption Errors

A bad password specified in the Decryption Private Key Password field fails the operator. The reason field contains a message:

`Unable to apply WSS security on incoming message (SOAP Response).`

The signature or decryption is invalid. The nested exception is: java.security.

`UnrecoverableKeyException: Get Key failed:`

The given final block is not properly padded.

Output Parameters

The Invoke SOAP Method operator includes the following output parameters.

SOAP Call Data

serviceURL

Returns the service URL the SOAP call uses.

methodName

Returns the method name to call.

userName

Returns the user name for basic HTTP authentication.

password

Returns the password for basic HTTP authentication.

httpAuth

Displays true or false, depending on your Use HTTP Basic Authentication? selection.

soapVersion

Returns the SOAP version that the product uses to make SOAP calls, either SOAP 1.1 or SOAP 1.2.

inputSource

Returns one of the following values for the SOAP call input source:

■ InlineText

■ Preformatted SOAP File

■ Expression

inlineText

Returns the inline text data that is used for the SOAP call. This variable only populates when you select Inline Text as the input source.

Example:

```
<checkServerStatus xmlns="http://www.ca.com/itpam">
CA AuthMinder
<token>token__</token>
<user>user__</user>
<password>password__</password>
</auth>
</checkServerStatus>
```

FileName

Returns the inline text data that is used for the SOAP call. This variable is only populated when you select Preformatted SOAP File as the input source.

Expression Value

Returns the inline text data that is used for the SOAP call. This variable is only populated when you select Expression as the input source.

SavedCallFileName

Returns the file name from the Saved Call File field. This file contains the actual SOAP envelope that the product uses to make a SOAP call.

Example:

```
<?xml version="1.0" encoding="UTF-8"?><SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><SOAP-ENV
:Header/><SOAP-ENV:Body><checkServerStatus
xmlns="http://www.ca.com/itpam">
CA
AuthMinder<token>token__</token><user>user__</user><password>pa
ssword__</password></auth></checkServerStatus></SOAP-ENV:Body><
/SOAP-ENV:Envelope>
```

Dynamic Parameters

Paramsyle

Returns the parameter style that you selected under the Dynamic Parameter field: Xpath Assignments or MacroExpansion.

DynamicParamsInfo

An array of value maps; displays the number of value maps it contains.

DynamicParamsdata

Returns query, value, and type.

Call Results Parameters

responseFileName

Returns the file name that you provided in the Response File field. This file contains the Response received from a SOAP call.

xPathQuery

Returns the xpath query that is defined for extracting the data from the call results.

datasetVar

Returns the variable that is created to hold the extracted call results.

Type

Returns the variable that holds the data type that the user defined to hold the call results.

isExtractToDataSet

Returns true when you select Extract SOAP Response Body to Dataset Variable, or false otherwise.

isExtractHeadersToDataSet

Returns true when you select Extract SOAP Header to Dataset Variable, or false otherwise.

isExtractHeadersToIndividualDataSet

Returns true when you select Extract First-Level SOAP Header Elements into Individual Dataset Variable, or false otherwise.

isExtractToIndividualDataSet

Returns true when you select Extract First-Level SOAP Header Elements into Individual Dataset Variables, or false otherwise.

isStripXMLNamespaces

Returns true when you select Strip XML Name Spaces from Response, or false otherwise.

Operation Results

The operation results return the output of the SOAP Call Results depending on the Call Results parameters you selected.

SOAPResponseBody

Stores the complete SOAP response body.

SOAPResponseHeaderData

Stores the response header.

SoapResponseHeader

Returns the stripped data of the SOAP header response.

SOAPResponseData

Returns the stripped data of the SOAP call results.

MIME Attachments

isResolvedByExpression

Returns true when you select the Is An Expression? option for the MIME Attachment, or false otherwise. Selecting the Is An Expression? option indicates that the MIME attachment is given as an expression and that expression refers to the MIME attachment.

Expressionfield

Returns the expression that refers to the MIME attachment.

attachmentFields

Returns a ValueMap array that holds the number of elements it contains.

Content Type

Returns the content type of the MIME attachment.

ContentID

Returns the ContentID with which the MIME attachment is uniquely identified.

FileURL

Returns the path of the MIME attachment.

Invoke SOAP Method Async Operator

The Invoke SOAP Method Async operator supports asynchronous SOAP-based Web service methods. The operator in this case sends a message asynchronously to a receiver expecting a response at a later time. The sender tags the request with an identifier allowing the response to be correlated with the originating request. This operator is used and configured very much like the Invoke SOAP Method operator.
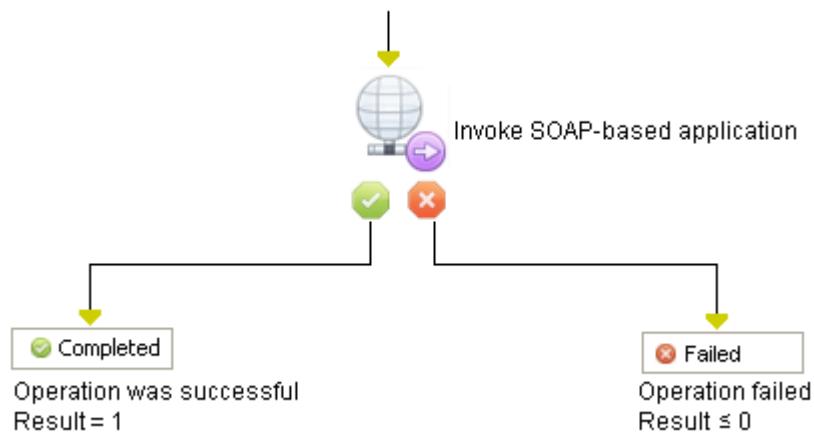
Input Parameters

The Invoke SOAP Method Async operator provides the following input parameters:

WSDL Explorer

WSDL Wizard

A wizard that lets you select SOAP methods. Click to open the WSDL Explorer window, where you can enter:

WSDL URL

- Enter a WSDL URL, then click Load. Examples:

  `http://`*`server`*`:8080/itpam/soap?wsdl`

  `https://`*`server`*`:8443/itpam/soap?wsdl`

- Select a recently entered WSDL URL from the drop-down list, then click Load.

  Note: The WSDL URL drop-down list contains the last ten applied entries.

The wizard populates the remaining fields. Revise as needed by selecting other entries from the drop down lists and then click Apply.

WSDL services

WSDL ports

WSDL operations

- Use the default entry that was loaded

- Select an entry from the drop-down list.

  For details on the listed Web Services methods, see the *Web Services API Reference*.

Response area, for example:

```
<tns:AsyncSoapResponse xmlns:tns="http://www.ca.com/itpam">
<tns:MessageID>MessageID__</tns:MessageID>
<!--xsd:Any Type Here-->
</tns:AsyncSoapResponse>
```

SOAP Call Data Properties

Service URL

Specifies the URL for the SOAP service. The URL is typically accessed over HTTP or HTTPS. The URL is typically an entry point for one or more methods.

Method Name

Defines the method or function to start. The product passes the method to the SOAP service as a MIME SOAPAction header.

Authentication Type

Specifies one of the following authentication types with which to call the SOAP server:

No Authentication

Do not authenticate.

Basic SOAP

Use the authorized user name and password for basic SOAP authentication.

HTTP Authentication

Use the authorized user name and password for basic HTTP authentication if this feature is enabled. WS Security applies if the user provides it on the WS Security input page (see page 470).

NTLM Authentication

Use the authorized user name, password, and Domain name to connect to the SOAP server with NTLM authentication.

Important! The user name, password, and Domain name that are supplied are for the SOAP web service server.

SOAP Version

Specifies the SOAP server version the product uses to make SOAP calls, either SOAP 1.1 or SOAP 1.2. This option provides a hint to the underlying logic, which uses the appropriate SOAP MessageFactory to make the SOAP request.

Input Source

Specifies the SOAP service input source. Each of the following methods includes a complete, properly formatted XML message that can include a SOAP envelope:

Inline Text

The SOAP Client Call operator defines the input request. If you select this option, use the Inline text option to specify the formatted SOAP input message.

Preformatted SOAP File

A preformatted SOAP file defines the input request. If you select this option, specify the path to the file in the File Name field. Verify that the file contains a message in a valid XML document.

Expression

A CA Process Automation expression defines the SOAP request. If you select this option, specify the CA Process Automation expression in the Expression field.

The input request can contain macros and XPath assignments that dynamically modify the SOAP request at run time. These macros and XPath assignments enable the product to update the SOAP request if necessary with values obtained at run time.

Inline Text

Click (...) to open the Inline Text dialog if you set Input Source to Inline Text. Enter a formatted SOAP input message. To read a SOAP message from a text file on a local or network drive, click Import from File in the Inline Text dialog.

File Name

Defines the fully qualified file name relative to the touchpoint running the Web Services operators. The product uses this property when you set Input Source to Preformatted SOAP File.

Expression

Defines a CA Process Automation expression from which to extract the SOAP request.

Saved Call File

Defines the full path to the SOAP envelope that the product uses to make a SOAP call. Use this option to validate that the product sent the data to the Web Services operators. If the SOAP request is rejected, the saved file can assist you in debugging the request before you resend it to the Web Services operators.

Dynamic Parameters

The Dynamic Parameters specify which values to update in a SOAP request.

Parameter Style

Specifies the method with which the Parameters List updates values in a SOAP request at run time.

Macro Expansion

The Parameters list uses macros to update values in a SOAP request.

XPath Assignments

The Parameters list uses XPath expressions to update values in a SOAP request.

Parameters List

Specifies the parameters that specify unique data entries in the SOAP request. Click Add to add a parameter, Edit to edit the currently selected parameter, or Delete to delete the currently selected parameter. A best practice is to add the parameters in the order in which the WSDL specifies them for the SOAP call.

To open the Dynamic Parameters dialog, click Add or Edit, and then set the following parameter values:

Macro Name-X-Path

Defines the name of the macro or the XPath query used to extract data from the body of the SOAP response.

- If the parameter defines a macro name, then the value replaces the macro name.

- If the parameter defines an XPath query, then the value updates the node that the query returns.

Value

Defines an expression that returns a run-time value for the parameter.

Type

Specifies one of the following data types for the parameter:

- Integer Value

- String Value

- XML Fragment

Call Results Properties

The following Call Results parameters determine how to save the results of a SOAP call.

Response Save File

Specifies the fully qualified path for the file that restores the response to the SOAP request. The new response overwrites an existing response file.

Extract SOAP Response First-Level Elements to Individual Dataset Variables

If the SOAP response exceeds 12 KB, this parameter saves the first-level element in the body of the SOAP response to a separate dataset variable.

Extract SOAP Response Body to Dataset Variable

Saves the response body to a dataset variable.

Extract SOAP Header to Dataset Variable

Saves the response header to a dataset variable.

Extract SOAP Header First-Level Elements to Individual Dataset Variables

Saves the response header to individual dataset variables.

Strip XML Namespaces from Response

Strips the XML namespaces from the response so that you do not need to use the local-name() function.

Additional Extracted Data (from Entire Response)

Defines XPath expressions to extract data from the SOAP response body. For each expression specified, specify a data type and a dataset variable to which to store the extracted data. Click Add to add an expression, Edit to edit a selected existing expression, or Delete to delete a selected expression. The expression order has no significance to CA Process Automation.

To modify the following parameters, click Add or Edit:

XPath Expression

Defines the XPath expression.

Dataset Variable

Defines the name of an operator dataset variable in which to save values extracted based on the selected XPath expression.

Type

Specifies the type of element being extracted from the response:

- Integer
- String
- Integer Array
- String Array
- XML Fragment
- XML Fragment Array

MIME Attachments

If the content that you want to send is already in a dataset variable, use the Expression field.

Is an Expression?

If selected, an expression must resolve the attachment.

Expression

Defines the MIME (Multipurpose Internet Mail Extensions) expression that extracts the attachment from the body of the SOAP response. For each expression defined, specify a dataset variable to which to store the extracted data and a data type. The ordering of the expressions has no significance to CA Process Automation. Click Add to add an attachment, Edit to modify a selected attachment, or Delete to delete a selected attachment.

Content Type

Defines the content type of the MIME attachment (for example, text).

Content ID

Defines the unique identifier for the MIME attachment.

File URL

Defines the URL of the MIME attachment.

WS Security

Web services (WS) security enables CA Process Automation to conduct secure SOAP message exchanges with a Web service that requires more security.

WS security features:

- Timestamps
- UsernameTokens
- Signatures
- Encryption

Note: The WS security parameters can only be set in the operators. No operator category parameters are available for WS security.

After they are defined, the SOAP request header <wsse:Security> tag includes the following parameters (or portions of the parameters). The Web service then:

- Reviews the parameters for authentication.
- Verifies that the SOAP request was not modified while in transit between the client and the server.

Common WS Security Parameters

Actor

Sets the actor attribute of the SOAP request <wsse:Security> header. If you set any of the following attributes, the product sets the actor attribute:

- Add Encryption

- Add Signature

- Add Timestamp

- Add Username Token

Note: Leave this parameter blank if no actor is specified or if you use SOAP 1.2.

Must Understand

If you set any of the following attributes, this parameter sets the mustUnderstand attribute of the <wsse:Security> header to "true":

- Add Encryption

- Add Signature

- Add Timestamp

- Add Username Token

Note: The mustUnderstand attribute specifies whether the <wsse:Security> header entry is mandatory or optional for the recipient to process.

Timestamp

Add Timestamp

Adds a timestamp to the <wsse:Security> header and enables all of the Timestamp Parameters.

Timestamp Parameters

Includes the following values:

Time to Live (sec)

Defines the time difference between when the SOAP request was created and when it expires (in seconds). If you leave this field blank, it defaults to 0 and the product does not set the Expires time. If the timestamp expires, the web service rejects the SOAP request.

Set Timestamp Precision to Milliseconds

If selected, the product sets the timestamp precision to milliseconds.

Username Token

Add Username Token

Adds a Username Token to the <wsse:Security> header and enables all of the Username Token parameters.

Username Token Parameters

Includes the following values:

User Name

Defines the value of the Username token.

Password

Defines the password that is associated with the Username Token.

Password Type

Specifies the type of password.

Values:

■   **0:** The product delivers the password in clear text.

■   **1:** The product delivers the password in digest form (the password is encrypted and not delivered in clear text).

■   **2:** The product delivers no password.

Add Nonce?

If selected, the product adds a nonce element (such as a hash value) to the Username Token. The web service may not require this element.

Add Created?

If selected, the product adds a Created element that indicates when the Username Token was created. The web service may not require this element.

Note: If you set the Password Type to Digest, the product automatically adds the Nonce and the Created elements to the Username Token.

Keystore Parameters

When you sign or encrypt a SOAP request, the product requires many keys to handle various responsibilities. The product uses keys in WS Security to sign or encrypt the SOAP request and to validate the SOAP response signature or decryption (if applicable).

CA Process Automation uses a *keystore* (a repository of security certificates) to maintain the keys that are used in WS Security. Keystores provide organization and consolidation for keys, and prevent other users from accessing the unique private keys. The product requires that you create a keystore (see page 472) or use an existing one.

WS Security includes the following keystore options:

■ Sign the SOAP request (or parts of it). The keystore contains the following items:

  – The private key to use to sign the request.

  – The associated public key with which the receiver validates the signature. The product adds a reference to this public key to the signed SOAP request.

■ Encrypt the SOAP request (or parts of it). The keystore contains the public key with which to encrypt the symmetric key. The product uses the symmetric key to encrypt the request.

■ Validate the SOAP response signature. The keystore contains the public key with which to validate the signature (if applicable).

■ Decrypt the SOAP response. The keystore contains the private key with which to decrypt the response.

The following parameters define the keystore with which to sign or encrypt a SOAP request:

Signature/Encryption Keystore Path

Defines the path to the keystore.

Signature/Encryption Keystore Password

Defines the clear text password with which to access the keystore.

Note: This parameter is not the password to access a private key in the keystore, but rather the password to access the keystore itself.

Signature/Encryption Keystore Type

Specifies the type of keystore.

Values:

■ JKS

■ PKCS12 (which typically has a .p12 extension)

■ Blank (the product uses the default value)

Default: JKS

Create a Keystore

You can use a third-party tool to create and build your keystore, or import new certificates/private keys to an existing keystore. One keystore management tool is Keytool, which comes with the Java JRE or JDK. Some keytool commands can be found here:

- http://download.oracle.com/javase/1.5.0/docs/tooldocs/solaris/keytool.html

- http://download.oracle.com/javase/1.5.0/docs/tooldocs/windows/keytool.html

You can also execute:

`keytool –help`

Signature

Add Signature

If selected, the product signs the SOAP request and then it adds a signature to the <wsse:Security> header. A private key in the keystore signs the SOAP request content. The product also enables all of the Signature Parameters fields.

Signature Parameters

The following parameters define the signature:

Private Key Alias

Defines the key alias in the keystore that the product uses for signing.

Private Key Password

Defines the key password in the keystore.

Canonicalization Algorithm

Defines the canonicalization method with which to serialize the data (the SOAP request body or the parts to be signed) before applying the signature. Leave this field blank to use the implementation default exclusive XML canonicalization algorithm xml-exc-c14n#.

Signature Algorithm

Defines the algorithm with which to define the signature. Leave this field blank if you want the product to try to detect and use a signature algorithm that matches the data in the key.

Public Key Identifier Type

Specifies the key identifier that sets up the certificate (public key) identification elements in the signature. The receiver uses this value to identify the signature certificate (public key) that validates the SOAP request signature.

Values:

- **1 (Binary Security Token):** The product adds <wsse:SecurityTokenReference> to the Signature element. The Signature element uses a URI fragment in a <wsse:Reference> element to reference the signature certificate (public key). The URI fragment references the signature public key. The product includes the signature public key as binary data in the <wsse:Security> header <wsse:BinarySecurityToken> element.

- **2 (Issuer Name and Serial Number):** The product adds <wsse:SecurityTokenReference> to the Signature element. The Signature element uses a <ds:X509Data><ds:X509:IssuerSerial> element to reference the signature certificate (public key). This element uniquely identifies a certificate by its X.509 issuer name and serial number.

- **3 (X509 Certificate Identifier):** The product adds <wsse:SecurityTokenReference> to the Signature element. The Signature element uses a <wsse:KeyIdentifier ValueType="oasis-200401-wss-x509-token-profile-1.0#X509v3"> element to reference the signature certificate (public key).

- **4 (Subject Key Identifier):** The product adds <wsse:SecurityTokenReference> to the Signature element. The Signature element uses a <wsse:KeyIdentifier ValueType="#oasis-200401-wss-x509-token-profile-1.0#X509SubjectKeyIdentifier"> element to reference the signature certificate (public key).

Default: 0. The operator uses the default key identifier (the Issuer Name and Serial Number) from the implementation.

Parts to Sign

Specifies which parts of the SOAP request to sign. Click Add Parameter to enter either a security ID or a Name/Namespace combination of the element to sign.

Values:

- **WSU ID:** Defines the wsu:id attribute of the element to sign. You can add wsu:id as an attribute of an element in the SOAP request and you can specify your own value. For example:

  `<token wsu:id="123"> </token>`

  The following statement shows the definition of the WSU namespace:

  `xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200`
  `401-wss-wssecurity-utility-1.0.xsd"`

- **Name:** Defines the name of the element to sign.

- **Namespace:** Defines the namespace URI (not the local name of the namespace) of the element to sign. For example:

  `"http://www.ca.com/itpam"`

Note: Leave this parameter blank to sign the body of the SOAP request. If you specify the WSU ID, the product ignores the Name and Namespace values.

Encryption

Add Encryption

Encrypts the SOAP request and adds a new encrypted symmetric key to the SOAP request <wsse:Security> header. CA Process Automation uses a symmetric key to encrypt the content of the SOAP request. The certificate (public key), provided in the keystore, encrypts the symmetric key itself and includes it in the <wsse:Security> header. If this field is selected, then all the fields in the Encryption Parameters are enabled.

Encryption Parameters

The following parameters define the encryption:

Public Key Alias

Defines the certificate (public key) alias with which to encrypt the symmetric key in the keystore.

Canonicalization Algorithm

Defines the canonicalization method with which to serialize the data before applying the encryption. Leave this field blank to use a standard serialization.

Symmetric Encryption Algorithm

Specifies the type of symmetric algorithm with which to encrypt the data.

Values:

- **Tripledes-cbc:** Use triple DES. This method uses a key that is 8 bytes - 24 bits long.

- **aes128-cbc:** Use AES with a 128-bit key.

- **aes192-cbc:** Use AES with a 192-bit key.

- **aes256-cbc:** Use AES with 256-bit key.

Default: AES128

Note: If you set this parameter to aes192-cbc or aes256-cbc, the following error can occur:

```
Illegal key size or default parameters.
```

If this error occurs, download the following Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files from the Oracle web site:

- US_export_policy.jar

- local_policy.jar

Overwrite the existing jars of the same name at C:*path_to_JRE_used_by_PAM*\lib\security with the new ones.

Encrypt the symmetric key?

Encrypts the symmetric key with which the data was encrypted. The product then includes the key in the <wsse:security><xenc:EncryptedKey> header.

Symmetric Key Encryption Algorithm

Defines the algorithm with which to encrypt the symmetric key. This parameter is only applicable if the key is to be encrypted.

Default: RSA15

Public Key Identifier Type

Specifies the key identifier that sets up the certificate (public key) identification elements in the <xenc:EncryptedKey> element. The receiver uses the private key that corresponds to this certificate (public key) to decrypt the symmetric key. The product then uses the symmetric key to decrypt the SOAP request.

Values:

- **1 (Binary Security Token):** The product adds <wsse:SecurityTokenReference> to the <xenc:EncryptedKey> element. The <xenc:EncryptedKey> element uses a URI fragment in a <wsse:Reference> element to reference the certificate (public key). The URI fragment references the public key. The product includes the public key as binary data in the <wsse:Security> header <wsse:BinarySecurityToken> element.

- **2 (Issuer Name and Serial Number):** The product adds <wsse:SecurityTokenReference> to the <xenc:EncryptedKey> element. The <xenc:EncryptedKey> element uses a <ds:X509Data><ds:X509:IssuerSerial> element to reference the certificate (public key). This element uniquely identifies a certificate by its X.509 issuer name and serial number.

- **3 (X509 Certificate Identifier):** The product adds <wsse:SecurityTokenReference> to the <xenc:EncryptedKey> element. The <xenc:EncryptedKey> element uses a <wsse:KeyIdentifier ValueType="oasis-200401-wss-x509-token-profile-1.0#X509v3"> element to reference the certificate (public key).

- **4 (Subject Key Identifier):** The product adds <wsse:SecurityTokenReference> to the <xenc:EncryptedKey> element. The <xenc:EncryptedKey> element uses a <wsse:KeyIdentifier ValueType="#oasis-200401-wss-x509-token-profile-1.0#X509SubjectKeyIdentifier"> element to reference the certificate (public key).

- **8 (Thumbprint SHA1 Identifier):** The product adds <wsse:SecurityTokenReference> to the <xenc:EncryptedKey> element. The <xenc:EncryptedKey> element uses a <wsse:KeyIdentifier ValueType="#oasis-wss-soap-message-security-1.1#ThumbprintSHA1"> element to references the certificate (public key).

Default: 0. The operator uses the default key identifier (the Issuer Name and Serial Number) from the implementation.

Parts to Encrypt

Specifies which the parts of the SOAP request to encrypt. Click Add Parameter to enter either a security ID (WSU ID) or a Name/Namespace combination of the element to encrypt.

Values:

- **WSU ID:** Defines the wsu:id attribute of the element to encrypt. You can add wsu:id as an attribute of an element in the SOAP request and you can specify your own value. For example:

  `<token wsu:id="123"> </token>`

  The following statement shows the definition of the WSU namespace:

  ```
  xmlns:wsu=http://docs.oasis-open.org/wss/2004/01/oasis-2004
  01-wss-wssecurity-utility-1.0.xsd
  ```

- **Name:** Defines the name of the element to encrypt.

- **Namespace:** Defines the namespace URI (not the local name of the namespace) of the element to encrypt. For example:

  `http://www.ca.com/pam`

- **Encode:** Select Content to encrypt the content of the element, or Element to encrypt the entire element.

Note: Leave this field blank to encrypt the body content of the SOAP request. If you specify WSU ID, the product ignores the Name and Namespace values.

Signature First?

Specifies whether to apply the signature before encrypting the data. This parameter is useful if the product encrypts and signs the same data in the SOAP request.

Decrypt and Validate Signature of SOAP Response

Specifies whether to decrypt the SOAP response content and (if applicable) validates the signature. When you select this option, the product enables the Decryption Private Key Password parameter.

Decryption Private Key Password

Defines the password with which to access the decryption private key in the keystore. Use this password to access the private key with which to decrypt an encrypted SOAP response.

Encryption/Signature Process for the SOAP Request

The Invoke SOAP Method operator and the Invoke SOAP Method Async operator sign or encrypt all or part of the SOAP request body.

Encryption

1. CA Process Automation uses the Canonicalization Algorithm to serialize the data to encrypt. This data includes either the entire request body or the parts that you specify in Parts to Encrypt.

2. CA Process Automation uses a symmetric key (depending on the Symmetric Encryption Algorithm) to encrypt the data. This data includes either the entire request body or the parts that you specify in Parts to Encrypt.

3. CA Process Automation uses the Public Key Alias to retrieve the public key from the keystore. The public key then uses the algorithm that you specify in the Symmetric Key Encryption Algorithm to encrypt the symmetric key.

4. CA Process Automation adds the encrypted symmetric key to the SOAP request in the <xenc:EncryptedKey> element.

5. Depending on the reference type you specified in the Public Key Identifier Type, CA Process Automation adds to the SOAP request a reference to the public key used to encrypt the symmetric key.

6. The receiver decrypts the symmetric key and then uses the decrypted symmetric key to decrypt the SOAP request.

Signature

1. CA Process Automation uses the Private Key Alias and the Private Key Password to retrieve the private key from the keystore.

2. CA Process Automation uses the Canonicalization Algorithm to serialize the data to sign. This data includes either the entire request body or the parts that you specify in Parts to Sign.

3. CA Process Automation uses the private key from the keystore to sign the SOAP request content using the Signature Algorithm that the user specifies. The product signs either the entire request body or the parts that you specify in Parts to Sign.

4. CA Process Automation adds a reference to the certificate/public key that is associated with the private key to the SOAP request. The user specifies the type of reference in the Public Key Identifier Type.

5. The receiver uses the public key to validate the signature in the SOAP request.

Troubleshooting WS Security

The Invoke SOAP Method and Invoke SOAP Method Async operators can fail when applying WS Security in the following cases.

Unable to Build a Crypto to Manage the Keystore

An error displays if the operator is unable to build a crypto to manage the keystore:

```
SOAP invocation failed: Unable to build a Crypto for the keystore containing the
Signature/Encryption keys…
```

This error can be due to:

- Bad keystore password

- Bad keystore type

- The keystore path points to a file that is not a keystore.

Monitor the CA Process Automation logs, in the error stack, to gain an indication of the nature of the problem. For instance:

- Bad keystore password: Keystore was tampered with, or password was incorrect.

- Bad keystore type: java.security.KeyStoreException: *x* not found (where *x* is the type provided)

- Keystore path points to a file that is not a keystore: Invalid keystore format.

Signature Errors

When signing the SOAP request, problems can occur when:

- The keystore path does not exist on the computer that contains the touchpoint.

- The keystore path is missing.

- The private key alias is missing.

- You attempt to sign a non-existent part of the SOAP request. The reason field contains a message:

  ```
  Element to encrypt/sign not found…
  ```

- The private key alias does not exist in the keystore. The reason field contains a message:

  ```
  No certificates for user x were found for signature…
  ```

  Where *x* is the private key alias provided.

- The private key password is required but not provided, or the provided password is wrong. The reason contains a message:

  ```
  Cannot recover key…
  ```

- You provide a bad canonicalization algorithm. The reason field contains a message:

  ```
  Unknown canonicalizer. No handler installed for URI x…
  ```

  Where *x* is the name of the canonicalization algorithm provided.

- You provide a bad signature algorithm. The reason field contains a message:

  ```
  The requested algorithm x does not exist...
  ```

  Where *x* is the name of the signature algorithm provided.

- You provide a bad public key identifier type. The reason field contains a message:

  `Unsupported key identification…`

Encryption Errors

When encrypting the SOAP request, problems can occur when:

- The keystore path does not exist on the computer that contains the touchpoint.
- The keystore path is missing.
- You attempt to encrypt a non-existent part of the SOAP request. The reason field contains a message:

  `Element to encrypt/sign not found…`

- You attempt to use symmetric encryption algorithm aes192-cbc or aes256-cbc without upgrading to the unlimited strength jurisdiction policy jars. The reason field contains a message:

  `Illegal key size or default parameters.`

- The public key alias does not exist in the keystore. The reason field contains a message:

  `No certificates for user x were found for encryption…`

  Where *x* is the public key alias provided.

- You provide a bad encryption algorithm. The reason field contains a message:

  `SOAP invocation failed: Unable to encrypt the SOAP message.null.`

  The WSS4J library throws a null pointer error in this case.

- You provide a bad symmetric key encryption algorithm. The reason field contains a message:

  `unsupported key transport encryption algorithm: x`

  Where *x* is the symmetric key encryption algorithm provided.

- You provide a bad public key identifier type. The reason field contains a message:

  `Unsupported key identification…`

Note: If you are encrypting a part of the SOAP request, then signing it, be sure to encrypt it as content. This action ensures its wsu:id (or name and namespace) remains in the SOAP request *after* encryption and *before* signing. Otherwise, that part is not found when attempting to sign it.

Decryption Errors

A bad password specified in the Decryption Private Key Password field fails the operator. The reason field contains a message:

`Unable to apply WSS security on incoming message (SOAP Response).`

The signature or decryption is invalid. The nested exception is: java.security.

`UnrecoverableKeyException: Get Key failed:`

The given final block is not properly padded.

Output Parameters

serviceURL

methodName

userName

password

httpAuth

soapVersion

inputSource

inlineText

SOAP Call Data

serviceURL

Returns the service URL the SOAP call uses.

methodName

Returns the method name to call.

userName

Returns the user name for basic HTTP authentication.

password

Returns the password for basic HTTP authentication.

httpAuth

Displays true or false, depending on your Use HTTP Basic Authentication? selection.

soapVersion

Returns the SOAP version that the product uses to make SOAP calls, either SOAP 1.1 or SOAP 1.2.

inputSource

Returns one of the following values for the SOAP call input source:

■   InlineText

■   Preformatted SOAP File

■   Expression

inlineText

Returns the inline text data that is used for the SOAP call. This variable only populates when you select Inline Text as the input source.

Example:

```
<checkServerStatus xmlns="http://www.ca.com/itpam">
CA AuthMinder
<token>token__</token>
<user>user__</user>
<password>password__</password>
</auth>
</checkServerStatus>
```

FileName

Returns the inline text data that is used for the SOAP call. This variable is only populated when you select Preformatted SOAP File as the input source.

Expression Value

Returns the inline text data that is used for the SOAP call. This variable is only populated when you select Expression as the input source.

SavedCallFileName

Returns the file name from the Saved Call File field. This file contains the actual SOAP envelope that the product uses to make a SOAP call.

Example:

```
<?xml version="1.0" encoding="UTF-8"?><SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><SOAP-ENV
:Header/><SOAP-ENV:Body><checkServerStatus
xmlns="http://www.ca.com/itpam">
CA
AuthMinder<token>token__</token><user>user__</user><password>pa
ssword__</password></auth></checkServerStatus></SOAP-ENV:Body><
/SOAP-ENV:Envelope>
```

Dynamic Parameters

Paramsyle

Returns the parameter style that you selected under the Dynamic Parameter field: Xpath Assignments or MacroExpansion.

DynamicParamsInfo

An array of value maps; displays the number of value maps it contains.

DynamicParamsdata

Returns query, value, and type.

Call Results Parameters

responseFileName

Returns the file name that you provided in the Response File field. This file contains the Response received from a SOAP call.

xPathQuery

Returns the xpath query that is defined for extracting the data from the call results.

datasetVar

Returns the variable that is created to hold the extracted call results.

Type

Returns the variable that holds the data type that the user defined to hold the call results.

isExtractToDataSet

Returns true when you select Extract SOAP Response Body to Dataset Variable, or false otherwise.

isExtractHeadersToDataSet

Returns true when you select Extract SOAP Header to Dataset Variable, or false otherwise.

isExtractHeadersToIndividualDataSet

Returns true when you select Extract First-Level SOAP Header Elements into Individual Dataset Variable, or false otherwise.

isExtractToIndividualDataSet

Returns true when you select Extract First-Level SOAP Header Elements into Individual Dataset Variables, or false otherwise.

isStripXMLNamespaces

Returns true when you select Strip XML Name Spaces from Response, or false otherwise.

Operation Results

Depending on the selection of check boxes in the Call Results parameters, the operation results holds the output of the SOAP Call Results.

Soap Response Body

Stores the complete SOAP response body.

Soap Response Header Data

 Stores the response header.

SoapResponseHeader

Contains the stripped data of the SOAP header response.

Soap Response Data

Contains the stripped data of the SOAP call results.

AsyncSoapIntermediateResponse

Contains the complete response along with the headers received from the SOAP call.

Example (using the CheckServerStatus method):

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<SOAP-ENV:Body>
<checkServerStatusResponse xmlns="http://www.ca.com/itpam">
<serverStatus>Server status ok.</serverStatus>
</checkServerStatusResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

AsyncSoapInterimResponseBody

Contains the body of the SOAP call response.

Example (using the CheckServerStatus method):

```
<checkServerStatusResponse xmlns="http://www.ca.com/itpam">
<serverStatus>Server status ok.</serverStatus>
</checkServerStatusResponse>
```

AsyncSoapInterimResponseHeader

Contains the header of the SOAP call response.

Example (using the CheckServerStatus method):

```
<SOAP-ENV:Header xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"/>
```

MIME Attachments

isResolvedByExpression

Returns true when you select the Is An Expression? option for the MIME Attachment, or false otherwise. Selecting the Is An Expression? option indicates that the MIME attachment is given as an expression and that expression refers to the MIME attachment.

Expressionfield

Returns the expression that refers to the MIME attachment.

attachmentFields

Returns a ValueMap array that holds the number of elements it contains.

Content Type

Returns the content type of the MIME attachment.

ContentID

Returns the ContentID with which the MIME attachment is uniquely identified.

FileURL

Returns the path of the MIME attachment.

Chapter 16: CA Process Automation System Functions

System functions can be used in:

■ Pre-execution and post-execution of any operator.

■ The source code of the Run JavaScript operator.

■ Any field that accepts an expression.

In each of these areas, system functions can be accessed by pressing Ctrl+Alt.

Return Types

System functions returns values of several types. In a few cases, the return type depends on the input parameters or on whether the function is returning a "null". In most cases, system functions return Java types rather than pure Javascript values. For instance, system functions that are described as returning a String are actually returning a Java.lang.String. Typically, you can ignore the difference between Java.lang.String and String. In some cases, this difference impacts behavior when executing and expression or script. In the case of typeof x, where x is a variable holding the returned value from a system function, the return type is "object", rather than "string".

For each system function return type, the following table describes the effect on the following variables:

■ A Javascript variable being assigned the return value

■ A CA Process Automation dataset variable being created

System Function Return Type	Javascript Variable Type after Assignment	Dataset Variable Type Created
boolean	boolean	Boolean
C2OString	object	String
C2OValue	object	ValueMap
C2OValueArray	object	ValueMap
C2OValueMap	object	ValueMap
Date	object	Date
int	number	Integer

System Function Return Type	Javascript Variable Type after Assignment	Dataset Variable Type Created
String	object	String
String []	object	String []
void	undefined/a	n/a

absPath

The absPath function returns the absolute path that was created with the help of base path and relative path. If the base path is not provided, the base path to the current process is used.

Syntax

```
sAbsPath = absPath (path2).
sAbsPath = absPath (path1, path2).
```

Arguments

path1 (String)

Specifies the base path.

path2 (String)

Specifies the relative path.

Return Value

sAbsPath (String)

The absolute path is returned as a String.

Examples

1. A process "Process" is in a folder "Folder". To get the full path of a "Process1" Object (present in the same folder) inside "Process" Object, use absPath('Process1'). In this case, the base path is automatically taken as the base path of "Process" object.

    ```
    sPath = absPath('Process1')
    ```

2. Here, the first path is the base path of the object, and then the second is the relative path with respect to the base path. The answer in this case is /folder1/process.

    ```
    sPath = absPath("/folder1/folder2","../process");
    ```

adjustDate

The adjustDate function adjusts a date by a specified number of days, weeks, months or years.

Syntax

```
newDt = adjustDate(dt, number, type)
```

Arguments

dt (java.util.Date)

Specifies the date that needs to be adjusted.

number (Int)

Specifies the value that needs to be adjusted.

type

Specifies the type of the value to be adjusted. Can be one of the following types:

- "y"(year)
- "d"(day)
- "w"(week)
- "m"(month)

Return Value

newDt (java.util.Date)

The adjusted date is returned as a java.util.Date.

Examples

Assume that today is 2/16/2012

1. The following example returns 2/18/2012, the date that is two days after today.

   ```
   adjustDate(today(),2,'d')
   ```

2. The following example returns 2/16/2013, the date that is one year after today.

   ```
   adjustDate(today(),1'y')
   ```

3. The following example returns 2/9/2012, the date that is one week before today.

   ```
   adjustDate(today(),-1,'w')
   ```

4. The following example returns 1/16/2012, the date that is one month before today.

   ```
   adjustDate(today(),-1,'m')
   ```

adjustResourceVals

This adjustResourceVals function modifies the values and state of a resource.

For more information about Resources, see the *Content Designer Guide*.

Syntax

```
bSuccess =
adjustResourceVals(resPath,resName,freeAmount,total,lock,unlock,re
set)
```

Arguments

resPath (String)

Specifies the path of the resources object.

resName (String)

Specifies the name of the resource in the resources object which must be adjusted.

freeAmount (Int)

Specifies the amount of free resources.

total (Int)

The value given in this argument adds up to the existing number of total resources.

lock (boolean)

Specifies True if the resource must be locked. When Lock is set to true, the values freeAmount and total for that resource cannot be set, and the system function returns false.

unlock (Boolean)

Specifies True if the resource must be unlocked.

Note: You cannot lock and unlock the same resource. Switch between lock and unlock when using this system function.

reset (Boolean)

Specifies True if the resources object must be reset.

Note: Specifying true takes precedence over all the other operators. All the parameters of the respective resource are reset to the default values (regardless of the given inputs in the system function).

Return Value

bSuccess (Boolean)

This function returns true if successful or false if it fails.

Example

```
bSuccess =
adjustResourceVals("\Resources","LockResource",20,20,false,true,fa
lse);
```

In this example, the variables are set as follows:

```
respath = "\Resources"
renName = "LockResource"
freeAmount = 20;
total = 20
lock = false
unlock = true
reset = false
```

applyXPath

The applyPath function uses an XPath query to extract XML data and returns an XML fragment that lists the nodes that result from the query.

Syntax

```
sXML = applyXPath(xmlData, xpathQuery, namespaceAware, returnArray)
```

Arguments

xmlData (String)

Defines the XML on which to run the XPath query.

xpathQuery (String)

Defines the XPath query to run.

namespaceAware (Boolean)

(Optional) Specifies whether to strip name spaces before applying the XPath query.

Values:

True: CA Process Automation does not strip name spaces before applying the XPath query.

False: CA Process Automation strips name spaces before applying the XPath query, making the query simple.

Default: True

returnArray (Boolean)

(Optional) Specifies whether to return an array of results or a single string.

Values:

True: The function returns an array of results.

False: The function returns a single string with multiple results concatenated in the string.

Default: False

Note: If the XPath query does not find a match, it returns a blank array.

Return Values and Examples

One of the following return values, based on the value of the returnArray argument.

- When returnArray is false:

sXML (String)

If you omit the returnArray argument or you set it to false, the query returns this value. The value that returns is an XML fragment that lists the nodes. For example:

```
Process.xpathResult = applyXPath(Process.bookXML,
"/bookstore/book[2]", true, false);
```

- When returnArray is true:

sXMLArray (C2OValueArray)

If you set the returnArray argument to true, the query returns this value. The value that returns is a C2OValueArray in which each element is an XML fragment for a node. For example:

```
Process.xpathArrayResult = applyXPath(Process.bookXML,
"/bookstore/book", true, true);
```

applyXPathToUrl

The applyXPathToUrl function uses an XPath query to extract XML data from a specified URL. The function returns an XML fragment that lists the nodes that result from the query.

Syntax

```
sXML = applyXPathToUrl(url, xpathQuery, namespaceAware, returnArray)
```

Arguments

url (String)

>   Defines the URL of the XML document on which to run the XPath.

xpathQuery (String)

>   Defines the XPath query to run.

namespaceAware (Boolean)

>   (Optional) Specifies whether to strip name spaces before applying the XPath query.

>   **Values:**

>   >   **True:** CA Process Automation does not strip name spaces before applying the XPath query.

>   >   **False:** CA Process Automation strips name spaces before applying the XPath query, making the query simple.

>   **Default:** True

returnArray (Boolean)

>   (Optional) Specifies whether to return an array of results or a single string.

>   **Values:**

>   >   **True:** The function returns an array of results.

>   >   **False:** The function returns a single string with multiple results concatenated in the string.

>   **Default:** False

>   **Note:** If the XPath query does not find a match, it returns a blank array.

Return Values and Examples

One of the following return values, based on the value of the returnArray argument.

- When returnArray is false:

sXML (String)

If you omit the returnArray argument or you set it to false, the query returns an XML fragment that lists the nodes. For example:

```
sXML =
applyXPathToUrl("http://localhost:8080/itpam_tutorials/book
.xml", "/bookstore/book[2]", true, false);
```

- When returnArray is true:

sXMLArray (C2OValueArray)

If you set the returnArray argument to true, the query returns a C2OValueArray in which each element is an XML fragment for a node. For example:

```
sXMLArray =
applyXPathToUrl("http://localhost:8080/itpam_tutorials/book
.xml", "/bookstore/book", true, true);
```

checkCalendarDate

The checkCalendarDate function checks whether the specified date is inside the specified condition. The condition includes the Include Calendar, Exclude Calendar, Delta, Open Days and Max shifts. See Calendar Properties for more information about these parameters.

Syntax

```
bIsAvail = checkCalendarDate(date, includeCalendar, excludeCalendar,
delta, openDays, maxShifts)
```

Arguments

date (java.util.Date)

Specifies the input date.

includeCalendar (String)

Specifies the path of the include calendar object.

excludeCalendar (String)

Specifies the path of the exclude calendar object.

delta (Int)

Specifies the delta.

openDays (Int)

> Specifies the openDays value.

maxShifts (Int)

> Specifies the maxshifts value.

Return Value

bIsAvail (Boolean)

> This function returns true if the specified date is open and false if it not.

Example

```
Process.sIncCal=absPath("IncCal");
Process.sExcCal=absPath("ExcCal");Process.bCaldate =
checkCalendarDate(today(),sIncCal,sExcCal,0,false,0);
```

convertJson

The convertJson function converts a valid JSON string into ValueMap. Use this method to convert a JSON response from a REST service to a ValueMap object. The Valuemap object can then be traversed and accessed using the standard expressions.

Syntax

```
vmResult = convertJson(jsonString)
```

Arguments

jsonString (String)

> Specifies the JSON string that needs to be parsed.

Return Value

ValueMap

> ValueMap representation of the data contained in the JSON string which can be traversed or referenced with expressions.

If you pass a null value, it returns null value without an exception. If you pass an invalid JSON string, the method returns a Null value and the exceptions are logged in the logs in the server.

Example

```
vmResult = convertJson(Process.jsonString)
```

Consider an example where you invoked a REST service. The following response that is received is stored in a variable name restResponse in a process dataset:

```
{
    "UserName": "pamadmin",
    "age": 25,
    "address": {
        "streetAddress": "CA Technologies, 115, IT Park Area",
        "city": "Hyderabad",
        "state": "AP",
        "postalCode": "500084"
    },
    "phoneNumber": [
        {
            "type": "Office",
            "number": "04066812345"
        },
        {
            "type": "Home",
            "number": "04066854321"
        }
    ]
}
```

To access the values of the UserName, streetAddress, and phoneNumber, write the following code:

```
// Code starts
// Parse the REST response using convertJson() method and store it in
a Process dataset variable named as "resultData"
Process.resultData=convertJson(Process.restResponse);
// Access "UserName" from the resultData varaible
Process.userName = Process.resultData.UserName;
// Street Address is inside address object hence it will be accessed
using the following syntax
Process.streetAddress = Process.resultData.address.streetAddress;
// Phone Number is an array and Office number is stored in the first
element, hence index [0] is used.
Process.officePhoneNumber =
Process.resultData.phoneNumber[0].number;
// Code ends
```

convertValueToXml

The convertValueToXml function returns an XML fragment based on an array of simple types or a ValueMap of simple types.

When using a ValueMap, the XML elements are created using field names as tags and field values as the contents. You can specify a string or null for the tag parameter when using ValueMaps. If you specify a string, the string is used to create a root element with the specified tag. The elements that were created from the ValueMap are contained within that root element. If you specify null, the elements are at the root level.

When using an array, specify a string for the tag parameter. That string is used to create the element tags with the array values as the element contents.

Syntax

```
sXML = convertValueToXml(arrayOrVmap, tag)
```

Arguments

arrayOrVmap (String)

Specifies an array or ValueMap.

tag (String)

Specifies the mandatory tag to use with an array or the optional tag to use with a ValueMap.

Return Value

sXML (String)

The return type is String.

Examples

1. In this example, Process.array contains values 1, 2, and 3.

   ```
   Process.xml = convertValueToXml(Process.array, "test")
   ```

   Process.xml contains the following XML fragment:

   ```
   <test>1<test><test>2<test><test>3<test>
   ```

2. In this example, Process.valuemap has two fields with the names "field1" and "field2" and values "value1" and "value2".

   ```
   Process.xml = convertValueToXml(Process.valuemap, null)
   ```

   Process.xml contains the following XML fragment:

   ```
   <field1>value1</field1><field2>value2</field2>
   ```

convertXml

The convertXML function converts an XML fragment to a ValueMap.

Syntax

```
vmResult = convertXml(xmlString)
```

Arguments

xmlString (String)

Specifies the XML that needs to be parsed.

Return Value

vmResult (ValueMap)

ValueMap representation of the data contained in the XML string.

Example

```
vmResult = convertXml(Process.xmlString)
```

convertXmlUrl

The convertXmlUrl function converts the XML document accessible through a URL into a ValueMap.

Syntax

```
vmResult = convertXmlUrl(url)
```

Arguments

url (String)

Specifies the URL of the XML document that needs to be parsed.

Return Value

vmResult (ValueMap)

ValueMap representation of the data that was retrieved from the XML document that the URL identifies.

Example

```
vmResult = convertXmlUrl("
http://localhost:8080/itpam_tutorials/book.xml");
```

createHyperLink

The createHyperLink function creates an HTML hyperlink element with the specified parameters and returns a string that will be formatted as ""+name+"".

Syntax

```
sLink = createHyperLink(url, name)
```

Arguments

url (String)

Specifies the HTTP URL whose hyperlink needs to be created.

name (String)

Specifies the name of the hyperlink.

Return Value

sLink (String)

A hyperlink with a URL and name as defined by the arguments passed into the function.

Example

```
sLink = createHyperLink("http://www.ca.com","CA Technologies");
```

createResourceObject

The createResourceObject function creates a resources object.

Syntax

```
bSuccess = createResourceObject(resourcePath)
```

Arguments

resourcePath (String)

Specifies the path of the resources object.

Return Value

bSuccess (Boolean)

This function returns true if successful or false if it fails.

Example

```
bSuccess = createResourceObject("SyncRes")
```

deleteAttachments

The deleteAttachments function deletes attachments from the CA Process Automation database given an array of unique IDs.

Syntax

```
bSuccess = deleteAttachments(AttachmentIDArray)
```

Arguments

AttachmentIDArray (Array)

Specifies an array of unique IDs. The IDs can be Strings, longs, or integers.

Return Value

bSuccess (Boolean)

- This function returns false only if its arguments are an empty array.

- This function throws an exception (that is, the operator fails) if it is unable to delete attachments or if invalid arguments are passed.

- This function returns true if it is able to process the delete attachment request successfully (including the case where the function is unable to delete a few or all of the attachments).

Examples

```
Process.rglAttachIDs = new Array(1,2,3,4,5);
Process.del = deleteAttachments(Process.rglAttachIDs);
```

deleteObject

The deleteObject function deletes and purges the library object specified by the "objectName" parameter. The input parameter can be a full or relative path. The relative path is relative to the process in which the script is executed.

Syntax

```
bSuccess = deleteObject(objectName)
```

Arguments

objectName (String)

> Specifies the full/relative path of the library object which must be deleted and purged.

Return Value

bSuccess (Boolean)

> This function returns true if successful or false if it fails.

Examples

1. This example deletes the process object found on the path /folder/Process and returns the value True.

   ```
   bSuccess = deleteObject('/folder/Process')
   ```

2. This function is being executed inside the process "Process_1", and Process_1 is in folder "Folder1" which is present inside the root folder. This function deletes the process with full path "/Folder1/Process" and returns the value True.

   ```
   bSuccess = deleteObject('Process')
   ```

deleteResource

The deleteResource function deletes a resource from a resources object.

Syntax

```
bSuccess = deleteResource(resourcePath, resourceName)
```

Arguments

resourcePath (String)

> Specifies the path of the resources object.

resourceName (String)

> Specifies the name of the resource in the resources object which needs to be deleted.

Return Value

bSuccess (Boolean)

> This function returns true if successful or false if it fails.

Example

```
bSuccess = deleteResource("/folder/ResObject", "fileLock");
```

deleteValueMapField

The deleteValueMapField function deletes a field from a ValueMap.

Syntax

```
bSuccess = deleteValueMapField(vMap, fieldName)
```

Arguments

vMap (ValueMap)

Specifies the ValueMap whose field needs to be deleted.

fieldName (String)

Specifies the name of the field that needs to be deleted.

Return Value

bSuccess (Boolean)

Returns true if the deletion was successful and false otherwise.

Example

```
bSuccess = deleteValueMapField(Process.vMap,"price");
```

existsCalendar

The existsCalendar function checks whether a calendar object exists in the given path. The path can be an absolute/relative path. The relative path is relative to the process in which the script is executed.

Syntax

```
bExists = existsCalendar(calendarPath);
```

Arguments

calendarPath (String)

Specifies the full/relative path of the calendar object whose existence needs to be checked.

Return Value

bExists (Boolean)

Returns true if the calendar object exists or false if it does not.

Example

```
bExists = existsCalendar("WorkCalendar");
```

existsCustomIcon

The existsCustomIcon function checks whether a custom icon object exists in the given path. The path can be an absolute/relative path. The relative path is relative to the process in which the script is executed.

Syntax

```
bExists = existsCustomIcon(customIconPath)
```

Arguments

customIconPath (String)

Specifies the full/relative path of the custom icon object whose existence needs to be checked.

Return Value

bExists (Boolean)

Returns true if the custom icon object exists or false if it does not.

Example

```
if (existsCustomIcon(customIconPath))
{
        Process.mseg_custom_icon= "CustomIcon exists" ;
}
else
{
        Process.mseg_custom_icon= "CustomIcon does not exist" ;
}
```

existsCustomOperator

The existsCustomOperator function checks whether a custom operator object exists in the given path. The path can be an absolute/relative path. The relative path is relative to the process in which the script is executed.

Syntax

```
bExists = existsCustomOperator(customOperatorPath)
```

Arguments

customOperatorPath (String)

> Specifies the full/relative path of the custom operator whose existence needs to be checked.

Return Value

bExists (Boolean)

> Returns true if the custom operator exists or false if it does not.

Example

```
if (existsCustomOperator(customOperatorpath))
{
        Process.mseg_custom_operator= "CustomOperator exists" ;
}
else
{
        Process.mseg_custom_operator= "CustomOperator does not exist"
;
}
```

existsDataset

The existsDataset function checks whether a dataset object exists in the given path. The path can be an absolute/relative path. The relative path is relative to the process in which the script is executed.

Syntax

```
bExists = existsDataset(datasetPath)
```

Arguments

datasetPath (String)

> Specifies the full/relative path of the dataset whose existence needs to be checked.

Return Value

bExists (Boolean)

> Returns true if the dataset object exists or false if it does not.

Example

```
if (existsDataset(datasetPath))
{
        Process.mseg_dataset= "Dataset Common exists" ;
}
else
{
        Process.mseg_dataset= "Dataset Common does not exist" ;
}
```

existsFolder

The existsFolder function checks whether a folder object exists in the given path. The path can be an absolute/relative path. The relative path is relative to the process in which the script is executed.

Syntax

```
bExists = existsFolder(folderPath)
```

Arguments

folderPath (String)

Specifies the full/relative path of the folder whose existence must be verified.

Return Value

bExists (Boolean)

Returns true if the folder object exists or false if it does not.

Example

```
if(existsFolder(folderpath))
{
Process.mesg_folder = "Folder test exists";
}
else
{
Process.mesg_folder = "Folder test does not exist";
}
```

existsInteractionRequestForm

The existsInteractionRequestForm function checks whether an interaction request form object exists in the given path. The path can be an absolute/relative path. The relative path is relative to the process in which the script is executed.

Syntax

```
bExists = existsInteractionRequestForm(irfPath)
```

Arguments

irfPath (String)

Specifies the full/relative path of interaction request form object whose existence needs to be checked.

Return Value

bExists (Boolean)

Returns true if the interaction request form object exists or false if it does not.

Example

```
if (existsInteractionRequestForm(irfPath))
{
        Process.mseg_irf= "Interaction Request Form exists" ;
}
else
{
        Process.mseg_irf= "Interaction Request Form does not exist" ;
}
```

existsProcess

The existsProcess function checks whether a process object exists in the given path. The path can be an absolute/relative path. The relative path is relative to the process in which the script is executed.

Syntax

```
bExists = existsProcess(processPath)
```

Arguments

processPath (String)

Specifies the full/relative path of the process object whose existence needs to be checked.

Return Value

bExists (Boolean)

> Returns true if the process object exists or false if it does not.

Example

```
if (existsProcess(processPath))
{
        Process.mseg_process= "\'Pass control to previous oper.\'
exists" ;
}
else
{
        Process.mseg_process= "\'Pass control to previous oper.\' does
not exist" ;
}
```

existsProcessWatch

The existsProcessWatch function checks whether a process watch object exists in the given path. The path can be an absolute/relative path. The relative path is relative to the process in which the script is executed.

Syntax

```
bExists = existsProcessWatch(processWatchPath)
```

Arguments

processWatchPath (String)

> Specifies the full/relative path of process watch object whose existence needs to be checked.

Return Value

bExists (Boolean)

> Returns true if the process watch object exists or false if it does not.

Example

```
if (existsProcessWatch(processWatchPath))
{
        Process.mseg_process_watch= "ProcessWatch exists" ;
}
else
{
        Process.mseg_process_watch= "ProcessWatch does not exist" ;
}
```

existsResource

The existsResource function checks whether a resources object exists in the given path. The path can be an absolute/relative path. The relative path is relative to the process in which the script is executed.

Syntax

```
bExists = existsResource(resourcePath)
```

Arguments

resourcePath (String)

Specifies the full/relative path of resources object whose existence needs to be checked.

Return Value

bExists (Boolean)

Returns true if the resource object exists or false if it does not.

Example

```
bExists = existsResource("/Resources/Locks");
```

existsSchedule

The existsSchedule function checks whether a schedule object exists in the given path. The path can be an absolute/relative path. The relative path is relative to the process in which the script is executed.

Syntax

bExists = existsSchedule(schedulepath)

Arguments

schedulepath (String)

> Specifies the full/relative path of the schedule object whose existence needs to be checked.

Return Value

bExists (Boolean)

> Returns true if the scheduleobject exists or false if it does not.

Example

```
if (existsSchedule("testSchedule"))
{Process.mseg_schedule= "testSchedule exists" ;}
else
{Process.mseg_schedule= "testSchedule does not exist" ;}
if (existsSchedule("testSchedule"))
{Process.mseg_schedule= "testSchedule exists" ;}
else
{Process.mseg_Schedule= "testSchedule does not exist" ;}
```

formatDate

The formatDate function returns a string based on a date and a format specifier. See this list for allowable date and time patterns.

Syntax

```
dateString = formatDate(dt, format)
```

Arguments

dt (java.util.Date)

> Specifies the date object to be formatted.

format (String)

> Specifies the format required (for example, MM/dd/yyyy).

Return Value

dateString (String)

> This function returns the date as a string using the format specifier.

Examples

```
Process.logMessage = "Date value is " + formatDate
(Process.CurrentFootprintDate, 'yyyy-mm-dd hh:mm:ss');
```

formatString

The formatString function returns a string after formatting the specified string with the mentioned arguments.

Syntax

```
resultString = formatString(format, args)
```

Arguments

format (String)

Specifies the format string.

args (Array)

Specifies the arguments for formatting.

Return Value

resultString (String)

This function returns a string, formatted according to the arguments provided in the function arguments.

Example

```
var myArray = new Array();
myArray[0] = "a";
myArray[1] = "b";
myArray[2] = "c";
myArray[3] = "d";
sString = formatString("%4$s %3$2s %2$2s %1$2s", myArray);
```

This results in sString = "d c b a".

getAllAttachments

The getAllAttachments function returns information for all attachments that are present in the CA Process Automation database.

Syntax

```
vmArrayAttachments = getAllAttachments()
```

Return Value

vmArrayAttachments (C2OvalueArray)

This function returns an array containing the following information for all attachments that are present in the CA Process Automation database:

contentType

Attachment content type.

contentID

Attachment contentID, if present.

fileURL

URL that can be used to view or download the attachment.

name

Name of the attachment.

attachmentID

Unique ID for this attachment. This ID can be passed to other system functions.

Example

```
Process.attachments = getAllAttachments();
```

getAttachmentContent

The getAttachmentContent function retrieves the content from an attachment and places it in a CA Process Automation dataset variable. This function has a 64KB limit to the size of the content it can retrieve.

Syntax

```
sAttachment = getAttachmentContent(lAttachmentID)
```

Arguments

lAttachmentID (long)

> The unique ID that identifies the attachment where the content resides that is being retrieved.

Return Value

sAttachment (C2OString)

> The content is returned as a C2OString.(com.optinuity.c2o.bean.C2OString).

Example

```
var i = Process.attachments[0].attachmentID;
Process.cont = getAttachmentContent(i);
```

getCountOfProcessStates

The getCountOfProcessStates function returns the count of processes in all possible states.

Syntax

```
Process.processStates = getCountOfProcessStates();
```

Return Value

processStates (ValueMap)

> Returns a ValueMap consisting of all the states and the number of processes in that state. If no processes are in the specified state, the count is 0.

Example

```
Process.processStates = getCountOfProcessStates();
```

getEEMArtifactToken

The getEEMArtifactToken function generates a CA EEM artifact token, typically for a single use.

Syntax

```
getEEMArtifactToken (certificateFilePath, {certificatePassword |
KeyFilePath})
```

Arguments

certificateFilePath

The relative path (File Path) to the certificate file. The certificate file is uploaded using Manage User Resources in the Configuration tab.

certificatePassword (String) | KeyFilePath (String)

One of the following arguments:

■ When FIPS support is enabled in CA EEM, set this argument to the relative path of the key file (KeyFilePath). The key file is uploaded to CA Process Automation using Manage User Resources in the Configuration tab.

■ When FIPS support is not enabled in CA EEM, set this argument to the certificate password. This password is used as a String for the certificate that is referenced in the certificateFilePath argument.

Return Value

CA EEM artifact token (String)

Examples

The following example shows FIPS support is enabled in CA EEM:

```
Process.artifactToken = getEEMArtifactToken
(".c2ouserresources/mycerts/pam.12", "mypassword")
Process.artifactToken = getEEMArtifactToken
(".c2ouserresources/mycerts/upload/pam.cer",
".c2ouserresources/mycerts/keys/pam.key")
```

getEEMArtifactTokenForUser

The getEEMArtifact function generates a CA EEM token for a single use.

Syntax

```
getEEMArtifactTokenForUser (username , password)
```

Arguments

username

The username for a CA EEM user.

password

The password for a CA EEM user.

Return Value

CA EEM artifact token (String)

Example

```
Process.artifactToken = getEEMArtifactTokenForUser ("pamadmin",
"pamadmin")
```

getEEMCredentialsToken

The getEEMCredentialsToken function generates a CA EEM credential token, typically for multiple uses. The certificateFilePath argument expects a relative path (File Path) of the certificate file. This file is uploaded using Manage User Resources within the Configuration tab.

In the case where FIPS is not enabled, the second argument is the certificate password. This password is used as a String for the certificate referred within the first argument (certificateFilePath).

In the case where FIPS support is enabled within CA EEM, the second argument is KeyFilePath. KeyFilePath is the relative path of the key file. This file is uploaded to CA Process Automation using Manage User Resources.

Syntax

```
getEEMCredentialsToken (certificateFilePath, {certificatePassword |
KeyFilePath})
```

Arguments

certificateFilePath

The relative path of the certificate file.

certificatePassword (String) | KeyFilePath (String)

One of the following arguments:

- For non-FIPS mode, this argument should be the certificate password.

- For FIPS-enabled mode, this argument should be the certificate KeyFilePath. The KeyFilePath is file is uploaded to CA Process Automation using Manage User Resources.

Return Value

CA EEM artifact token (String)

Example

Before you write the code, first verify if CA EEM is FIPS-enabled or not by using the isFIPSMode (see page 548) () function, so you can pass certificateFilePath, certificatePassword, or certificatepath and KeyFilePath.

For example:

```
If (isFIPSMode ().equals('true')){
Process.credentialToken =
getEEMCredentialsToken("/mycerts/upload/pam.cer",
"/mycerts/keys/pam.key")
} else {
Process.credentialToken = getEEMCredentialsToken("/mycerts/pam.12",
"mypassword")
}
```

getEEMCredentialsTokenForUser

The getEEMCredentialsTokenForUser function generates a CA EEM token for multiple uses.

Syntax

```
getEEMCredentialsTokenForUser (username , password)
```

Arguments

username

The username for a CA EEM user.

password

The password for a CA EEM user.

Return Value

CA EEM artifact token (String)

Example

```
Process.credentialToken =
getEEMCredentialsTokenForUser("pamadmin","pamadmin")
```

getEnvVar

The getEnvVar function returns the value of environment variable from the (OS) environment which must have been set before starting the JVM.

Syntax

```
sEnvValue = getEnvVar(vname)
```

Arguments

vname (String)

Specifies the name of the environment variable whose value is required.

Return Value

sEnvValue (String)

The value of the specified environment variable, in string form.

Example

```
Process.username=getEnvVar("username");
Process.path = getEnvVar("path");
```

getOrchestratorURL

The getOrchestratorURL function returns the name of the Orchestrator.

Syntax

```
getOrchestratorURL()
```

Return Value

URL of the Orchestrator (String)

Specifies the URL of the Orchestrator or the Load balancer (in the case of a cluster).

Example

```
Process.x = getOrchestratorURL();
```

Where *x* is the process name. Once the process runs, variable x inside the process dataset includes the Orchestrator name.

getPartialAttachmentContent

The getPartialAttachmentContent function is used to retrieve content from an attachment. The function has a 64 KB limit to the amount of content it can retrieve. Its purpose is to allow the retrieval of a subset of the content.

Syntax

sContent = getPartialAttachmentContent(*lAttachmentID*,*nStartIndex*, *nRetrieveLength*)

Arguments

lAttachmentID (long)

The unique ID that identifies the attachment where the content resides that is being retrieved.

nStartIndex (Int)

The location, in bytes

nRetrieveLength (Int)

Return Value

sContent (C2OString)

Contains the content, from the specified attachment, beginning with the nStartIndex byte and containing at most nRetrieveLength bytes. The return type is C2OString (com.optinuity.c2o.bean.C2OString).

Example

```
var  i = Process.attachments[0].attachmentID;
Process.part=getPartialAttachmentContent(i,i+1,100000);
```

getResourceAvail

The getResourceAvail function returns the value of the free field of a resource in a resources object.

Syntax

nAvail = getResourceAvail(*resourcePath*, *resourceName*)

Arguments

resourcePath (String)

Specifies the path of the resource object.

resourceName (String)

Specifies the name of the resource in the resources object whose free field needs to be returned.

Return Value

nAvail (Int)

This function returns the number of resources available from the specified resource in the resource object found on the specified path.

Example

```
nAvail = getResourceAvail("/Resources/Locks", "DiskLock");
```

getResourceName

The getResourceName function returns an array of strings containing the names of the resources inside a resources object.

Syntax

```
resourceNames = getResourceNames(resourcePath)
```

Arguments

resourcePath (String)

Specifies the path of the resource object.

Return Value

resourceNames (String Array)

Array of resource names contained in the specified resources object.

Example

```
resourceNames = getResourceNames("/Resources/Locks");
```

getResourceTotal

The getResourceTotal function returns the value of total amount for a particular resource in a resources automation object.

Syntax

```
nTotal = getResourceTotal(resourcePath, resourceName)
```

Arguments

resourcePath (String)

Specifies the path of the resource object.

resourceName (String)

Specifies the name of the resource in the resources object whose total amount needs to be returned.

Return Value

nTotal (Int)

Returns the total number for the specified resource. If the resource does not exist, -1 is returned.

Example

```
nTotal = getResourceTotal("/Resources/Locks", "DiskLock");
```

getTouchpoints

The getTouchpoint function returns a list of touchpoints referenced by a touchpoint name, or a touchpoint group name. This method returns an array of strings. An empty array means that the touchpoint group is empty, or the touchpoint with the given name does not exist.

Syntax

```
rgsTouchpoints = getTouchpoints(TouchpointOrGroupName)
```

Arguments

TouchpointOrGroupName (String)

Specifies the touchpoint or touchpoint group name.

Return Value

rgsTouchpoints (String Array)

An array of touchpoint string names.

Example

```
rgsTouchpoints = getTouchpoints("localhost");
```

getValueFromValueMapArray()

The getValueFromValueMapArray returns one field or column from an array that is based on the provided parameters.

Syntax

getValueFromValueMapArray(*groupName*, *arrName*, *fieldName*, *fieldValue*, *requiredFieldName*)

Arguments

groupName (String)

Defines the name of the Custom Group that you publish.

arrName (String)

Defines the name of a C2OValueMap array.

fieldName (String)

Defines the name of the field for which to retrieve the values of other columns or fields in an array.

fieldValue (String)

Defines the actual value of the field for which you retrieve the values of other columns or fields in an array.

requiredFieldName (String)

Defines the name of the field or column that you require. For example, username, password, or URL.

Return Value

The return value is a C2OValue Return Type.

Example

Consider a group named DemoGroup and an array as the following illustration shows:

The following syntax retrieves the value of the mUsername field from the array for the named connection DemoGroup:

```
getValueFromValueMapArray("DemoGroup", "MailServerCredentials",
"mNamedConnection", "demo", "mUsername")
```

The output is a C2OValue where:

- The fieldName value is mUsername
- The fieldValue value is test@mydomain.com

getValueMapFields

The getValueMapFields function returns the list of field names inside a ValueMap as an array of string variables.

Syntax

```
fieldNames = getValueMapFields(vmap)
```

Arguments

vmap (ValueMap)

Specifies the ValueMap object whose field names needs to be returned.

Return Value

fieldNames (String Array)

An array of Strings containing the names of the fields found in the ValueMap.

Example

```
fieldNames = getValueMapFields(Process.vmBooks);
```

getValuesFromValueMapArray()

The getValuesFromValueMapArray returns all the columns or fields of an array.

Syntax

```
getValuesFromValueMapArray(groupName, arrName, fieldName, fieldValue)
```

Arguments

groupName (String)

Defines the name of the Custom Group that you publish.

arrName (String)

Defines the name of an array of C2OValueMap.

fieldName (String)

Defines the name of the field against which you retrieve the value of other columns or fields in an array.

fieldValue (String)

Defines the actual value of the field against which you retrieve the value of other columns or fields in an array.

Return Values

The return value is a C2OValue Return Type.

hasField

The hasField function determines if a field exists in a ValueMap.

Syntax

```
bHasField = hasField(valuemap, fieldName)
```

Arguments

valuemap (ValueMap)

Specifies the input ValueMap.

fieldName (String)

Specifies the field name.

Return Value

bHasField (Boolean)

Returns true if the field exists in the ValueMap or false otherwise.

Example

```
bHasField = hasField(Process.vmBooks, "author");
```

include

The include function loads JavaScript code that is defined in a file for use in the pre-execution or post-execution section of any operator or in the SourceCode section (see page 406) of the Run JavaScript operator. The loaded JavaScript code is only good for the duration of the pre-execution, post-execution or SourceCode section that it is loaded in.

Syntax

```
include(jsFile)
```

Arguments

jsFile (String)

This parameter must refer to a JavaScript file. This parameter can be an HTTP URL in which case the parameter must start with a file URL or it can be a path inside the c2ouserresources folder. The path must be relative to the ".c2ouserresources" folder itself. ".c2ouserresources" is present inside the .c2orepository folder in the CA Process Automation installation folder.

Return Value

No value returned (void).

Examples

The following example is an HTTP URL; the parameter starts with a file URL..

```
include('http://test.ca.com/test.js')
```

The following examples load the test file from a path relative to the .c2ouserresources folder.

```
include('file://c:/test.js')
Include('test.js')
```

isFIPSMode

The isFIPSMode function lets you programmatically determine whether the CA EEM server has FIPS mode enabled. This function returns true if the CA EEM server is running when FIPS mode is enabled, and false if the server is not running.

Syntax

```
isFIPSMode()
```

Return Value

True

Returned if the CA EEM server is running when FIPS mode is enabled. (String)

False

Returned if the CA EEM server is running when FIPS mode is not enabled. (String)

isTouchpointUp

The isTouchpointUp function determines if a touchpoint is active.

Syntax

```
bIsUp = isTouchpointUp(touchpointName)
```

Arguments

touchpointName (String)

Specifies name of the touchpoint.

Return Value

bIsUP (Boolean)

Returns true if the touchpoint is active or false otherwise.

Example

```
bIsUp = isTouchpointUp("AccountingTouch");
```

load

The load function loads JavaScript code that is defined in a dataset variable for use in the pre-execution or post-execution section of any operator or in the SourceCode section (see page 406) of the Run JavaScript Operator. The loaded JavaScript code is only good for the duration of the pre-execution, post-execution, or SourceCode section where it is loaded.

Syntax

```
load(jsCode)
```

Arguments

jsCode (String)

Specifies the JavaScript code to load.

Return Value

No value returned (void).

Examples

1. Define a dataset object that is called Common with a parameter jsCode that contains the following JavaScript:

```
function convertToUpperCase(sValue) {return
sValue.toUpperCase()};
```

2. Use the load function to load that piece of code and make functions in that code available to you:

```
load(Datasets["Common"].jsCode);
Process.ucValue = convertToUpperCase("helloworld");
```

lockResource

The lockResource function locks or unlocks one or more resources in a resource object. If you specify a value for resourceName, it resets only that resource. If you leave resourceName empty, it resets all of the resources in the resources object.

Notes:

- If resources that do not exist are provided as input, resources are still created.

- If you specify a value for resourceName and that resource does not exist, the resource is created with an amount of zero and set to the specified state.

Syntax

bSuccess = lockResource(*resourcePath*, *resourceName*, *state*)

Arguments

resourcePath (String)

Specifies the path of the resources object.

resourceName (String)

Specifies the name of the resource in the resources object.

state (Boolean)

Specifies whether the resource should be locked or unlocked. Set to true for locked and false for unlocked.

Return Value

bSuccess (Boolean)

Returns true if the function succeeds or false if it fails.

Example

bSuccess = lockResource("/Resources/Locks", "InvLock", true);

logEvent

The logEvent function inserts a custom message into the logs of a process instance.

Syntax

logEvent(*level*, *category*, *msg*)

Arguments

level (Int)

Specifies one of the following log levels:

4 = Error

3 = Warning

2 = Notice

1 = Normal

category (String) (this value is optional)

Specifies one of the following log categories:

- "CUSTOM" (the default)
- "FLOW_CATEGORY"
- "AGENDA_CATEGORY"
- "ICON_CATEGORY"
- "HANDLERS_CATEGORY"
- "RESPONSE_CATEGORY"
- "OTHERS_CATEGORY" (PROCESS,AGENDA,OPERATOR,HANDLER, RESPONSE,OTHERS, CUSTOM,OTHERS)

msg (String)

Specifies the log message.

Return Values

No value returned (void).

Example

```
logEvent(1, "FLOW_CATEGORY", "Start New Hire Process has completed");
```

newValueMap

The newValueMap function creates and returns a new ValueMap.

Syntax

```
vmData = newValueMap()
```

Arguments

None.

Return Value

vmData (ValueMap)

Example

```
Process.myVmap = newValueMap()
```

newDataset

The newDataset function dynamically creates and returns a new ValueMap that can be assigned to a dataset object. If you create a dataset that already exists with this function, that existing dataset is not affected.

Syntax

```
Datasets["/mydataset"]=newDataset()
```

Arguments

None.

Return Value

ValueMap

Example

```
Datasets["/mydataset"]=newDataset()
```

nextOpenDate

The nextOpenDate function returns an open date given a targetDate by considering the includeCalendar, excludeCalendar, and maxShifts. If no open date is found with the given inputs, the result is null.

Syntax

```
dtNextOpenDate = nextOpenDate(targetDate, includeCalendar,
excludeCalendar, maxshift)
```

Arguments

targetDate (java.util.Date)

Specifies the desired date.

include_calendar (String)

Specifies the path of the include calendar object.

exclude_calendar (String)

Specifies the path of the exclude calendar object.

maxshifts (Int)

Specifies the maximum acceptable number of shifts when searching for an open date. Specify positive numbers to increment the date and negative numbers to decrement the date. The system caps the maximum number of shifts at 5.

Return Value

dtNextOpenDate (java.util.Date)

now

The now function returns the current date including the time.

Syntax

```
dtNow = now()
```

Arguments

None.

Return Value

dtNow (java.util.Date)

The return value is the current date and time.

parseDate

The parseDate function returns a date object after parsing the specified string in the required format.

Syntax

```
dtDate = parseDate(dateStr, format)
```

Arguments

dateStr (String)

Specifies the string that needs to be parsed as a date.

format (String)

Specifies the format required to interpret the date string provided; for example: MM/dd/yyyy.

Return Value

dtDate (Date)

Date equivalent of input dateStr String as parsed using the input format String.

Examples

1. Sets dtDate to a value of Dec 10, 2009 12:00:00 AM, when the format is MM/dd/yyyy.
    ```
    dtDate = parseDate('12/10/2009','MM/dd/yyyy')
    ```

2. Sets dtDate to a value of Oct12, 2009 12:00:00 AM, when the format is dd/MM/yyyy
    ```
    dtDate = parseDate('12/10/2009','dd/MM/yyyy')
    ```

3. Fails the following input because the input date string does not match the input format.
    ```
    dtDate = parseDate('12-10-/2009','dd/MM/yyyy')
    ```

resetResource

The resetResource function resets one or more resources in a resource object by unlocking them and setting the used count to zero. If you specify a value for resourceName, only that resource is reset. If you leave resourceName empty, all of the resources in the resources object are reset.

Note: If you specify a value for resourceName and that resource does not exist, the resource is created with an amount of zero and set to the unlocked state.

Syntax

```
bSuccess = resetResource(resourcePath, resourceName)
```

Arguments

resourcePath (String)

Specifies the path of the resources object.

resourceName (String)

Specifies the name of the resource in the resources object.

Return Value

bSuccess (Boolean)

Returns true if the function call is successful and false if it fails.

rollDate

The rollDate function is used to roll a particular value that is based on a date. For example, perhaps you want to send a feedback email one day after a service desk request was closed. In that case, to write the automation logic, use this function.

When this function is executed, the value num is added to the date dt based on the type.

Syntax

```
dtRollDate = rollDate(dt, num, type)
```

Arguments

dt (java.util.Date)

Specifies the date object that is based on the rolling to take place.

num (Int)

Specifies the value that must be rolled.

type (String)

Specifies the one of the following values:

- "y"(year)
- "d"(day)
- "w"(week)
- "m"(month)

Return Value

dtRollDate (java.util.Date)

Examples

1. Returns the date which is two days from today.

    ```
    dtRollDate = rollDate(today(),2,'d')
    ```

2. Returns the next year from today; for example, if today is November 12, 2009, this example will return January 1, 2010.

    ```
    dtRollDate = rollDate(today(),1 'y')
    ```

3. Returns the first day of the previous week.

    ```
    dtRollDate = rollDate(today(), -1,'w')
    ```

4. Returns the first day of the previous month.

    ```
    dtRollDate = rollDate(today(),-1 ,'m')
    ```

rollTime

The rollType function rolls the current hour into the provided value and returns the value in hours based on a 24-hour clock. In this convention of timekeeping, the day runs from midnight to midnight and is divided into 24 hours, numbered from 0 to 23.

Syntax

```
nHTime = rollTime(num, type)
```

Arguments

num (Int)

Specifies the value that needs to be rolled.

type (String)

This value can only be "h".

Return Value

nHTime (Int)

Example

Returns the time three hours before the current time. For example, if it is currently 9 PM, this example returns 18, which is 6 PM on a 24-hour clock.

```
nHTime = rollTime(-3 ,'h')
```

saveAttachmentToFile

The saveAttachmentToFile function saves the content of an attachment, identified by a unique ID, to the specified file location. The function returns the absolute path of the new file with the attachment content.

Syntax

sFileName = saveAttachmentToFile(*nAttachmentID*, *sFileDirName*)

Arguments

lAttachmentID (long)

Specifies a unique ID that identifies the attachment containing the desired content.

sFileDirName (String)

Full path and file name to the location where the file will be written.

If a file path is not provided, then the file will be written to the *install_dir*/server/c2o directory.

If a file is not specified, a unique file will be generated.

If only a path is specified, the path must include the path separator character at the end of the path ('\' for Windows or '/' for UNIX).

Return Value

sFileName (C2OString)

The full path to the file, including the file name, is returned if the function is successful. If the function fails, NULL is returned. The return type is C2OString.

Example

```
var  i = Process.attachments[0].attachmentID;
Process.save =  saveAttachmentToFile( i, "attach.txt");
```

setOperatorStatus

The setOperatorStatus function is used to either force fail or force pass the operator.

Syntax

setOperatorStatus (o*peratorStatus*, *operationResult*, *reason*)

Arguments

Operator Status (String)

Specifies the state of the operator. This argument can take either success or failure values only.

Operation Result (Int)

Specifies the operation result. This argument overrides the operator result in the operator dataset.

Reason (String)

Specifies the reason that overrides the operator reason in the operator dataset.

Return Value

No value returned (void).

Example

The following example performs a force success on the operator with an operation result of 1 and the reason as "force success".

```
setOperatorStatus("success",1,"force success")
```

setProcessProgress

The setProcessProgress function sets the progress of a process in the following areas:

- Pre-execution or post-execution operator code
- A RunJavaScript operator

Syntax

```
setProcessProgress (ProcessProgress)
```

Arguments

ProcessProgress

Defines the percentage completion of a process.

Return Value

No value returned (void).

Example

Use the setProcessProgress method to set the process progress to 30 percent as follows:

```
setProcessProgress(30)
```

When a process reaches the operator, the operation dashboard or the process dataset displays the progress as 30.

setResourceTotal

The setResourceTotal function sets the total amount of resources with the resource name "resName" to the "amount" specified in the resource object on the path "resPath".

Notes:

■ If you provide resources that do not exist as input, this function creates the resources.

■ If the resName parameter is blank, this function sets the total amount for all of the resources in the resources object.

Syntax

```
bSuccess = setResourceTotal(resPath, resName, amount)
```

Arguments

resPath (String)

Specifies the path of the resource object.

resName (String)

Specifies the resource name that you set in the resources object.

amount (Int)

Specifies the total amount that you set on the resource.

Return Value

bSuccess (Boolean)

Returns true if the function is successful or false if the function fails.

Example

```
bSuccess1=setResourceTotal(Process.ResObjName, Process.ResName_1,
1);
bSuccess2=setResourceTotal(Process.ResObjName, Process.ResName_3,
3);
```

today

The today function returns the current date and time. The time returned is 12:00 AM.

Syntax

```
dtToday = today()
```

Arguments

None.

Return Value

dtToday (java.util.Date)

 Returns the current date.

Example

If today is December 12, 2009, returns the date December 12, 2009 12:00 AM.

Index