CA Process Automation

Content Designer Guide

Release 04.2.00

This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the “Documentation”) is for your informational purposes only and is subject to change or withdrawal by CA at any time. This Documentation is proprietary information of CA and may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA.

If you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION “AS IS” WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with “Restricted Rights.” Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2013 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

CA Technologies Product References

This document is part of a bookshelf with 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 1: Introduction to Process Automation 11

Process Development Phases....13
Phase I: Requirements Analysis....14
- Identify Processes to Automate....15
- Identify Where Processes Run....15
- Identify Steps in Processes....16
- Identify Interdependencies....16
- Identify External Dependencies....17
- Identify Runtime Constraints....17

Phase II: Design and Implementation....17
- Identify Process Objects and Operators....18
- Configure Operators and Steps....20
- Optimize for Modularity and Component Reuse....22
- Define Process Initiation and Monitoring....23
- Define Alerts....26

Phase III: Testing and Deployment....27
- Test the Process with Related Objects....27
- Generate the Process Documentation....29
- Set the Release Version of the Process and Related Objects....29
- Assemble the Process with Related Objects....29
- Deploy a Release Version of the Process with Related Objects....30
- Specify Import Instructions....30

Chapter 2: Getting Started 31

Browse to CA Process Automation and Log In....32
Configure User Settings....33
The CA Process Automation User Interface....34
- Main Application Pages....35
- Common User Interface Controls....48

Browse Out-of-the-Box Content....50
Use CA EEM to Change Your CA Process Automation Password....51
Web Browsers....52

Chapter 3: Library Browser 53

Customize the Library Browser....53
Search the Library Browser....53

Search for Release Version Information 55
Search for Version Information 56
Search for Audit Trail Information 57
Working with Objects and Folders 58
Automation Object Types 58
Create a Folder 60
Create an Object 61
Edit an Object 62
Versions 71
Delete or Restore an Object or Folder 83
Copy and Paste an Object or a Folder 83
Cut and Paste an Object or a Folder 84
How to Work with Nonmodifiable Content 85

Chapter 4: Designing Processes 87

The Process Designer 88
Operators and Links: The Building Blocks 90
Create a Process Object 91
Design a Process 92
Process Operators 93
The Start Operator 95
Add Operators to a Process 96
Logical Operators 97
The Stop Operator: Success or Failure 99
Process Operator Ports and Links 100
Add Operator Ports and Links 102
Custom Exit Ports and Expressions 103
Break a Link for Readability 105
Process Loops and Iterations 105
System Variables for Looping 106
Loop an Operator in a Process 107
Loop a Series of Operators 111
Loop a Process 114
Process Control 116
Child Processes 117
Inline Process 120
Process Lanes 123
Create Horizontal or Vertical Lanes 124
Manage Swim Lanes 125
Lane Handling Rules 126
Process Versions 126

Document a Process 127
Add Comments to a Process 127
Set the Name for an Operator in a Process 128
Change and Display Operator Information in a Process 128
Self-Contained Content 129
Self-Contained Content Links 129
Navigate to a Specific Part of a Process 132
Multi-Tenancy and CA Process Automation 132
Make a Process Aware of Multiple Tenants 133
Inherit Security in Subprocesses 134
Add Variables at Time of Initialization 134
Multi-Tenant Processes Using Process Watch 134
The CA Process Automation Code Editor 134

Chapter 5: Operators and Icons 141

Operators 142
Configure Operator Properties 144
Auto Recovery 145
Java and External JARs 146
Custom Operators 152
Your Favorite Operators 175
Connectors 177
Operator Icons 178
Operator Status Icons 179
Creating, Editing, and Applying Custom Icons 180

Chapter 6: Datasets and Parameters 189

Datasets 189
Create a Named Dataset Object 191
Define Dataset Pages and Variables 193
Modify a Dataset 199
View or Copy a Dataset Expression 200
Read Operating System Values into Dataset Variables 200
Process Parameters 208
Operator Properties 208
Calculated Parameters 222

Chapter 7: Forms 257

Start Request Forms 257
Monitor Start Request Form Instances and Process Instances 258

Interaction Request Forms 259
The Form Designer 260
Form Elements 261
Form Element Properties 264
Form Element Events 275
Form Element Functions 277
Initialize Form Variables 342

Chapter 8: Resources 343

How Resources Work 343
Create a Resource Object 345
Edit a Resource Object 346
Monitor and Edit Resources 348
Add a Manage Resources Operator to a Process 349
Define Resource Actions 350
Check for and Respond to Unavailable Resources 351
Specify a Time-Out Interval 351
Specify Resource Availability and Action Settings 352
Check for Resource Availability without Executing Actions 353

Chapter 9: Calendars, Schedules, Tasks, and Triggers 355

Calendars 355
Create a Calendar Object 356
The Basic Calendar Designer 357
The Advanced Calendar Designer 358
The Calendar Designer: Preview Tab 368
Exclude Calendars 369
Schedules 370
Create a Schedule Object 372
Schedule Process and Operator Tasks 373
Preview All Occurrences of a Scheduled Task 375
Using Schedules 376
Monitor Active Schedules 376
Monitor All Occurrences of All Scheduled Tasks 377
Task Management 378
Assign a Task to a User 378
The Task List 380
Administer Triggers 381
Controlling Processes from an External Application with SOAP Calls 382
How File and Mail Triggers Work 382
SNMP Trap Input Considerations 386

Chapter 10: Running, Testing, and Debugging Processes 389

The Operations Page 390
Filters for Process Instances 391
Filter Objects Displayed by a Shortcut 392
Process Watch Objects 393
Content Package Objects 394
Execution Rules 396
Runtime Security 397
Properties Affecting Security of Running Processes 397
Guidelines for Setting Runtime Security for a Process 399
Exception Handling 399
Create Exception Handlers 401
Run Processes Interactively 404
Start a Process from the Library 405
Start a Process as Suspended from the Library 406
Start a Process While Editing 407
Open an Instance of a Process 407
Process States 408
Debug a Process 408
Suspend a Process 408
Change whether Processes are Unloaded on Completion 409
Set and Remove Breakpoints in a Process 409
Debug a Java Process 411
Reset a Process 412
Abort a Process 412
Control a Process Branch 412
Disable Operators or Deactivate Branches 413
Abort an Operator 413
Reset Operators in a Process 414
Resume Execution of a Suspended Process 414
Simulate Processing of Operators 415
Simulate Processing of a Selected Operator 416
Simulate Processing of an Entire Process 417

Chapter 11: Generating and Using Process Documentation 419

About Process Documentation 419
Generate Process Documentation 420

Chapter 12: Release Objects to Another Environment 423

Release Versions 424

View Release Version Information 424
Set the Release Version of Objects to Export 425
Baseline the Release Version of the Objects 426
Set the Release Version of Content Package as Current 427
Exporting and Importing 428
Exporting 428
Importing 442
Values Maintained in the Initial Version of a Copied Object 449

Appendix A: Format Specifiers for Dates 451

Appendix B: Using Masks to Specify Patterns in Strings 455

Mask Syntax 455
Sample Mask 458

Appendix C: How Targets of an Operator are Processed 459

How Targets for an Operator Can Be Specified 459
Processing a Target Specified as an IP Address or FQDN 460
Processing a Target Specified as the ID of an Agent or Orchestrator 463
Use Case: Track Recovered Processes through Logs 465
Operators Auto Recovery Example 465

Index 473

Chapter 1: Introduction to Process Automation

Information Technology teams like yours are automating their IT processes. By defining, automating, and orchestrating processes across systems, you can improve productivity while enforcing standards across departments. Automation helps your organization to:

- Reduce operational expenses
- Increase staff productivity
- Speed IT service delivery
- Improve service quality
- Enforce compliance policies

Automate operational processes that are otherwise manual, time-consuming, inconsistent, or error-prone. You can automate IT processes that span multiple systems across multiple organizations.

Welcome to CA Process Automation

Use CA Process Automation to design, test, manage, and report on automated processes that support IT operations and production environments. CA Process Automation speeds the delivery of IT services while reducing manual errors. You can

Use CA Process Automation in your web browser to manage, design, and deploy processes in an easy-to-use interface that supports the following essential features:

- Visual authoring including familiar drag-and-drop icon-based designs.
- Enterprise process monitoring with the ability to pause, modify, and resume live running processes.
- Role-based views with flexible layout options.
- User-friendly objects. Processes, calendars, schedules, datasets, forms, and other automation objects are represented as objects you can click.
- Easy organization of automation objects through a library of hierarchical folders.
- Object version control including check in, check out, and selection of current and baseline versions.
- Active assistance features to help you work with datasets, variables, and expressions.
- Process operators that integrate with other applications and systems.

This section contains the following topics:

Process Development Phases (see page 13)
Phase I: Requirements Analysis (see page 14)
Phase II: Design and Implementation (see page 17)
Phase III: Testing and Deployment (see page 27)

Process Development Phases

Before working with CA Process Automation, it is important for you as a content developer or designer to be familiar with the concepts behind process development. The remainder of this section walks you through the phases and steps involved in automating IT processes.

Process development includes the following phases:

Phase I: Requirements Analysis (see page 14)

Identify and characterize the automation processes required at your organization.

Phase II: Design and Implementation (see page 17)

Map requirements to automation objects and configure the elements. Identify existing elements to reuse, modules you need from other parties (for example, database-related processes or data from database administrators), and modules you must develop yourself. You can then use CA Process Automation development tools to import or define those automation objects in your library.

Phase III: Testing and Deployment (see page 27)

Export all relevant automation objects to a file and facilitate their import into the CA Process Automation production environment. Provide production administrators with instructions explaining how to configure the elements to work in the production environment.

Phase I: Requirements Analysis

This phase focuses on what information to gather and which issues to address before you design a CA Process Automation management package or process. Your specific sites and management solutions could require additional tasks.

Include the following tasks in a typical requirements analysis:

- ☐ Identify processes to automate.
- ☐ Identify where the processes run.
- ☐ Identify steps in the processes and describe the outcomes of each step.
- ☐ Identify the interdependencies.
- ☐ Identify the external dependencies.
- ☐ Identify the runtime constraints.

Good candidates for process automation meet one or more of the following initiation requirements:

- You can identify multiple, often interdependent, tasks.
- You can identify overlapping resource requirements.
- The process can run over the network or on different platforms.
- You can apply date and time constraints to schedule the process.
- Other processes, applications, or users can trigger the process.
- System or database administrators, operators, or other users can run the process on demand.

Identify Processes to Automate

The first step in the CA Process Automation development process is to identify processes for automation. A process typically consists of multiple subprocesses.

Follow these steps:

1. Identify the primary task or use case, such as *move data from distributed transactional databases to a single data warehouse*.
2. Identify subprocesses for subordinate tasks, such as:
   a. *Extract data from different sources*
   b. *Transform extracted data*
   c. *Load the transformed data*
3. Separate the components that are common to multiple processes into modular subprocesses. Similar to code reuse, modularization of shared subprocesses saves development and maintenance time.

Note: Later in this guide, you will learn that subprocesses correspond to separate process objects (see page 58) in the Library Browser.

Identify Where Processes Run

Identify where you want to run the following components:

- The process itself
- Subprocesses
- Tasks in the process and subprocesses

Note: Later in this guide, you will learn that these locations correspond to *touchpoints* running processes or individual steps in processes. Your administrator configures touchpoints for you.

Identify Steps in Processes

List each step that is performed as part of the process or subprocess.

Follow these steps:

1. Identify each step. Examples include:
   - Running an application
   - Obtaining data from a remote source
   - Performing one or more calculations
   - Prompting a user for information with a form
   - Starting a subprocess
2. For each step, determine the following possible results or outcomes:
   a. Normal outcomes
   b. Abnormal outcomes
   c. Any conditional outcomes

   A step can produce a result with conditional outcomes. When the result is greater than a specified value, the step yields one particular outcome. When the result is less than a specified value, the step yields a different outcome. Multiple calculated outcomes determine the subsequent paths through the branches of a process.

Note: Later in this guide, you learn that steps in a process design correspond to *operators* (see page 90).

Identify Interdependencies

Interdependencies determine the logical flow through a process.

Follow these steps:

1. Determine which steps in a process must precede or follow other steps.
2. Determine how the outcome of each step affects subsequent steps.
3. For a step with more than one outcome, identify each outcome and list the separate sequence of steps that must run after each outcome.

   Note: The application performs each outcome separately. For example, an abnormal outcome of a step can cause a process to alert another step to wait for input. For the same step or operator, a custom outcome could activate a separate branch of the process to handle an error condition.

Identify External Dependencies

Identify external dependencies of steps in a process, such as:

- Remote systems and applications.
- Shared resources such as file systems or databases.
- Synchronized processes, and other processes that run in parallel.

Identify Runtime Constraints

Identify the resources that a process requires at runtime. Because they are shared, resources are always in limited supply; for example, processors, memory, or access to other files, databases, or hardware. Be aware of two main types of runtime constraints:

- **Time constraints** define when a process must be completed. Examples include backups that run at night or system maintenance activities that run only during a specific time window.
- **Configuration constraints** include process environment requirements. For example, data that must be available to a process, including files, programs, user profiles, and account data such as user names and passwords.

Phase II: Design and Implementation

In the design phase of process development, you map requirements to CA Process Automation automation objects and operator functionality. The steps in the design and implementation of a process include:

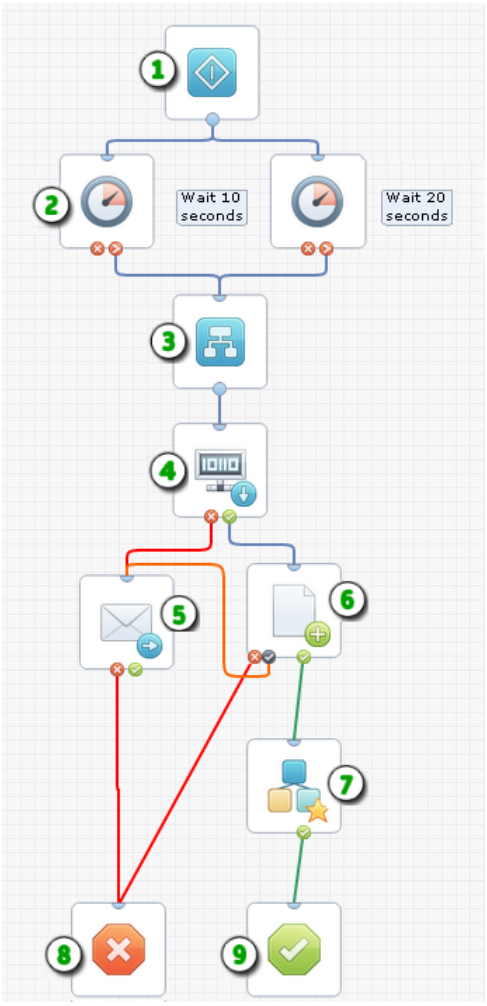
- ☐ Identify the automation objects and process operators to implement the process.
- ☐ Configure all operators and logical steps, including all tasks, parameters, outcomes, resources, and dependencies. Define any required runtime configuration information (such as user names, passwords, and file names).
- ☐ Optimize your design for modularity and component reuse.
- ☐ Define process initiation and monitoring including the permissions for starting, monitoring, and modifying the process.
- ☐ Define the alerts and determine which users or groups are alerted.

Identify Process Objects and Operators

Each process or subprocess that you identify corresponds to a process object in the Library Browser. Process objects define a sequence of linked steps. A process operator represents each step. After you map a step-by-step process, identify the operators that perform each step. Sometimes a single operator can perform a given step. Other times, a step can require two or more separate operators.

Define steps in a process by placing icon-based operators that represent actions that CA Process Automation performs. Start operators begin processes. Stop operators end processes. Logical and control operators define starting and stopping points, branching, and iterations in a process. The design for a process reveals its structure and also maps, synchronizes, and defines both the sequence and dependencies between tasks.

Example: Operators in a Process



Item:	Description:
1	**Start**: The Start operator represents the single entry point where the process begins. Start operators can also indicate the first step in an independent branch.
2	**Delay**: The Delay operator delays processing subsequent branches of a process until a specified interval of days, hours, minutes and seconds passes.
3	**Or Operator**: *Or* and *And* operators control steps logically. The Or operator only completes (and lets processing continue) when one of the input operators completes. The And operator only completes (and lets processing continue) when all of the input operators complete.
4	**Get SNMP Variable**: This operator returns the value of an SNMP variable.
5	**Send Email**: Use this operator to notify other users by email.
6	**Write File**: This operator writes a dataset variable to a file. The custom exit port specifies a predefined course of action for a specific exit condition.
7	**Start Process**: This operator starts another process. You can set an optional Loop property to run another process repeatedly.
8	**Stop Failure**: The Stop Failure operator stops all branches in a process and sets the process to a *Failed* state. Use a Stop Failure operator for the abnormal outcome of a process or branch.
9	**Stop Success**: The Stop Success operator stops all branches in a process and sets the process to a *Completed* state. Use a Stop Success operator for the normal outcome of a process or branch.

More information:

Getting Started (see page 31)
Designing Processes (see page 87)

Configure Operators and Steps

After you identify operators for each step, define their properties. Think about the types of data they need and the source of that data. Some operator properties are common to all operators and others are limited to specific types of operators. For example, a database operator requires parameters for the type of database, the database name, a valid user name, and password.

You can set the value of a property to a literal entity such as *5* or *Australia* or to an expression such as *Var1.quantity* or *GetCountry*. Expressions often include variables or functions. A variable stores a value. For example, a variable can contain a password or a threshold setting. A function inspects some logical statement and returns a value. For example, a function can get or set a field value.

Expressions can include any of the following items:

- User and system variables
- Functions
- Literal values
- Logical, string, arithmetic, and comparison operators
- JavaScript expressions

Some fields accept only literal values. The labels for these fields appear in italics. Most fields support calculated expressions.

Note: Previous versions of CA Process Automation identify fields that accept calculated expressions with an asterisk (*).

Operators also share the following common properties:

- Execution target: *Where should this operator run?*
- Time-out settings: *What should happen when the operator takes too long to run?*

To learn more about the functionality and properties of each operator, see the *Content Designer Reference*.

Datasets store CA Process Automation variables. You can add and edit variables in any of the three dataset types:

- **Named Datasets**: The dataset objects that you create and maintain in the Library Browser.
- **Process Datasets**: The dataset for each process.
- **Operator Datasets**: The dataset for each operator in a process.

Additionally, the read-only system dataset makes system data available to any expression. Use variables and more complicated expressions to define values that can change, such as user credentials. You can assign values for expressions and variables globally or dynamically change a value in code. Use literal values to configure parameters only if the values are unlikely to change.

Named Datasets

Use global variables in named datasets to define parameter settings. Named datasets are automation objects that define variables having scope across the entire library. Depending on the security settings for a dataset object, its variables are globally accessible to any expression in any automation object. A field in another dataset can reference a dataset variable.

Use named datasets to specify information such as accounts and passwords used in multiple processes. When information saved in a named dataset changes, you only have to change it once, in the named dataset, instead of in each automation object that uses it.

You can use Start Request Forms to gather information from authorized users and then update the values of dataset variables. You can use a Run JavaScript operator in a process to calculate and store values in named variables.

More information:

Start Request Forms (see page 257)
Access Dataset Fields in Expressions (see page 236)

Resources

Some operators or processes draw heavily on memory, processor time, or other limited resources. For example, if a child process calls on other processes, monopolizing many CPU cycles, you can limit the number of instances of the child process running at any given time.

To manage resources, define a resource object that the process draws on. When the resource is locked or consumed by running processes, additional requests for the resource must wait until the resource is available or free. Use resources to handle load balancing in your processes.

More information:

Resources (see page 343)

Assign Exit Conditions to Outcomes

You can define an exit condition for each outcome of the operator. An operator can have multiple exit conditions for different outcomes, such as *Completed* or *Failed*.

Operators support specific exit conditions. Some operators also allow you to define a custom port. For example, you can define a custom port that only runs if the operator result is *True, False*, "blue", *500, -2,* or some other designated value.

Define a custom port when you have a distinct course of action in mind for a particular exit condition. One such condition is when a database import fails for a particular reason. When a process runs and you encounter a situation with no specific exit condition, the process is suspended and switches into a blocked state, pending user action. An administrator or other user with sufficient permissions can change parameter settings if necessary and can restart a suspended process either where it failed or from the beginning.

Optimize for Modularity and Component Reuse

Break processes into separate subprocesses or *child* process objects. Use a main *parent* process to control the subprocesses. Each subprocess performs a distinct task or set of tasks, such as extracting data from a database and writing it to separate files. When multiple processes share a common subprocess, maintain flexibility in the design to allow the subprocess to continue to be used in the different parent processes.

Add flexibility by using variables or expressions to set parameters on the process and on each operator. This way, if a process is reused in different environments or systems, adjusting the process is as easy as updating only the relevant variables.

CA Process Automation also helps you control the flow among different paths or subprocesses. For example, when processes require input at runtime, you can use *Interaction Request Forms* to prompt users for information. Use one or more interaction request forms to implement process checkpoints where specific users or groups are required to authorize a specific subprocess before it can proceed. The Assign User Task operator in a process object pauses the process and then opens the form. The user can then enter the required values.

Custom Operators

You can base a custom operator object on any of the other operators available in your environment. Use custom operators to:

- **Facilitate reuse**: Use the same operator with the same configuration settings in different processes.
- **Save the settings for a task**: You can preconfigure custom operators to perform specific tasks. For example, you can define a custom operator to work with an application that is already installed on your system.
- **Interact with enterprise applications**: You can export a folder as a content package to support common enterprise applications.

You can modify your custom operator without changing the original operator.

Define Process Initiation and Monitoring

Process Initiation

When designing processes, decide how each process starts. You can design a process to start in several ways:

- **Manual Start**: You can manually start a process in the Library Browser or the Process Designer. As a designer and content developer working in CA Process Automation, you routinely start processes manually while designing and testing them.
- **Automatic Start**: You can construct a process that automatically starts another process using the Start Process operator.
- **Automatic Start by External Entity**: You can set a process to start from an external application or system using Web services, a command line utility, or *triggers*. For example, you can specify that another program starts a process using Web Services (*executeProcess* or *executeStartRequest*). You can set a process to start by an external event through the use of supported triggers including file creation, incoming e-mail, SNMP traps, and events from UCF connectors.
- **Start by Schedule**: You can schedule a process to start by using a Schedule object.
- **Start by User and Form**: You can design a Start Request Form object or Interaction Request Form object that can prompt a user to respond to the form resulting in the initiation of another process. An example of this method for starting a process is an expense report submission form. An employee can fill out the form and start an *UpdateExpenses* process on demand.

Process Control

Designers, administrators, and production staff use CA Process Automation to control processes. To learn more, refer to the following content for your role:

- **Designers:** To run, test, and debug processes, see Running, Testing, and Debugging Processes (see page 389).
- **Administrators:** To administer application or system content, see the *CA Process Automation Content Administrator Guide.*
- **Production Users:** To start processes interactively in a production environment, see the *CA Process Automation Production User Guide*.

Schedule Processes

Two automation objects that are used for scheduling processes are *calendars* and *schedules*. Use calendar objects to define date conditions for starting tasks that launch processes or individual operators. Use *schedule* objects to schedule tasks using calendar objects and/or explicit dates.

- Creating date conditions with calendar objects is described in Calendars (see page 355).
- Scheduling tasks with schedule objects is described in Schedules (see page 370).

More information:

Calendars (see page 355)

On Demand Processes

The development and administrative tools available on the Operations tab expose the underlying structure and logic of libraries and applications. The Operations tab might feature a Start Request Form object to prompt a user for information required to start a process. Authorized users can use Start Request Forms to run on-demand processes without knowing the technical details behind:

- How CA Process Automation works.
- How the system architecture is designed.
- How each process is structured.

More information:

Start Request Forms (see page 257)

Triggered Processes

CA Process Automation supports events and *triggers* that can start processes from external applications and systems. You can designate applications, email messages, web pages, or other processes to trigger a process. Any of the following methods can be used to trigger a process:

- FTP
- HTTP/SOAP post
- Custom SNMP traps
- SMTP (email)
- UCF event

To learn more, see Triggers (see page 381).

Monitor Processes

You can graphically monitor processes using a *process watch* object. Process watch objects provide portals or shortcuts to all processes and related objects for a particular category of ownership. For example, a data warehouse team requires access to a process watch object containing shortcuts to all extract, transform, and load (ETL) processes for populating data warehouses.

Consider roles and ownership when managing a process watch. For example, an end user must typically see and respond to processes as they are occurring. In contrast, an administrator might need to view history. A process watch accounts for ownership by including only those objects that the person using it must monitor.

More information:

The Operations Page (see page 390)

Define Permissions

Permissions depend on who owns the components being managed or who is responsible for particular management functions. You can assign permissions by functional group (such as administrators, operators, or a data warehouse team) or by ownership. You can then add or remove individual users in a group.

For example, a data warehouse team would monitor the extraction, transformation, and loading (ETL) processes to populate a data warehouse. Create a *DW* group containing the data warehouse team members. Then, give this group List and Open permissions to a process watch object providing all shortcuts related to the ETL processes.

Administrators are assigned greater permissions than general users. A database administrator can have control over processes to update a data warehouse or to restore or back up certain databases. If you are developing for a complicated enterprise environment with many different roles and ownership, consider this approach. Map out ownership in a spreadsheet or other format for the groups, users, and roles to assign appropriate permissions.

Ownership for a process (or any automation object) is initially assigned to the user under which it is created. Ownership can be changed.

Defining permissions for actions users can take at runtime is handled by the permissions editor in CA EEM. Members of the default *PAMAdmins* group have full permissions on all folders and all automation objects. Members of the default *PAMUsers* group have no permissions on any folder or automation object. Content designers are typically members of the Designers group.

A content administrator or the owner of a folder can assign you permissions on the folder with the Set Owner property. Folder permissions are inherited by automation objects added to the folder. An administrator with CA EEM credentials can assign you permissions on automation objects and folders in CA EEM.

Note: See the *Content Administrator Guide* for details.

Define Alerts

You can use email alerts in processes to notify users or administrators about errors or incidents, or to inform users that a process or task is complete.

When you design alerts in a process, first define the critical recipients. For example, you might notify a single administrator when a process succeeds, fails, or is waiting for user input. You can also create a recipient group to alert managers that might need to receive monthly reports.

Create separate processes that not only send alerts, but handle escalation and perform other tasks related to the alert. Then, invoke the alert process from other processes as needed using the *Start Process* operator.

Always use named dataset variables in CA Process Automation to specify email recipients, users, groups, profiles, and account passwords. Use the variables to set parameters in the *Send Email* operator. When changes occur, you need only edit the variables in a named dataset rather than in all the processes using the values.

Because individuals change, you can define user email groups or email aliases based on function or role. When personnel change, you can then change individuals assigned to the aliases without having to edit variables. Your organization may already have aliases for the appropriate roles or groups.

Phase III: Testing and Deployment

Deploying your CA Process Automation management package to a production environment includes the following activities:

1. Assemble the automation objects in a folder. Administrators deploy a folder to another Orchestrator, typically in a different environment.
2. Test, debug, and verify all content for transitioning.
3. Export the folder as a content package from the development environment.
4. Import the content package to a production environment.
5. Provide the instructions and training so users can activate schedules, and use forms to start processes on demand.

Test the Process with Related Objects

Testing is essential to deploying the CA Process Automation management package that represents the processes you want to automate. Verify that all components work together as designed before you deploy the content package from the design environment to the production environment.

CA Process Automation has several features for testing processes:

- You can start a process in *suspended* mode in the Design palette and conduct runtime testing. You can start and pause the process instance as it runs.
- You can set *breakpoints* to pause and debug process steps.
- You can run processes in *simulation* mode, which lets you define specific outcomes for a step. Simulation mode returns the simulated results of each step without performing any real actions.

Testing occurs on the following levels:

- Content designers test and verify that each process and the automation objects it uses work properly in the design environment.
- Administrators test the deployment after export and import to verify that each process and the automation objects it uses work properly in the production environment. This process is seamless when administrators create touchpoints in the production environment with the same names as in the design environment. Each member of the touchpoint pair targets agents in their respective environments.

The entire deployment can pass all tests and can be operational while a specific form is still being redesigned and tested. The following terms reflect this difference:

- *Unit testing* as opposed to *integration testing*
- An *internal release candidate build* as opposed to a *customer proof-of-concept build*
- *Development* as opposed to *production*

Periodically retest mission-critical processes that do not run frequently (such as system failovers). Periodic testing verifies that the processes and users stay current and functional in the production environment and prevents incidents from becoming crises.

More information:

Run Processes Interactively (see page 404)
Debug a Process (see page 408)
Simulate Processing of Operators (see page 415)

Generate the Process Documentation

Several roles can generate the process documentation at various times during testing and deployment:

- During design, process documentation is useful for one content designer to hand off the development of an in-progress automation to another designer. The content administrator can also use the process documentation to track progress of the automation process.
- At the end of the design process, content designers can generate the process documentation to reference later.
- During subsequent process planning, content administrators can refer to the process documentation for information about datasets or other reusable components.
- When a new process is imported into the production environment, the production administrator can generate the process documentation to review before running the process for the first time.
- After using an automated process for a while, the production administrator can use the generated production documentation when requesting enhancements. With the process flow diagram, the administrator can easily indicate exactly where in a process a change is requested.

Set the Release Version of the Process and Related Objects

You can set a value for the Release Version attribute when you prepare to deploy release-specific objects to a production environment. The file that you export contains only the selected version of each object being exported. If you export objects in a content package, users cannot modify their release versions after import. All objects are baselined during the import. As a result, users cannot modify the imported version of any object. if production users need to modify an imported object, they must save it as a new version.

Assemble the Process with Related Objects

A folder assembles the automation objects that are related to a process or processes being deployed. A folder object includes objects the process uses (such as datasets) and objects that use the process (such as start request forms).

Deploy a Release Version of the Process with Related Objects

After designers test a process in the design environment, they set a release version on the process object and all related objects. Related objects include all objects that the process uses and all objects that use the process.

An administrator or designer assembles the version of each object to release in a folder. An administrator exports a folder of related automation content to an XML file. The administrator then imports this XML file to the production environment.

Specify Import Instructions

To help facilitate the deployment of the content package, provide import instructions. Write your instructions for production environment administrators and any other process users. Indicate how the process elements must be configured to work in the production environment.

These instructions can include the following sample topics:

- How to Install Any Required Applications or Updates
- How to Set Up User Names, Groups, and Passwords
- How to Set Permissions for Users or Groups
- How to Configure Datasets for the Production Environment
- How to Configure Additional Hardware for Operators
- How to Activate Schedules
- How to Set Up External Triggers
- How to Use a Form to Start Processes On Demand

Chapter 2: Getting Started

This section orients you to CA Process Automation and provides basic information about logging in and becoming familiar with the user interface.

This section contains the following topics:

Browse to CA Process Automation and Log In (see page 32)
Configure User Settings (see page 33)
The CA Process Automation User Interface (see page 34)
Browse Out-of-the-Box Content (see page 50)
Use CA EEM to Change Your CA Process Automation Password (see page 51)
Web Browsers (see page 52)

Browse to CA Process Automation and Log In

The URL you use to access CA Process Automation depends on whether the Domain Orchestrator is configured with one node (nonclustered) or multiple nodes (clustered). You can browse directly to a nonclustered CA Process Automation. For a clustered CA Process Automation, browse to the associated load balancer. You can reach all Orchestrators in the domain by launching the URL to the Domain Orchestrator or to the load balancer for the Domain Orchestrator.

Follow these steps:

1. Browse the CA Process Automation.

   - For secure communication, use the following syntax:
     `https://server:port/itpam`

     Examples:
     `https://Orchestrator_host:8443/itpam`
     `https://loadBalancer_host:443/itpam`

   - For basic communication, use the following syntax:
     `http://server:port/itpam`

     Examples:
     `http://Orchestrator_host:8080/itpam`
     `http://loadBalancer_host:80/itpam`

   The CA Process Automation login page opens.

2. Enter the credentials from your user account.

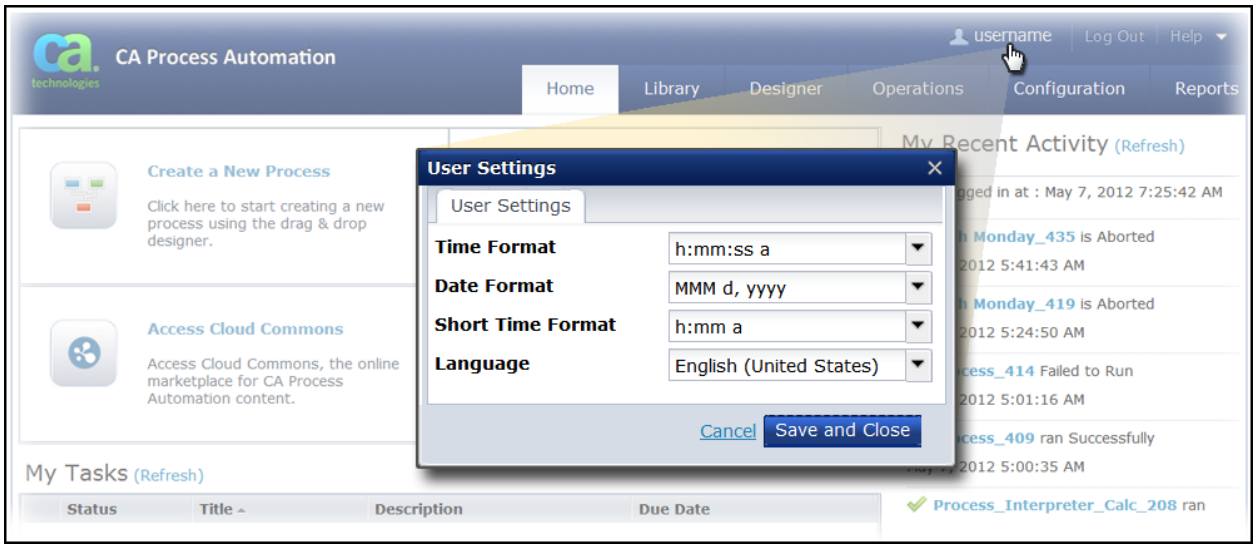
   Note: If CA EEM is configured to reference users from multiple Microsoft Active Directories and CA Process Automation does not accept your unqualified user name, enter your principal name. One format for a principal name is *domain_name**user_name*.

3. Click Log In.

   CA Process Automation opens. The Home tab displays.

Configure User Settings

After you log in for the first time, configure your user interface display settings.



Follow these steps:

1. From the list of links at the top of the main page, click your username.
2. On the User Settings dialog, indicate your display preferences for the date and time formats, and language.
3. Click Save and Close.

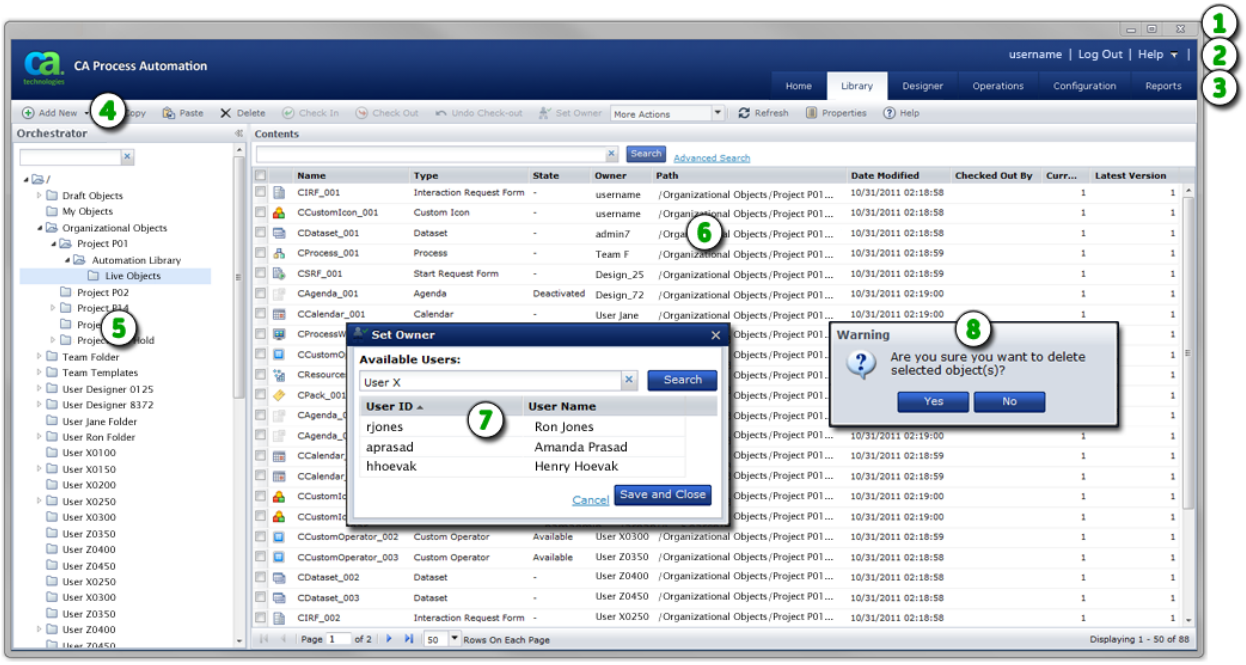
Note: CA Process Automation automatically saves and restores other personalized adjustments that you make as you work. For example, if you dock a palette or customize the columns you want to view in a table or list.

The CA Process Automation User Interface

The CA Process Automation interface provides an integrated development and administrative environment to view, manage, and run all objects in your automation systems. CA Process Automation is a web application that can be opened on any computer with access to the CA Process Automation Orchestrator.

Each major tab at the top of the page presents a unique section or functional area of the application. Common controls throughout the application make it easy to use. For example, you use the same basic steps to sort a list of entries and configure which columns appear.

Equation 1: This graphic highlights how to work with the user interface to perform CA Process Automation functions.



Item:	Description:
1	**OS and Browser Controls**: Although not part of CA Process Automation, your operating system provides controls for working with the current window for tasks such as Minimize, Maximize, Restore, and Close. Your browser also displays its own menus, toolbars, panes, and search areas. You can sometimes use browser features to supplement built-in CA Process Automation features such as refreshing a page or adjusting the view magnification (Zoom).
2	**Links**: CA Process Automation provides common application links including User Settings, Help, and Log Out. Individual pages include appropriate links to related content.

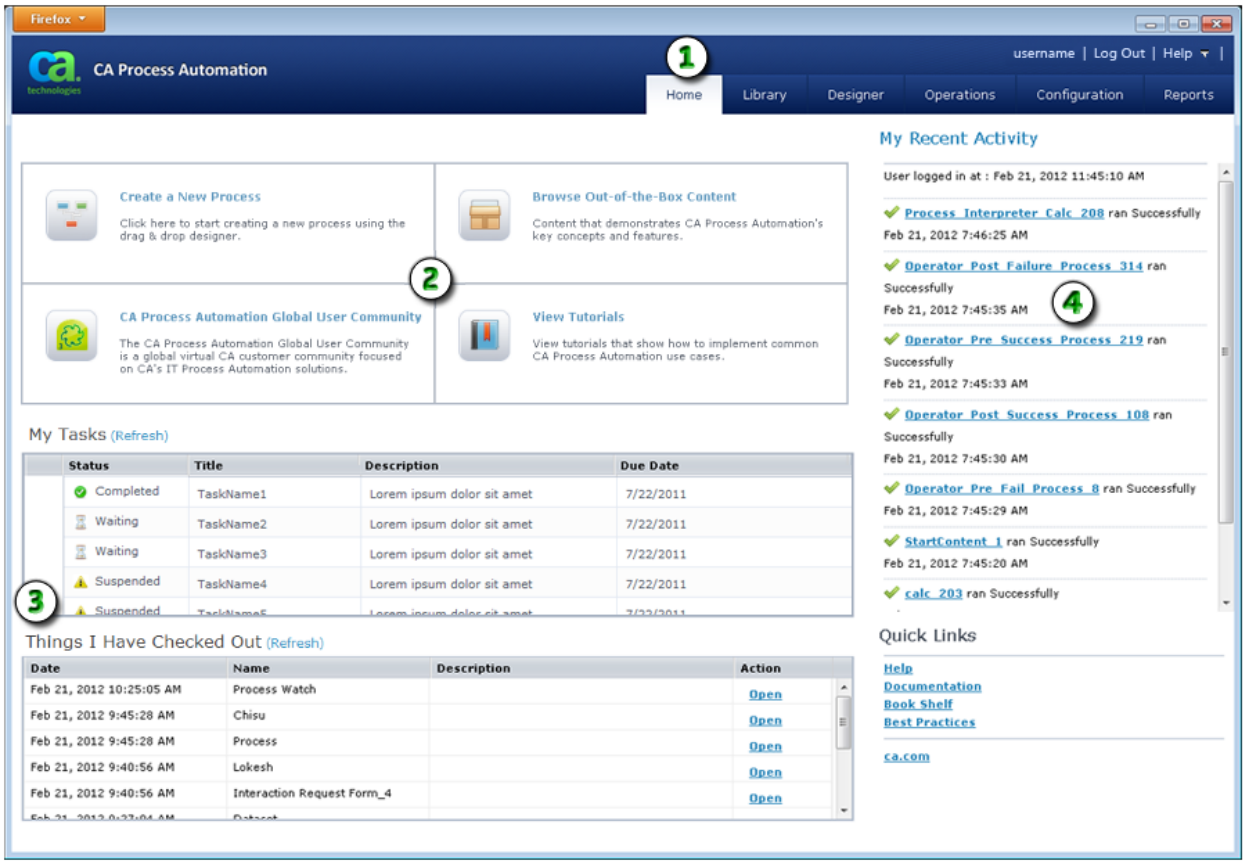
Item:	Description:
3	**Main Application Tabs**: Click a tab to focus on a specific CA Process Automation section. The Library tab is selected in the sample image, so the application displays folders and objects in the Library Browser.
4	**Toolbar**: Many pages and dialogs display specific toolbars with appropriate tool buttons and icons.
5	**Panes**: *Panes* divide a window or page. In this example, the Library Browser pane appears at left and features both a filter input field and an expandable folder hierarchy. The main page displays detailed information about the entry you select from a pane.
6	**Main Page**: The main area of a page displays essential information about an item selected from a pane. CA Process Automation typically presents the data in a table, list, form, design canvas, or chart. As necessary, the main area is further divided into palettes, tabs, panes, or other visual controls.
7	**Dialog**: When you click specific buttons, apply actions, or issue commands, the application often presents dialog boxes to collect additional input.
8	**Message**: Predefined logic and events that you or the system initiate can result in the appearance of messages. Most of these alerts are informative and provide necessary feedback. However, some messages display critical warnings designed to protect your data. Error messages provide useful information that you can combine with log file data to troubleshoot issues.

Main Application Pages

Click any of the six main tabs to navigate within the main pages of the application. As you work, CA Process Automation applies logic and permissions to determine what you can view. At times, CA Process Automation automatically changes your current tab selection, or opens another window. For example, when you open a process in the Library Browser you automatically see the process in the Designer. When you open a schedule object from the Library tab, you see the Schedule Editor window. When you click a process instance from the Operations tab, you automatically see a separate Process Instance window.

Home

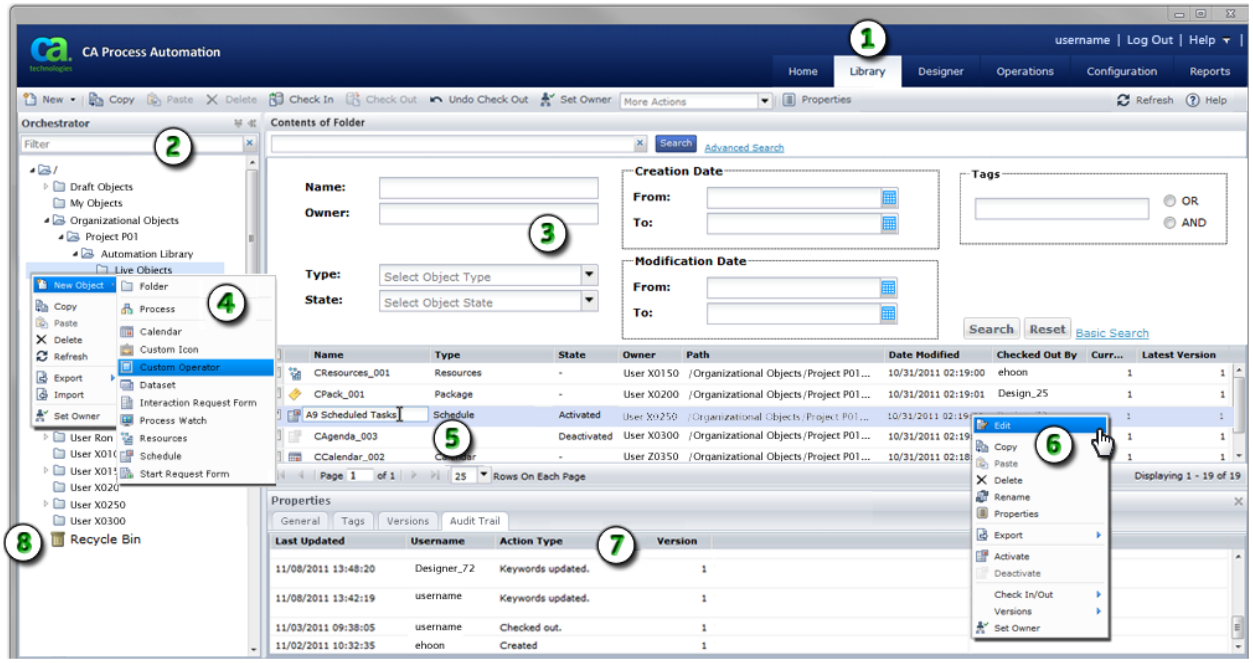
Use this page as a convenient starting point and personal dashboard for your CA Process Automation work session. Available features are highlighted in the following example.



Item:	Description:
1	**Home Tab**: Click the Home tab to view the Home page. The Home page appears when you log in to CA Process Automation.
2	**Quick Buttons**: CA Process Automation provides options to help you save time and start working. The lower right corner of the page also lists Quick Links.
3	**Tasks and Automation Objects**: CA Process Automation displays tables that list your tasks and the automation objects you have checked out (that is, the working versions of objects).
4	**My Recent Activity**: A list of recent process activity appears sorted by date and time. Click a link to open the process instance.

Library Browser

To manage automation objects in folders, use the Library Browser.



Item:	Description:
1	**Library Tab**: Click this tab to navigate to the Library Browser. Use the Library Browser to view, create, edit, or remove automation objects and folders.
2	**Folder Pane**: CA Process Automation displays the orchestrators and folders that you can access in this resizeable pane. Apply a filter to view only matching folders. For example, *Folder_2* would find Folder_2, Folder_200, and MyFolder_2.
3	**Search Area**: Enter criteria to perform a basic or advanced search. For example, locate one or more objects by name, type, or keyword. You can also search for and replace objects.
4	**Create Automation Objects**: In the Library Browser, you can create folders to store automation objects. Right-click a folder to view a menu of commands you can perform in that folder. You can also create new automation objects such as a processes, schedules, or start request forms.
5	**Rename an Object**: Right-click an object and select Rename from the shortcut menu to edit its name.
6	**Edit an Object**: Use the toolbar or right-click an object to view a shortcut menu of commands.

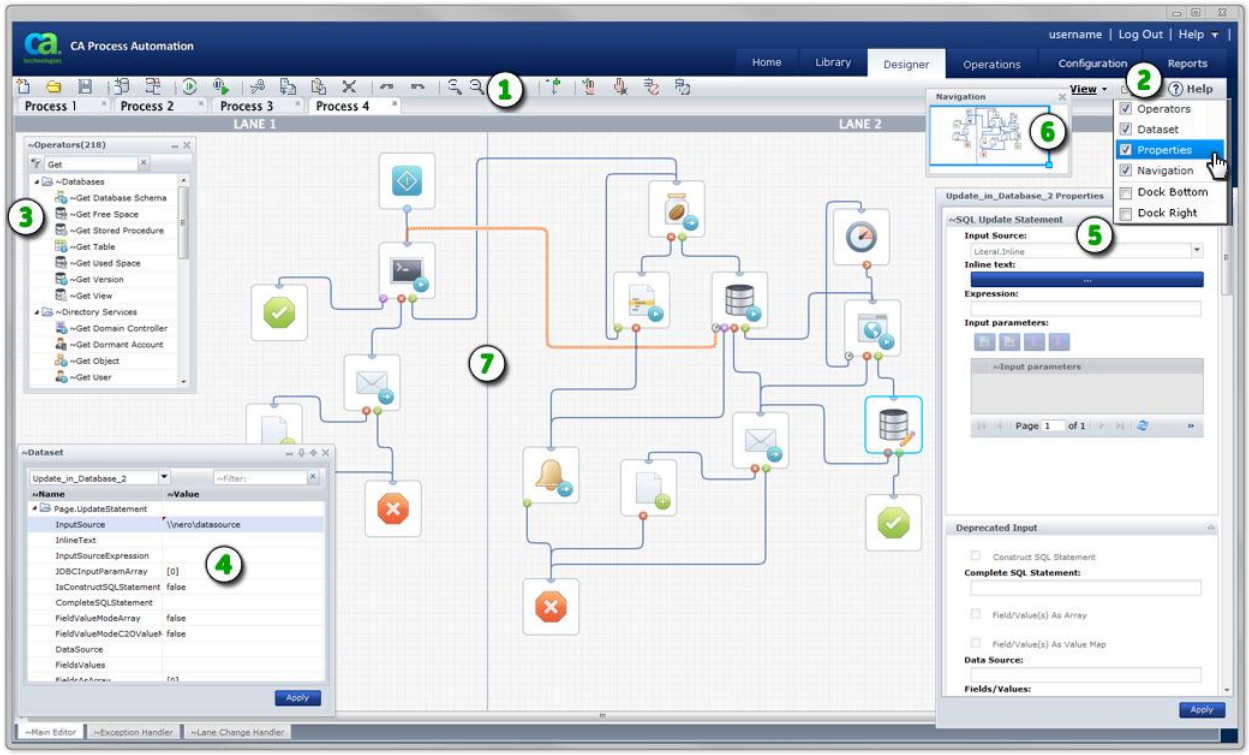
Item:	Description:
7	**Properties**: This tabbed panel displays general data, tags, versions, release, and an audit trail for each object.
8	**Recycle Bin**: Use the Recycle Bin to manage objects and folders that have been deleted. The application lets you restore them, leave them in the Recycle Bin, or permanently purge them.

Note: For information about configuring security for objects and folders, see the *Content Administrator Guide*.

Designer

To design, edit, and test a process, use the Designer page. The Designer supports the following process details:

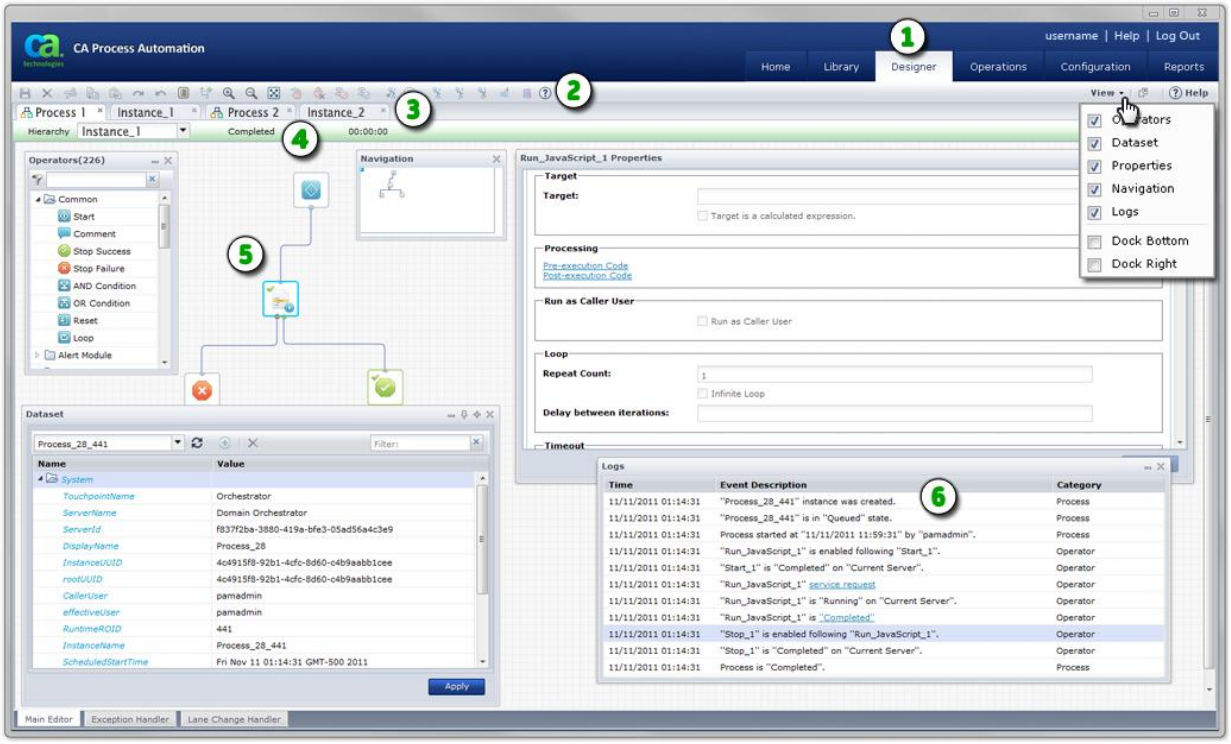
- Basic operator flow
- Documenting the process objects
- Logical results
- Connecting lines
- Browsing datasets
- Defining properties
- Writing code
- Monitoring process instances
- Stepping through, debugging, testing, and controlling process instances



Item:	Description:
1	**Designer Tab**: When you open a new process or edit an existing one from the Library Browser, this tab appears. If this tab is not available, it means you have not yet opened a process or do not have security permissions to open a process.

Item:	Description:
2	**Process Designer Toolbar**: Use the tool buttons to design and test processes. Use the View settings at right to show or hide the Operators, Dataset, Properties, and Navigation palettes.
3	**Process Tabs**: Each process you open appears in its own tab. You can copy and paste objects between tabs.
4	**Process Designer**: The actual process design appears in this work area, canvas, or layout. The process designer includes the grid, lanes, all operators, ports, and connecting lines.
5	**Operators Palette**: Drag and drop operators with specific functions from this palette to your process layout.
6	**Dataset Palette**: Use this palette to view or edit the variables in process or operator datasets.
7	**Properties Palette**: Use this palette and its links and windows to manage the properties for an operator. This example displays Run Script operator properties. The user has also added code in the Post-execution Code dialog.
8	**Navigation Palette**: Use this palette to navigate to specific regions in larger processes. To save time, try panning within this palette instead of scrolling the main designer layout.

You can also run process instances on the Designer page.



Item:	Description:
1	**Designer Tab**: This tab appears when you open an existing process instance from another page in the application. If this tab is not available, you have not yet opened a process instance or do not have security permissions to open a process instance.
2	**Process Instance Toolbar**: Use the tool buttons to stop, start, debug, and test actual instances of a process. Use the View settings at right to show or hide the Operators, Dataset, Properties, Navigation, and Logs palettes.
3	**Process and Instance Tabs**: Each process and each instance of a process that you open appears in its own tab. Process instance tabs appear adjacent to their source process design tab. Process tabs show an icon. Process instance tabs do not show an icon. CA Process Automation assigns a unique process ID number to each instance name to help you identify different instances. You can copy and paste objects between tabs.
4	**Process Instance Status Bar**: This bar displays the status of the instance. When applicable, the status duration also appears. For example, a Waiting instance will also show a live clock indicating how long the instance has been in this state. You can also use the Hierarchy control to focus on specific subprocesses.

Item:	Description:
5	**Process Instance**: The design for the process instance appears in this work area, canvas, or layout. The process instance includes the grid, lanes, and all operators, ports, and connecting lines. Use it to trace the path of your process as it runs.
6	**Logs Palette**: Use this palette to verify or troubleshoot process instances.

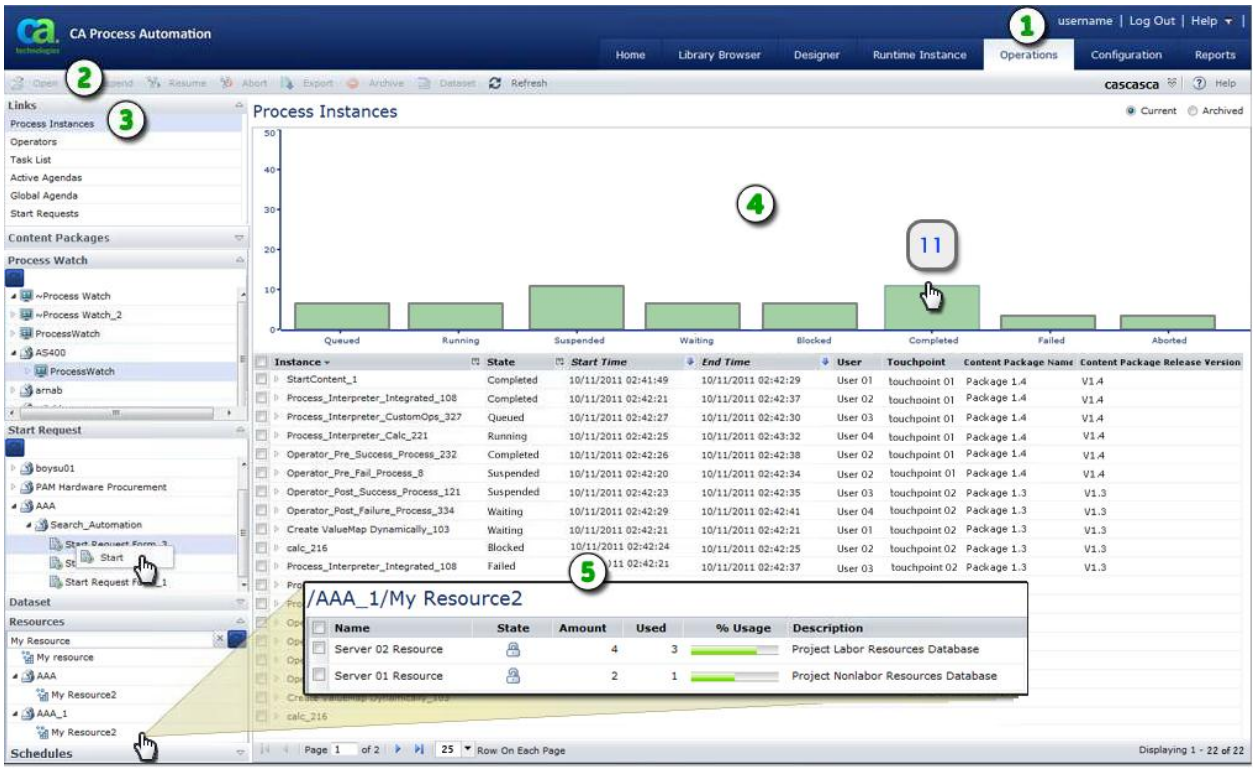
Operations

The Operations page displays a dashboard of the following key elements:

- Links to operators, processes, and other objects
- Content packages
- Process Watch
- Start request forms
- Datasets
- Resources

The Operations page helps you manage running or completed processes. Start the processes on demand and interact with running processes through interactive forms. The Operations page is an automation dashboard that answers the following types of questions:

- What is running in my environment?
- What has been running in my environment?
- Which of the processes are available to start on demand?
- Which of the processes are waiting for user input or approval?

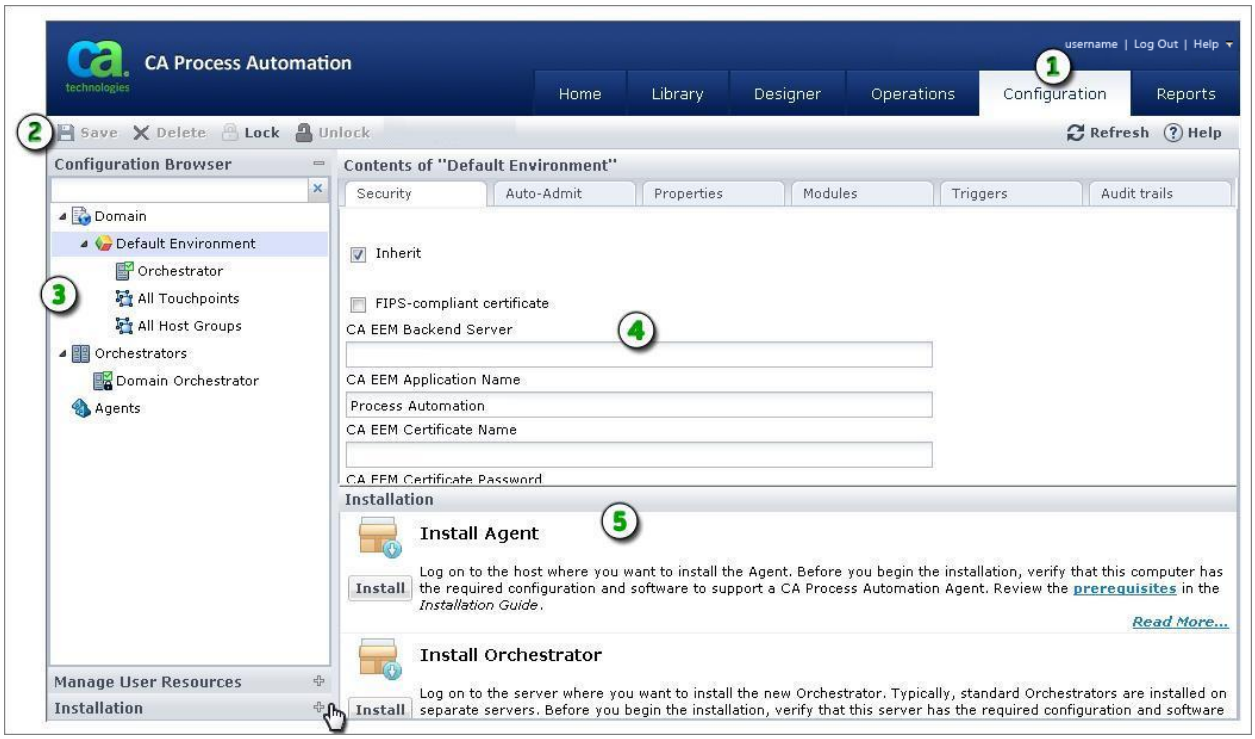


Item:	Description:
1	**Operations Tab**: Click this tab to view the Operations page.
2	**Operations Toolbar**: Use these commands to work with items in the various Operations page lists.

Item:	Description:
3	**Operations Pane**: Select an entry from the expandable list in this pane. For example: ■ Click Process Instances to view a chart and list of instances. ■ Click Content Packages to view a list of the content packages that were imported and the objects they contain. ■ Click a Start Request Form to start it. ■ Enter an optional filter to find a resource object. In the illustration, the user has entered My Resource to find matching entries arranged in folders in the Resources section of the pane.
4	**Graph or Chart Area**: The product displays the data as a graph or chart for process instances, process watches, and process instances that are in content packages. In the illustration, the user is pointing to the Completed instances bar to see the total count (11). Click a bar in the chart to add the associated status value to the filter for the result list below it. Click the bar again to remove the value from the filter.
5	**Main Page or List**: The main area of the Operations page displays essential information. In addition to the chart, the product often presents this data in a table, list, or form. For example, you can view, filter, and sort a list of instances by their state. Click a resource object to view the current status of the resources it represents.

Configuration

The Configuration Browser palette displays the logical hierarchy of the Domain. Administrators use the Configuration tab to install and configure multiple agents and orchestrators. Content designers have read-only access to the Configuration Browser palette on the Configuration tab.

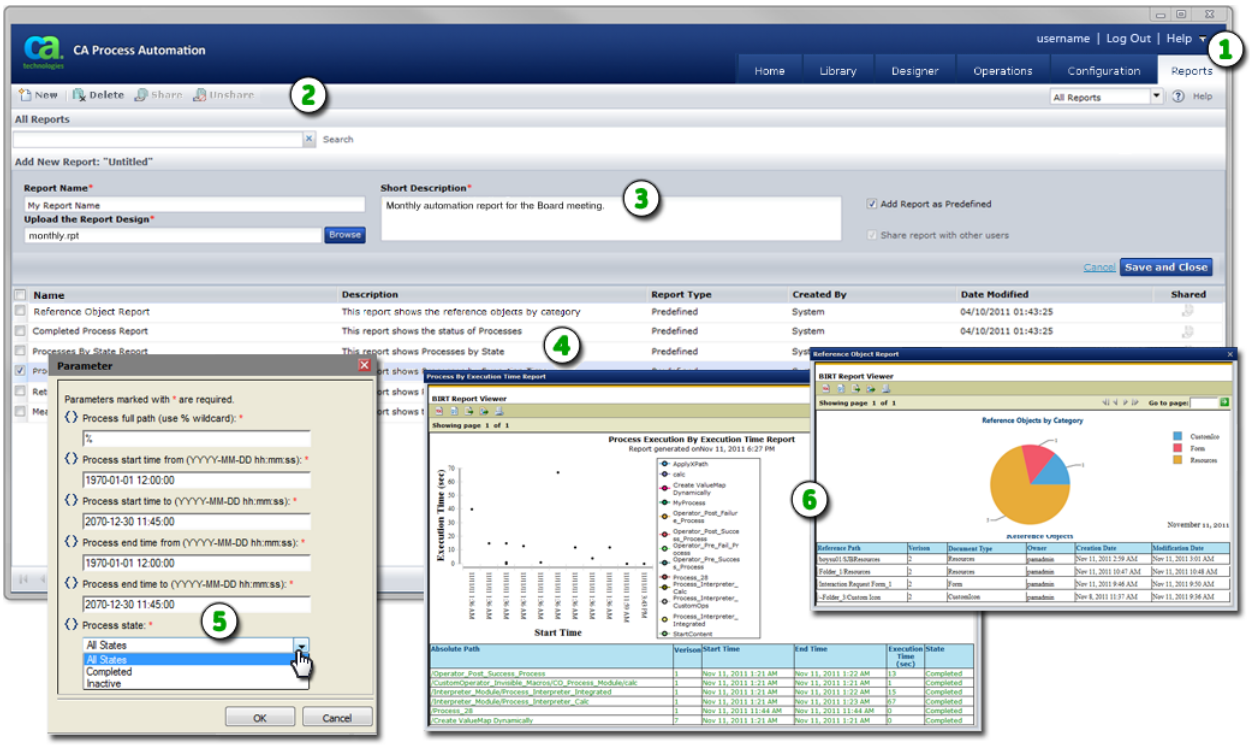


Item:	Description:
1	**Configuration Tab**: Only administrators can use this tab to update configuration settings.
2	**Configuration Toolbar**: Only administrators can lock objects in the Configuration Browser, save configuration changes, and unlock the objects.
3	**Configuration Browser Domain hierarchy**: Only administrators can configure the product.
4	**Configuration Browser Main Pane**: The main pane displays configuration data.
5	**Installation and Manage User Resources**: Only administrators can view these palettes. The Installation and Manage User Resources palettes do not appear when content designers click the Configuration tab.

Reports

Use this page to perform reporting tasks. These include:

- Generate an existing report
- Upload a new report
- Set custom parameters
- Export report data



Item:	Description:
1	**Reports Tab**: Click the Reports tab to view or create reports.
2	**Reports Toolbar**: Use this toolbar to work with the Reports page. A separate standard toolbar is also available for working with report output.
3	**Add New Report Area**: If you click New in the toolbar, this area appears so you can define and save report details.
4	**Reports List**: Lists available reports.

Item:	Description:
5	**Parameter Dialog**: With the exception of the Reference Object Report, when you click a report in the list, this dialog appears. Use it to define the range of data to include in the selected instance of the report. In this example, the user has selected All States from the { } Process State field.
6	**Report Output**: Once you click OK, an on-screen preview of the output appears on the Parameter dialog. You can use this output as is, print it to another device, or export it for use in another application.

Note: To design new reports, install the BIRT designer on a separate application server and connect it to the reporting database.

Common User Interface Controls

The CA Process Automation web interface offers a consistent set of controls. Tables, lists, pages, commands, and filters work in the same standard way from page to page.

Note: Some features are not available on all pages.

Item:	Description:

Item:	Description:
1	**Commands**: As a shortcut, you can right-click a list item to select the same commands that are available on a toolbar. In this example, you can click **Properties** in the toolbar or right-click a task row in the table and choose **Properties** from the resulting shortcut menu.
2	**Pagination**: When CA Process Automation presents tables of data, you can control how much fits on a page. The top and bottom tables in this example both display 170 items. On the top page, the user has selected **200** Rows On Each Page. This condenses all 170 items to a single page. For the bottom page, the user has elected to view 10 rows. This setting increases the number of pages required to display all 170 items to 17 pages. Use the controls at the bottom of the window to navigate to the first, last, previous, or next on-screen page. You can also jump to a specific page by entering the page number in the Page field.
3	**Multi-Select**: Check one or more rows to apply the same command to similar items. The application will only allow you to perform valid commands for multiple items at the same time.
4	**Filters**: Enter an optional word or text string and press the Enter key to view only matching items. For example, type *net* and only the *netstat* and *netstatw* objects appear in the list. Click the X to clear the filter.
5	**Sort Order**: Click in a field column header row to select **Sort Ascending** or **Sort Descending**. As a shortcut, just click the column header to toggle the sort order. A small triangle indicates the sort order direction: ascending (down), descending (up), or unsorted (no triangle).
6	**Column Arrangement**: Click and drag an entire column to change the sequence of columns in a table. In this example, the user has elected to position the Type column in between Name and State.
7	**Column Size**: Position your mouse over the borderline dividing two columns. When it changes to a resize cursor, drag the column edge left or right to adjust the width of the column and its data.
8	**Show or Hide Columns**: You can show (checked) or hide (not checked) the listed fields to control the columns that appear in a table.
9	**Quick Filters**: If available, you can apply a quick filter to a column. Check the field values you want to include and clear the values you want to exclude.
10	**Date-Time Filters**: For date-time fields in a table, you can define a filter as a range or span in time. The filter excludes data not in the range. In this example, the user only wants to include rows with End Time values from November 30, 2012 at 10:00 a.m. to November 30, 2013 at 11:59 p.m.

Browse Out-of-the-Box Content

The Out-of-the-Box Content link on the Home page links the content designer to sample content. Sample content illustrates what you can do with various automation objects.

Follow these steps:

1. Click the Home tab and then click Browse Out-of-the-Box Content.

   The Library tab opens with the PAM_PreDefinedContent folder selected in the navigation pane.

2. Expand the folder that is titled PAM_PreDefined Content.
3. Expand the subfolder that represents the operator, the object, or the design process of interest.
4. In the main pane, double-click an object where the Type is Process.

   The process opens in the Designer tab. The process includes annotations describing the purpose of the process, other relevant information, and directions for customizing the process.

5. Use the selected sample process as a learning tool or as a model for a process of your own.

Use CA EEM to Change Your CA Process Automation Password

The administrator typically assigns a temporary password when setting up user accounts for the internal user store. All CA Process Automation users with user accounts created in CA EEM can change this password before you log in to CA Process Automation. Then, you can change your CA Process Automation password at the interval that password policies define.

Note: This ability does not apply when CA EEM references user accounts from an external user store such as Microsoft Active Directory. In this case, maintain your password within the referenced directory.

Use CA EEM to change your CA Process Automation password.

Follow these steps:

1. Open a browser and enter the URL for the CA EEM server that CA Process Automation uses. For example:
   https://*hostname_or_IPaddress*:5250/spin/eiam/

   To identify the host name or IP address of the CA EEM that CA Process Automation uses, see the CA EEM Backend Server field on the CA Process Automation Configuration tab Security subtab.
2. Log in to CA Embedded Entitlements Manager (CA EEM) from the log In dialog:
   a. For Application, select <Global>.
   b. Delete EiamAdmin if this default name appears in the User Name field.
   c. Enter your CA Process Automation user name and password.
   d. Click Log In.
3. Under Self Administration, click Change Password.
4. Reset your password:
   a. Enter your CA Process Automation user name and old password.
   b. Enter your new password in both the New password and Confirm password fields.
   c. Click OK.

   CA Process Automation accepts your updated credentials when you log in.

Web Browsers

Even among supported web browsers, CA Process Automation may behave differently due to differences in third party products. For example, when you attempt to leave a page or tab with unsaved changes, each of the following supported browsers may display a slightly different message:

- Microsoft Internet Explorer
- Google Chrome
- Mozilla Firefox

Essentially, all three give you the option of remaining on the page to save your data or leaving the page and discarding any changes.

Microsoft Internet Explorer

When you set Microsoft Internet Explorer 8 or 9 to Compatibility Mode (CM), CA Process Automation may exhibit poor performance or behavior. For example, the drop-down menus for the field columns on the Operations page may disappear if you click Start Requests (or another option in the Operations pane) and then go back to Links and choose Process Instances. Once in this state, the column header drop-down menu button may not appear for other views. This may result in the inability to set a filter.

We recommend that you do not explicitly enable CM or Intranet Settings. If you observe odd behavior when using Internet Explorer, ensure that CM is not enabled.

Note: If you must enable Intranet settings, manually disable the *Display intranet sites in Compatibilty View* option in Compatibility View Settings under the Tools menu in Internet Explorer.

Chapter 3: Library Browser

This section contains the following topics:

Customize the Library Browser (see page 53)
Search the Library Browser (see page 53)
Working with Objects and Folders (see page 58)

Customize the Library Browser

You can customize your view of the columns in the Library Browser. Columns display field values for entries in a table.

Follow these steps:

1. Click the Library tab.
2. In the Library Browser folders pane, select a folder.
3. In the main window, to show or hide a column, click in the column header and choose Columns.

   A checklist of available field names appears.
4. Clear the field columns you want to hide. Check the field columns you want to show.
5. To sort the list, click in a specific column header and choose Sort Ascending or Sort Descending.
6. To rearrange the columns, click a column header and drag it to the left or right.

Search the Library Browser

Use the search features to find specific items in the Library Browser. You can then perform actions on multiple related objects in the search results.

Follow these steps:

1. Click the Library tab.
2. In the Library Browser folder pane, select the root folder or a specific subfolder.

   The product limits your search to objects and folders in the selected folder.
3. In the Search field, enter the name of an operator or folder.

4. (Optional) Click Advanced Search to base a refined search on any of the following properties:
   - Name
   - Owner
   - Type
   - Check-In State
   - Creation Date
   - Modification Date
   - Tags
5. Click Search.

   The search results appear.

   Note: If you search in a folder hierarchy with a content package, the product returns only objects in the current release version of the content package. The product also returns objects that reside in the folder hierarchy but are not in the content package.
6. Work with objects and folders in the search results as you would in any other library folder.
   - To open an item for editing, double-click it.
   - To view a shortcut menu of available commands, right-click a single item.
   - To apply an action to multiple objects simultaneously, select the items and then right-click a selected item.
7. To begin a new search, take one of the following actions:
   - Click Reset.
   - Click Basic Search or Advanced Search.
   - Clear or modify the criteria that are specified in the search fields.
   - Click the x in the Basic Search box.

Search for Release Version Information

You can optionally associate each object version with a release version. The Versions tab uses its displayed associations to filter for those objects and versions that are associated with a string you specify. You can search a specific release version to display a version of interest. Alternatively, you can search for a generic string that returns multiple object versions. The Versions tab supports wildcard searches with or without a wildcard character (*), as the following example shows:

Search String	Search Results
5	Any object version in which the Version field or the Release Version field contains the character 5.
*5	
5*	
5	

Follow these steps:

1. Click the Library tab.
2. Right-click an object, and then select Edit.
3. On the object editor, click the Versions tab.

   Note: See the tooltips or the *User Interface Reference* for field descriptions and examples.
4. On the Version toolbar, click the magnifying glass icon on the rightmost side.
5. Enter search criteria in the search field, and then click Search.

   The search returns the objects and versions that are associated with the specified string.

Search for Version Information

The Version tab lets you filter the displayed versions to objects that contain the string you specify.

- For search criteria other than dates, the search results include rows for object versions that contain the specified string in any of the following fields:
  - Version
  - Modified By
  - Created By

  Note: To display objects containing 5, for example, specify 5, *5, 5*, or *5* as search criteria.
- For date search criteria, the search results include rows for object versions that have the date you entered in either of the following fields:
  - Created On
  - Last Modified On

Follow these steps:

1. Click the Library tab.
2. Right-click an object and select Edit.

   The object editor appears.
3. Click the Versions tab.
4. Click the magnifying glass icon on the right-hand side of the Version toolbar.

   The search field and Search button appear.
5. Enter search criteria in the search field and click Search.

   The search process displays objects meeting the specified search criteria. Search criteria other than dates returns all versions containing the specified string in the Version, Modified By, and Created By fields. Date search criteria returns all versions with an exact match in the Created On and Last Modified On fields.

Search for Audit Trail Information

The Audit Trail tab lets you display only object versions that meet one or both of the following search criteria:

- The Version field or the Username field contains the string that you specify in the Filter field.

  For example, if you specify 5 in the Filter field, the search returns any Audit Trail object that has 5 in the Version field or has 5 in the Username field

- The Last Updated field contains a date within the date range that you specify in the From and To fields.

Follow these steps:

1. Click the Library tab.
2. Right-click an object and select Edit.

   The object editor appears.
3. Click the Audit Trail tab.
4. Click the magnifying glass icon on the right-hand side of the Audit Trail toolbar.

   The search fields and Search button appear.
5. Enter search criteria in either the Filter field, the From and To fields, or both. Then, click Search.

   Filter

   Specifies the numeric or alphanumeric search criteria for the following fields, where the search results include any object versions that contain the specified string.

   - Version
   - Username

   From - To

   Specifies date range criteria for searching the Last Updated data.

   The search process displays objects meeting the specified search criteria.

Working with Objects and Folders

Folders are containers for objects.

Use this process to edit an object:

1. Identify the location of an object or folder in the Library Browser.
2. Open the object.
3. Check out the object to a working version you can edit.
4. Edit the object.
5. Save the changes.
6. Check in the object (in this step you also decide whether to apply the changes to the same or a new version of the object).

When you only want to inspect an object, either click Properties or open it. You can then view its properties in detail and decide whether to edit it.

Objects open in an editor or designer that is based on the object type. For example, a process opens in the Process Designer and a dataset object opens in a Dataset dialog. Each editor has specific tabs, palettes, and toolbar buttons.

Note: This section includes only basic information about opening and editing objects and managing versions. Other sections describe in greater detail how to edit various types of objects.

Automation Object Types

The Automation objects are application components that define configurable elements of a CA Process Automation package. These objects define system operations and include runnable software. Create and configure CA Process Automation automation objects in the Library Browser in specific automation library folders that are associated with a specific Orchestrator.

You can take the following actions in an automation library:

- Browse an automation library that is associated with an Orchestrator.
- Create, edit, and view objects in an automation library.
- Create folders in a library to group related objects. Folders let you define a hierarchical structure so you and your coworkers can locate objects. This structure is similar to computer operating system directories or folders.

You can create the following automation object types in the Library Browser:

Process

A *process* object graphically depicts the order of and the dependencies between operators and some other processes. Processes represent operators graphically with links that show the sequence and logic behind the steps that the process performs.

Schedule

A *schedule* object applies date and time conditions to when modules (including processes) run. To group tasks by application, ownership, or other criteria, use multiple schedules.

Calendar

A *calendar* object defines rules that describe complex date conditions. A calendar object graphically specifies dates, time intervals, and conditional elements that determine when and how frequently the product performs operations.

Custom Icon

A *custom icon* object specifies graphic images that uniquely identify operators.

Custom Operator

A *custom operator* object lets you extend the presentation and configuration of existing operators. You can base a custom operator on an existing operator and then optimize it with specific parameters that are designed for reuse in various processes.

Dataset

A *dataset* object defines and groups variables that the product uses as parameters that other processes, operators, and resources require. Examples include application locations, passwords, and profile names. You can easily configure these variables so that the product can update processes and scheduling efficiently to reflect changes in an application environment.

Package

Beginning in CA Process Automation 04.2.00, the *package* object is no longer available to bundle shortcuts to other automation objects. Packages are not available for reference through any of the CA Process Automation object browsers. To export and then import content that is unmodifiable from one environment to another in release 04.2.00, use a content package.

Content Package

The *content package* automation object bundles the CA Process Automation objects that a content designer exported as a content package. The content designer adds the automation objects to a folder and exports the folder as a content package. The production user can view the imported content packages from the Library or from the Operations tab.

Process Watch

A *Process Watch* object lets users define and monitor selected applications in the production environment. A Process Watch object is a collection of shortcuts to other automation library elements. A user can open a Process Watch object to view the state of process instances and other objects. A Process Watch object lets a user monitor operators without necessarily permitting access to underlying objects or data.

Resource

A *Resource* object is a model that represents elements of your system architecture. Use them to synchronize independent processes that rely on common infrastructure elements and to quantify and control access to the specific IT entities. Include multiple resources that represent related entities in a single Resource object.

Start Request Form

A *Start Request Form* object defines shortcuts to let a production user start processes manually. A custom dialog prompts users for the values of parameters that are required to start their associated processes.

Interaction Request Form

An *Interaction Request Form* object lets you prompt users to provide responses in data fields and other user interface controls. Users enter the required information to continue a process. For example, use an Interaction Request Form to prompt a stakeholder to review each step in an approval process.

Create a Folder

Folder hierarchies let you categorize and organize related objects and allow you to apply selected permissions recursively to subordinate objects and folders. To organize and secure groups of objects, you can add folders to structure the library hierarchy.

Follow these steps:

1. Click the Library tab.
2. In the Library Browser folder pane, select the folder where you want to create a new folder.
3. In the toolbar click New, and then choose Folder.

   The new folder appears.
4. Rename the folder.

Note: To set rights and perform other object tasks, the best practice is to organize the objects in folders. Do not create objects at the root level because you cannot manage such objects as a group.

Create an Object

Create new automation objects in specific folders in the library that is associated with an orchestrator. After you create an object, you can edit its properties.

Follow these steps:

1. Click the Library tab.
2. In the Library Browser folder pane, select a folder to store the new object.
3. In the toolbar click New, and select an object type.

   A new object is created.
4. Provide a unique name for the object.

   The application alerts you when the specified name is already associated with another object in the folder.

Note: To set rights and perform other object tasks, the best practice is to organize the objects in folders. Do not create objects at the root level because you cannot manage such objects as a group.

Note: The package object is no longer supported CA Process Automation 4.2.00. A content designer can now bundle the automation objects in a folder and export objects as content package for deployment in a production environment. The usage of the existing package object is kept for backward compatibility. See How to Migrate Objects From a Package to a New Folder to Prepare for Export.

Edit an Object

After you create an object, you can modify its configuration at any time. For example, you can:

- Edit the tags for an object
- Add a resource to a resource object
- Remove an object from a process watch.

CA Process Automation stores multiple versions of an object. By default, when you check out and you edit an object, the application uses the designated *current* version. After you edit it, check in the object. You can specify whether to apply the changes to the same version or a new version when you check in the object. In either case, the application applies the changes by default.

Each object can have an associated release version (see page 424). The release version identifies the specific version of an object to deploy to a production environment. CA Process Automation requires that the object is delivered to the production environment in a content package object (see page 432). You can set a flag that prevents the release version from being modified, and then CA Process Automation baselines the object upon import to the production environment. When you set a release version for an object, the internal object version becomes associated with that release number.

Follow these steps:

1. Click the Library tab.
2. In the Library Browser folder pane, click a folder.
3. Double-click an object.

   The current version of the object opens.
4. If the object is not already checked out, click Check Out.
5. Edit the object.
6. Click Save or click Save and Close.

   The application saves your changes. Because the object remains checked out, you can continue to edit it.
7. Click Check In.
8. Complete the following procedure on the Check In dialog:
   a. In the Check In As field, select New Version or Same Version.
   b. (Optional) Enter comments about the changes made in this version.

c. (Optional) Select the Baseline check box to use this version as a baseline.

d. (Optional) Clear the Current check box if you want a new version but you do not want to mark it as current. The previously current version remains current.

e. Click Check In.

View or Change General Properties of an Object

You can view the general properties of any library object or process.

Follow these steps:

1. Click the Library tab.
2. Select a folder, right-click an object, and then select Properties.
3. On the Properties pane, click the General tab.
4. View the read-only object properties.

   Note: See the tooltips or the *User Interface Reference* for field descriptions.
5. (Optional) Edit the Description field for a checked-out or checked-in object. Click Save to store your changes.

Specify Tags or Keywords for Objects

The Tags tab lets you assign keywords to organize objects according to any meaningful naming system that your organization recognizes. You and other users can then perform an advanced search for objects or folders that use the tags or keywords.

Follow these steps:

1. Click the Library tab.
2. Select a folder, right-click an object, and then select Properties.
3. On the Properties pane, click the Tags tab.
4. On the Tags tab:

   a. Click Edit.

   b. Enter one or more comma-separated text values with no spaces between the list items. For example:

   ```
   testing,security,Project Code Beta
   ```

   c. Click Save.

Change Ownership for Automation Objects

The user who creates an automation object or folder is, by default, the owner. The owner has full control of the automation object or folder. An owner can switch the ownership to another CA Process Automation user.

Note: The CA EEM Environment_Content_Administrator permission grants full control of all automation objects and folders. All administrators who belong to the PAMAdmins group have this permission.

If you enable Runtime Security, then only the process owner (or an administrator) can start that process.

Follow these steps:

1. Click the Library tab.
2. Select one or more objects including folders.
3. Click the Set Owner toolbar button.
4. In the Available Users list, select the user account to set as the new owner. Use search to find matching user accounts.
5. Click Save and Close.

Specify an Archival Policy

After an automated process runs and completes successfully, the product archives the run-time data for reference and metrics. By default, the object inherits Archival Policy properties from the Orchestrator. You can, however, define manually the number of days the server retains completed or failed process instances or schedule objects. You can also define the number of completed and failed process instances that the product retains.

Follow these steps:

1. Click the Library tab.
2. Select a folder, and then select a Process or Schedule object.
3. Right-click the object, and then select Properties.
4. On the Properties pane, click the Archival Policy tab.
5. Clear the Inherit Archival Policy From Orchestrator check box. This task activates the other fields.

   Note: See the tooltips or the *User Interface Reference* for field descriptions and examples.

6. Enter a value for Process History.

   This value defines the interval (in days) that the product retains completed and failed process instances in the archive before automatically purging the records. The product maintains the process history for this interval regardless of the number of instances in the archive.

7. Enter the values for failed and finished instances.

   These values define the number of process instances that the product retains before automatically purging records with the following statuses:

   - Failed status only
   - Finished status

   From the instances that exceed the specified retention limit, the product deletes the oldest first.

8. Click Save.

Specify the Folder Release

A content designer can provide release information for a folder. The Release tab in the folder properties provides the release version of the folder. Specifying the release version for a folder is required to export the folder as a content package. Adding information that describes the folder content can help the content package consumer when the content is imported to a new environment.

Follow these steps:

1. Click the Library tab.
2. Right-click the folder on the right pane of the Library, and then select Properties.
3. On the Properties pane, click the Release tab to display the folder properties.
4. Click the ReleaseVersion attribute Value field.
5. On the Value window, enter the folder Release Version, and then click OK.

   Note: You cannot delete or edit the ReleaseVersion attribute, but you can insert the appropriate release version in the Value column.

6. Click Save.
7. Click Add Property or Remove Property to define or remove more folder properties.

   For example, you can add the following information in the Release tab:

   - Owner who created the content package.
   - Users who can use the content package objects after it is imported to a new environment.
   - Support information.
   - Online help (URL to online help).

Specify ROI Properties

When you set the Return on Investment (ROI) properties for a process object, the values you enter appear in the process instances dataset.

Follow these steps:

1. Click the Library tab.
2. Select a folder, right-click a process object, and then select Properties.
3. On the Properties pane, click the ROI tab.
4. Select the Enable ROI check box.
5. Enter the following values:
   - In the Manual Labor Time field, enter the total staff time (in hours and minutes) required to trigger multiple process operations manually.
   - In the Manual Process Elapsed Time field, enter the total staff time (in hours and minutes) required to run the entire process manually.
   - From the Criticality drop-down list, select the process criticality that appears on the ROI report (High, Medium, or Low).
   - (Optional) In the App/System Group Name field, enter the common group name through which the product aggregates processes in the ROI report.

   Note: See the tooltips or the *User Interface Reference* for field descriptions and examples.
6. Click Save.

Determine When to Select Run as Owner

CA Process Automation provides fine-grained access control to operations on specific automation objects such as processes, datasets, and schedules. Access control includes not only traditional read and write rights but also rights to start a process and monitor its instances. Access to other objects such as child processes, datasets, or calendars is also controlled. The access rights are enforced at all external interfaces, including the UIs and Web Services.

Consider the following example with these assumptions:

- Process P1 has a child process P2.
- Owner X owns P1 and P2.
- P1 requires access to other elements that Owner X does not want to expose directly to other users.
- The default access is that a process runs as the user who starts the process.

If User A starts a P1 instance, P1 can start P2 only if User A is authorized to start P2 directly. To protect automation objects from exposure, Owner X can require process P1 (or any of its child processes or operators) to run as owner, that is, to run under the identity of Owner X. When Owner X configures a process to Run as Owner, owner X is authorizing other users to run process P1 without granting them access to the elements P1 needs.

Similarly, Owner X can force operators to run under the identity of the user who started the process instance. Although process P1 is configured to Run as Owner, Owner X can force the child process P2 to run under the identity of the user who directly or indirectly starts process P1. In this case, CA Process Automation uses the identity of the originating user to validate access to child process P2.

Consider this scenario: A DBA user creates a database administration process that requires externalizing the DBA credentials to a named Dataset. The DBA configures this process to run as the owner. Run as Owner grants instances of the process access to the dataset that stores the DBA credentials. The DBA authorizes selected users to execute a specific DBA process, without granting these users direct access to the DBA credentials.

In summary, a running process instance has two related identities:

- The real user who starts the process instance from the UI, an agenda, or a web service.
- The effective user who is determined by how Run as Owner is applied to process definitions and operators in those processes.

  Note: The identity of the current, effective user is used to validate access to child processes, operators, and datasets.

More information:

Guidelines for Setting Runtime Security for a Process (see page 399)

Specify Run-Time Security Properties

You can specify run-time security properties for both Process and Schedule objects. These properties establish whether and how the system verifies user permissions when users attempt to access automation objects from the database.

Follow these steps:

1. Click the Library tab.
2. Select a folder, right-click a Process or Schedule object, and then select Properties.
3. From the Properties pane, click the Runtime Security tab.

   Note: See the tooltips or the *User Interface Reference* for field descriptions.
4. Under Runtime Security, perform one of the following tasks:
   - Inherit enabled security settings from the Orchestrator, if applicable. This value is the default.
   - Select **Enable** from the drop-down list. This option lets the product verify user permissions when accessing objects in the process.

     The Process objects are typically designed to access other objects to read or run while it runs, such as when one process triggers another process. If it is enabled, the product uses run-time security to verify that the user (the owner or caller) has permissions to access the object. For example, User A has permissions to start a process. This process attempts to access Object B and Object C. The product cross-checks permissions for each object. If the run-time security determines that the user does not have permissions to Object B, the process fails.
   - Select **Disable** to disable the validation and enforcement of process ownership at run time. This option provides backward compatibility for existing processes and lets these processes run as they did before this release.

5. Complete the run-time security configuration for your Process objects using the following options. Maintenance of the Run as Owner check box requires that you either own the process or are an Environment Content Administrator.
   - Run the process as the user who started the process instance.

     a. From the Runtime Security field, click Enable.

     b. Clear Run as Owner.

     This option enforces run-time security rights with the identity of the user who starts the process instance.
   - Run the process as the owner.

     a. From the Runtime Security field, click Enable.

     b. Select Run as Owner.

     This option enforces run-time security rights with the identity of the owner, regardless of who starts the process instance. This allows users to start a process that gains access to elements that the owner wishes to protect.
   - Select the Enable Operator Recovery check box to recover automatically specific operators that fail with a SYSTEM_ERROR.
6. Click Save.

More information:

Guidelines for Setting Runtime Security for a Process (see page 399)

Define the Run Duration for a Process

You can define the expected run duration for a process. If the process run time exceeds the defined duration, the Operations tab displays a red (warning) indicator.

Follow these steps:

1. Click the Library tab.
2. Select a folder and then select a process object for which to define the run duration.

   Note: If the appropriate process object does not exist, create it. For more information about creating a process, see Designing Processes (see page 87).
3. Right-click the process object and then select Properties.
4. On the Properties pane, click the Duration tab.

   Note: See the tooltips or the *User Interface Reference* for field descriptions and examples.

5. Select the Enabled check box.

   The product enables the Duration tab fields.

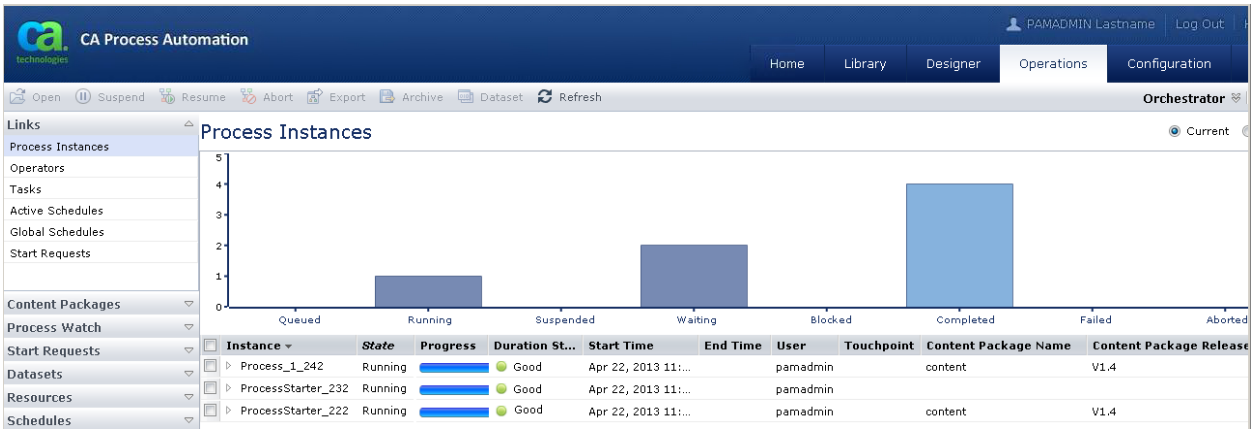
6. Set the expected process duration.

7. Set how long before the process run exceeds the expected duration that the product warns the user.

8. Click Save.

The Duration Status displays one of the following status indicators for the associated runtime process instance:

- **Red:** The process run has exceeded the expected duration.
- **Yellow:** The process is running but is nearing end of the expected duration.
- **Green:** The process ran during the expected duration. The status is green while a process is completed and while a process is running and not in yellow or red status.
- **No indicator:** The process has no expected duration set.

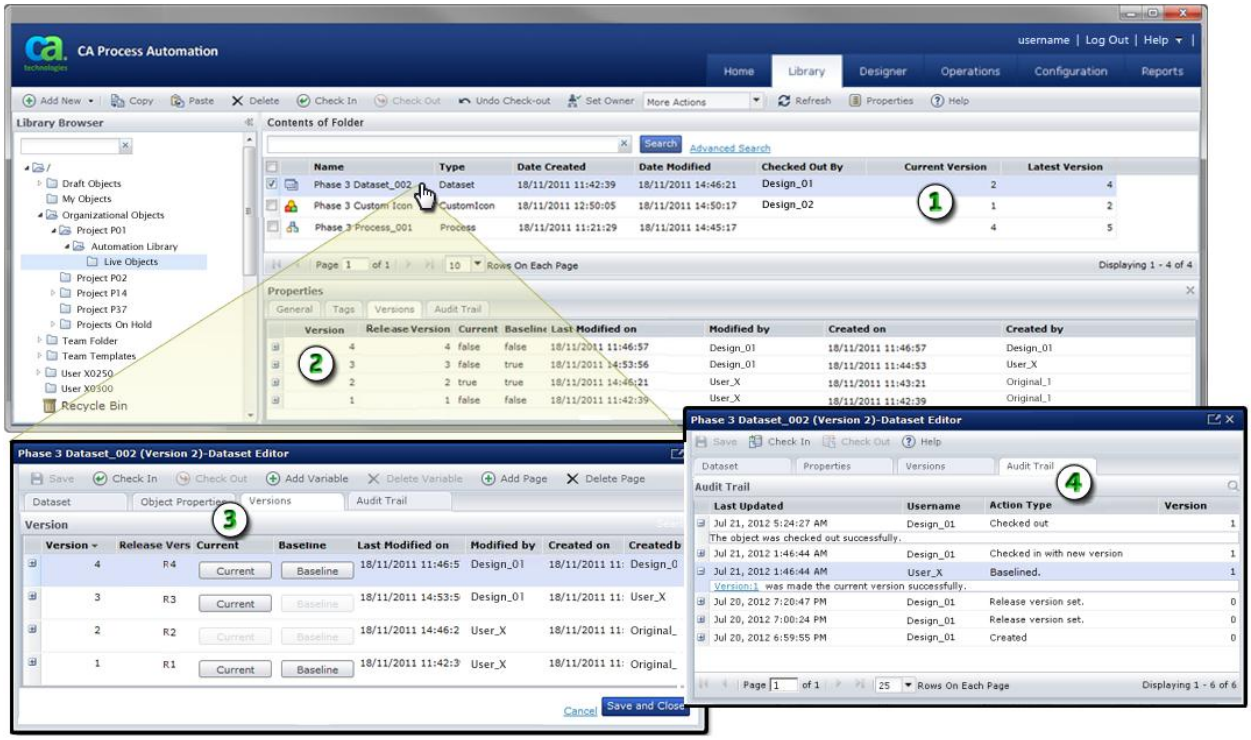


Versions

CA Process Automation uses a version control system that tracks your changes to the objects in your library. The version tracking system always maintains a single *current* version for each object. When you open an object, the application automatically uses the object that you designated as the current version. The application retains other versions of an object for backup or archive. You can open archived versions of an object to view, edit, or designate as the current version. The object owner or another user with sufficient permission can change which version is the current version.

You can check out then edit any version of an object. When your edits are complete, you can check in changes either to that version or to a new version of the object. To prevent users from changing a specific version, you can designate it as a *baseline* version. A baseline version can be used like any other version. You can also check out a baseline version and edit it, but you can only check in the object as a new version. Baseline versions are a benchmark or template for further development.

To release a new version of a process with all of its components, set the release version attribute for the process and each associated object. The release version (see page 424) typically identifies a version of an object that you deploy to a production environment. CA Process Automation maintains the release version attribute that was set before export in the import environment. You set a release version attribute for each object, add each object to the folder, and initiate the export of the folder. If you export the folder as a content package, the release version is nonmodifiable in the production environment. The import process automatically baselines the release version of a content package to prevent users from checking in changes in the imported version.



Item:	Description:
1	**Object Versions**: For each object, the library browser displays the user who checked out the object and the current and latest object versions. In this example, the current version of the selected dataset is 2, which is the version that user Design_01 checked out. The latest version of the object is version 4.
2	**Versions Tab of the Properties Panel**: The Versions tab of the Properties panel lists the original and all subsequent versions of an object. You can see which version is current, which version or versions are baselines, and creation and modification information. The Versions tab of the Properties panel also provides an editable Release Version field. Optionally, you can take the following action: 1. Add a release-specific value to the Release Version field as part of preparing for export. 2. Select this version of the object for export. **Note:** If you exported the object as part of a content package, then the Release Version field is nonmodifiable once the object is imported into a new environment. A tooltip on the field indicates that it is locked.
3	**Versions Tab in an Object Editor:** When you open or edit an object, the Versions tab lets you select a single current version and one or more baseline versions. The other information is the same as the information displayed in the Properties panel.

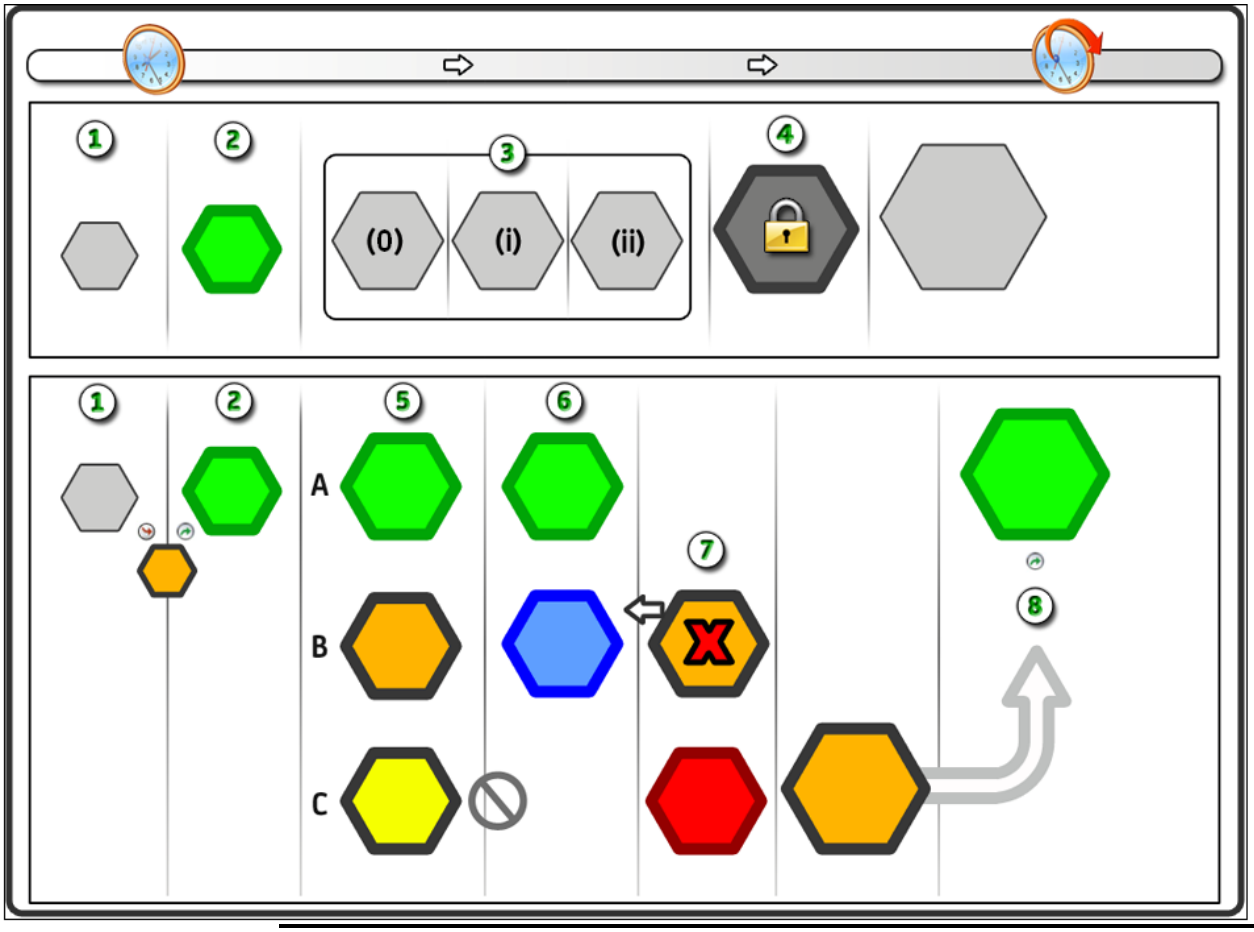
Item:	Description:
4	**Audit Trail Tab in the Object Editor**: The Audit Trail tab provides a history of all changes to each automation object.

Understanding Versions

Refer to the following graphics and examples to learn the essential concepts about working with versions.

Symbols:

- Object version.
- Working version.
- Open a previous version.
- Version checked out by user 1.
- Current version.
- Discard working version.
- Baseline version.
- Latest version checked out by user 2.



Item:	Description:
1	**Typical Versioning Sequence**: A new automation object begins at version 0. The object is checked out as working version 0, with the option of applying pending changes to the same version 0 or new version 1.

Item:	Description:
2	**Current Version**: In this typical sequence, working version 0 is checked in as new version 1. The new version is also typically designated as the *current* version. The current version is the one that is used when the process actually runs. The original version 0 is now considered just a backup or archive.
3	**Changes Applied to the Same Version**: In this example, version 2 is checked out multiple times. Each time, the user has elected to apply the changes to the same version rather than create a new one. Although changes were made in working versions 3(0), 3(i), and 3(ii), the user has decided to group them all together into the same single version. The user cannot go back to view the object in between changes 3(0) and 3(i). No separate version exists to go back to.
4	**Baseline Version**: Version 4 has been designated by the user as a baseline version. Any checked-out working version must be checked in as a new version, version 5. When you specify one or more baseline versions, you are essentially locking each version. This practice prevents any modifications to the actual baseline versions. They can only be used as templates or ancestor class objects to facilitate the creation of new versions.
5	**Current, Working, and Open Versions**: current version 5 has three simultaneous branches users might follow. In branch A, its changes are applied as new and current version 6. In branch B, working version 5 can be checked in as new version 6, with version 5 remaining as the current version. In branch C, the user elects to open version 5. The user can view the version but cannot save any changes to it.
6	**Multiple Users**: While running processes use current version 6, it is possible for the object to also be checked out to user 1 with pending changes in working version 7. While running processes use current version 6, the latest changes to the object may be made by user 2 and reflected as *latest* version 7.
7	**Discard Working Version**: The changes to working version 7 can be discarded. Similar to an Undo Checkout operation, this rolls back the object to the last checked in version.
8	**New Current Version**: The latest version 7 can be checked out as working version 7. During check-in, the user can elect to make it the new current version 8.

Set the Object Version as Current or as a Baseline

The Versions tab displays information about the versions of an object. You can perform the following tasks on this tab:

- Determine when and by whom a specific version was created and last updated
- Determine which is the current version of an object and, optionally, designate a different version as current
- Determine which versions are baselined and, optionally, designate one or more versions as baselined

The product initially designates an object that you create as (working) Version 0. When you save the object and you check it in for the first time, the product designates it as (current) Version 1. From this point on, the product designates one version as current and uses this version whenever the process runs. The product starts the current version by default when you open the object from the Library. By default, modified versions of objects assume a status of current upon check-in. Optionally, you can use the Current button to assign the status of current to any previous version. This option lets you create a version of the object that the product does not access or open by default. You do not have to check out an object to set the current version. Only the object owner or a user in the Environment Content Administrator role has permissions to designate a version as current.

You can baseline an object version to lock it from further updates. Users can use the baselined version as a template from which to create a checked-in, newer version. Consider baselining a version when you do not anticipate making more changes to the object. For example, it is a good practice to baseline objects before you package them for export. The version that you baseline can be the current version or another version.

Follow these steps:

1. Click the Library tab.
2. Select a folder, right-click an object, and then select Properties.
3. On the Properties pane, click the Versions tab.

   Note: See the tooltips or the *User Interface Reference* for field descriptions and examples.
4. (Optional) Click the active Current button of a version to set it as the current version and deactivate the button.
5. (Optional) Click the active Baseline button for one or more versions to set them as baseline versions and deactivate the button.

   Note: You cannot revert setting a version as a baseline. If you update a baselined version, you must save it as a new version.

Set the Current Version of an Object

The current version of an object is the default version used by CA Process Automation in development or production. You do not have to check out an object to set the current version, but you must be the owner of the object or a member of the Environment Content Administrator role.

Follow these steps:

1. Click the Library Browser tab.
2. Double-click an object.

   The object opens in its editor.
3. Click the Versions tab.
4. In the row for a specific version, click the Current button.

   The disabled Current button indicates that the selected version is now current. You can directly change the current version by clicking the Current button in any row.

Open the Current or Working Version of an Object

When you double-click an object in the library that you have checked out, the working version opens. The working version will display the changes you make. If the object has not been checked out, the current version will open in read-only mode. Check out the object to edit it.

Open a Selected Version of an Object

You can open a selected version of any object to view or edit it instead of opening the current version. For example, you can open a previous version or open a new branch of the object with your latest changes.

To open a selected version of an object

1. Click the Library Browser tab.
2. In the Library Browser folders pane, click a folder.
3. Right-click an object and choose Versions and then Open a Version.
4. In the Edit Version dialog, click a version of the object and then click Open.

   The object opens in its editor.

Check Out an Object

To edit and save changes to an object, check it out. Checking out an object allows you to edit versions of the object while preventing other users from changing it at the same time. You can open and view an object without checking it out, but you cannot edit it.

You can check out one or more objects before you open them.

Follow these steps:

1. In the Library Browser, select one or more objects.
2. Click Check Out.

   You can now double-click the checked-out object to open and edit it.

You can also check out a single object after you open it.

Follow these steps:

1. In the Library Browser, double-click an object.

   The object opens in its editor.
2. From the object editor or the Process Designer, click Check Out.

   You can check out, edit, check in, test the changes, and then check out and continue editing the object.

Save Changes to a Checked-Out Object

Saving the working version of an object prevents you from losing changes as you work. For example, you could inadvertently close the object editor. Saving changes for a checked-out object affects only the working version of the object. A new version of the object is not created and the object is not checked in.

Follow these steps:

1. Check out an object (see page 78).
2. Edit the working version.
3. Click Save.

Once changes are saved to the working version, you can close the editor. The object is still checked out. The working version remains editable and continues to reflect your changes as long as it is still checked out. When you open the object again, the application automatically opens the working version.

Discard Changes to a Checked-Out Object

After you check out, modify, and save an object, you may decide that you do not want to continue working with that specific version. Even if you saved the working version numerous times, you can perform an "Undo Checkout" to discard the saved working version.

Follow these steps:

1. Click the Library tab.
2. Right-click the object in the main window and select Check In/Out.
3. Select Undo Check Out.

   The application ignores the changes that you saved to the working version and checks in the object.
4. To continue working with the object, check it out again.

Check In an Object

When you complete your changes, check in the object to save them to the same or a new object version.

After you complete the following procedure, the Designer remains open and you can continue viewing the object in read-only mode. Check out the object again to make more changes.

Follow these steps:

1. Check Out an Object (see page 78).
2. Edit the object.
3. Save Changes to a Checked-Out Object (see page 78).
4. On the object editor toolbar or Library Browser toolbar, click Check In.

   The Check In dialog appears with the object name in the Object Name field.

5. To specify how to check in your changes, configure the following settings:

Current

Selected: Make the checked-in version the *current* version of the object. The current version is the version that CA Process Automation uses.

Cleared: Retain the previously current version as the current version. Clear the check box if you are checking in an object for which design is in progress.

Baseline

Selected: The object version that you are checking in cannot be changed during a future checkout. Baseline versions can only be the basis of new versions. You can check out a baseline version, change it, and check in the object as a new version.

Cleared: The object version that you are checking in can be changed during a future checkout.

Version

Displays the object version that you are checking in.

Release Version

Defines a unique indicator for an object version that is selected for export as part of a release.

If the Release Version field is read-only, the object was imported previously with the release version in nonmodifiable mode. The tooltip provides details.

Enter a value for Release Version only when you are preparing the object version for export (alone, in a folder). Enter a release-specific value so that there is no conflict with any release version value for an existing object in the import environment.

Note: If a conflict occurs, the import fails. Specifically, the import fails under the following conditions:

- The object is imported with the option Import as a New Version and Keep Existing Object.
- The existing object has a version that has the same Release Version value as the value you enter in the Release Version field.

Comments

(Optional) Enter descriptive comments to save with this version of the object.

Check In As

Specifies whether to create a version of the object or apply the changes to the same version.

New Version: Creates a version of the object.

Same Version: Overwrites the current version of the object with changes made since it was last checked out.

6. Click Check In.

Note: If you try to check in changes for a baselined object, CA Process Automation displays an error message.

Understanding Baselines

The purpose of baselining an object is to lock a version.

Consider the case where you are editing some version of an automation object and saving the changes. After you complete all the planned changes, you baseline the current version of the object to prevent future changes to that version. You can check out a baseline version, modify it, and save the changes, but you cannot check in the object as the same version. You must check in the changes as a new version.

For example, assume you baseline version 0 of a process, then continue as follows:

1. You check out version 0.
2. You edit version 0.
3. You save version 0.
4. You check in version 0.
   - If you check in the changes as New Version, CA Process Automation creates the version successfully.
   - If you try to check in the changes as Same Version, the product displays the following message:

     ```
     You checked out a baseline version. To check in your changes,
     you must select New Version.
     ```

   You cannot overwrite a baseline version by checking in changes to the same version.

Consider baselining a version when you do not anticipate making more changes to the object. For example, it is good practice to baseline objects before you export them in a content package. You can equate the packaging process to a release process. So, before you release an object, you can lock it by selecting baseline at the final check-in. You can still modify the object later as long as you check in your changes to a different version.

Note: During an export of a folder as a content package, CA Process Automation exports Release Version attributes in nonmodifiable mode. The import process baselines all objects so that users in the target environment cannot modify the exported object versions.

Create a Baseline Version of an Object

To prevent users from changing a particular version, you can designate it as a *baseline* version. Users can check out a baseline version and edit it, but can only check it in as a new version.

You do not have to check out objects to set them as baselines, but you must be the owner or have administrative permission to the object.

Follow these steps:

1. Click the Library Browser tab.
2. Double-click an object.

   The object opens in its editor.
3. Click the Versions tab.
4. In the row for a specific version, click the Baseline button.

   The selected version is now a baseline indicated by the disabling of the Baseline button. You can select multiple baseline versions of an object.

Note: You cannot undo or reverse the baseline status of a version. If you want to reset a version so that it is no longer a baseline, set it to current, check it out, and then check it in as a new version with the Baseline check box not marked.

Delete or Restore an Object or Folder

You can delete an automation object, an empty folder, or a folder that contains objects that you no longer need. The application moves deleted items to the Recycle Bin.

If you inadvertently delete an object you need (or a folder containing an object you need), you can restore that object with its folder structure.

Note: Only an administrator can permanently purge objects (and their folders) from the Recycle Bin.

Follow these steps:

1. Click the Library tab.
2. Select one or more objects or folders.
3. Click Delete.

   To prevent immediate loss of data, the items automatically move to the Recycle Bin.
4. In the Library Browser folders pane, click Recycle Bin.
5. (Optional) Sort by Date, Name, or Type, or use the search features to locate specific items.
6. To restore objects or folders, select them and click Restore Selected.
7. To delete objects or folders permanently, select them and click Purge Selected.

   Important! After objects or folders are purged, you cannot recover them. When you select folders or a set of objects for purging, the purge process lists any checked-out objects. The administrator reviews the list. The administrator then continues or cancels the purge based on analysis results.

Copy and Paste an Object or a Folder

You can copy objects or folders. You can use a current object version as the basis for similarly configured objects on the same Orchestrator. Copy a current object version to the same or a different folder where you can rename and edit it as a new object. For example, you can configure a custom operator object and then add copies of it within a library.

Follow these steps:

1. Click the Library tab.
2. Identify the objects to copy in the main window. Verify that the version you want is marked as the current version.

3. Select one or more source objects or folders and click Copy.
4. In the Library Browser folder pane, click a destination folder in the same automation library, then click Paste.

   Notice the following results:

   - CA Process Automation adds the copied objects with the same names as the original objects.
   - Each object has one version, the version that was copied.
   - If the copied version has a Release Version value, the copy retains that value.
   - If the copied version was baselined, the copy is baselined.
   - If this location is the same as the source folder, then CA Process Automation appends a number to each object name (for example, Process_1).
5. (Optional) Rename the new objects.

Cut and Paste an Object or a Folder

You can cut objects or folders and then paste them to the same or a different folder. For example, you can cut a custom operator object and then paste it within the same library.

Follow these steps:

1. Click the Library tab.
2. Identify the objects to cut in the main window.
3. Select one or more source objects or folders and click Cut.
4. In the Library Browser folder pane, click the destination folder in the same automation library and click Paste.

   If the names of the objects to paste exist at the target location, select objects at the target location to replace with the objects you cut.

How to Work with Nonmodifiable Content

CA Technologies can release nonmodifiable content. The not icon indicates that the content is nonmodifiable:

The use of nonmodifiable objects is different from the use of modifiable objects in the following respects:

- You cannot modify the imported version.
- You can copy and paste the imported version to another folder and then modify the copy.
- You cannot modify the release version of an imported object.
- You cannot modify the release version of a copy of the imported object.

Action-level Details on the Use of Nonmodifiable Content

The following actions are permitted on nonmodifiable content:

- Copy the object with a new name and paste it in a different folder.
- Cut the object and paste it in a different folder.
- Delete the imported object.
- View the object properties.
- Perform any actions shown in the More Actions drop-down list for imported objects.

The following actions are not permitted on nonmodifiable content:

- Edit the object.
- Rename the object.
- Check in or check out the object.
- Open the object version for edit.
- Edit the object properties.
- Set the owner on the object.
- Export the object.

If nonmodifiable content is exported in a folder, the following actions are not permitted in the folder:

- Import
- Create object
- Paste

Chapter 4: Designing Processes

CA Process Automation process objects graphically represent operators, ports, links, logic, and constraints. Each process contains one or more chains of operators that you can run in a sequence or in parallel. Lanes separate parent processes from any triggered secondary processes. For example, a branch of a process that switches to a subprocess is often depicted in a separate lane. Exception handlers control operators that abort or terminate due to system errors or unidentified exit conditions.

Each process defines the configuration and management of operators on touchpoints in an environment. Process objects are stored with other objects in a library that is associated with an orchestrator. You can create process objects in the Library Browser or directly in the Designer. You also open them from the Library Browser. However, you view and edit processes in the Process Designer.

This section contains the following topics:

The Process Designer (see page 88)
Operators and Links: The Building Blocks (see page 90)
Create a Process Object (see page 91)
Design a Process (see page 92)
Process Operators (see page 93)
Process Operator Ports and Links (see page 100)
Process Loops and Iterations (see page 105)
Process Control (see page 116)
Process Lanes (see page 123)
Process Versions (see page 126)
Document a Process (see page 127)
Self-Contained Content (see page 129)
Navigate to a Specific Part of a Process (see page 132)
Multi-Tenancy and CA Process Automation (see page 132)
The CA Process Automation Code Editor (see page 134)

The Process Designer

The Process Designer provides an integrated development environment where you can drag and drop operators and links to design processes. The Process Designer also provides property and dataset configuration, testing, and debugging capabilities.

Use the Process Designer to:

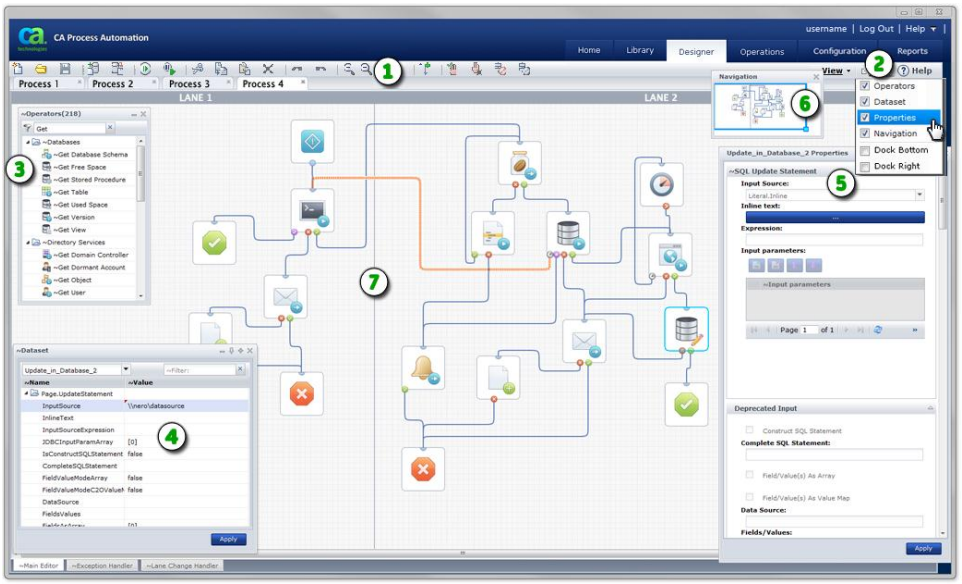
- Design and edit processes.
- Run, monitor, and control the execution of processes in production or test environments.
- Modify a running instance of a process to recover from an incident.

Add an operator by dragging it from the Operators palette to the design layout. You can also directly view and configure the properties or dataset values for an operator or process.

The process designer layout includes the following process elements:

- **Canvas**: Drag an operator to view guides to help your operator snap to positions in the grid.
- **Lanes**: The example process shows two lanes to arrange segments. You can add more, merge, or remove lanes. You can also refer to lanes as *swim lanes*.
- **Operators**: The functional entities within a process. Each operator except for those operators at the end shows its exit ports and other smaller status icons.
- **Ports**: These small connection points represent the exit ports for each operator. Every operator except the Start operator also has a single entry port.
- **Links**: These lines connect the exit port from one operator to the entry port of another operator. You can customize the appearance of these lines.
- **Handler Editors**: In addition to the Main Editor, the designer also includes two other tabs along the bottom for editing exception and lane change handlers.

The following graphic depicts the Process Designer.



Item:	Description:
1	**Designer Toolbar and Process Tabs**: When you open a process from the Library Browser, the Designer tab appears. Each open process appears in its own tab. Use the toolbar buttons to check out, edit, zoom in or out, save, test, and check in the selected process object. You can copy and paste operators from one process tab to another. The toolbar also features icons for creating a process or opening an existing process (screen images may vary).
2	**View Menu**: Use the View settings at top right to show or hide the Operators, Dataset, Properties, and Navigation palettes. You can also dock the properties and datasets palettes to the right or bottom. Click Tear Off to open the current page in its own window in your browser to maximize your view.
3	**Operators Palette**: Drag and drop operators from this palette to your process layout. You can also enter search criteria (for example, "Get") to filter out nonmatching operators.
4	**Dataset Palette**: Use this palette to view, edit, and add variables in process or operator datasets.
5	**Properties Palette**: Use this palette and its additional buttons and windows to manage the properties of the currently selected operator.
6	**Navigation Palette**: Use this palette to navigate to specific regions inside large processes with multiple lanes. As a convenience, you can pan in any direction within this palette instead of scrolling the main designer layout up or down.
7	**Process Designer**: The actual process design appears in this work area, canvas, or layout. The Process Designer includes the background grid and one or more lanes.

Operators and Links: The Building Blocks

The general structure of a CA Process Automation process consists of two basic items: operators and links. *Operators* perform tasks or conditional tests. *Links* connect operators and determine the processing sequence.

The following list describes some examples of operators:

- File Management operators monitor file sizes, patterns, and other parameters relative to files on a system.
- File Transfer operators use FTP to perform file transfers and remote file operators.
- Email operators notify system administrators in the event of errors or other conditional states requiring human intervention.
- Network Utilities interface operators get, update, and monitor SNMP variables and send SNMP traps for network devices and monitors.
- Resource operators represent resources in limited supply.
- Web Services (SOAP methods) represent an interface between third party products and CA Process Automation.

Links connect operators and carry out the processing flow. The point where an operator and link intersect is called a *port*. A link originates at the exit port for one operator and ends at the entry port for another operator.

When a process between two operators runs, the processing sequence can be summarized as follows:

- Activate the first operator.
- Apply logic, get a result, and flow to the appropriate exit port.
- Activate the second operator.

An operator can have multiple exit ports to handle various results. Each exit link can initiate a separate branch of operators in a subprocess or *child* process. Exit and entry ports enforce a linear sequence in which operators are processed. They can direct the process flow to a particular branch of a process depending on the outcome of a single operator.

To direct processing based on the outcomes of multiple operators, use the following operators:

- The *Loop* operator.
- The *Exception* operator in Exception Handler mode.
- Conditional operators, such as the *And* operator or the *Or* operator.

These operators allow you to design multiple entry and exit branches for other operators. You can also create links that depend on the outcomes of multiple operators.

Create a Process Object

Use the Library Browser to create a *Process* object in a folder. You can also create a process in the Designer and specify a folder when you save it. Create a Process object for every discrete automation sequence, flow, or subprocess that you want to automate. After creating the Process object, design the process by adding operators and connecting them with links.

Note: To set rights and perform other object tasks, the best practice is to organize the objects in folders. Do not create objects at the root level because you cannot manage such objects as a group.

Follow these steps:

1. Click the Library tab.
2. In the Library Browser folders pane, select a folder.
3. In the toolbar, click New and then choose Process.

   A new process object appears with a default name. The process is automatically checked out to you to capture your exclusive changes.
4. Click the process name and change it to a unique name. The name is directly editable until you deselect it. To edit it again, right-click it and choose Rename.

   You have created a new process object. Next, you can design the process.

Design a Process

Complete the steps in Create a Process Object (see page 91) first. After you understand the concepts and steps behind creating your first few new process, you can easily edit any process. Use the Process Designer to design and configure all process objects.

Follow these steps:

1. Double-click the process in the Library or open it from the Process Designer.

   The Process Designer opens with a default set of basic start and stop operators.

2. Use the Process Designer to design and configure the process. This includes adding operators, ports, and links. Refer to the remaining topics in this section in any order to guide you.

3. To configure properties for a process, click any neutral space in the canvas, and choose Properties from the View menu in the toolbar.

   The Properties palette opens. The process properties determine the default behavior for all operators added to the process.

   Configuration

   Specifies whether to display horizontal or vertical swim lanes.

   Link

   Specifies the weight, color, and shape of the lines that link operators.

   Simulation

   Specifies the default simulation options for operators added to a process. You can also override these settings for a specific operator.

   Label Options

   Specifies the default display options for operators added to a process. You can also override these settings for a specific operator.

4. When finished, click Apply to view the changes.

5. Continue with any other tasks in this section of this guide.

6. When you are done editing an object click Save and then Check In.

Process Design Tips

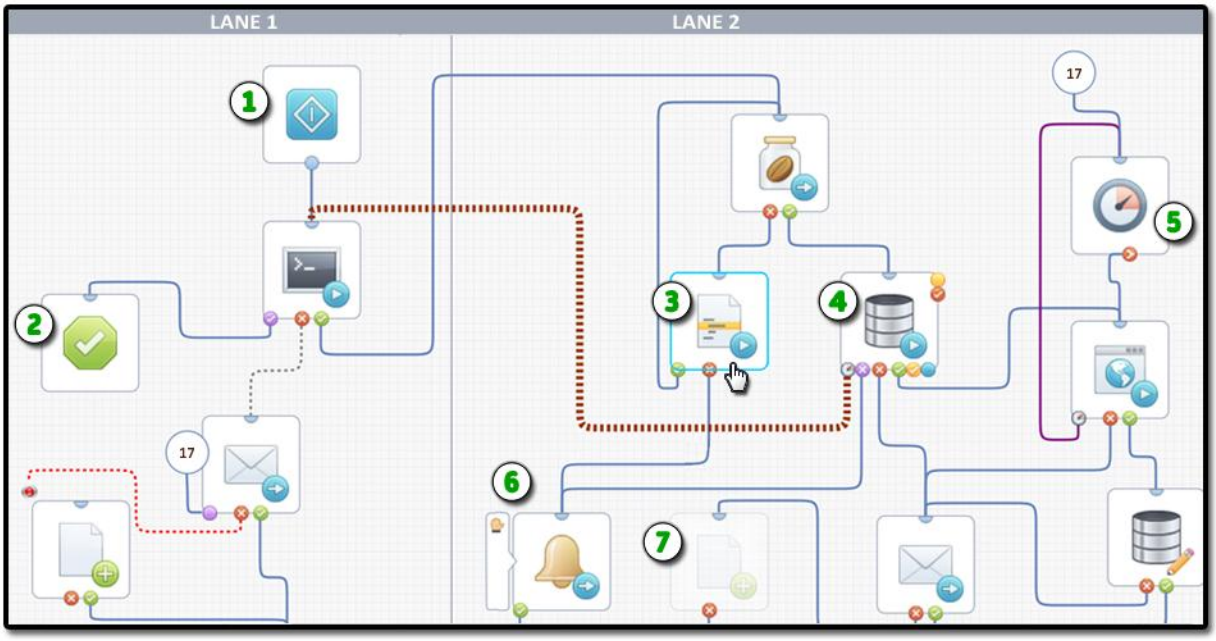
Keep the following tips in mind when working with processes:

- You can repeat any of the topics in this entire chapter in almost any order. For example, add a swim lane, add a port, or connect a link.
- As a basic rule, you can edit a process using the same steps for editing any automation object in the library. See Working with Objects (see page 58).

- Always check out the process to prevent others from overwriting your changes. You can check out a process before opening it (in the Library Browser) or after opening it (in the Process Designer).
- CA Process Automation always runs the current version of a process; however, if you have an object checked out, CA Process Automation is smart enough to run your working version.
- When you check in a changed process, decide how its versions are handled. You can either replace the version that you opened or you can create a separate distinct new version.
- Before you test changes to a process, check in the edited version that you had previously checked out.

Process Operators

Process operators apply functions or perform actions to yield a result. The collective results of a series of operators determine the automated flow through the process.



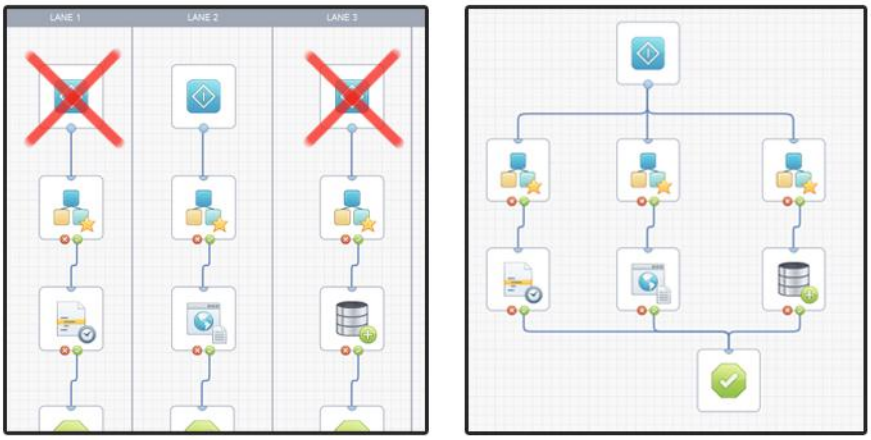
Item:	Description:
	Start Operator: All processes include at least one *Start* operator with no entry port and a single exit port. It is possible, although rare, to have more than one Start operator in a process.
2	**Stop Success Operator**: All processes include at least one *Stop* operator. Stop operators come in two flavors or types: *success* (shown here) or *failure*.

Item:	Description:
3	**Selected Operator**: The currently selected operator border appears as a dashed outline. The Properties palette displays the currently selected operator's settings.
4	**Database Query Operator with Multiple Exit Ports**: As you add ports, they appear around the operator's bottom and right edges. Each port represents the process flow for a particular result. **Note**: Right-click an operator to add a port.
5	**Delay Operator with After Port**: Delay operators do not include successful *Completed* or *Timeout* ports. In addition to a custom result port, they feature just two default ports: *After* and *Failed*. Different operators include different ports.
6	**Send Event Operator with Breakpoint Indicator**: Status indicators appear when you click Set Breakpoint.
7	**Disabled Operator**: When you disable an operator, the designer gives it a transparent appearance. You can decide later to enable it.

The Start Operator

A default *Start* operator is automatically applied when you create and open a new process. The Start operator has no entry port. Configure the properties for the Start operator in the Properties palette. For example, you can change the name, specify a custom icon, or change the location and text in the icon label.

The Start operator starts a chain of operators in any process. When a new process begins, it immediately activates the Start operator. All operators must be linked either directly or indirectly to this chain of operators. A process can have multiple Start operators to initiate multiple processing paths. These paths are also known as *branches* or *subprocesses*. However, as a general rule, a process really only needs one Start operator. As the example shown at right illustrates, a more efficient design utilizes the same Start operator and perhaps even the same Stop Success operator. Multiple links can share the same operator.



Note: Running multiple branches in parallel with multiple Start operators can result in confusing log entries. The logs may be difficult to analyze because the run sequence of branches with multiple starts may appear to be in random order. In most cases, use a single Start operator for each process.

Add Operators to a Process

After you place the Start operator in a process, drag the next operator, and then join the links that determine the process flow.

Follow these steps:

1. Do one of the following to open a process
   a. In the Library Browser, double-click a Process object.
   b. In the Designer toolbar, click Open.

   The Process Designer tab appears. If not already checked out, click Check Out.
2. Drag an operator from the Operators palette onto the process layout anywhere below or to the right of the *Start* operator. Operators are arranged in folders for each *module* or operator category supported by CA Process Automation. Enter the name of an operator in the Search field to search for matching operators.
3. Repeat the previous step as often as necessary to create a chain of operators.
4. Create the first link in the chain. Click the small exit port under the Start operator and drag the link to the destination operator's entry port.
5. Continue to logically connect exit ports to the entry ports of operators in sequence.
6. Configure each operator by double-clicking it to open its Property palette.

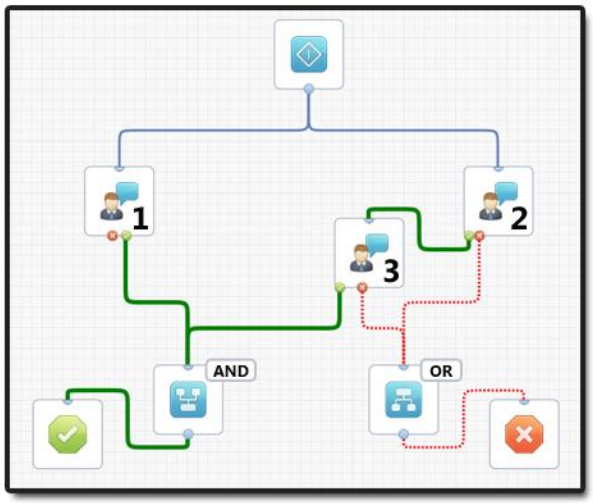
Logical Operators

The logical operators (*And*, *Or*) are used to synchronize, unite, and split processing according to conditions. In this release, the And and Or operators are available in the Standard folder in the Operators palette.

Note: Beginning in CA Process Automation r4.0, the Derivation operator is no longer available. When imported or migrated to this release from a previous release, the Derivation operator is converted to an Or operator.

Example: Conditional Logic

The following process example shows eight operators and nine links (two blue, four green, and three red).



- The Start operator at the top initiates simultaneous processing of the first two linked operators (Task 1 and Task 2).
- Task 3 starts *only* after Task 2 successfully completes.
- The And operator on the Task 1 and Task 3 exit links is activated only after *both* Task 1 and Task 3 successfully complete. This path ends at the green Stop Success operator.
- The Or operator on the Task 2 and Task 3 exit links is activated after an abnormal exit from *either* Task 2 or Task 3. This path ends at the red Stop Failure operator.
- The Process Control operators running on an orchestrator run a process. You can run individual process tasks on any agent touchpoint that is in the same environment as the Orchestrator. For example, Task 1 can flow on a Windows touchpoint while Task 2 and Task 3 flow on a UNIX touchpoint.

The 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 the entry links to it have been activated. Use an And operator to synchronize multiple branches of a process when all branches must be completed before beginning one or more additional branches.

The Or Operator

The *Or* operator activates exit links when any one of its entry links are activated. The Or operator can also be implemented with a single entry link to enable two or more output links for separate parallel branches.

You do not have to use an Or operator to implement a logical "or" condition in a process. Two or more links entering the same operator behave the same as an Or operator. However, to reduce confusion, and to document the logic in a process, it is recommended that you use an Or operator rather than merging links in an operation. The *Name* property on an Or operator allows you label the operator and document it in context of the logical sequence of operators in the process.

The Stop Operator: Success or Failure

Terminate a process by linking the final operator in a sequence to a Stop operator. A process can have multiple Stop operators on different branches. A Stop operator processed on any branch has no exit and terminates processing of the entire process. Stop operators can optionally be configured to terminate a calling loop in another process.

Follow these steps:

1. Open and check out a process.
2. In the Operators palette, drag a *Stop Success* or a *Stop Failure* operator to your process.
3. Double-click the Stop operator to open the Properties palette.
4. In the Stop section, specify the integer value returned by the process in the Result field and choose a value for the End Type field.
   - To end the process normally, click Stop Success and enter a Result value of 1. The Result value 1 causes a completed process to exit when the process is called from a Start Process operator in a parent process.
   - To end the process abnormally, click Stop Failure and enter a Result value of 0. The Result value 0 causes an aborted process to exit when the process is called from a Start Process operator in a parent process.
5. (Optional) Instead of using these default result settings, you can specify any expression that returns an integer. The expression should return a non-zero integer to indicate that the process completed normally or zero to indicate that the process completed abnormally.
6. Select the *Break Calling Loop* check box to break a calling loop. If the process was called from a looping operator in another process, this option breaks the loop when processing returns from this Stop to the other process. Clearing this check box allows a calling loop to continue.
7. Select the *Ignore Running Tasks (Immediate Stop)* check box to stop processing the process immediately when the Stop is executed. This interrupts any other operators that are still active elsewhere in the process. Clear this check box to allow ongoing operators to terminate normally before stopping the process.

Process Operator Ports and Links

Links between operators define dependencies. The links act on the results that each operator produces. Links define the order and logic of a process as it flows.

Different kinds of actions have different predefined results or outcomes:

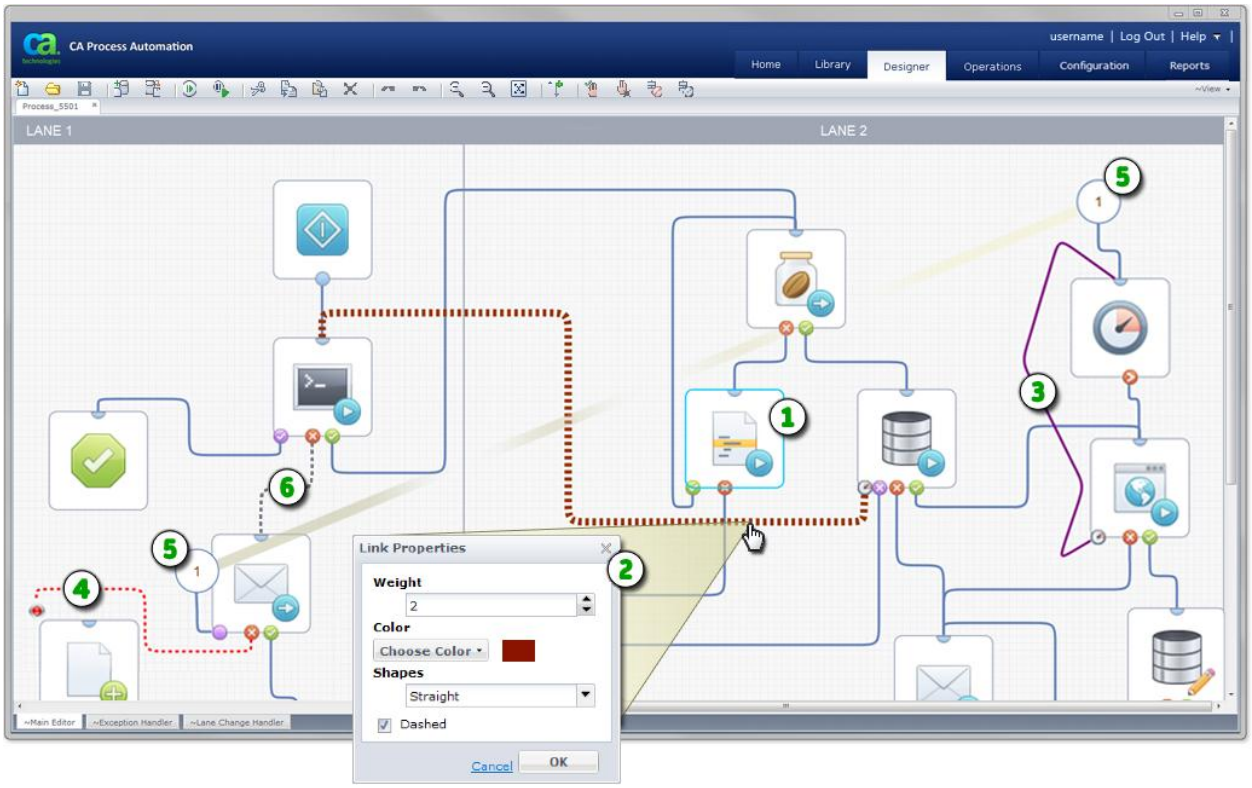
- Successful
- Completed
- Aborted
- Failed
- Timed out

The application calculates these outcomes to determine the next exit conditions, ports, and links to activate, in a logical sequence. For example, you can add a custom port on some operators and can define the port to activate when an expression returns a True value.

Exit conditions on an operator are not mutually exclusive. If the product evaluates more than one exit condition as True, all of exit conditions are processed. Processing multiple exit conditions on a single operator can start subsequent simultaneous processing of multiple branches.

When a process runs, the product activates its operators only once. When a link leads to a previously activated operator, the product does not reprocess the destination operator and the branch that the link extends ends.

Note: Links are joined to operators at small connection points called *ports*.



Item:	Description:
1	**Selected Operator**: Click an operator to view its dataset variables, pages, and properties. Right-click an operator to add an exit port.
2	**Link Properties**: Do one of the following to adjust the appearance of a link: ■ Double-click a link ■ Right-click a link, then select Link Properties Select the thickness, color, shape, and dashed appearance of each link.
3	**Link Line Shape**: Instead of ordinary orthogonal lines, this purple link appears with straight line segments. You can stretch and position all links as necessary.
4	**Stopped Link**: This link has been forced to stop. As an example, consider a process that is looping, waiting for some event, processing that event, and looping repeatedly. When a parallel process branch determines that the original looping process must stop, it can use the stopped port and link to stop the loop.
5	**Broken Link**: Break a link to split a long circuitous route into two numbered stubs. The split links are easier to view and manage. To rejoin the numbered stubs, right-click the circled link number, then select Join Links.

Item:	Description:
6	**Disabled Link**: This dashed gray link to indicates that it is temporarily disabled. Right-click the link to re-enable it.

Add Operator Ports and Links

Place a link between operators to establish the logical flow. For example, link the Start operator to the next operator to begin a process flow. Links connect one of any number of exit ports on one operator to the single fixed entry port of another.

Follow these steps:

1. Open a process.

   The Process Designer tab appears. If not already checked out, click Check Out.

2. Identify a source operator, its exit port, and a destination operator's entry port.
3. On the source operator, if the exit port you want does not appear, right-click the operator and then click the port you want to add. Port types vary by operator. Examples include Failed, Completed, After, and Custom.

   A new color-coded exit port appears on the operator's border.

4. Click the exit port on the source operator and drag the link to the entry port of the destination operator.

   A new link appears between the two operators.

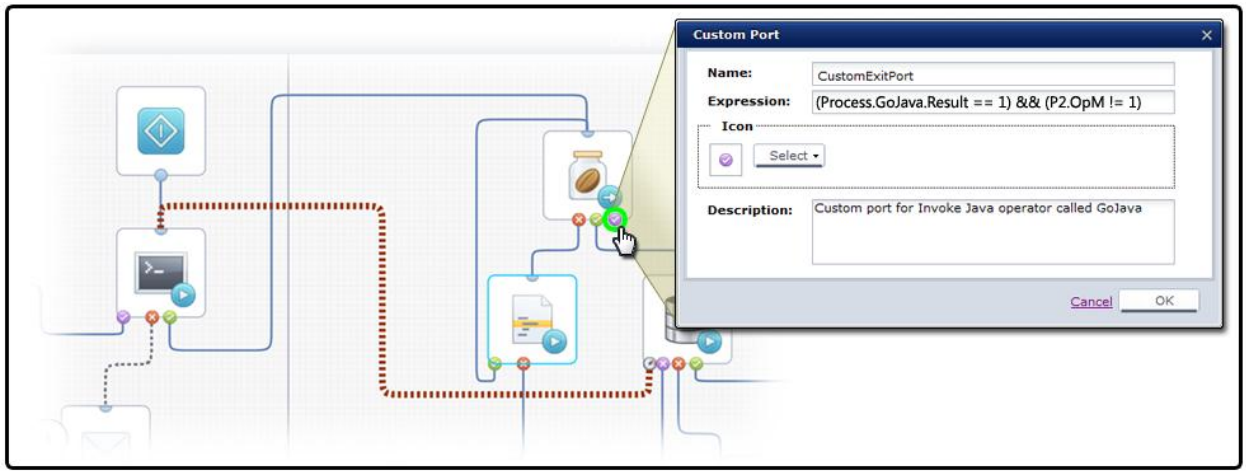
5. Repeat these steps, adding links and, if necessary, ports between operators to define the process flow.
6. In the toolbar, click Save.

Custom Exit Ports and Expressions

Most CA Process Automation operators support custom exit ports. When you add a custom port, specify the following:

- a name for the condition
- an icon to distinguish it from other ports on the operator
- a valid Boolean expression that returns *True* or *False* when the operator finishes processing
- an optional description

You can also move the port to improve your view of the link. Press the Ctrl key while clicking the port to move it to supported positions along the right or bottom edges of the operator.



In the Expression field, enter any valid JavaScript expression that returns a Boolean value, either *True* or *False*. You can use the custom port expression to evaluate the result code of an operator. The result code indicates the outcome of the operator and is returned by the *Result* variable in the operator dataset. The custom link is activated only if the expression returns a *True* condition.

To activate a link based on a specific value for the result code, use the following syntax:

```
Process.Operator.Result == value
```

The keyword *Process* refers to the process dataset. *Operator* is the value specified by the *Name* parameter of the operator. *Result* is the field name for the result code variable in the operator dataset.

To activate a port, you can also construct an expression as a comparison between multiple statements. In this example, the value for *GoJava* must be *1* and the value for operator *OpM* in process *P2* must not be *1:*

```
(Process.GoJava.Result == 1) && (P2.OpM != 1)
```

When a process runs and encounters an operator with no defined exit port, the process goes to a *Blocked* state.

When an operator has multiple custom ports, the application runs all the ports with an exit condition that returns a *True* value. Avoid overlapping logic for exit ports if you do not want to activate more than one exit link at a time. This condition is more likely to occur if you include both standard and custom ports on the same operator. For example, if you include a custom port activated by the expression *Process.A==5* and a standard successful port on an operator, an operator activates both exit links when Process.A returns 5. To trap values and route processing to a single port, it would be better to use more than one custom exit port to specify exclusive expressions, such as:

```
(Process.A == 5) AND (Process.OperatorName.Result == 5)
(Process.A == 5) AND (Process.OperatorName.Result != 5)
(Process.B == "finance") AND (Process.OperatorName.Result == 7)
(Process.OperatorName.Result == 1) AND (Process.A <> 5)
```

In this example you could also include the standard failure link to cover when the operator result variable returns 0. If you are uncertain about the results of different outcomes, you can use the simulation function on an operator to test outcomes of different permutations of settings and values.

Note: Use variable assistance features (press Ctrl + Space) to ease constructing these expressions.

You can edit a custom port any time after you add it to an operator by double-clicking the port. If you cannot predefine a course of action for a particular exit condition (for example, when a database import fails), you can omit a link for it. When an exit condition for an operator is not specified by any exit link, the process enters a suspended state until a user can take corrective action.

More information:

Simulate Processing of Operators (see page 415)

Break a Link for Readability

Links can become tangled in a complicated process and clutter your view. If you have many links crossing one area, or you want to link an operator across other branches to a distant operator, you can break a link.

Note: The link itself is not broken; only the visual representation of the link. Instead of a full line winding its way between two operators, the line is split into two matching numbered link symbols at the source and destination operators.

Follow these steps:

1. Identify the link you want to break between a source operator and a destination operator.
2. Right-click the link, and select Break Link.

   The link is broken, replaced by matching link markers at both ends of the link. The split link behaves the same way as an unbroken link.

   Note: To rejoin a broken link, right-click either numbered link marker, and click Join Link.

Process Loops and Iterations

Surveillance, monitoring, and other cyclical processes often run repeatedly. You can control these cycles using loops. You can apply one or more methods for running operators in loops:

- You can cycle through or loop an operator until some condition is met.
- You can use the Loop operator to loop a sequence of operators.
- You can loop an entire process. A looped process can consist of multiple linked operators.

System Variables for Looping

You can create custom loop variables and manage them yourself, or use the available system variables for loops included with this release of CA Process Automation.

For example, you could create logic to calculate the duration from the start time of the first iteration of a loop to the current time for each loop. You could even use pre and post execution code to set up input into a loop such as

- initializing variables
- setting loop counts
- processing results when the loop is complete

Instead of creating and updating your own loop counter variables, you can take advantage of built-in loop variables. The Loop operator and any other operator with loop settings support the following dataset variables:

- CurrentLoopIteration
- OverallLoopDuration

Use *CurrentLoopIteration* when you need a standard loop counter. The *CurrentLoopIteration* variable contains the value *0* during the first iteration of the loop and increments by *1* at the beginning or end of each additional iteration. For example, if the operator is configured to loop 3 times, at the end of execution of all iterations, *CurrentLoopIteration* is equal to *3*. Specifically, it is *0* in the first iteration, *1* in the second iteration, *2* in the third iteration, and *3* in the last iteration. The last iteration is not executed because it violates the loop condition.

Use *OverallLoopDuration* when you need to loop for some fixed time duration; for example, to loop for a maximum of 5 minutes and then exit. This variable contains the number of seconds between the start of the first iteration of the loop and the end of the last iteration. *OverallLoopDuration* is updated at the beginning and end of every loop iteration. It includes any delay set between iterations of the loop.

Note: You cannot modify the *CurrentLoopIteration* and *OverallLoopDuration* system variables. Although they appear in the operator dataset, their values do not change unless they are looping (operator Repeat Count > 1).

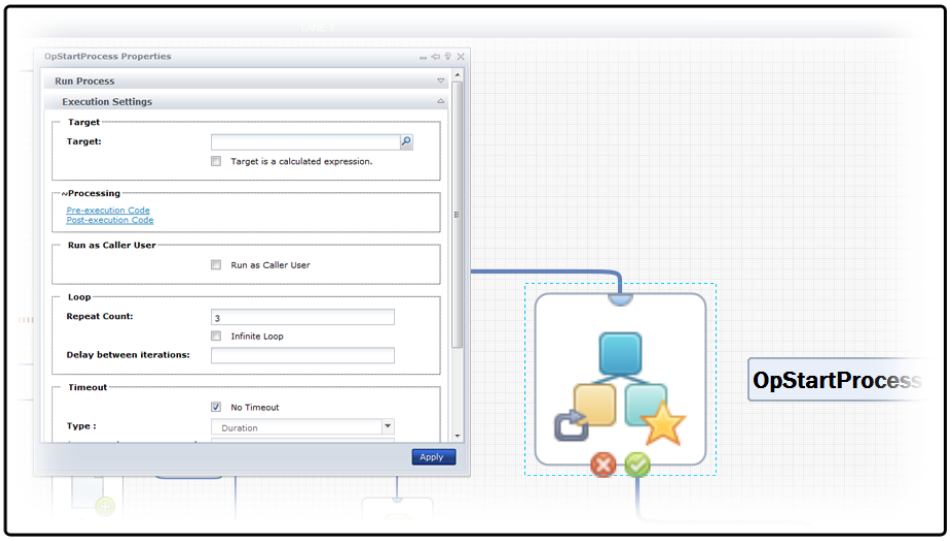
Loop an Operator in a Process

A simple method for looping a process is to set the Loop parameters on an operator that supports looping. Open the Properties palette for the operator. In the Execution Settings section, in the Repeat Count field, enter the number of times for the operator to repeat. Repeat Count is a calculated field, so you can use a variable or expression to specify the count at runtime. Repeat Count accepts either an integer (the number of times to loop) or a Boolean (the loop continues as long as the condition evaluates to true). Examples of valid entries include:

```
3
Process.var < 3
Process.var == false
```

You can repeat the operator indefinitely by selecting the Infinite Loop check box.

The Loop indicator appears on operators that you have decided to loop:



You can configure the loop parameter on the Start Process operator to run a process repeatedly. This works well for looping a few iterations of a process and saving a historical snapshot of the process for each loop. However, avoid calling a process many times (as in an infinite loop) from another process. CA Process Automation keeps a history of all process instances. Calling a process in an infinite loop from another process can use a considerable amount of disk space to save irrelevant data.

The preferred method for running a process repeatedly is to loop cyclically within a process. When necessary, you can still save a historical snapshot of a looping process by branching to a Start Process operator that starts a new detached instance of the process before executing a Stop operator at the end of the branch.

You can specify the retention period, or length of time to save for the history, in the library policy settings for an orchestrator or its associated touchpoint.

Note: If you set an operator to loop with a timeout followed by an action of *Reset*, the loop condition is checked when moving from one iteration to another, not when resetting an iteration. The *OverallLoopDuration* variable contains the number of seconds since the start of the first iteration, including time spent in all the reset iterations. Loop iteration resets do not also reset *OverallLoopDuration*.

If you set an operator to loop with a timeout followed by an action of *Continue, OverallLoopDuration* will contain the number of seconds from the start of the first iteration until the end of the last successful iteration. If the operator times out, *OverallLoopDuration* will not contain the number of seconds from the start of the first iteration until the time the operator times out.

Interrupt a Looping Operation

You can interrupt a looping operator by adding a stop link from another branch of the process.

Follow these steps:

1. Open and check out a process.
2. Create a link from an operator in an independent branch and attach it to a looping operator.
3. Right-click the link, and click Stop Link.

   The link appears as a dashed red line with a red stop symbol near the looping operator.
4. Click Save.

At runtime, the separate branch reaches the looping operator, and the following actions occur:

- The looping operator runs and completes itself.
- Any post-execution actions for the operator are performed.
- The now-merged processing from the two branches proceeds to the next operator in the process.

Loop Through Indexed Elements of a Dataset Field

Instead of using the built-in *CurrentLoopIteration* and *OverallLoopDuration* system variables, you can create custom loop variables and manage them yourself. Previous versions of CA Process Automation required this method. For example, you could create logic to calculate the duration from the start time of the first iteration of a loop to the current time for each loop. You could even use pre and post execution code to set up input into a loop such as

- initializing variables
- setting loop counts
- processing results when the loop is complete

To loop through all the elements of an indexed dataset field, first use a Run JavaScript operator in the Utilities group to initialize the CurrentIndex element for the dataset field to 0. For example, the following expression initializes the *CurrentIndex* element on the process variable *X* to *0*:

```
Process.X.CurrentIndex=0;
```

To loop through indexed elements of a dataset field

1. Connect the Successful exit link from the Run JavaScript operator to the operator you want to process in a loop.
2. In the Dataset palette of the operator that you want to loop, use a Size element setting for the indexed field as the Repeat Count value on the Loop tab. For example:
   ```
   Process.X.Size.
   ```
3. To increase the CurrentIndex setting after completing each iteration of the loop, use a post-execution code expression. For example:
   ```
   Process.X.CurrentIndex=Process.X.CurrentIndex+1;
   ```
   In this case, the CurrentIndex element is the counter for the loop. You can use the CurrentIndex setting to access elements of the indexed field in calculated expressions. For example:
   ```
   Process.X[Process.X.CurrentIndex];
   ```
   Note: If you are accustomed to programming languages such as Visual BASIC that use one-based arrays, remember that an indexed dataset field is a zero-based array. In one-based arrays the first element is indexed by 1 and the last element is indexed by the number of elements. The first element of an indexed dataset field is indexed by 0 and the last element is indexed by one less than the value of the Size element for the field.

In addition to accessing elements in an indexed dataset field, you can track the iteration number during loop processing for other purposes. Use a process variable such as *process.i* for the index variable. In an Interpreter Module Calculation operator that precedes the looped operator, initialize the process variable to its starting value. For example, *process.i=1.* To increase the index variable after completing each iteration of the loop, use an expression in the post-execution code for the operator. For example:

```
Process.i=Process.i+1;
```

Note: The steps in this topic are considered no longer necessary; however, they are included for reference with legacy code. If using the newer system variables, only step 2 applies and step 3 is replaced as follows. Use the *CurrentLoopIteration* variable of the looping operator to access elements of the indexed field in calculated expressions. For example:

```
Process.X[Process[OpName].CurrentLoopIteration];
```

Index the Loop Count for Other Purposes

You can track the iteration number during loop processing for accessing elements in an indexed dataset field, or for other purposes. Use a process variable for the index variable (for example, process.i). In an Interpreter module Calculation operator that precedes the looped operator, initialize the process variable to its starting value for the loop:

```
Process.i=1;
```

To increase the index variable after completing each iteration of the loop, use an expression on the Post-Execution Actions tab of the operator properties. For example:

```
Process.i=Process.i+1;
```

Note: This topic is considered no longer necessary; however, it is included for reference when working with legacy code.

Loop Errors and Exceptions

If an error occurs with respect to the pre or post-execution code:

- The process logs indicate that the Loop operator pre or post condition failed to execute.
- The process exception handler is triggered with an *abort* exception. The *Source* and *SourceROID* point to the Loop operator.

Loop a Series of Operators

You can also loop a series of operators inside a Loop operator.

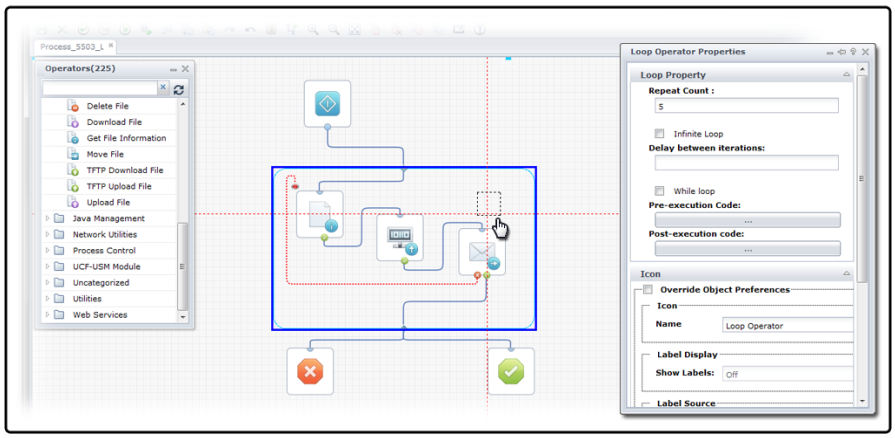
Follow these steps:

1. Open a process.
2. Drag the Loop operator from the Standard operators group onto the process.
3. Resize the bounding box for the Loop operator to accommodate the sequence of operators that you want to add to the loop.
4. Drag operators into the box.

   The bounding box color changes to indicate it is the selected destination for operators you drag.
5. Add ports and links.
6. Link the entry point on the box to the first icon in the branch and link the last icon to the exit port on the box.
7. Add links to and from the Loop operator:

   a. Link at least one operator outside the box to the entry port.

   b. Link the exit port to at least one operator outside the box.



8. Double-click the Loop operator to view its Properties.

   The Properties palette for the Loop operator appears. If it does not appear, choose Properties from the View menu.
9. Set the Loop Properties and click Apply.

   Note: You can create an infinitely looping branch by selecting the Infinite Loop check box. As with any infinitely looping operator, you can stop the loop by adding a Stop link from another operator.

Note: Values for the *CurrentLoopIteration* and *OverallLoopDuration* system variables are updated for each process instance and appear in the dataset. Even if the Repeat Count for the Loop operator is set to 1, *CurrentLoopIteration* shows 1 and *OverallLoopDuration* shows 10 at the end of a single instance. By contrast, the *CurrentLoopIteration* and *OverallLoopDuration* system variables are only updated for other operators when Repeat Count is greater than 1. This is because the Loop operator *always* loops, even if it is only one time.

More information:

Interrupt a Looping Operation (see page 108)

While and Do While Loops

Use the While Loop check box of the Loop operator to manage loop behavior. When checked, the Loop operator behaves as a *while* loop. When unchecked the Loop operator behaves as a *do while* loop.

Any existing Loop operators in content developed before an upgrade to CA Process Automation 4.0 will have the While Loop field unchecked. They will continue to behave as *do while* loops. You can turn these Loop operators into *while* loops by checking the associated While Loop box.

The two main differences between *while* and *do while* Loop operators are:

- *While* loops check the loop condition specified in the Repeat Count field before it executes the first iteration and each subsequent iteration.
- *Do while* loops check the loop condition specified in the Repeat Count field at the end of every iteration, so it is guaranteed to always execute at least the first iteration of the loop.

Note: A Loop operator can be set to behave as a *while* or *do while* loop. The other operators that support looping can only behave as *do while* loops.

The Logical Sequence of a Loop Operator

A Pre-execution code and Post-execution code field are available in the Loop operator properties. You can enter JavaScript code in these fields to run with each iteration of the loop. CA Process Automation runs any pre and post execution code for a Loop operator for each iteration of the loop.

Note: In content developed before an upgrade to CA Process Automation 4.0, loop operators will have empty Pre-execution code and Post-execution code fields.

The processing sequence of any Pre and Post conditions depends on the type of loop.

While Loop Operators:

1. Run Pre condition.
2. Check *while* loop condition as indicated by the Repeat Count field.
   - If the loop condition succeeds:
     a. Reset the operators inside the Loop operator.
     b. Execute the operators inside the Loop operator.
     c. Execute the Post Condition at the end of the Loop iteration.
     d. Loop back to step 1.
   - If the loop condition fails:

     Get out of the Loop operator without executing the Post condition then execute the next operator after the loop operator.

Do While Loop Operators:

1. Reset the operators inside the Loop operator.
2. Execute Pre condition.
3. Execute the operators inside the Loop operator.
4. Execute Post condition.
5. Check *do while* loop condition as indicated by the Repeat Count field.
   - If the loop condition succeeds, loop back to step 1.
   - If the loop condition fails, get out of the Loop operator then execute the next operator after the Loop operator.

Loop a Process

To run a process many times, create a cyclical branch in a process object instead of using the Start Process operator to call a process repeatedly from another process. This method involves looping a process cyclically by placing two Start operators. One Start operator goes at the beginning and another Start operator goes at the end.

Follow these steps:

1. Open and check out a process.
2. Place the following operators:
   a. Start
   b. Stop Success
   c. Stop Failure
   d. A series of operators that you want to repeat.
3. Add a Start operator at the point in the sequence where it ends and where you want to restart the branch.
4. Link the last operator in the branch to the second Start operator. The Start operator has an entry link that allows it to be placed at the end of a sequence of steps.

   Note: When the processing sequence arrives at the second Start operator shown in the figure, it reinitializes all operators and restarts the process.
5. To interrupt this type of cyclical process, incorporate logic that leads out of the cyclical branch. Add an exit port on an operator which leads directly to a Stop operator or a different branch.

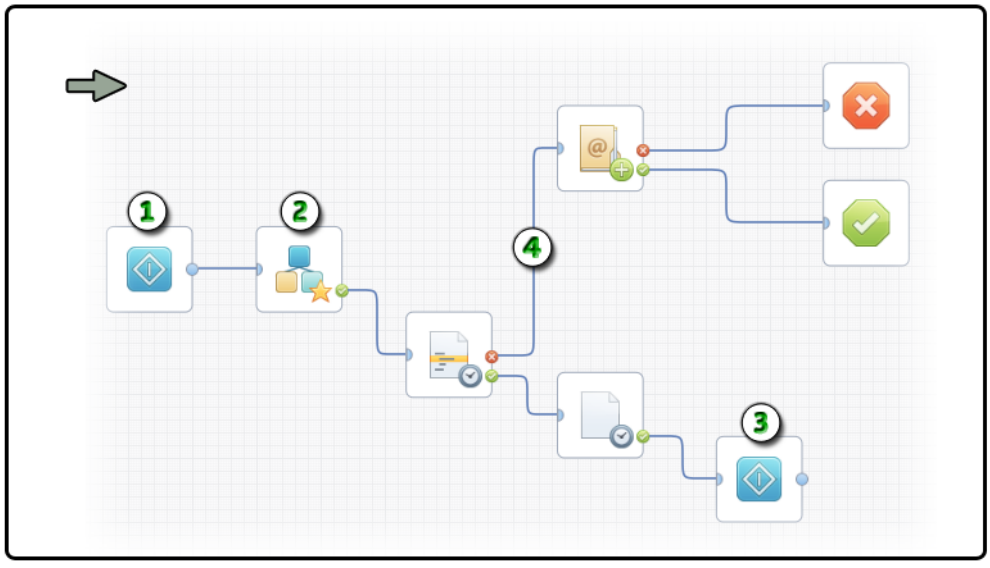
Daemons and Other Looping Processes

When you want to run a process over and over repeatedly, create a cyclical branch in a process object instead of using the Start Process operator to call a second process repeatedly. A cyclical process runs in an infinite loop. Each iteration does not create new instances of the process object. The primary advantage for implementing a cyclical branch is that the logic that exits the loop is in the process itself. The process does not depend on external factors in a parent process. The process performs as an autonomous object.

Examples of looping processes that run indefinitely include:

- Continuously looping processes for monitoring system or network usage
- Polling the state of networks or system components
- Organizing and checking events or messages
- Daemons
- Services

Looping Process Example



Item:	Description:
1	**Start Operator 1**: Begin the process with a standard Start operator.
2	**Start Process Operator**: Design a sequence of operators that represent the functionality you want to repeat or loop. This branch can even include a Start Process operator to launch instances of a second process.

Item:	Description:
3	**Start Operator 2**: The Start operator has an optional entry link. When the process flows to the second Start operator, it reinitializes all other operators and restarts the process back at Start Operator 1.
4	**Exit Port and Link**: To exit this looping sequence, specify an exit condition.

Note: An alternate method for interrupting a cyclical process is to use an independent branch with its own Start operator. The branch can wait for the looping portion of the process to change a variable or free a resource. After those events occur, the independent branch runs; for example, by sending an email alert or processing a Stop operator.

Process Control

Complex processes often require a hierarchy of subprocesses. In traditional programming, functionality is modular. Teams break down complexity into procedures, methods, or libraries that you can use repeatedly. Similarly, in CA Process Automation there are methods to simplify complex processes. You can model subordinate processes as branches or as separate processes. A separate subordinate or *child* process can then be called from a *parent* process by a Start Process operator. For example, a parent process managing computer systems for a bank could incorporate the following child processes using Start Process operators:

- During each business day, process automatic bill payments.
- During each night, perform a daily backup.
- At the end of each day, process and verify checks.
- At the end of each month, calculate interest for accounts.

Subordinate processes define logically or physically distinct child processes within a system. The structure of a parent process synchronizes child processes, manages resources that are shared across processes, and defines error handling and dependencies between processes on a system-wide scale.

Child Processes

Use the Start Process operator to start a secondary or *child* process from a running instance of a primary or *parent* process. The Start Process operator starts a new process *instance* on a specified touchpoint. The Start Process operator initiates process variables that are associated with the new instance.

The Start Process operator can start a new instance of a process in *attached* mode, *detached* mode, or as an *inline* process:

- **Attached mode**: The process that starts the new instance is referred to as the *parent* process. The new instance is referred to as the *child* process. The application finishes processing the entire new instance. A child process finishing its flow in attached mode can copy its dataset values to the dataset in the calling parent process.

  If a parent process activates a Stop operator on a branch before a child activates its own Stop operator, it is possible for the parent instance to finish *before* the child instance.

- **Detached mode**: The Start Process operator launches a new child process instance that behaves as if it has no parent. In detached mode, the workflow that starts a new instance of another process completes immediately after queuing the start request. A process finishing in detached mode cannot copy its dataset values to the dataset in the calling parent process.

- **Inline process**: The child process runs as a separate instance. A parent process has limited control over the child process. The inline child process is tightly linked to the parent with access to the parent context and lifecycle such as instantiation or archiving. You cannot run an inline process in detached mode.

Configure a Child Process

Configure the Start Process operator in a parent process to control the behavior of the child process it is starting.

Follow these steps:

1. Open and check out a process in the Process Designer.
2. In the Operators palette, expand Process Control and drag a Start Process operator into your process.
3. Double-click the Start Process operator to view its properties in the Properties palette.
4. Expand the Start Process group.
5. In the Process Name field, enter the full path to the process.
6. In the Process Dataset Initialization Code field, click the Browse button indicated by ... (ellipsis) to open the full scripting dialog.
7. Enter JavaScript statements to initialize variables in the dataset of the child process that you are starting. In this context, the *Process* keyword refers to the process dataset of the new instance being started. The *Caller* keyword refers to the dataset of the parent instance containing the Start Process operator. This context is the only one in which the Caller keyword is available. This context is also the only one in which the Process keyword does not refer to the process that contains an operator.
   - Parent Process: Process A
   - Child Process: Process B
   - Start Process operator in A starts child process B
   - Initialization of local dataset B occurs with dataset A as the *Caller*

   The Process and Caller keywords are mandatory for referring to parent or child process dataset variables. If you omit both keywords on a variable name, the application looks for a calculation-scope variable. It does not check for a similarly named variable in either the parent or child dataset. For example, the following code fails if no calculation-scope variable X was previously created in the local script dialog:

   ```
   Process.X = Caller.X;
   Process.Y = X + 100;
   ```
8. Click Save to close the Initialization Code dialog.
9. In the Mode field, select Attached, Detached, or Inline.

   If you select Detached, the Start Date field becomes enabled. It specifies the date when the detached instance of the process starts. The default value is the date when the operator runs, indicated as *System.Date*. Similarly, the Start Time field specifies the time when the detached instance starts. The default value is *System.Time*.

Note: Previous versions of this application included separate Run Process operators for attached mode and detached mode. Both operators actually performed the same function. The only difference was that for the operator placed with Run Process, the Detach after start or after queuing request check box is initially cleared. For the Run Detached Process operator, the check box was initially selected. In either case, you could select or clear the Detach After Start or After Queuing Request check box anytime after you placed the Run Process operator to change its start mode. In this release, all of these operators are imported as Start Process operators.

Initialize Child Process Variables with the Caller and Process Keywords

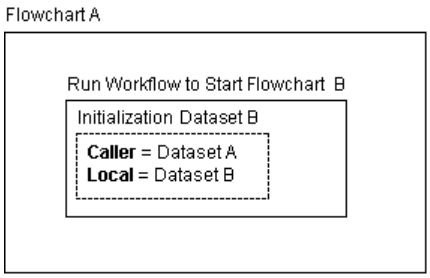
To enter a script to initialize Process variables in the child Process, click the Process Dataset Initialization Code browse button to open the Process script dialog. In the Process script dialog, you can enter JavaScript statements to initialize variables in the Dataset of the Process that you are starting.

Process keyword

Refers to the Process Dataset of the new instance being started. This context is the only one in which the Process keyword does not refer to the Process that contains an Operator.

Caller keyword

Refers to the Dataset of the parent instance that contains the Run Process Operator. This context is the only one in which the Caller keyword is available.



In the Process script dialog, the Process and Caller keywords are mandatory for referring to parent or child Process Dataset variables. If you omit both keywords on a variable name in the Process script dialog, the Interpreter Module looks for a calculation-scope variable. The Interpreter Modules does not look for a similarly named variable in either the parent or child Dataset. For example, the following code fails if no calculation-scope variable X was previously created in the Local script dialog:

```
Process.X = Caller.X;
Process.Y = X + 100;
```

Return Dataset Variables to the Parent Process

The dataset for a completed child process can be accessed by using an expression in the parent process. The name of the Start Process operator references the child process dataset in the local dataset of the parent process. The following code in the parent process references a dataset variable of a child process:

```
Process.Operator_Name.Field_Name
```

Operator_Name represents the name of the Start Process operator in the parent process and *Field_Name* is the dataset variable that you want to access in the child process.

This only works for processes started in attached mode. Processes started in detached mode become the root process in a call sequence and do not copy their datasets to the process that started them.

Start Processes Recursively

Other than memory usage, there is no restriction on the number of processes you can start in a chain. A process can even start another instance of itself recursively.

Avoid calling processes recursively in attached mode because this practice can result in an infinite call chain. However, it is often useful for a process to start another instance of itself in detached mode. For example, you can save an image of a monitoring process in a certain state, and continue running the process. In this case, the monitoring process can start a new instance of itself in detached mode, and then execute a Stop operator to terminate itself. The terminated instance is then saved and an administrator can examine it in its preterminated state.

Inline Process

The Start Process operator is used to invoke child processes. The child process that is invoked runs as a separate instance. A parent process has limited control over the child process and may lead to a performance overhead at execution time.

Inline mode lets you execute a child process and expand it into the parent process. An inline child process has access to the parent context. The lifecycle of the inline child, including instantiation, archiving, and so on, is tightly linked to the parent.

CA Process Automation also permits users to decide whether they want to run a process as an inline process.

Configure an Inline Process

You can configure a process operator to run a child process in inline mode.

Follow these steps:

1. In the Start Process operator Properties palette, click the Object Browser (...) button.

   The Object Browser appears.

2. Select the child process.

3. In the Mode field, select *Inline*.

   The child process is configured to run as an inline process.

4. Select the Inherit Lane Change Handler from parent process check box.

   The lane change handlers of the parent are loaded for the child process.

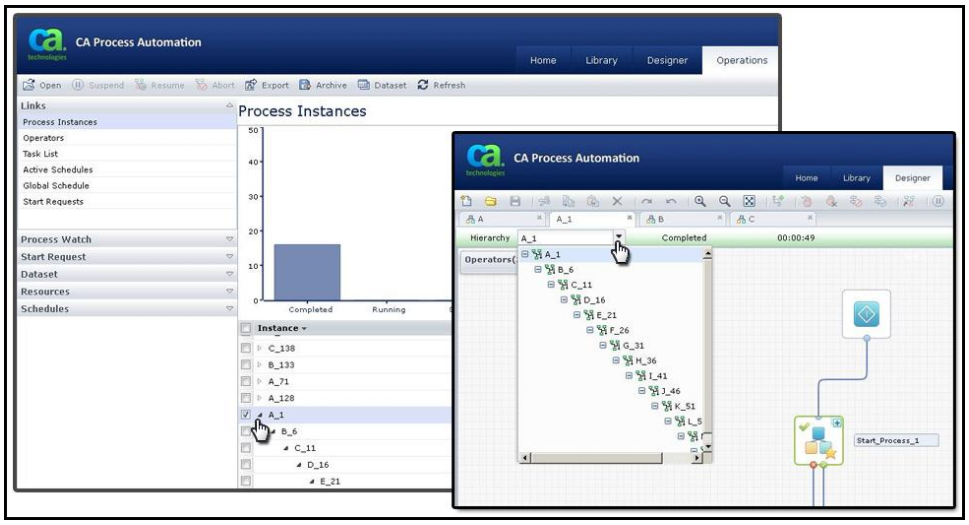
   Note: The Inherit Lane Change Handler from parent process check box is enabled only for inline child processes. CA Process Automation evaluates the child process when the parent process starts. The inline child process is loaded and expanded in the parent process.

View an Inline Child Process

You can view an inline child process at runtime using one of the following methods:

- In the Designer, click the plus icon on the top right-hand side of the Start Process operator in the parent instance.
- In the Designer, click the Hierarchy drop-down list and select a child process listed under the parent.
- On the Operations page, click a parent instance.

The following graphic demonstrates these three methods.



Process Lanes

The lanes in a process, often called *swim lanes*, provide a way for you to divide your process into different logical parts. You can add, insert, resize, and remove swim lanes in the Process Designer. You can define rules that are triggered when links between steps in a process cross a boundary between lanes. The rules define additional steps to run when processing crosses swim lane boundaries.

You arrange process operators visually in one or more swim lanes. Parallel lines show the lane boundaries. The default lane orientation for a new process is vertical. Initially, a new process has a single lane, labeled *Lane_0.* Additional lanes are named in sequence as you add them, for example, *Lane_1*, *Lane_2*, and so forth. You can customize the names to arrange branches of a process in your own meaningful way.

When there are multiple lanes in a process, you can define lane change rules. These rules specify operator sequences that are invoked only when execution of a process crosses between two lanes. A lane change rule looks like any sequence of operators on the Main Editor tab, except that each rule starts with a Change Lane operator instead of the Start operator. Properties on the Change Lane operator define the transition between source and destination lanes that invoke the sequence of operators linked to the Change Lane operator.

You can add any number of lanes on the Main Editor tab and then form links that cross lane boundaries between operators in the process. Then, on the Lane Change Handler tab, you define sequences of operators invoked when execution crosses boundaries. When execution of a process crosses a lane boundary, it invokes a process on the lane change handler that is defined to occur for the particular transition (such as from lane A to lane B).

If you define more than one rule for a particular transition, the application uses the first existing rule in the following order of priority:

Priority	Matches and runs
1	Rule defined for specific source and destination lanes (A to B)
2	Rule defined for a specific source lane (A to *any*)
3	Rule defined for a specific destination lane (*any* to B)
4	Rule defined for any two lanes (*any* to *any*)
5	No rule if none of the matches listed above exist.

If no lane change rules are defined in the process object for the current instance of a process, the application looks for a matching lane change rule in the default process object specified in the property settings for the orchestrator. This occurs only if no lane change rules are defined in the current process object. The application does not check the default process object if there is any lane change rule defined in the current process object, even when no match occurs.

When processing crosses a transition between lanes, the application:

- Suspends the process after completing currently executing operators.
- Tries to match a rule defined in the process object or in the default process for the orchestrator in the following order:

Lane Change Rules	Matches	Action
Are defined in the process object	A rule in the process object in the order of priority listed above	Run the rule defined in the process object.
Are not defined in the process object	A rule in the default process object for the orchestrator in the order of priority listed above.	Run the rule defined in the default process object.
	No match.	Ignore the transition.

Then, the application continues processing the process.

Create Horizontal or Vertical Lanes

Swim lanes are oriented either horizontally or vertically, as configured in the process properties. Select the orientation of swim lanes before you start to add operators to a process. You must configure the orientation before you add additional swim lanes to a process because you cannot change the orientation of multiple swim lanes in a process.

Follow these steps:

1. Open and checkout a process.
2. In the Process Designer, click Properties.
3. In the Process Properties palette, expand the Configuration section.
4. Under Lane Orientation, click Horizontal or Vertical.
5. Click Apply.

Manage Swim Lanes

You can add, insert, resize, and remove swim lanes in the Process Designer.

To add a new lane to the right of vertically orientated lanes or to the bottom of horizontally orientated lanes:

1. On the Main Editor panel of the Process Designer, right-click the Process pane.
2. Click Lanes, Add Lane.

   Note: Alternately, you can click the Add Lane button on the toolbar.

   A new lane is added to the Main Editor panel.

To insert a lane anywhere else on the Main Editor tab:

1. On the Main Editor panel of the Process Designer, right-click the lane adjacent to where you want to add the new lane.
2. Click Lanes, Insert Lane, click Insert Lane on Left Side or Insert Lane on Right Side.

   A new lane is inserted.
3. If lanes are oriented horizontally, click Insert Lane Above or Insert Lane Below

   A lane is inserted above or below the existing lane based on your selection.
4. To resize an existing lane, click the separator line between lanes and drag it left or right for vertically arranged lanes or up or down for horizontally orientated lanes.

   Note: The minimum width of a swim lane is 50 pixels.

You can remove a lane from a process by merging it into an existing lane. When merging a lane, you can merge with a lane either to the left or right of a vertically oriented lane or to above or below a horizontally orientated lane.

To remove a lane from a process

1. On the Main Editor panel of the Process Designer, right-click within the lane that you want to delete.
2. Click Lanes, click Merge Lanes, and click either Merge with Left Side or Merge with Right Side.

   Note: If lanes are oriented horizontally, click Merge Lane Above or Merge Lane Below to merge.

Lane Handling Rules

When a process contains multiple lanes, you can define lane change rules. These rules specify operator sequences that are invoked only when running a process crosses a lane boundary. Each rule is defined to occur for one of the following lane transitions:

- From a specific lane to another specific lane (from lane A to lane B)
- From a specific lane to any other lane (lane A to *any*)
- From any lane to a specific lane (*any* to lane B)
- From any lane to any other lane (*any* to *any*)

Lane change rules look like any operator sequence on the Main Editor tab, except each rule starts with a Lane Change operator instead of a Start operator. Lane Change operator properties define the transition between source and destination lanes that invoke the operator sequence that is linked to the Lane Change operator.

Follow these steps:

1. At the bottom of the Process Designer, select the Lane Change Handler tab.
2. From the Operators palette Standard group, drag a Change Lane operator to the Lane Change Handler panel.
3. Double-click the Change Lane icon, and click the Properties pane Change Lane properties panel.
4. In the Name field, type a name to identify the rule.
5. For Source and Destination, select the combination of lanes that triggers the rule.
6. Click Apply.
7. From the palette, drag more operators to the process that completes the rule for the transition, and configure each one as necessary.
8. In the toolbar, click the Save button.

   The new transition rule is complete.

Process Versions

CA Process Automation always runs the checked-in copy of the current version of a process object. When the orchestrator starts running the current version of a Process object, it creates a copy of that version of the process in the automation library. The system processes operators in an instance and creates or references process dataset variables within the instance. Changes to an instance of a process do not affect the base definition of the process. Base definitions are accessed using the Library Browser. You view or edit both the base definition and instances of a version of a process using the Process Designer.

Document a Process

You can generate process documentation. The process documentation can include a graphic of the process design as it appears on the canvas in the Designer tab.

Use comments, your own operator names, and object labels to supplement the process documentation you generate. Appropriate comments, naming, and labels help other designers understand what your process does and how it is constructed. Inserting comments in a process provides details on chains, processes, or regions in a process. Comments remain stationary in a process. Do not use comments to label specific operators because operators are often relocated on the workspace to adjust for links and other operators. Instead, use the Name property in the Information properties of the specific operator to label it.

More information:

Generating and Using Process Documentation (see page 419)

Add Comments to a Process

Use the Comment operator in the Standard group of the Operators palette to add comments to a process. You can change the Name property string for the operator. Comments are important for documenting steps in a process and allow more space than labels.

To add a comment to a process

1. Drag a Comment operator from the Standard group onto the process.

   A comment object appears with a default name of *Comment*.

2. Double-click the comment to open the Comment Properties dialog.
3. Replace the initial text of the comment with the text that you want to appear on the canvas.

   You can change the background color, text format, and alignment.

4. Click OK to apply your changes.
5. Click Save to save your process design.

Set the Name for an Operator in a Process

The operator *Name* property identifies an operator placed in a process. Expressions use the name to access the operator dataset in the format:

```
Process.operator_name.field_name
```

By default, the *Name* property is also used to label an operator in the Process Designer when you turn on the Icon Information option for a process.

When you add an operator to the process, a default name is generated, indicating the task performed by the operator. You can change this text to provide more meaningful or specific information about the operator relative to your system.

To change the name of an operator

1. Double-click an operator in a process or click the operator and choose Properties from the View menu.
2. In the Information properties on the Properties pane, in the Name field, enter a short description for the operator.

   Operator names can be composed of alphanumeric characters and the underscore (_).

Change and Display Operator Information in a Process

The name is included in a text field that optionally labels an operator on a process. The operator Text property defines this field. The Operator Information option on the View menu hides or displays this text field next to operators in a process.

To generate a default value for the Text property, CA Process Automation combines the Operator Name and Operator Parameter settings.

Follow these steps:

1. Double-click an operator in a process.
2. In the Information properties Name field on the Properties palette, enter the text to appear next to the operator in the process.

Note: Use the other settings on the Properties pane Information tab to:

- Specify a custom icon instead of the default icon for the operator.
- Override the object preferences for automatically or manually updated text that is displayed with the operator.
- Set the text position, background color, font, and alignment.

Self-Contained Content

You can embed a process design, runtime instance, automation object, or a particular view into other products and dynamic interfaces using designated URLs. This seemless integration offers the following functionality:

- Other development teams can adopt products and solutions that leverage CA Process Automation and that offer integrated views on one page in one window. For example, as a customer, administrator, or service provider, you can integrate specific portions of the CA Process Automation user interface into existing portals such as web sites, intranets, and Sharepoint repositories.
- As a process designer, you can construct a process workflow that sends email with a direct link to a specific task so that the task can be approved directly.
- Self-contained content facilitated by shortcut links to specific views saves time accessing important and relevant information.

Self-Contained Content Links

The following links support self-contained content in other views, frames, web parts, or portals. You can also send email with these links to facilitate direct views on specific objects.

Instead of [Server URL], use the URL for your CA Process Automation deployment:

```
http://server:port/itpam/Web.jsp
```

Instead of <path>, specify the absolute path to the automation object and its name in the Library Browser:

```
/MyProjectFolder/Folder_1/MyStartForm
```

Note: These links are case-sensitive.

Automation Objects:

CA Process Automation objects are URL addressable. These tear off URLs can be embedded within iframe or portlets of other products. For example, you can trigger a process and have the tracking UI for that instance embedded into a custom portlet.

Interaction Request Form

```
[Server URL]?page=Form&refPath=<path>
```

Start Request Form

```
[Server URL]?page=Commander&refPath=<path>
```

Schedule

```
[Server URL]?page=Agenda&refPath=<path>
```

Calendar

```
[Server URL]?page=Calendar&refPath=<path>
```

Custom Icon

```
[Server URL]?page=CustomIcon&refPath=<path>
```

Custom Operator

```
[Server URL]?page=Template&refPath=<path>
```

DataSet

```
[Server URL]?page=Dataset&refPath=<path>
```

Package

```
[Server URL]?page=C2OPackage&refPath=<path>
```

Process Watch

```
[Server URL]?page=AppMonitor&refPath=<path>
```

Resource

```
[Server URL]?page=Resources&refPath=<path>
```

Process

```
[Server URL]?page=processeditor&refPath=<path>
```

Other Entities:

Process Watch (default)

The following format accesses the Process Watch as seen on the Operations tab:

```
[Server URL]?page=processwatch
```

Task Lists

The following format accesses *all* tasks as seen on the Operations tab:

```
[Server URL]?page=tasklist&tasklist=alltasks
```

The following format accesses *group* tasks as seen on the Operations tab:

```
[Server URL]?page=tasklist&tasklist=grouptasks
```

The following format accesses *my* tasks as seen on the Operations tab or the Home tab:

```
[Server URL]?page=tasklist
```

Process Instances

The following format accesses a runtime instance of a process in its own window:

```
[Server URL]?page=runtimeeditor&ROID=<runtime_instance_ID>
```

Forms

The following format accesses the Start Request Form or Interaction Request Form that is used to reply to a task:

```
[Server
URL]?ROID=<runtime_object_ID>&tasklist=ALL_TASK_FILTER
&page=replytask
```

The following format accesses a list of Start Request Form instances as seen on the Operations tab:

```
[Server URL]?refPath=<path_to_SRF_object>&page=srflist
```

Object Versions

To access a specific checked-in version of any library object except a process, append the *versionid* parameter to the URL:

```
&versionid=<version_number>
```

For example, to access version 3 of the MyResource object:

```
[Server
URL]?page=Resources&refPath=/TestFolder/MyResource&versioni
d=3
```

Process Versions

To access a specific checked-in version of a process, append the *version* parameter to the URL:

```
&version=<version_number>
```

For example, to access version 4 of the MyProcess object:

```
[Server
URL]?page=processeditor&refPath=/TestFolder/MyProcess&versi
on=4
```

Navigate to a Specific Part of a Process

When working with long processes, use the Navigator window to help you adjust your current view. Panning a smaller view of a process is more convenient than scrolling through the entire process in the main window.

Follow these steps:

1. Open a process.
2. In the Process Designer, check Navigator from the View menu.

   The Navigator window shows a miniature image of your process.
3. Drag the rectangular frame over the portion of the process that you want to view.

   The main window shows an enlarged view of the selected area.
4. To adjust the miniature view of your process, drag the square at the bottom right corner.
5. To resize the Navigation palette, drag any edge or corner the same way you resize any palette.

Multi-Tenancy and CA Process Automation

In a multi-tenant deployment, administrators want to control user access to process instances based on a tenant or a set of tenants. You can use common CA Process Automation processes across multiple tenants. This feature allows for access control of process instances. You can prevent a user with access to one tenant from accessing an instance related to a different tenant. This feature also results in enhanced process duplication and synchronization and reduces related maintenance tasks.

You can limit access to the process instances, based on access to a tenant, using common processes across tenants. This is accomplished by setting a tenant ID (as a new well-defined variable) as part of a process instance. Then validate access to that tenant ID when access to that process instance is requested.

Note: Setting the tenant ID is up to the process designer as part of designing the process or as part of the input parameters to the process. Any process that does not have a tenant ID already set must follow the current access control restriction based on the process definition.

Make a Process Aware of Multiple Tenants

Making an existing process aware of multiple tenants involves the following steps:

- Add a security-related variable to the context of a running process instance at run time
- Create a policy for multi-tenancy

Follow these steps:

1. To add a security-related variable to the context of a running process instance at run time, do one of the following actions:
   - Add a security-related variable to the context of a running process instance at run time using JavaScript:

     ```
     Process.SECURITY_CONTEXT_ID=<ID>;
        Process.SECURITY_CONTEXT_GRP=[set the product group or family];
     ```

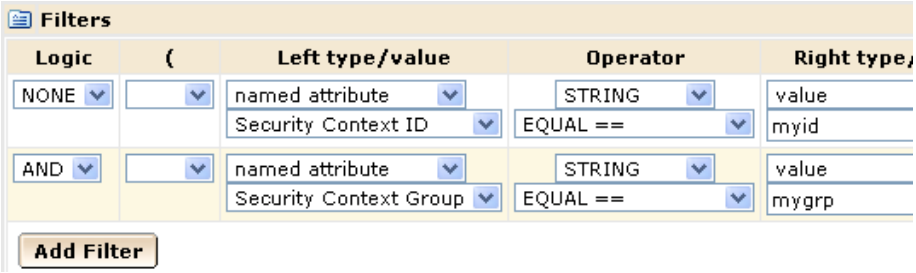
     For example:

     ```
     Process.SECURITY_CONTEXT_ID="myid";
        Process.SECURITY_CONTEXT_GRP="mygrp";
     ```

   - Pass values to the process instance using <params> tags while starting the process through the executeProcess web service. For example:

     ```
     <executeProcess xmlns="http://www.ca.com/itpam">
        <flow>
        …
        <params>
          <param name="SECURITY_CONTEXT_ID">myid</param>
          <param name="SECURITY_CONTEXT_GRP">mygrp</param>
        </params>
     </executeProcess>
     ```

2. To create a policy for multi-tenancy, complete the following steps:
   a. Add a policy in EEM using Object as the Resource Class Name.
   b. Specify the Identities for which this policy is valid.
   c. Specify Resources so that this policy matches the path of the process for which you are creating the policy.
   d. Specify the required policy actions.
   e. Add filters and specify values for the Security Context ID and Security Context Group named attributes.



This policy applies to all process instances when the values of the named attributes match the ones in the process instance at run time.

Inherit Security in Subprocesses

Subprocesses automatically inherit the values of security attributes (that is, SECURITY_CONTEXT_ID and SECURITY_CONTEXT_GRP) from their parent process.

Add Variables at Time of Initialization

You can specify new values in the Process Dataset Initialization Code option of the Run Process service operator. Values for the security context specified in the Process Dataset Initialization Code take precedence over those specified in the parent process.

Multi-Tenant Processes Using Process Watch

You can view runtime process instances in Process Watch if a policy for multi-tenancy allows it. You can also perform required actions such as Open, Abort, Suspend, and so on, if a policy allows it.

The CA Process Automation Code Editor

The advanced Code Editor in CA Process Automation lets you easily create, edit, and debug code for various scripting languages in the following places:

- The pre- and post execution code of an operator.
- The Source Code input parameter for the Run Script operator.

- The Source Code input parameter for the Run JavaScript operator.
- The Data Initialization Code parameter for the Start Process operator.
- Form script editors for the interaction request form and start request form.
- The Source code input parameter for the Invoke SOAP Method operator.
- The Source code input parameter for the Invoke SOAP Method Async operator.
- The Source code input parameter for the Apply XSLT operator.
- The Source Code input parameter for any custom operators where the base operator is one of the previously mentioned operators.

The Code Editor supports the following languages:

- JavaScript (.js) (available for both client-side JavaScript and server-side JavaScript)
- XML (.xml)
- SQL (.sql)
- Java (.java)
- batch (.bat)
- Windows Command File (.cmd)
- Visual Basic script (.vbs)
- Windows Script Host Settings (.wsh)
- PerlScript File (.ps)
- HTML (.html)
- Shell (.sh)

Some additional operators contain text editors that use the Code Editor functionality, when applicable. When the specific language used within the operator is supported by the Code Editor and CA Process Automation, their text editors automatically adapt to the specified language. Specifically, their text editors highlight language-specific keywords. However, if the operators use a language that is *not* supported by the Code Editor or not implemented by CA Process Automation, a plain-text editor is available for coding.

If a process is checked in, the Code Editor starts in read-only mode and the editor and toolbar are disabled. However, some toolbar buttons (for example, Search, Help, and Script validation) are still available.

The Code Editor includes the following features:

Drag and Drop

Drag any file or selected text (from inside or outside of CA Process Automation) and drop it into the editor to populate the editor.

Search/Replace

Locate specific code (with Case Sensitive and Regular Expressions options), then replace specific instances or all occurrences.

Undo

Reverse your previous action.

Redo

Reverse your previous Undo action.

Jump to line number

Advance to a specified line of code.

Reformat selection

Adjust indentation of the selected text.

Reformat whole document

Adjust indentation of the entire code document.

Import from file

Browse to locate a specific file to import into the editor.

Supported editor modes for imported files include:

- .bat
- .cmd
- .js
- vbs
- .wsh
- .ps1
- .sh
- .pl
- .xml

CA Process Automation provides an error if any other file extensions are imported in the advanced Code Editor.

Note: Any file type can be imported for the generic editor.

Export to file

Save the file to your CA Process Automation downloads.

Show Error Log

Display errors in your code. This feature is only available in the following cases:

- The pre- and post execution code of an operator.
- The Source Code input parameter for the Run JavaScript operator.
- The Data Initialization Code parameter for the Start Process operator.

The editor highlights incorrect code in a scrollable error log at the bottom of the editor. Each error contains the line number and a hyperlink to navigate to the specific line of code. When you fix the error, the editor automatically clears it from the error log.

Note: The code should be entered properly, in valid expressions. However, you can save code even if it contains errors, although CA Process Automation issues a warning. Some validations (such as the following examples) are skipped:

"strict"

Expects 'Use Strict' in the JavaScript function declaration.

"eqeqeq"

Expects '===' instead of '==' in the condition.

"smarttabs"

Validates a mix of tabs and spaces.

Code folding

When you click the line number of any "{", the Code Editor folds the code up to its matching bracket "}".

When editing XML, code folding is based on the matching tags. You can expand or collapse the XML fragment based on the matching tags. The Code Editor automatically closes the ending tags for you while editing.

The Code Editor also uses the following conventions:

- Displays line numbers.
- Removes empty lines.
- Highlights matching text across the document when you select specific text.
- Highlights the current line.

Color Coding

The Code Editor uses the following colors to display JavaScript:

Orange

Indicates:

- Dataset keywords for server-side JavaScript
- Form variables for client-side JavaScript

Example: `Datasets`

Blue

Indicates CA Process Automation system functions. Client-side and server-side JavaScript have different sets of system functions.

Example: `adjustResourceVals`

Purple

Indicates JavaScript keywords.

Example: `comment`

Shortcuts

The Code Editor uses the following shortcuts:

- Ctrl+C (copy)
- Ctrl+V (paste)
- Ctrl+X (cut)
- Ctrl+Z (undo)
- Ctrl+Y (redo)
- Ctrl+A (Select All)
- Delete (to delete selected text)
- Home (to go to the beginning of the current line)
- End (to go to the end of the current line)

The following shortcuts are only valid for JavaScript:

- Press Ctrl+Space to display the following items:
  - A list of dataset keywords (displayed first) and JavaScript keywords for server-side JavaScript.
  - A list of form variables (displayed first) and JavaScript keywords for client-side JavaScript
- Press Ctrl+Alt to display a list of system functions. Client-side and server-side JavaScript have different sets of system functions.

  Note: Only client-side JavaScript can be used in the Forms Designer (for start request forms and interaction request forms). CA Process Automation-specific system functions are not available. However, form functions are available.

Chapter 5: Operators and Icons

CA Process Automation carries out the instructions in the operators that you add to a process or to a schedule. This section describes how to configure operators after you have added them to a process or schedule object. This section also includes information on custom operators and connectors.

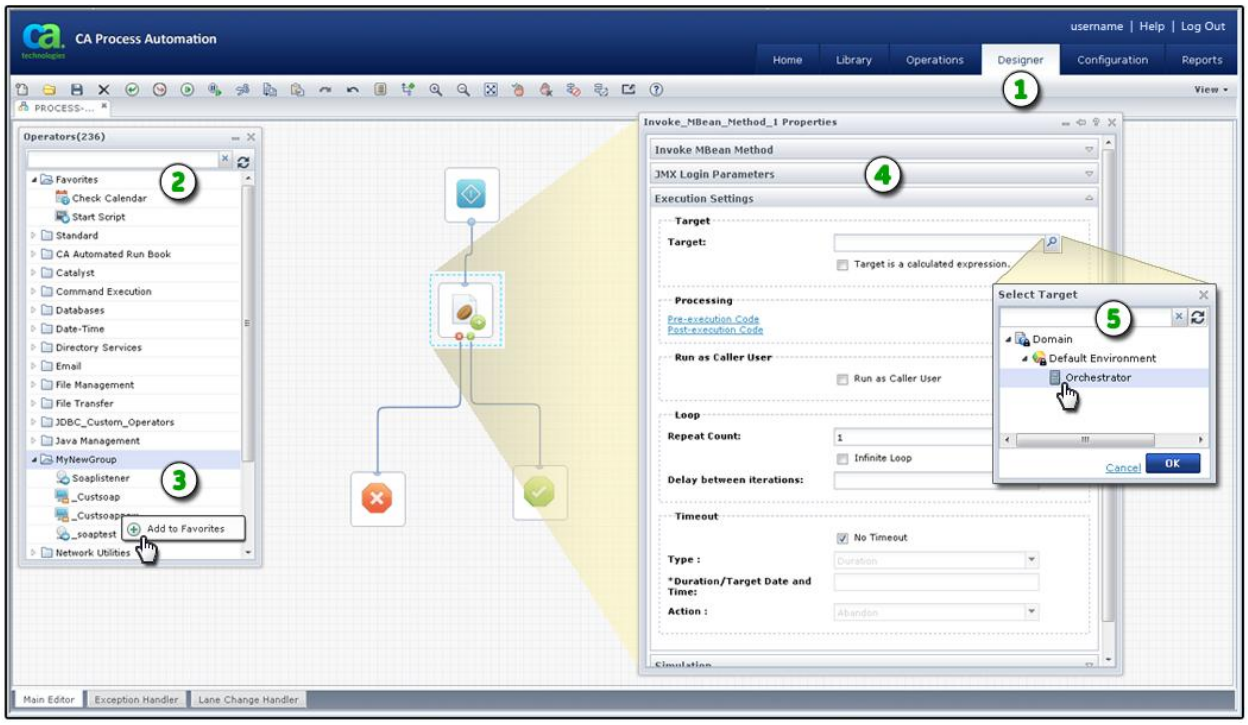
This section contains the following topics:

Operators (see page 142)
Operator Icons (see page 178)

Operators

A previous section of this guide introduced process operators (see page 93). To learn more about each operator, see the *Content Designer Reference*.

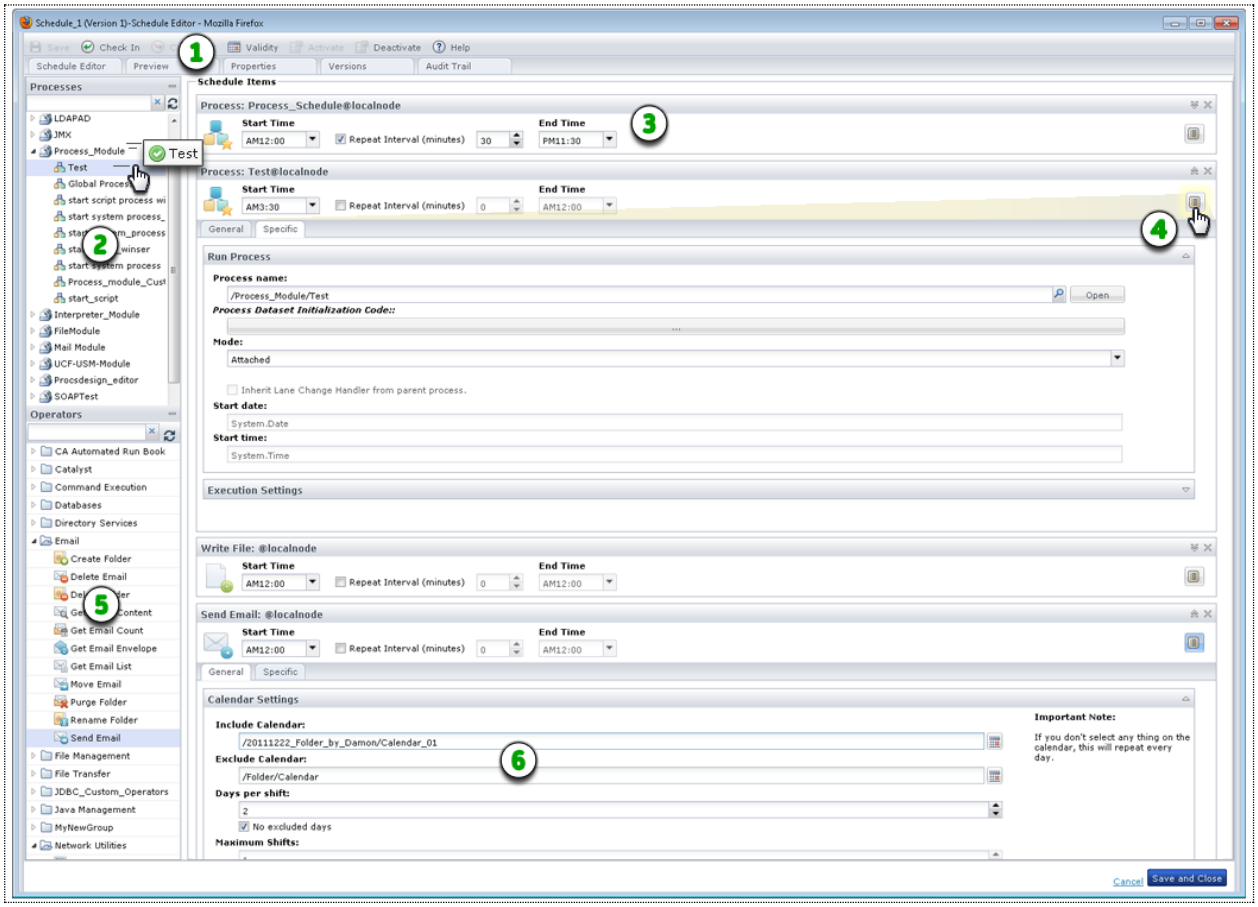
This section provides general guidelines for configuring properties and working with different types of operators, including custom operators.



Item:	Description:
1	**Designer Tab and Toolbar**: When you open a new process or you edit an existing one from the Library Browser, the Designer tab appears. In the toolbar, click View and select the Operators and Properties palettes.
2	**Operators Palette**: Search for operators by folder or name.
3	**Custom Operators in a Custom Group**: Drag-and-drop a custom operator from this palette to your process layout. Right-click an operator to add or remove it from the Favorites group.
4	**Properties Palette**: Use this palette and its buttons, links, and windows to manage the properties of the currently selected operator. Expand and collapse panels as you work. Each panel consists of similar properties.

Item:	Description:
5	**Additional Windows**: Some properties appear in their own windows.

You also work with operators to configure schedules.



Item:	Description:
1	**Schedule Editor**: When you open a schedule from the Library Browser, the Schedule Editor appears. Use the toolbar to activate, set validity of, check in, or save the schedule.
2	**Processes Pane**: Select the processes to include and drag them to the Schedule Items page.
3	**Scheduled Process in Queue**:Set the duration and frequency for running the process in a single day.
4	**Process Properties**: Click Properties to view process properties on the General and Specific tabs.

Item:	Description:
5	**Operators Pane**: Select the operators to include and drag them to the Schedule Items page.
6	**Operator Properties**: Click Properties to view operator properties on the General and Specific tabs.

Configure Operator Properties

Operator properties appear in the Process Designer on the Properties palette. Similar fields are organized in group boxes on expandable panels with familiar titles such as Process, Execution Settings, Simulation, and Icon. Some panels and groups are common to all operators and others are unique to a particular operator. For example, the properties for every operator include an Icon panel. The Asynchronous SOAP Call Data panel however, only appears for the Invoke SOAP Method Async operator.

Operator properties are also available when you include operators in Schedule objects.

Follow these steps:

1. Click the Library tab.
2. Open a process object or open a schedule object.

   Processes open in the Process Designer. Schedules open in the Schedule Editor dialog.
3. In the toolbar, click Check Out if the object is not already checked out.
4. Add an operator to the process or schedule:
   a. For a process, drag an operator onto the process layout from the Operators palette.
   b. For a schedule, collapse the Processes pane, expand the Operators pane, and drag an operator into the list of Schedule Items.
5. View the operator properties:
   a. In a process, double-click the operator.

      The Properties palette appears.
   b. In a schedule, click Properties and then click the General or Specific tabs.
6. Click Save.

   The properties values you entered for the selected operator are saved.

Note: For detailed information about operator properties, see the *Content Designer Reference Guide*.

Auto Recovery

The *Operator Recovery* feature controls what processes can recover as part of the manual or automatic recovery of a touchpoint, orchestrator, or host group. You can enable or disable this feature.

- Enable recovery on a process and define an exception handler that notifies you of a failure.
- Disable recovery on a process and define an exception handler that remedies the system error. For example, the handler could run the operator on a different touchpoint.

  Note: For more information see Exception Handling (see page 399).

CA Process Automation checks the *Enable Operator Recovery Settings* for each object before running a process instance. CA Process Automation uses the settings to determine whether the process instance is recoverable. The process instance is not affected if the object settings change after running the process instance.

If enabled, the recovery procedure acts on operators and recovers process instances that fail with a SYSTEM_ERROR. The operators' processes must be set to be recoverable and must be in the *Blocked, Running,*or *Waiting* state when the recovery is triggered. Operator recovery resets the operator and then resumes the processes. An operator in the *Blocked* state should resume operation and run again during touchpoint recovery.

New processes created in CA Process Automation version 4.0 or later have this option selected by default. *Enable Operator Recovery* is unchecked by default only for existing processes created before a CA Process Automation version 4.0 upgrade.

Java and External JARs

The Java module allows you to specify Java code to run inside a *BeanShell* Interpreter in the CA Process Automation JVM. BeanShell is an embedded dynamic Java source interpreter, scripting language, and flexible environment. Using the CA Process Automation Java module and operators, you can:

- Import and reference JAR files at the module or operator level.
- Configure global settings for all Run Java Code operators or specific settings for a particular operator. You can specify paths to the external JAR files that operators can use. You can also set the default log setting in the module. For each operator, you can specify the code that you want to run, the input parameters, and the output variable names. Log settings you specify for a single Run Java Code operator override the module-level log settings.
- Write Java code that references classes in these JARs. CA Process Automation automatically creates new Java Object data types when you run a Java program. Because new Java objects are invoked inside the Run Java Code operator, you do not need to use any create or destroy object methods. After execution of the operator, CA Process Automation automatically collects any garbage resources, classes, and objects in memory.
- Run the Java code using the Run Java Code operator. You can invoke classes in an external Java Archive (JAR) file from a Run Java Code operator. Use this operator to leverage the functionality that your existing Java code provides.
- Save Java objects to the operator dataset to make them available to subsequent Run Java Code operators. You have the option of saving an entire Java object in the operator dataset before the end of execution of the operator. You can then leverage the Java object saved in the operator dataset in subsequent operators by passing it to them.

Configure the Java Module

You can invoke classes in an external Java Archive (JAR) file from all Run Java Code operators. Configure the Java module to apply settings to these operators to leverage the functionality that your existing Java code provides. Then use the Run Java Code operator to create a Java object.

Follow these steps:

1. Specify the JARs you want to work with on a CA Process Automation orchestrator or agent machine. Locate the Default External Jars field for the module. Enter the paths to the external JARs to load for use by all the Run Java Code operators running on the orchestrator or agent. For each path, you can enter:

   a. The full path to a JAR file that resides on the machine where the orchestrator or agent is running. The full path starts with either of the following slash marks:

   ```
   /
   \\
   ```

   You can also designate the full path using a regular expression that starts with one character, then a colon (:), and then the rest of the string, including dot syntax as in:

   ```
   ^.:.*
   ```

   b. The full path to a JAR file available over http:// or https://. The path does not require authentication and is not accessible through an http proxy.

   c. A relative path to a JAR file that was uploaded to the CA Process Automation User Resources folder. Unless you specify a full path, the application considers the path that you enter to be a relative path.

   The Java Module Class Loader, which all the running *Run Java Code* operators of the Java module share, loads the JARs that you list in this field. The JARs are loaded once when executing the first Java module operator, and later after any changes made to the default external JARs. Any JAR entered in this list are available to the Java code that the *Run Java Code* operators run. Any classes you define in the operator-level JARs override the same classes specified in the module-level JARs.

2. In addition to external JARs, enter paths to any .class files that any Run Java Code operators running on the orchestrator or agent are using.

   a. For .class files in an unnamed package, enter a path that ends with the directory that contains the .class files. For example, MyAccount.java does not belong to a package, and MyAccount.class is in the following location:

   ```
   C:\java\tests\MyAccount.class
   ```

   Set the operator to use the following path:

   ```
   C:\\java\\tests
   ```

b. For .class files in a named package, enter a path that ends with the directory that contains the *root* package. The root package is the first package in the full package name. For example, MyAccount.java belongs to package com.ca.tech. MyAccount.class is at the following path:

```
C:\java\othertests\com\ca\tech\MyAccount.class
```

Set the operator to use the following path:

```
C:\\java\\othertests
```

Note: Specify the path to a folder as a full path or as a relative path to CA Process Automation User Resources. Do not specify an http path. Specify the path to a folder 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).

3. (Optional) Upload the JARs you want to work with to the CA Process Automation User Resources.

   CA Process Automation automatically mirrors the JARs.

   Note: Resources, including user resources, are mirrored within the mirroring interval of the orchestrator or agent. Ensure that the JAR files you upload in the user resources are already mirrored before using them in the Java module operators.

4. Verify that the JAR files containing the Java classes you want to work with are available to the orchestrator/agent whose touchpoint is running the operator.

5. (Optional) Configure the module default logger. You can override this configuration at the operator level.

6. (Optional) Override any module level settings by configuring individual operators. See the next procedure, Configure a Run Java Code Operator (see page 149).

7. Run any Run Java Code operators. The Java module captures any exceptions or errors that are encountered during an operation and alerts the user in the Reason field of the problem operator.

Configure the Run Java Code Operator

You can invoke classes in an external Java Archive (JAR) file from a Run Java Code operator. The Run Java Code operator can use both operator-level and module-level JARs. You can also configure the Run Java Code operator to override module-level settings. Use the operator to leverage the functionality in your existing Java code.

Follow these steps:

1. If you did not already configure the Java module, complete this task first. See Configure the Java Module (see page 147).
2. Configure the Run Java Code operator.

   Specify the paths to the external JARs that the Run Java Code operator uses. For each path, you can enter:

   - The full path to a JAR file that resides on the host where the orchestrator or agent is running. The full path starts with either of the following slash marks:

     `/`

     `\\`

     You can also use a regular expression (including dot notation) that starts with one character, then a colon (:), then the remaining string as in:

     `^.:.*`
   - The full path to a JAR file available over http:// or https://. The path cannot require authentication and must not be accessible through an http proxy.
   - A relative path to a JAR file that resides in to the CA Process Automation User Resources folder.

   Unless you specify a full path, the application interprets the path that you enter as a relative path.

   The operator loads the JARs listed in this field and makes them available to the Java code in the running operator. The classes defined in these JARs override the same classes specified in the module-level JARs.
3. In addition to external JARs, enter paths to any .class files for the Run Java Code operator.

   a. 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 in:

      `C:\java\tests\MyAccount.class`

      Set the operator so it uses the following path:

      `C:\\java\\tests`

b. For .class files in a named package, enter a path that ends with the directory that contains the root package. The root package is the first package in the full package name. For example, if MyAccount.java belongs to the package com.ca.tech and MyAccount.class is in:

```
C:\java\othertests\com\ca\tech\MyAccount.class
```

Set the operator so it uses the following path:

```
C:\\java\\othertests
```

Note: Specify the path to a folder as a full path or as a relative path to CA Process Automation User Resources. Do not specify an http path. Specify the path to a folder so it loads .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).

4. (Optional) Upload the JARs you want to work with to the CA Process Automation User Resources.

   CA Process Automation automatically mirrors the JARs.

   Note: Resources, including user resources, are mirrored during the mirroring interval of the orchestrator or agent. Ensure that the JAR files you upload in the user resources are already mirrored before using them in the Java module operators.

5. Specify the code to run.
6. Specify the input parameters to pass to the Java code.
7. Specify the output names of the variables created in the operator Java code. The output variable names must be saved in the operator dataset when the code finishes running.

   Note: CA Process Automation serializes Java objects that are not Boolean, date, integer, number, string, character, or an array of these types and saves them as JavaObjects.

8. (Optional) Specify the logger setting of this Run Java Code operator. These settings override the module-level logger settings.
9. Run the Run Java Code operator.

   The Java module captures exceptions or errors that are encountered during an operation and alerts the user in the Reason field of the problem operator.

Using a JavaObject

Java objects are saved after a Run Java Code operator has completed in a *JavaObject* data type. You can use a *JavaObject* dataset variable in the following ways:

- Pass the *JavaObject* dataset variable in the parameters list of the Run Java Code operator.
- Pass the path of the *JavaObject* dataset variable in a string variable from an Interaction Request Form or Start Request Form. For example:
  `JavaObjectPath = Datasets["/GlobalDatasets"].acct`

  Then, you can use the *eval* function when passing the JavaObjectPath variable to the parameters list of the Run Java Code operator. For example:
  `eval(Process.IRF.JavaObjectPath)`
- Copy a JavaObject in Javascript. You can also assign one JavaObject to another.

  Note: Do not modify the actual value of a CA Process Automation JavaObject (the serialized string of the original Java object). Your changes could prevent the Run Java Code operator from loading the JavaObject.

Observe the following constraints when working in JavaScript:

- Similar to passwords, you cannot concatenate a JavaObject to a string.
- You cannot load a *JavaObject* and use its methods in JavaScript. Instead, pass the *JavaObject* in the Run Java Code operator parameters list and access it in the operator Java code.

The Java code that you write can consist of normal Java statements and expressions. You can also define your own methods and use them inside the code. For example:

```
// 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(1000.00);

// Use the public methods of the MyAccount object
acct.addFunds(34.44);
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));
```

After you run this Java code, the log message shows the account balance, the date, and the time:

```
Account Balance: 124.44 on date: Wed Jul 13 12:53:37 EDT 2011
```

Custom Operators

You can define c*ustom operator* objects that users can add to their processes or schedules as they would add any other operator. You create custom operators by reusing an existing base operator. Configure the settings of that base operator in the Properties palette. Use custom operators to share new functionality across your IT organization.

For example, you could use the Get SNMP Variable operator as the basis for a custom operator that retrieves specific information from a network router. Configure the appropriate SNMP variable OID and community string, then set the properties so a user can configure the IP address of the SNMP variable. After it is published, users can add the custom operator to a process or schedule, then configure the IP address to access specific network router information. Users do not have to know the SNMP variable OID or the community string. Those values have already been configured and are probably hidden from the user.

A more powerful use of custom operators is to develop interfaces to new enterprise applications and systems including:

- Mission-critical applications; organizations often develop and deploy such applications internally
- Web services (SOAP and RESTful)
- Command line and scripting applications
- Databases
- Java

Develop custom operators that perform common actions that interface with your applications and systems. Reuse your own custom operators to enable your organization to work easily with these applications and systems from automated processes. Users can perform actions without expert knowledge of all application and system interfaces.

Custom operators also provide the advantage of easily adapting to changes in your IT environment. For example, you need change only the original custom operator object if IT environment changes require you to change how you interface with an application. All processes and schedules that use the custom operator automatically apply the changes in the latest designated version. Therefore, changes in the IT environment are transparent to all processes and schedules.

Create a Custom Operator Object

You can create custom operators in any library folder.

Follow these steps:

1. Click the Library tab.
2. Click a folder.
3. In the toolbar, click New and then choose Custom Operator.

   The *Select Base Operator* dialog appears.
4. Select the base operator for your custom operator and click OK. Expand folders or enter an operator name to search.

   A custom operator object with a default name is created in the library.
5. Rename the custom operator.

Note: The application identifies custom operators by a unique location in the library using the object path and name. Avoid changing the location and custom operator object name if it is currently being used in a process. Renaming or moving a custom operator while it is used in a process can result in the loss of changes or updates.

Custom Operator Properties

You can configure a custom operator by modifying the default settings available for the base operator. You can also add input pages and parameters to present a user of the custom operator with specific input settings.

Seven main types of properties exist for each custom operator:

- Form
- Preview
- Settings
- Dataset
- Group Config
- Properties
- Versions
- Audit Trail

For each setting, you would typically:

- Leave it blank and let the user configure it when they use your custom operator.
- Configure it with a predefined value and mark it as invisible, which hides the setting altogether from the user when they use your custom operator.

You can also:

- Configure it with a predefined value, but let the user change the value.
- Configure it with a predefined value and mark it as read-only, which lets the user see but not edit the value.

Example: A Basic Custom Operator

You can create a custom operator that retrieves a specific piece of information from a network router using the Get SNMP Variable operator as its base. As part of your custom operator, configure the appropriate SNMP variable OID and community string, and then set the properties so a user can configure the IP address of the SNMP variable. The Get SNMP Variable operator has the following specific settings:

- Agent Host (IP Address)
- Community
- Object ID (OID)
- Retry Count
- Timeout
- SNMP Version

You would typically configure predefined values and mark as invisible all the settings except for Agent Host. Another designer using this custom operator only sees the Agent Host setting in the Properties palette. All other settings are hidden. Other users can specify the Agent Host to determine the network router where information is sourced. But other users do not need to know the other settings for a Get SNMP Variable operator. As long as they configure the correct IP address, the information in question appears.

Example: An Advanced Custom Operator

As for a more advanced example, you can create an interface to an in-house application using scripting. In this example, you would use the Start Script operator as the base for your custom operator. Then, you would typically specify the extension of the script, the script itself, and other settings such as the parameters to pass to the script. You would typically set these parameters (in fact, every parameter that comes from the base operator) as invisible. You can also create your own settings page to ask a user of your custom operator for some settings that are specific to your operator. As in the simple example above, an end user can then use your custom operator to act upon your in-house application. This technique extends integrated processes without the need to know specifics about how you interface with that application.

More information:

Custom Operator: Properties Tab (see page 171)

Custom Operator: Form Tab

When designing a custom operator, you can permit custom parameters and pages to receive input from end users as they configure your custom operator in a process or schedule. Use the Custom Operator Form tab to:

- Add, remove, and rename property pages
- Add, remove, and rename parameters on the property pages
- Configure the characteristics of a parameter
- Reorder parameters on a property page
- Move parameters between pages

Each custom operator can have one or more pages of parameters that are based on its ancestor or base operator. You can modify and configure these parameters. For example, if the base is the Run Script operator, you can configure the following parameters:

- The *scriptType* field to define the extension of the script
- The *inLineScript* field to define the script itself
- The other fields and parameters standard for the Run Script operator

Add Property Pages

The Custom Operator Settings tab includes the standard base operator properties. These settings correspond to the parameters found on the Properties palette in the Process Designer. Use the Properties palette to configure the custom operator's base properties and the custom properties that you assign to it.

A custom operator may require additional parameters as input into the function of the operator. You can add property pages to group these additional parameters. When you add pages to the custom operator on the Form tab, they appear as expandable sections in the Properties palette of the Process Designer.

Follow these steps:

1. In the Library Browser, double-click a custom operator.
2. In the toolbar, click Check Out.
3. In the Custom Operator designer window, click the Form tab.
4. In the Form Elements pane, expand all of the following:
   a. Form Elements
   b. Page Layout under Form Elements
   c. Page Layout under your custom operator
5. Drag a Page element from the Form Elements Page Layout down to the Page Layout level for your custom operator.
6. Click the new page and then click Rename in the toolbar. Rename the new page to a meaningful identifier.

   Each page name corresponds to a named expandable section in the Custom Operator Properties palette in the Process Designer.
7. The order of the pages in the Properties palette also corresponds to the order shown on the Form tab. Select a page and click Move Up or Move Down from the toolbar to move the page up or down in the list.

To remove a property page including all parameters:

1. Select the page.
2. In the toolbar, click Delete.

Add Custom Parameters

After creating property pages, you can add parameters (fields) to them. Custom parameters are often used to give users a different presentation for a parameter needed for the base operator. For example, you can add a parameter to present a list of values that you want users to select from, rather than using an edit box with no defined values. Parameters are also used to capture input that is then used in an expression to calculate one or more base operator parameters. Also, parameters are used to present an interface to users with appropriate terminology.

Unless you set their Read Only, Disabled, or Hidden properties to true, custom parameters that you add to your custom operator are visible and configurable by end users in your Custom Operator Properties palette in the Process Designer or in the Schedule Editor.

Follow these steps:

1. In the Library Browser, double-click a custom operator.
2. In the toolbar, click Check Out.
3. In the Custom Operator designer window, click the Form tab.
4. In the Form Elements pane, expand all of the following:
   a. Form Elements
   b. Page Layout and any pages under your custom operator
5. Drag a form element from the Form Elements down to the page for your custom operator. You cannot drag a field directly onto the form layout.
6. Click the new field to edit its properties. Rename the new field to a meaningful identifier using the Label property. Use the Rename toolbar button to rename certain fields such as check boxes.

   Each field name corresponds to a named field in the Custom Operator Properties palette in the Process Designer.
7. The order of the fields in the Properties palette also corresponds to the order shown on the Form tab. Select an element and click Move Up or Move Down from the toolbar to move it up or down in the list.

To remove a parameter:

1. Select the page.
2. In the toolbar, click Remove.

Invisible Parameter Option

CA Process Automation allows you to configure and hide custom operator parameters from the end user. You can set the custom operator parameters to be invisible, read-only, or editable at run time. To pass information to the custom operator parameter, add macros to the custom operator. You can hide input parameters from the user and use macros to access the input values of the hidden custom operator parameters. Changes you make to custom operator parameters cascade down to all the processes using the custom operator.

Unless you set their Read Only, Disabled, or Hidden properties to true, custom parameters that you add to your custom operator are visible and configurable by end users in your Custom Operator Properties palette in the Process Designer or in the Schedule Editor.

Expand Macro in the Value Property

To use a custom parameter to define the value of a base operator parameter (a typical reason for using custom parameters), you must treat it as a macro. Use the *Expand macro in the value* property to permit user input for a custom operator field to be used as the value for the base operator parameter.

To do this, set the Hidden property first. Then configure the base operator parameter with the Expand macro in the value option. CA Process Automation searches the base operator parameter values (for any parameter with the Expand macro in the value set) for any custom parameter name, and replaces the custom parameter name with the customer parameter value. While this is a powerful feature, take care to determine which base operator parameters should have the Expand macro in the value option set and in naming your custom parameters so that they are unique enough that you do not accidentally replace a string with your custom parameter sharing the same name.

For example, you created a custom operator using the *Run Script* operator as a base. The script is defined to pass some parameters based on input supplied by a user of the custom operator. A form field is added to obtain this input from the end user and that field is named *for*. In the base operator, add one parameter and enter the value *for* and set the *Expand macro in the value* property to *true*. This correctly passes the user's input configured in the custom operator form field to the script as a parameter.

If a user enters the value *Steve* for this custom parameter, the script would receive one parameter with the value *Steve*. However, if you set the *Expand macro in the value* property for the base operator parameter *inLineScript*, this replaces the word *for* anywhere in the script with the word *Steve*. This is undesirable, as the scripting code for any For loops would be accidentally replaced with an unintended term, resulting in a syntax error in your script.

More information:

Custom Operator: Properties Tab (see page 171)

Test the Custom Operator Interface

After you add and configure pages and parameters, you can use the *Test* feature to preview the Custom Operator Properties pane.

Follow these steps:

1. Click Test on the toolbar.

   The parameters appear on the tabs in the same order that is listed on the corresponding pages in the Custom Operator Parameters palette.

2. Review the pages and parameters.

   You can preview the end-user view of the customer operator and available options for it.

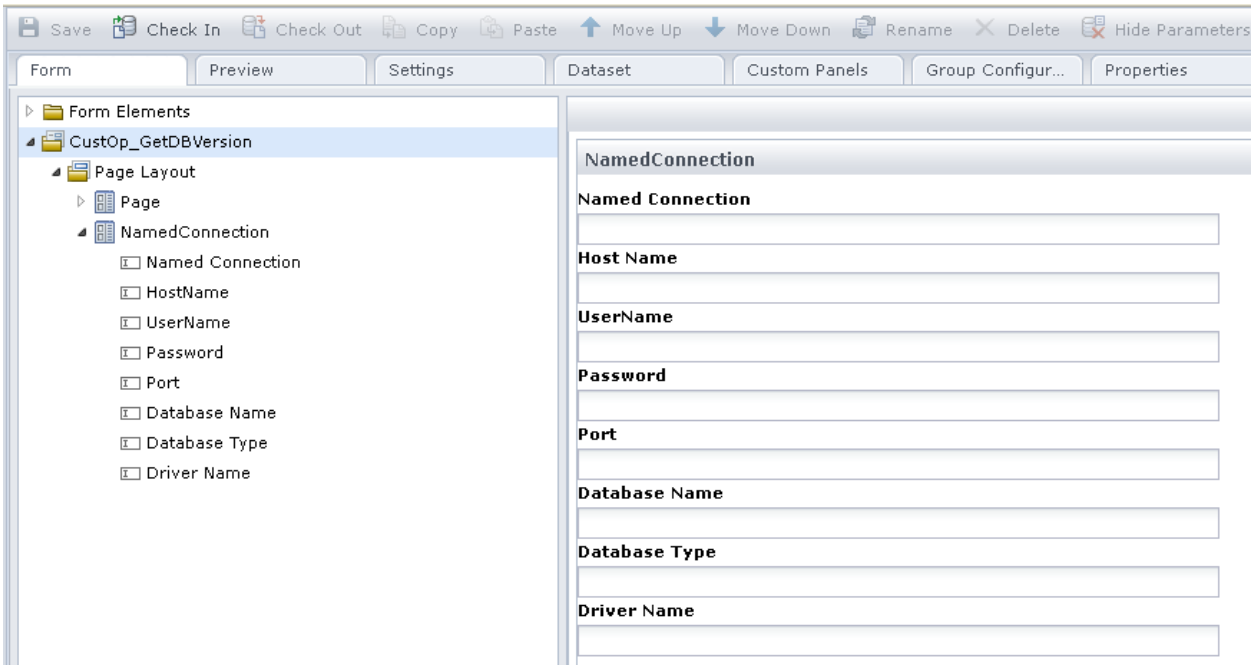
Example: Retrieve Valuemap Array Values with an Operator System Function

This example uses the following operator system function to retrieve the values of a valuemap array. The function returns one field or column from an array that is based on the provided parameters.

```
getValueFromValueMapArray(groupName, arrName, fieldName,
fieldValue, requiredFieldName)
```

Follow these steps:

1. Create a custom operator named CustOp_GetDBVersion with Get Version as the base operator.
2. Design the form as the following illustration shows:

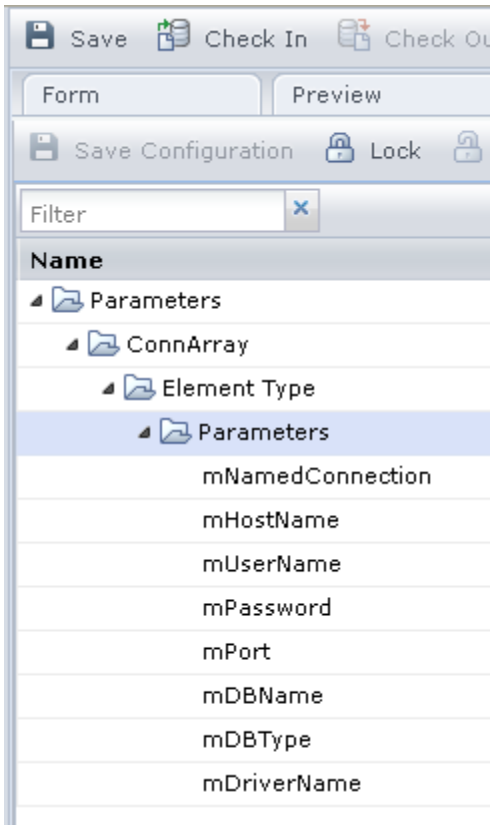


Name the form elements as mNamedConnection, mHostName, mUserName, mPassword, mPort, mDBName, mDBType, and mDriverName. Ensure that the form elements names are similar to the variable names defined in the Group Configuration tab.

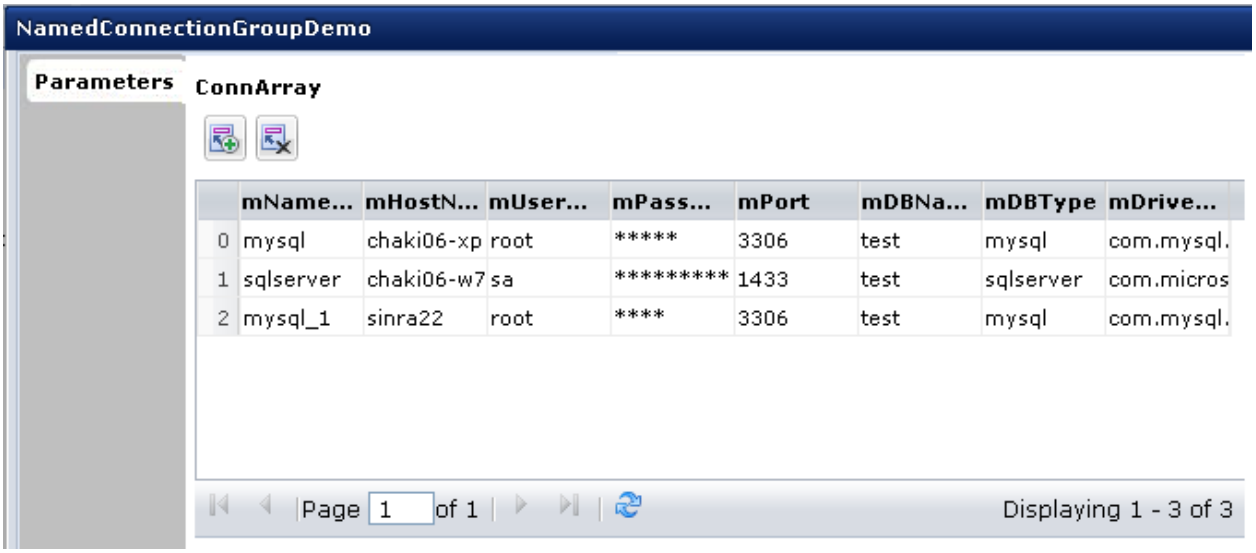
3. Click the Settings tab and create a group named NamedConnectionGroupDemo.
4. Click the Custom Operator Pre-execution tab and enter the following code:

```
Process.HostName =
getValueFromValueMapArray("NamedConnectionGroupDemo",
"ConnArray", "mNamedConnection", Operator.mNamedConnection,
"mHostName");
Process.UserName =
getValueFromValueMapArray("NamedConnectionGroupDemo",
"ConnArray", "mNamedConnection", Operator.mNamedConnection,
"mUserName");
Process.Password =
getValueFromValueMapArray("NamedConnectionGroupDemo",
"ConnArray", "mNamedConnection", Operator.mNamedConnection,
"mPassword");
Process.Port =
getValueFromValueMapArray("NamedConnectionGroupDemo",
"ConnArray", "mNamedConnection", Operator.mNamedConnection,
"mPort");
Process.DBName =
getValueFromValueMapArray("NamedConnectionGroupDemo",
"ConnArray", "mNamedConnection", Operator.mNamedConnection,
"mDBName");
Process.DBType =
getValueFromValueMapArray("NamedConnectionGroupDemo",
"ConnArray", "mNamedConnection", Operator.mNamedConnection,
"mDBType");
Process.DriverName =
getValueFromValueMapArray("NamedConnectionGroupDemo",
"ConnArray", "mNamedConnection", Operator.mNamedConnection,
"mDriverName");
```

5. Select the Group Configuration tab and click Lock to lock the group.
6. Create a valuemap array named ConArray and add the parameters shown in the following illustration:


Save
Check In
Check Ou
Form
Preview
Save Configuration
Lock
Filter
Name
Parameters
ConnArray
Element Type
Parameters
mNamedConnection
mHostName
mUserName
mPassword
mPort
mDBName
mDBType
mDriverName

7. Click Save Configuration to save the group configuration.
8. Click Unlock to publish the NamedConnectionGroupDemo group at the Domain and Environment level.
9. Open the group from the Configuration Browser, Modules tab.
10. Add the parameter values to the array as the following illustration shows and save the array.



After the valuemap array values are retrieved, a production user can use the NamedConnectionGroupDemo operator in a process to reference a named connection.

1. Create a process that uses the CustOp_GetDBVersion custom operator.
2. Provide "mysql" as the connection field value in the custom operator properties to retrieve the related values from the array ConArray.
3. Run the process.

   The script in the custom operator pre-execution code runs. The process retrieves the values for the "mysql" connection field value entered in the named connection property and displays it in the Dataset palette.

Custom Operator: Preview Tab

Use the Preview tab to test the form elements.

Custom Operator: Settings Tab

Use the Settings tab to configure settings that are common to all operators. The Custom Operator Settings tab includes the standard base operator properties. These settings correspond to the parameters found on the Properties palette in the Process Designer.

Target

Defines where the custom operator runs.

Target is a calculated expression

Indicates the target uses a calculated expression.

Target is Read-Only

Indicates the target is read-only and cannot be changed.

Run as Caller User

Indicates the operator runs as if it were the calling entity,

'Run as Caller User' is Read-Only

Indicates the operator runs as if it were the calling entity, but as read-only so that it cannot be changed.

Group

Specifies a group name for your custom operator. This setting is used as the title of the group or folder in the Operators palette. You can use the same group name for related custom operators so that they all appear under the same folder in the Operators palette. The Custom Operator Group configuration defines common parameters and values for custom operators in the group.

Display Name

Indicates the name that is shown in the Operators palette with the icon for your custom operator. The name should be short and based on the function of your operator. Display Name is also used to provide the initial value for the Name field on the Information page of the Operator Properties. You can use any combination of letters, digits, spaces, and underscore characters.

Custom Operator Pre-execution

Specifies any code that must be performed before the custom operator runs.

Custom Operator Post-execution

Specifies any code that must be performed after the custom operator runs.

Current Display Icon

Specifies the icon that represents the operator. By default, the icon of your custom operator is the icon of the base operator.

Create a Custom Operator Group

Content designers or administrators can use the Settings tab to create a custom operator group. Only administrators can then configure the custom operator group on the Group Configuration tab.

Follow these steps:

1. Click the Library tab and navigate to the folder containing the custom operator on which you want to base the custom operator group.
2. Double-click the target custom operator.
3. Click the Settings tab.
4. Enter the custom operator group name in the Group field, and then click Save.

Assign the Group to Other Custom Operators

Content designers who are adding custom operators can use the Settings tab to assign an appropriate group to the custom operator.

Follow these steps:

1. To open the custom operator editor, double-click a custom operator on the Library tab.
2. On the custom operator editor, click the Settings tab.
3. From the Group drop-down list, select the group name.
4. Click Save.

Custom Operator-Specific Pre-execution and Post-execution Code

CA Process Automation lets you define pre-execution and post-execution JavaScript code for custom operators. Pre-execution code is processed before an operator runs; post-execution code is processed after an operator runs. Custom operator users cannot override pre-execution or post-execution code that designers or developers with more permissions have already defined.

The code runs in the following order:

1. User-defined pre-execution code.
2. Custom operator-specific pre-execution code.
3. The custom operator.
4. Custom operator-specific post-execution code.
5. User-defined post-execution code.

Define Custom Operator-specific Pre and Post Execution Code

To prevent accidental deletion or modification of execution code, you can define specific code for each custom operator. You can also use the custom operator input or output data in your code.

Follow these steps:

1. Click the Library tab.
2. Double-click a custom operator.

   The Custom Operator dialog appears.
3. Click the Settings tab.
4. Click one of the following two long buttons:
   - Click Custom Operator Pre-execution to enter pre-execution code.
   - Click Custom Operator Post-execution to enter post-execution code.

   A code dialog appears.
5. Enter your code. Click OK.
6. Click Save.

Run Order for Custom Operator-specific Pre-Execution Code

The run order for the custom operator-specific pre-execution code is as follows:

1. Run the user-defined pre-execution code (that is, the pre-execution code defined in the process).
2. Evaluate the User Parameters (this step includes evaluating expressions and expanding macros).
3. Run the custom defined pre-execution code. The User Parameters are exposed and you can use the Operator keyword to access the parameters.

   Note: The designer does not have the permissions to change User Parameter values.
4. Evaluate the Standard Parameters (this step includes evaluating expressions and expanding macros).
5. Evaluate the Base operator.

Custom Operator: Dataset Tab

To define and group operator dataset variables that contain information that a custom operator returns, use the Dataset tab.

For example, assume that your custom operator retrieves fields from a ticket in an in-house ticketing system. You can create a page to group all the retrieved fields and put the parameters in the page that corresponds to the returned fields.

Using the Dataset tab in this way provides the following benefits:

- Custom operator users can easily see how the output is defined without having to run the operator first to create the parameters at run time.
- You can group related parameters in a page with a descriptive name, which is not possible at run time.

In the Dataset palette, you can add, rename, delete, and move pages and parameters. You can also configure the parameter definitions to set the type, initial value, and other characteristics. The configurations and settings in the Dataset palette of a custom operator are the same as for any other dataset.

For each output parameter name/value pair on the left, you can configure whether to hide the parameter value from output.

Hide from output

Specifies whether to include the parameter with the output.

- Selected: Indicates to hide the parameter; the output parameter is not displayed in the dataset.
- Cleared: Indicates that the parameter is to be included in output parameters; the dataset includes the output parameter.

More information:

Datasets (see page 189)

Custom Operator: Group Configuration Tab

From the Settings tab, content designers or content administrators

- Create a custom operator group (see page 165) for a custom operator.
- Assign the group to other custom operators (see page 165).

From the Group Configuration tab, content administrators

- Configure a custom operator group (see page 168).
- Edit a custom operator group configuration (see page 169).

From the Modules tab in the Configuration Browser, administrators add default values for the configured variables, which are then inherited. Inheritance proceeds from the Domain to the environments, and from each environment to the operators that run in the environment. Administrators can override Domain level values at the environment level. Content designers can override environment level settings at the custom operator level.

Configure and Publish a Custom Operator Group

A custom operator group defines the parameters that are common to a group of custom operators. Content designers (or administrators) use the Settings tab to create a custom operator group. Administrators then use the Group Configuration tab to add the appropriate variables and publish the group to the Domain and its environments. Publication adds the custom operator group to the Modules tab in the Configuration Browser.

Note: After you publish the variables, you configure their values from the Configuration tab. Custom operators automatically inherit the default values of the custom operator group to which the custom operator belongs.

Follow these steps:

1. To open the custom operator editor, double-click a custom operator on the Library tab.
2. On the custom operator editor, click the Group Configuration tab.

   Note: You do not need to check out the custom operator to configure a group.
3. Click Lock to lock the group.

   The Add Variable and Add Page options are enabled.
4. Add variables or pages as appropriate.
5. Click Save Configuration.
6. Click Unlock.
7. Click OK on the following message:

   `Group configuration variables published to domain and environment.`

   CA Process Automation adds the new custom operator group to the Modules tab for the Domain and for each environment in the Domain.

   From the Modules tab in the Configuration Browser, administrators can open the custom operator group that you defined on the Settings tab. From an open custom operator group, administrators can add values to the variables you defined in the Group Configuration tab.

   Note: For more information about configuring values for custom operator groups, see the *Content Administrator Guide*.

Edit a Custom Operator Group Configuration

Administrators can edit a custom operator group configuration. Republishing does not update all changes that you make in the Modules tab of the Configuration Browser. If a group you edit was already published at least once or another user is using it, CA Process Automation only publishes parameters new to the group. The delete or change of data type operations on variables that were already published are not updated at the Domain or environment level when you republish. However, CA Process Automation does save the changes at the custom operator level. You can view the changed data type variables when you open the custom operator in a group.

Follow these steps:

1. To open the custom operator editor, double-click a custom operator on the Library tab.
2. Click the Group Configuration tab.
3. Click Lock to lock the group.

   CA Process Automation enables the Add Variable and Add Page options.
4. Add, delete, or edit variables or pages as appropriate.
5. Click Save Configuration.
6. Click Unlock.

   When you use a custom operator in the edited group in the Designer tab, verify that the custom operator includes all of your group changes (including data type changes).

Custom Operator: Custom Panels Tab

The Custom Panels tab appears for a limited subset of base operators only, such as the *Assign User Task* or *Invoke SOAP Method* operators. The sections on the tab vary by base operator. Two common base operators appear in the following examples.

For custom operators based on the Assign User Task operator, the Custom Panels tab is divided into the following sections:

Assignees

Specifies the users or groups that can interact with the custom operator and its form.

Transfer/Delegate Filters

Specifies the users or groups that are available for task transfer or delegation.

User Task

Specifies the Title, Description, and Form Data Initialization Code for the associated Interaction Request Form.

For custom operators based on the Invoke SOAP Method operator, the Custom Panels tab is divided into the following sections:

SOAP Call Data

Use this group of fields to specify the Service URL, method name, user name, password, version, source, and other details for basic SOAP or HTTP authentication. Use the WSDL Wizard to load a URL and select WSDL services, ports, and operations.

Dynamic Parameters

Use this group of fields to specify the parameter style and add, edit, or delete macros or XPath queries.

Call Results

Use this group of fields to specify the saved SOAP request response file path and add, edit, or delete additional extracted data. You can also check options to determine how portions of the extracted SOAP body, header, and XML namespaces are handled.

MIME Attachment

Use this group of fields to specify an expression or add, edit, and delete content as MIME attachments.

WS Security

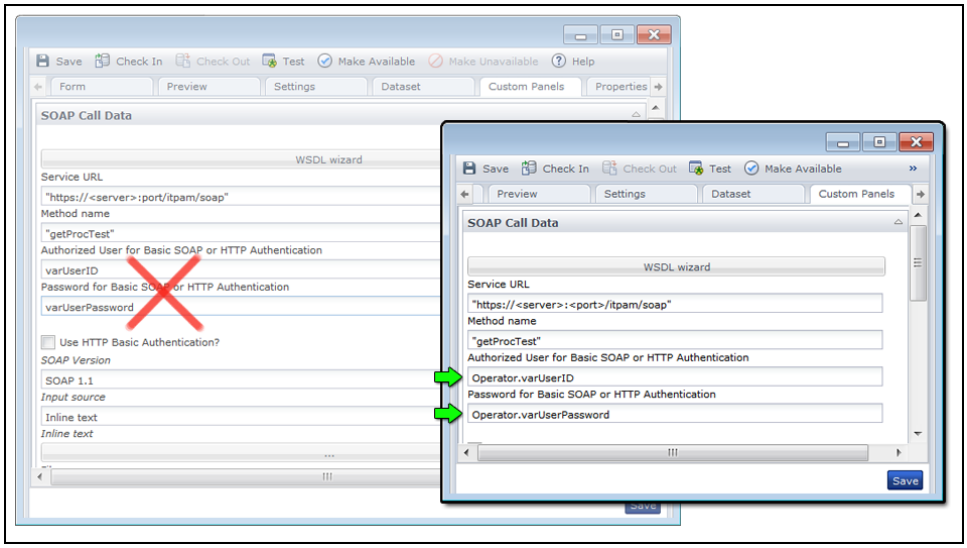
Use this large group of fields to manage security details. Expand each group box by clicking its title to view all the fields.

Macro Expansion Syntax

In previous releases of CA Process Automation, you could use a variable in a custom operator field that accepted an expression as input. The base operator supported the use of variables and dynamically replaced them with actual values, a concept that is known as *macro expansion*. Custom operators that you import from previous releases continue to support dynamic variables and macro expansion in this release of CA Process Automation.

For certain base operators, however, namely the ones with a Custom Panels tab, the fields do not support macro expansion with just a variable name alone. Examples include the Assign User Task and Invoke SOAP Method operators. Add the term *operator* to the field for evaluation of the expression and macro expansion, if supported. The following graphic demonstrates the syntax that is required to replace the variables *varUserID* and *varUserPassword* dynamically:

```
Operator.varUserID
Operator.varUserPassword
```



Custom Operator: Properties Tab

Use this tab to store the name, description, and keyword tags for your custom operator. Basic properties on this tab function the same way for all automation objects.

Custom Operator: Versions Tab

Use this tab to manage the versions of your custom operator. Versions function the same way for all automation objects.

Custom Operator: Audit Trail Tab

Use this tab to examine the history of your custom operator. Audit Trails function the same way for all automation objects.

Set Custom Operator Availability to All Users

By default, a custom operator is not available to other CA Process Automation designers. You can set the availability of custom operators for all users.

The availability of a custom operator only affects the ability to see and add the custom operator to a process or schedule. Once a custom operator has been added to a process or schedule, making it unavailable does not affect its existing inclusion and use.

Follow these steps:

1. Click the Library tab.
2. Double-click a custom operator.

   The Custom Operator dialog appears.
3. In the toolbar:
   a. Click Make Available to allow all users to see and use the operator.
   b. Click Make Unavailable to hide the operator from other users.

   The change in availability is instant whether you click Save or not.

Publish a Custom Operator Group Configuration to Another Domain

When you define a custom operator group, the group automatically publishes to the current Domain and to all environments in the Domain.

When you import to a different Domain, the process lets you publish the custom operator group to that Domain and to all of its environments. The Publish Custom Operator Group Configuration option requires Group_Config_Admin rights. Administrators (members of the PAMAdmins group) have this right.

Follow these steps:

1. In the Library browser for the selected Orchestrator, select the destination for the imported items.
2. Right-click the parent folder, then click Import.
3. Click Browse, browse for the XML file in the Open window, then click Open.

4. Select one of the following options that specify how to handle objects with conflicting names. If you import into an empty folder, there can be no conflicts.
   - Import
   - Import and Replace
   - Do Not Import
5. (Optional) Select the Set Imported Version as Current check box.

   This option applies if you selected *Import as a New Version and Keep the Existing Object* in Step 4.
6. Select the Make Imported Custom Operators Available check box.

   You cannot use custom operators until they are made available.
7. Select the Publish Custom Operator Group Configuration check box.

   Publication publishes the custom operator groups to the import Domain and to all environments in this Domain.
8. Click Submit.
9. Click OK on the successful import confirmation message.

   The import recreates the exported folder structure in the selected location, imports the custom operators, and publishes the custom operator group and related variables.

More information:

Custom Operator: Group Configuration Tab (see page 167)
Example: Retrieve Valuemap Array Values with an Operator System Function (see page 160)
Release Objects to Another Environment (see page 423)

Using Custom Operators

You can use a custom operator in a process or schedule like any other operator. Keep these points in mind when working with custom operators:

- Initial values for the custom operator parameters may be pre-configured or hidden.
- New custom parameter inputs may exist that require configuration.
- Your custom operator will appear in a folder called Uncategorized unless you set a custom group name on the Custom Operator dialog Settings tab.
- If a custom operator is not available at the time that your process was opened for editing, make the custom operator available, and refresh the operator group folder again.
- If a custom operator is not available at the time that your schedule was opened for editing, make the custom operator available, and refresh the operator group folder again.
- You must refresh the operator group folder in a process or schedule to see any changes made to the custom operator's name or other settings.
- You must close and re-open a process or schedule to see any changes to existing custom operators.
- Any process or schedule using a custom operator uses the latest checked-in version that is marked as *Current* in the Library Browser.

Edit Custom Operator Values

Custom operator groups provide custom variables with default values. Custom operators that you assign to a custom operator group share those variables and values. You can accept the group-level values or you can assign different values to the variables for a custom operator.

Note: Group values are set at the Domain level. All environments in the Domain inherit these group values. Unlike operator categories, you cannot override values for custom operator groups at the Orchestrator or agent level. However, you can change inherited values at the custom operator level.

Follow these steps:

1. Click the Designer tab.
2. Click the View drop-down list and select Operators, Dataset, and Properties.
3. Expand the custom operator group name in the Operators list and drag the selected custom operator to the canvas.
4. Double-click the custom operator.
5. Review the value for each custom variable. Either accept the inherited value or type the value that you require.
6. Click Save.

How to Work with Protected Custom Operators

CA Technologies can release protected custom operators. The lock icon indicates that the custom operator is protected:

The use of protected custom operators is restricted in the following respects:

- You cannot view any code defined to the protected custom operator. The input parameters, pre-execution code, and post-execution code are encrypted.
- You cannot modify the imported version.
- You cannot modify or view the code in a copy of the imported version.

Your Favorite Operators

As a convenience, you can add your favorite operators to the Favorites folder or group. The Favorites group appears at the top of the Operators palette in the Process Designer.

Click Refresh in the Operators palette to view your favorite operators when you design a process.

Add or Remove Your Favorite Operators

You can add and remove your favorite operators from the Favorites group folder in the Operators palette.

Follow these steps:

1. Click the Designer tab.
2. Open a process or create a process.
3. In the toolbar, click the View menu and select Operators.

   The Operators palette appears.
4. In the filter area at the top of the Operators palette, click Refresh.

   The list of operators, custom operators, and favorite operators is updated.
5. Expand the Favorites folder to view the operators you have added. For new users, no operators appear.
6. To add an operator to your Favorites folder:
   a. Expand any other folder of operators.
   b. Right-click an operator and select Add to Favorites.
   c. Click Refresh.
7. To remove an operator from your Favorites folder:
   a. Click Refresh.
   b. Expand the Favorites folder to view the operators you have added.
   c. Right-click an operator and select Remove from Favorites.

Connectors

Connectors are optional extensions to CA Process Automation that enable operators that interface with other CA and third-party solutions. When your administrator configures a connector, a new group of operators appears in the Operators palette in the Process Designer.

Connectors integrate CA and third-party products into workflow processes and provide bridge services from other products to CA products and solutions that embed CA Catalyst. You can download other connectors in addition to the connectors provided on the CA Process Automation installation media. Licensing restrictions may apply. A list of the most popular connectors follows:

- Amazon Web Services (AWS)
- BMC Remedy
- CA Client Automation
- CA CMDB
- CA Configuration Automation
- CA eHealth
- CA NSM
- CA Service Desk Manager
- CA Spectrum IM
- CA Workload Automation AE
- IBM AS/400
- IBM z/OS
- Microsoft Hyper-V
- VMware vSphere

Use application-specific connectors to perform tasks such as gathering data or applying actions on target systems and target applications. Connectors provide operators that run in the following locations:

- On the Orchestrator
- On agents that reside on the application server
- On proxy agents that can remotely perform the required task or collect the data on the application server

Each connector module typically includes multiple operators. Each operator performs one of the following specialized tasks:

Decision Tree Support

Returns a binary (true/false, success/failure) value that can be used to decide how to branch in the process. In some situations, the returned value can have more than two options. However, the returned value is always a small and well-defined set of possible values.

Data Collection

Collects more complex datasets from the target application. The result is typically stored in a local dataset where other operators can further analyze it. If so required, the data can also be made global so that other processes can use it.

Active Management

Performs actions on the target system, including all operators that change the behavior of the target system in any way. For example:

- Sending an event
- Reconfiguring the application
- Starting or stopping a related service

In some cases, a single operator can perform more than one of these functions. For example, a connector can run, then return a result set that is based on the connector action.

CA Process Automation includes many connectors. You can create connectors and related operators easily in either of the following ways:

- By calling standard, application-specific executables that are on the agent
- By using standard scripting languages for more complex functions

To minimize the application footprint and user interface complexity, only a set of generic and commonly used connectors are installed by default. Best practice is to install other connectors only when necessary.

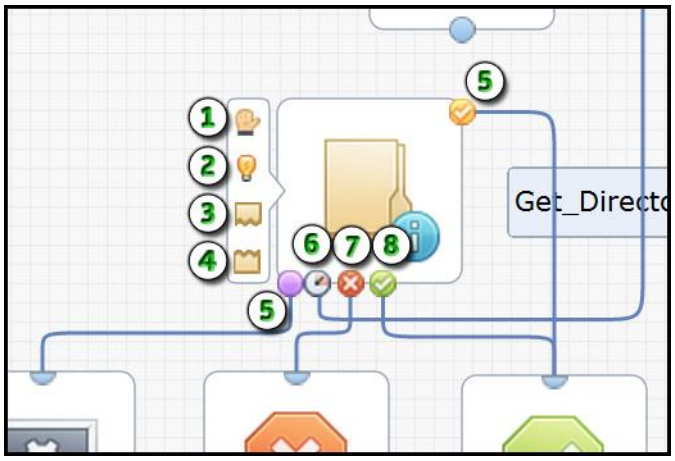
Operator Icons

CA Process Automation supplies default icons that are displayed for operators when they are placed in a process or schedule. You can replace the default icon with an icon that you specify (a c*ustom icon)*. You can assign a custom icon to any operator in a process, including custom operators.

CA Process Automation automatically handles the smaller visual indicators that represent the status of an operator in a process. For example, smaller graphics in the corners of all icons represent different execution states (such as idle, running, completed, and failed) in a process.

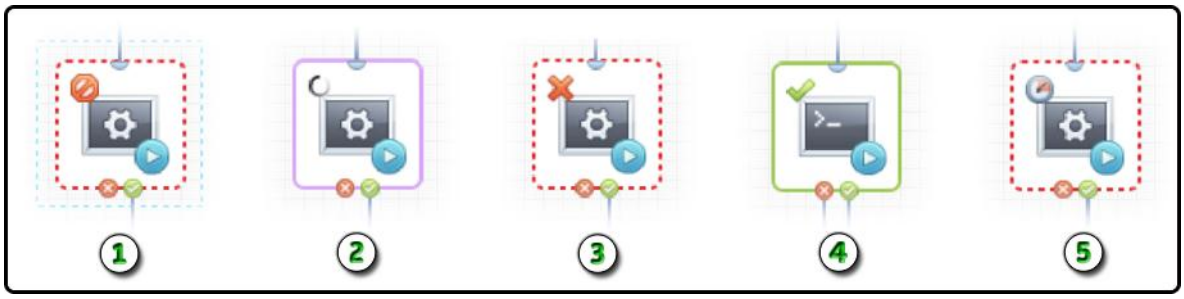
Operator Status Icons

The following graphic details the smaller subset of icons that an operator can display to indicate status and port options.



Item:	Description:
1	**Breakpoint**: This icon indicates that you set a breakpoint in the process at this operator. Click Set Breakpoint in the toolbar.
2	**Simulation Mode**: This icon indicates that you are overriding simulation options and have set the operator simulation type to *local* or *distant*. In the Properties palette, expand Simulation.
3	**Pre-Execution Code**: This icon indicates the presence of JavaScript code that runs before the operator runs. In the Properties palette, expand Execution Settings.
4	**Post-Execution Code**: This icon indicates the presence of JavaScript code that runs after the operator runs. In the Properties palette, expand Execution Settings.
5	**Custom Port**: The process flows through a custom port when its predefined expression is true.
6	**Timeout Port**: The process flows through this port when the operator times out.
7	**Failed Port**: The process flows through this port when the operator yields an unsuccessful result or fails.
8	**Completed Port**: The process flows through this port when the operator yields a successful result.

When you run a process, the following icons indicate the status of each operator.



Item:	Description:
1	**Aborted**: The process has been stopped.
2	**Running**: The process is currently running.
3	**Failed**: The process failed at this particular operator.
4	**Completed**: The process has successfully passed this particular operator.
5	**Timeout**: The process has timed out at this particular operator.

Creating, Editing, and Applying Custom Icons

Custom icons are visual identifiers for an operator. They assist you in identifying the specific function of an operator. You can create custom icons in any automation library folder, and then apply them to any operator. Each icon displays your chosen base image with a modifier image overlaid in the lower right corner. The base (object) and modifier (action) provide a consistent structure for all icons.

The topics in this section describe why and how a CA Process Automation Content Designer uses custom icons to customize the appearance of operators. This section also provides examples.

As a process designer, you rely on the *visual* cues that icons provide to determine the purpose of each operator. For example, what would you guess is the function of the following operator icon?

Even without a label, you can infer from the image that this icon represents a *Delete Email* operator. An appropriate icon helps you and other designers understand the functionality of an operator.

The initial investment of time you make in assigning an appropriate icon yields many benefits, including the following:

Standardization

Using a standard set of base and modifier images helps designers understand the object performing the action.

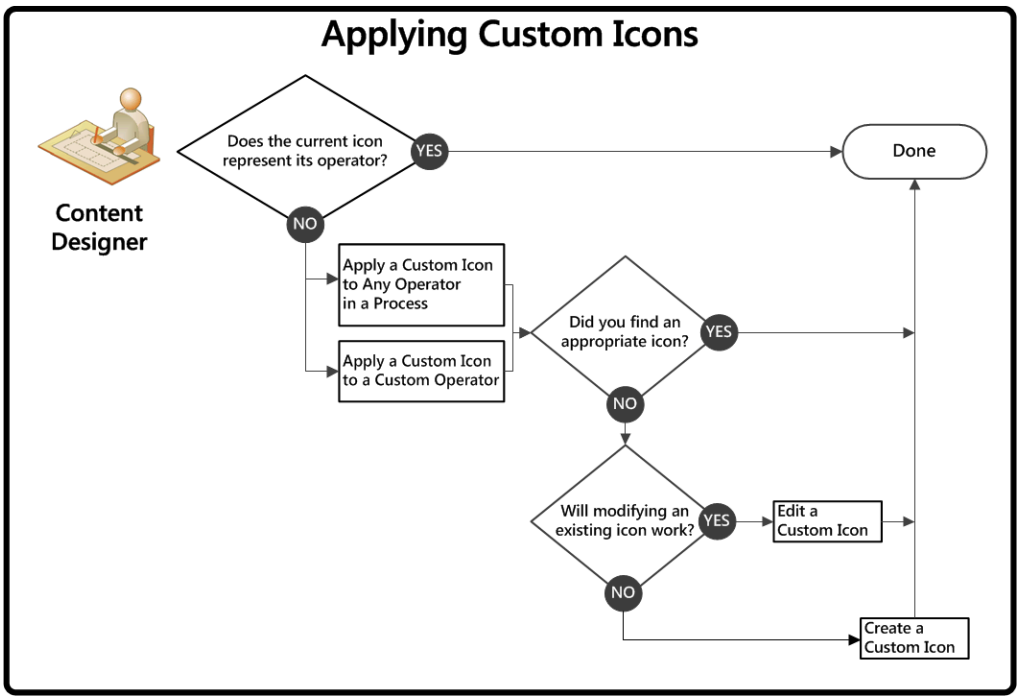
Simplicity

Designers can readily identify distinct operators in even the largest, most complex, processes.

Sharing

Teams of designers working in other native languages can share process designs because they are easier to understand.

Use the following flowchart as a guide when customizing icons for your process operators:



No sequence is required for performing the listed tasks. At any time, you have the following options for customizing icons:

- Apply a Custom Icon to Any Operator in a Process (see page 184): While you design a process, you can change the icon for any operator to customize its appearance only for a single instance. Select a specific instance of an operator and change its icon to one of the predefined custom icon objects in the library.
- Apply a Custom Icon to a Custom Operator (see page 185): When you change a custom operator icon, your choice of icon is applied to all future instances of that operator. In addition, all designers can see the new icon for the custom operator in the Operator palette.
- Edit a Custom Icon (see page 183): As time goes by, you can edit one or more predefined custom icons. Your changes apply wherever that custom icon is already used.
- Create a Custom Icon (see page 183): You can define one or more custom icons. You can create a series of custom icons that you plan to assign to operators or custom operators. Or, you can simply create and save custom icons without knowing in advance exactly where they will be used.

Create a Custom Icon

You can create custom icons in any automation library folder. Each icon pairs your choice of base image with a modifier image overlaid in the lower-right corner of the base. The base and modifier provide a consistent structure for all icons.

Follow these steps:

1. Click the Library tab.

   A hierarchical list of folders appears in the left pane and all your automation objects appear in the main window.

2. (Optional) If the new custom icon you want to create is similar to an existing custom icon object, select it, click Copy, and then Paste.

   You can now edit the copy to complete your custom icon. Skip the next step.

3. Right-click any folder in the Library Browser pane and click New Object, Custom Icon.

   The new icon appears in the browser with a default name.

4. Double-click the icon.

   The Custom Icon Editor opens

5. Select one base and one modifier image. You can browse the icons one by one, filter by the category drop-down menu, or enter a keyword search.

6. Click the Object Properties tab, enter or edit the icon name, and then click Save & Close.

   Your new custom icon is available in the library.

For example, if you want to represent an operator named *Upload Report*, you would:

- Select a base image that represents a report.
- Select a modifier image that represents the upload action.

Edit a Custom Icon

Over time, the custom icons you and other designers create can be modified to align with other icons. You can edit custom icons in any automation library folder. Each icon pairs your choice of base image with a modifier image overlaid in the lower-right corner of the base. The base and modifier provide a consistent structure for all icons.

Follow these steps:

1. Click the Library tab.

   A hierarchical list of folders appears in the left pane and all your automation objects appear in the main window.

2. Right-click a custom icon.

3. Click Action, Edit.

   The Custom Icon Editor opens.

4. Select a base and a modifier image, edit the icon name, and then click Save.

   Your modifications to the custom icon are applied.

For example, if you want to represent an operator named *Upload Report*, you would:

- Select a base image that represents a report.
- Select a modifier image that represents the upload action.

Apply a Custom Icon to Any Operator in a Process

You can change the icon for any single operator used in a process.

Follow these steps:

1. Click the Designer tab.
2. Click Open.

   The Open Process dialog opens.

3. Navigate to your process and click Open.

   The process that you open appears on a new tab.

4. Double-click a specific operator already shown on the canvas to view its Properties. In the toolbar, click the View menu and check Properties to view the Properties palette.
5. In the Properties palette, expand the Information section.
6. In the Information section:
   a. Clear the Use default Icon check box.

      A border appears around the current icon and a Browse button is available.

   b. Click the Browse button.

      The Select Custom Icon dialog appears.

   c. Select the custom icon object that you want to use for this specific occurrence of the operator.

   d. Click OK.
7. In the toolbar, click Save.

Note: You cannot change the icons that are associated with the execution state of an operator. For example, *waiting* or *completed*. CA Process Automation automatically manages these icons.

Apply a Custom Icon to a Custom Operator

You can change the icon for a custom operator. The icon that you select applies to all future occurrences of the custom operator in processes. Existing occurrences of the custom operator in processes continue to show the original default icon for the base operator.

Follow these steps:

1. Click the Library tab

   A hierarchical list of folders appears in the left pane and all your automation objects are listed in the main window by type.

2. Double-click a custom operator.

   The Custom Operator window appears.

3. In the toolbar, click Check Out.
4. Click the Settings tab.
5. On the Settings tab, click the Change Icon link.
6. In the Select Custom Icon dialog, browse to the custom icon, select it, and click OK.
7. Click Save.
8. (Optional) If none of the existing icons are appropriate, edit a custom icon or create a new one.

Note: You cannot change the icons that are associated with the execution state of a custom operator. For example, *waiting* or *completed*. CA Process Automation automatically manages these icons.

Custom Icon Examples

The following list shows examples of custom icons and the potential operators they could represent.

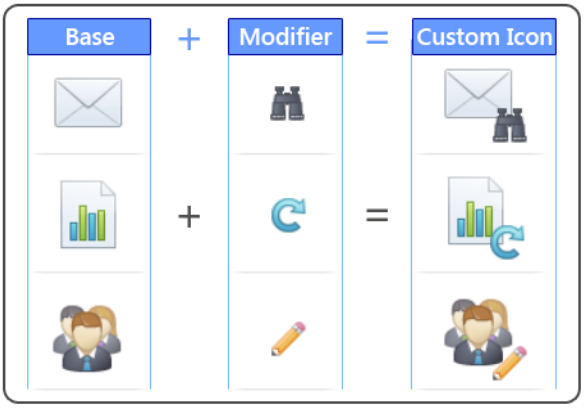
Diagnose (or Monitor) Performance

Add User Account

Debug Script or Code

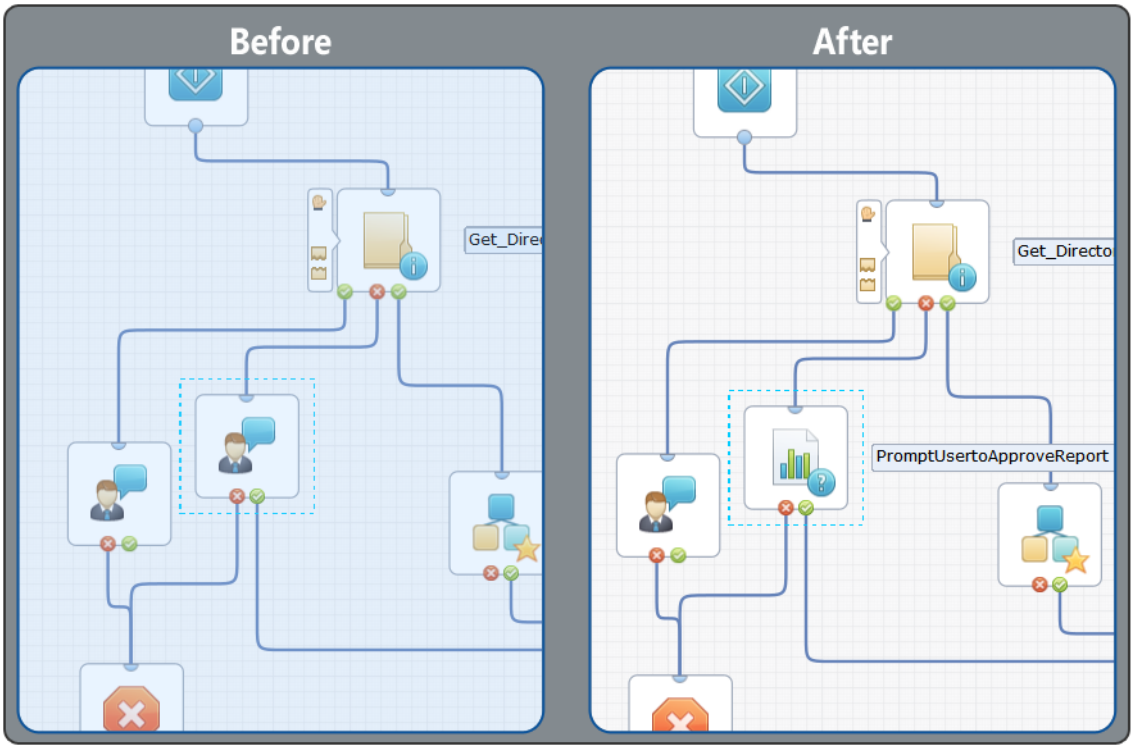
Example: Create or Edit a Custom Icon

This example demonstrates the key concepts behind the CA Process Automation Custom Icon Editor. The graphic shows the selection of a base icon that is combined with a smaller modifier icon to create a custom icon. The resulting combination is saved as a Custom Icon object that can later be applied to any operator.



Example: Apply a Custom Icon

In the following example, the designer has decided to change one of the two identical icons. The custom operator is based on the Assign User Task operator. However, it has a distinct new purpose. The operator prompts a user to approve a report. The old default icon appears within the process at left (*Before*) and the new custom icon appears at right (*After*). The new icon better represents the function of the currently selected operator within the process. In this example, the designer has also elected to show the long name for the operator.



Imagine a process with many similar operators arranged in a series. Each operator could perform a different action. In this situation, use custom icons to help you to identify each distinct operator.

Chapter 6: Datasets and Parameters

This section contains the following topics:

Datasets (see page 189)
Process Parameters (see page 208)

Datasets

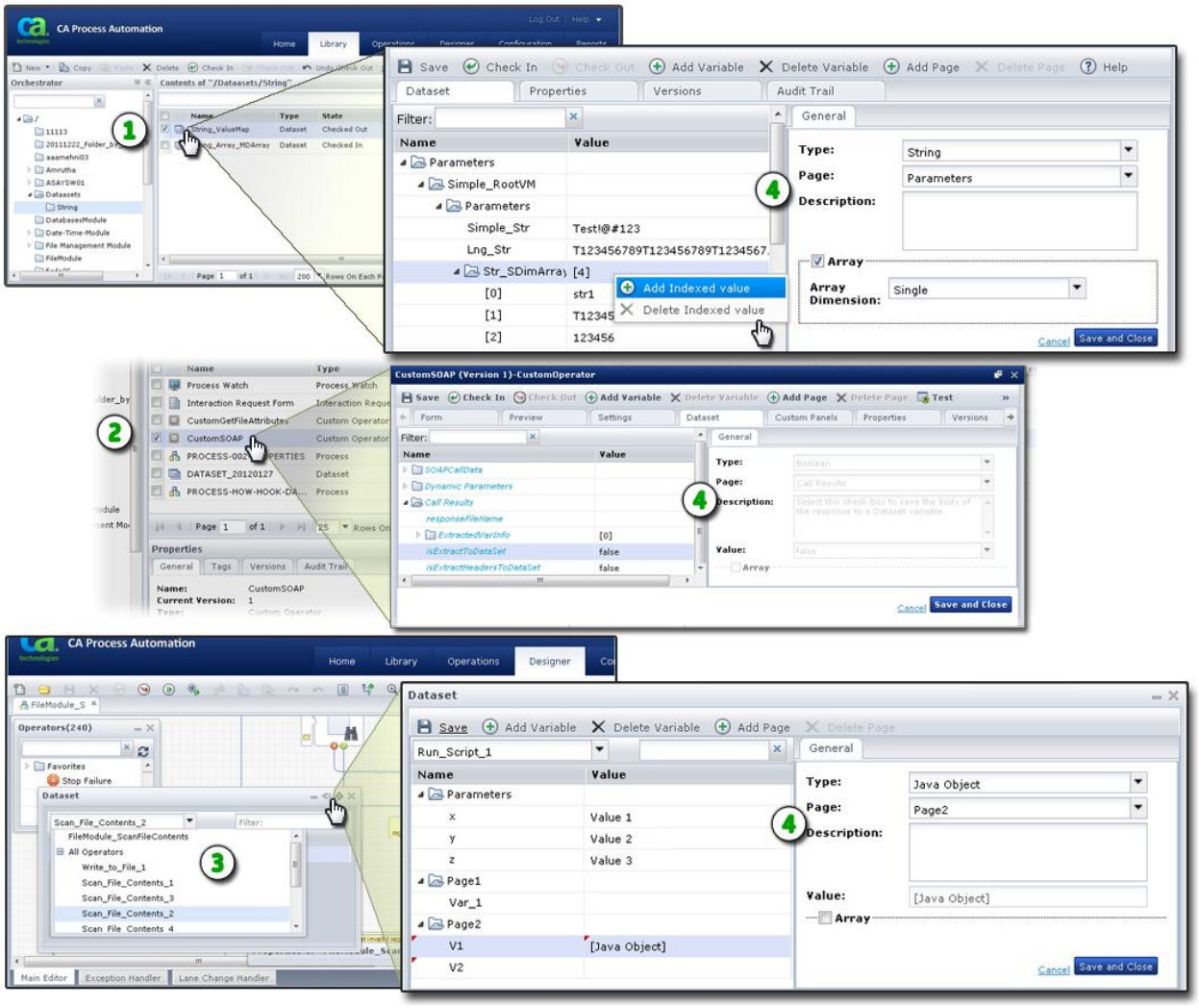
Datasets let you define groups of variables to store and organize data. Datasets provide a way to share data across multiple process instances.

A *dataset* object defines a collection of variables that you can reference by name. You create and manage datasets in the Library Browser just like any other automation objects for an orchestrator. A dataset can contain any number of fields, called *variables*. Assign each variable to one of the following data types by the kind of data the variable stores:

- *Boolean*
- *Date*
- *String*
- *Integer*
- *Java Object*
- *Long*
- *Double*
- *Password*
- *Object Reference*
- *ValueMap*

You can configure all data types to contain a single value or multiple indexed values (called an *array)*. You can define an indexed field as an array of one or more dimensions.

You can edit dataset objects and custom operator datasets from the Library Browser. You can edit datasets for processes and each operator in the Process Designer.



Item:	Description:
1	**Datasets in the Library Browser**: Create, edit, and manage your own dataset objects in the Library Browser. Open a dataset to edit it.
2	**Custom Operator Datasets:** Create, edit, and manage your own datasets for custom operator objects in the Library Browser. Open a custom operator to edit its dataset.
3	**Datasets in the Process Designer**: The Dataset palette displays the datasets available for the process and each operator. You can also refer to your own datasets in code. Click the icon to open a dialog for editing the selected process or operator dataset.

Item:	Description:
4	**Pages, Variables, Data Types, and Values:** Define the pages and variables for the dataset in the left half of the dialog. Define the data types and enter a description in the right half. Define values on either side. Right-click the variable name for an array to add or delete an indexed value.

Create a Named Dataset Object

You create and manage named dataset objects with your other automation objects in the Library Browser.

Follow these steps:

1. Click the Library tab.
2. In the Library Browser folders pane, select a folder.
3. In the toolbar, click New and select Dataset.

   A new dataset object appears and is checked out to you.
4. Enter a name for the new dataset.
5. Double-click the dataset to define its pages and fields.

   The Dataset dialog opens.

Dataset Types

Dataset *variables* (also called *fields*) can contain literal values that you explicitly define in the dataset object. You can also assign values to variables (fields) at runtime using expressions. Refer to dataset objects and their variables by name using JavaScript expressions in calculated parameters.

You can create and configure dataset variables for all types of datasets, except the system dataset. Refer to system dataset variables directly. You can refer to process and operator datasets through parent process or operator objects. The following table describes the dataset types.

Dataset Type:	Description and Scope:	To Reference in an Expression:
Named Dataset	Dataset objects store the definition for a named dataset in the Library Browser. Named dataset variables are accessible by any operator, process, or schedule in the same library. You can edit the current version of a named dataset by expanding the library folder and double-clicking the dataset object.	See Specify Named Dataset Variables (see page 241).

Dataset Type:	Description and Scope:	To Reference in an Expression:
Process Dataset	Process datasets contain variables that you or another designer defines. CA Process Automation can also define process variables automatically when a process instance starts. Process datasets appear in the Dataset palette of the Process Designer.	See Specify Process Dataset Variables (see page 242).
Operator Dataset	An operator dataset is included in every instance of an operator added to a process or schedule object. The operator dataset can contain operator-parameters, user-defined variables, and program-defined variables. An operator dataset is primarily accessible to the immediate operator and secondarily to other operators in a process. Operator datasets appear below process datasets in the Dataset palette of the Process Designer.	See Specify Operator Dataset Variables (see page 244).
System Dataset	Contains predefined variables that are available in the context of the entire CA Process Automation domain. These variables access system parameters and are made available by the *System* keyword.	See Specify System Dataset Variables (see page 245).

Define Dataset Pages and Variables

Define the pages, variables, and values in a dataset. When you run a process, operators can reference the values in datasets.

New named datasets and process datasets include a default root page called *Parameters*. The pages and variables for operator datasets vary by operator. You can create new variables (fields) or you can edit existing variables. You can also add pages to group variables in logical ways. To edit an existing variable or its value, click the variable or its value, and then make your changes. You can also change values in dataset variables (fields) programmatically.

Follow these steps:

1. Click the Library tab.
2. Select a folder and locate a dataset. Use the optional search features if necessary.
3. Double-click the dataset.

   The Dataset tab of the Dataset dialog appears.
4. Click Check Out.
5. To create a page, click Add Page.
6. To create a variable:
   a. Select a page.
   b. Click Add Variable.
7. To rename a page or variable, double-click it. Names must start with a letter and must have a maximum length of 32 characters.
8. For variables, select the data type and page, enter an initial value, and provide an optional description.

   Page

   Defines the page that contains the variable. To move the variable to a different page, select that page. For custom operators, the page that is specified here corresponds to an expandable properties group on the Properties palette.

   Description

   (Optional) Provides helpful information about the variable when it appears later in a dataset. If provided, a tool tip displays the text that you enter here when you move the mouse pointer over the field name or value.

   Value

   Specifies the default value if the field is blank (cleared). For fields of the integer, long, and double type, the default value is 0. For fields of the integer, double, long, or string type, you can specify your own default value here. To view the Value field contents in a separate window, right-click in the Value field and choose Expand.

Note: The Expand option is available for only the string data type.

You can also enter constraints for a field of these types:

- For fields of the *Boolean* type, you can select True or False.
- For fields of the *Object Reference* type, click the browse button (...) to select a dataset object.
- For fields of the *Date* type, click the browse button (...) to select a date.
- Java Object fields are read-only.

Note: The ValueMap field type cannot be assigned a default value.

9. To define arrays:
   a. On the General tab, select the Array check box and select *Single* or *Double* in the Array Dimension field.
   b. Right-click the variable and select Add Indexed Value.
   c. Click the Value field to enter a value.

Note: To edit the dataset for a custom operator, open it, check it out, and click the Dataset tab. You can also edit process and operator datasets by clicking in the title bar of the Dataset palette in the Process Designer.

Variable Data Types

Dataset variable data types map to JavaScript value types.

The different data types for variables are as follows:

Boolean

Stores and returns True or False.

In expressions, this type maps to the JavaScript Boolean data type.

Object Reference

References any type of object available in CA Process Automation, including objects available in an automation library, touchpoint, and touchpoint groups.

Date

Stores and returns a date in a date format specified in the parameter properties.

In expressions, this type maps to the JavaScript date object.

Double

Stores and returns a decimal value. Double values are entered in the following format:

[digits][.digits][(E|e)[(+|-)]digits]

In expressions, this type maps to JavaScript floating-point literal type. The literal has a minimum value of -1.7976931348623157E308 and a maximum value of 1.7976931348623157E308.

Integer

Stores and returns a 16-bit integer value. An integer field can return a single integer or an indexed list of integers. The integer allows you to represent all integers to ten digits from -2,147,483,648 to +2,147,483,647.

In expressions, this value type maps to the JavaScript integer type.

Java Object

Stores a Java object.

Long

Stores and returns a 32-bit integer. The long data type allows you to represent all integers to 19 digits from -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807.

In expressions, this value type maps to the JavaScript integer type.

Password

Returns a password in encrypted format. Password-field values can only be assigned to other password fields. Unauthorized users are prevented from viewing passwords in an unencrypted format.

Note: CA Process Automation protects passwords from any modification. Operations such as concatenation and string manipulation on the passwords results in a null value.

String

Returns a string of characters. A string field can return a single string or an indexed list of strings.

In expressions, this value type maps to the JavaScript string type.

ValueMap

Defines a dataset structure that is stored within another dataset. You can add pages and variables to a ValueMap field. ValueMap variables and pages can be nested one inside of another.

In calculated fields, ValueMap type fields are addressed hierarchically within a dataset path. The following example addresses the parameter *param1* in ValueMap field *VMap2*, which is in turn nested in ValueMap field *VMap1*, which in turn is a member of the dataset *MyDataset*.

```
Process.MyDataset.VMap1.VMap2.param1
```

Important! Changing the data type or changing the Array check box option eliminates any existing data in the field. Edit the dataset only if it is acceptable to discard any current values in the field.

Validation Settings for Dataset Variables

When you select the variable type, you can specify exactly what type of data a user can enter. The constraints that you can apply to a value depend on the selected data type. The following table lists the potential constraints for the different data types in an operator dataset.

Data Type	Validations
Boolean	None
Date	None
Double	Specifies minimum and maximum values for a Double type variable between minus 1.7976931348623157E308 and positive 1.7976931348623157E308. Minimum value is the minimum allowed value for double values. Maximum value is the maximum allowed value for double values.
Integer	Specifies minimum and maximum values for an Integer type variable between minus 2147483648 and positive 2147483647. Minimum value is the minimum allowed value for integer values. Maximum value is the maximum allowed value for integer values.
JavaObject	None

Data Type	Validations
Long	Specifies minimum and maximum values for a Long type variable between minus 9223372036854775808 and positive 9223372036854775807. Minimum value is the minimum allowed value for long values. Maximum value is the maximum allowed value for long values.
Object	None
Password	None
String	None
ValueMap	None

Arrays with Indexed Values

You can define a dataset variable to store a single value or an array of indexed values. You can access each value in an indexed array. You can define the indexed fields in a single or double dimension array.

Each dimension in an array begins with indexed value [0] and continues in sequence with [1], [2], then [3], and can continue with any number of additional values. In expressions, represent a dimension, or level, of an array with bracket notation. Each level of an array supports JavaScript array properties and methods.

The default type of field is to store a single value. To specify that a field store an indexed list of values, select the Array check box and configure the indexed values.

Define a ValueMap as an Array

You can create a ValueMap, define its variables, and can include arrays in the ValueMap. You can even decide to define a ValueMap as an array. Each variable of the ValueMap represents a structure that you define in the dataset under *Element Type*.

1. Click the Library tab.
2. Select a folder and locate a dataset. Use the optional search features or create a dataset if necessary.
3. Double-click the dataset.

   The Dataset tab of the Dataset dialog appears.
4. Click Check Out.
5. Select a page. The default page is named *Parameters*.
6. In the toolbar, click Add Variable.
7. On the General tab:
   a. Set the Type to ValueMap.
   b. Do not click the Array check box.
8. Expand the new ValueMap variable and select a page. The default page is named *Parameters*.
9. In the toolbar, click Add Variable.
10. On the General tab:
    a. Set the Type.
    b. Enter an initial value.
    c. Check the Array check box.
11. Repeat the last three steps. Select a page, click Add Variable, and define the variable as an Array.
12. Expand the ValueMap variable and page. Right-click the first variable and choose Add Indexed Value.
13. Repeat the last step.
14. Right-click the second variable and choose Add Indexed Value.
15. Repeat the last step.

    You have defined a ValueMap made up of two arrays.
16. Click the original ValueMap variable.
17. On the General tab, check the Array check box.

    The Element Type folder appears under the ValueMap variable.
18. Expand Element Type completely to view its pages, variables, and indexed values.

19. Right-click the original ValueMap variable and choose Add Index Value.

    Array index [0] appears.

20. Expand array index [0]. The pages, variables, and indexed values are copied from the Element Type structure.

When you add a variable to a page under Element Type, all existing valuemap array index entries immediately include the new variable under the corresponding page.

When you delete a variable from a page under Element Type, all existing valuemap array index entries no longer include the deleted variable under the corresponding page.

When you specify values for the variables under Element Type, the values become the default values for any new indexed entries of the ValueMap array. The new Element Type values are not propagated to existing index entries of the ValueMap array.

Modify a Dataset

You can modify a dataset by adding indexed values or modifying variable data. This procedure provides fewer capabilities than the more robust dataset tasks of defining pages and variables.

Follow these steps:

1. Click the Library tab.
2. Right-click a dataset and choose Modify Dataset.
3. In the Modify Dataset window:
    a. Right-click an array variable and choose Add Indexed Value to add an entry to the array.
    b. Click a value to directly add or change it.

View or Copy a Dataset Expression

Use this procedure to view or copy the full reference to a dataset array or a specific value.

Follow these steps:

1. Click the Library tab.
2. In the Library, do one of the following actions:
   - Double-click a dataset.
   - Right-click a dataset and click Edit.
   - Right-click a dataset and click Modify Dataset.
3. In the resulting window, click an array or a specific variable to view it.
4. Right-click the array or variable and click View Expression.

   The Dataset Expression window opens.
5. View or copy the expression. You can also drag the expression to a destination field that accepts expressions and supports drag-and-drop.

Read Operating System Values into Dataset Variables

CA Process Automation can read values that a shell process generates into dataset variables. Before a UNIX Script or Windows Script operator runs its associated script, it creates a folder to accept values that the script generates. The C2OSVD environment variable specifies the full path to the folder (for example, C:\TMP_VS_559) that is created for the script operator. A script can then copy data to text files in the folder to pass the data back to CA Process Automation. Data that is passed back to CA Process Automation using the C2OSVD directory populates variables in the script operator dataset.

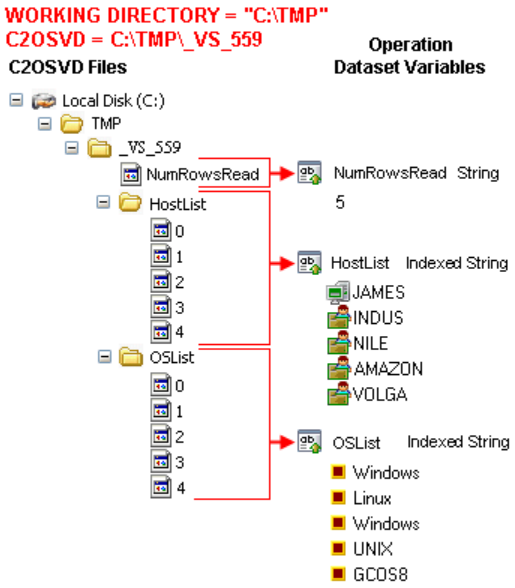
A script must save data to text files in the C2OSVD directory. After a script operator completes its script (but before it performs the post-execution actions), it determines whether the location to which C2OSVD points contains files. CA Process Automation then creates operator dataset variables according to the following rules:

- The operator saves the contents of a file from the C2OSVD location to a string variable in the operator dataset. The string variable has the same name as the file.
- A folder in the C2OSVD location generates an indexed variable in the operator dataset with the same name as the folder. The operator saves the contents of text files named in a numbered sequence located in the folder to corresponding elements in the indexed variable. The elements are numbered from 0 to as many elements as the highest numbered file in the folder. Missing files in the number sequence generate null elements in the indexed variable.
- CA Process Automation requires a script to create files in the C2OSVD location with the same names as the variables you want to appear in the operator dataset. If the script creates text files with file extensions (such as .txt), CA Process Automation includes the extension in the variable name.

The following illustration shows the file-to-variable conversion when the working directory is set to C:\TMP. The operator appends _VS_599 to the working directory path to create the C2OSVD value C:\TMP_VS_599. The folder name (in this case _VS_599) is unique for every instance of any operator.

The illustration also shows two folders %C2OSVD%/HostList and %C2OSVD%\OSList that contain five files named 0, 1, 2, 3, and 4. The script writes a computer name to each numbered file in the HostList folder. The script writes an operating system name to each numbered file in the OSList folder. CA Process Automation creates two zero-based indexed variables after running the script, HostList and OSList. The application assigns the contents of the numbered files in the HostList and OSList folders to the corresponding elements of the indexed variables.

In the example, the Windows script uses the C2OSVD variable to create the file %C2OSVD%\NumRowsRead. CA Process Automation creates a corresponding variable (NumFilesRead) in the script operator dataset after it runs the script. The product then assigns the contents of the NumFilesRead file to the variable.



When a script operator finishes, it deletes the C2OSVD folder and its contents. The script operator post-execution code can access the operator dataset variables. The code typically copies the operator dataset variable values to local variables in the process dataset or to operator dataset variables in subsequent process operators. The three example scripts in this chapter show how the example illustrated in this section is implemented using UNIX script, VBScript, or PerlScript.

Sample Scripts for Reading Operating System Values into Dataset Variables

A process can use UNIX script, VBScript, or PerlScript to read operating system variables into operator dataset variables. Each script reads lines from a text file that specifies a host name, a single space, and the operating system running on the host. For example:

```
JAMES Windows
INDUS Linux
NILE Windows
AMAZON UNIX
YANGTZE Solaris
GILA UNIX
```

The three scripts save the host names to an indexed field named HostList and the operating system names to an indexed field named OSList.

The process incorporates the following steps:

1. The UNIX or Windows module passes a text file name. The module gets the file name from a parameter on the appropriate Process tab of the script operator properties.
2. The UNIX or Windows module creates and initializes the C2OSVD environment variable with a path to a unique folder.

   One instance of a script operator uses the folder that the C2OSVD environment variable specifies. The instance is not repeated or overwritten by any subsequent instance of that or any other script operator.
3. The UNIX or Windows module creates the folder that the C2OSVD environment variable references.
4. The script creates folders named HostList and OSList in the folder that the C2OSVD environment variable references.
5. The script reads each line of the text file and takes the following actions:
   - The script writes the host names to sequentially numbered files in the %C2OSVD%\HostList folder.
   - The script writes the operating system names to sequentially numbered files in the %C2OSVD%\OSList folder.

   These files generate two indexed operator dataset variables HostList and OSList.
6. The script writes the number of lines it read to the files to a file named %C2OSVD%/NumRowsRead.

   This file generates an operator dataset variable named NumRowsRead.

UNIX Script Example: UNIXGetInfo Script Operator

The UNIXGetInfo operator runs UNIX script on a UNIX touchpoint.

Script (UNIX)

The UNIX script example creates two indexed fields, *HostList* and *OSList* in the UnixGetInfo operator dataset. It reads the source file line-by-line and assigns host names ($host) to indexed values in sequential *HostList* fields and operating systems ($opsys) to indexed values in sequentially numbered *OSList* fields, starting with 0, and finishing at one less than the number of rows read from the source file.

The UNIX Shell script parameter variables $1, $2, etc. are set by the first, second, etc., entries of the Parameters input area of the calling *Run Script* operator. In this case, only one parameter is being passed, which is used to set SourceFile. The number of rows read are assigned to the operator dataset variable named *NumRowsRead*. The sleep 30 line has no purpose other than to pause the operator and give the user 30 seconds to examine the folders and files created in the C2OSVD location. This line would not be included in a production script.

```
#!/bin/ksh
SourceFile=$1
#known to be reading 2 variables, host and operating system
mkdir $C2OSVD/HostList
mkdir $C2OSVD/OSList
integer counter=0
while read host opsys; do
        echo -n $host > $C2OSVD/HostList/${counter}
        echo -n $opsys > $C2OSVD/OSList/${counter}
        counter=$counter+1
done < $SourceFile
echo -n $counter > $C2OSVD/NumRowsRead
sleep 30
exit 0
```

VBScript Example: WinGetInfo Script Information

The WinGetInfo script operator runs VBScript on a Windows touchpoint.

Script (VBScript)

The script creates two indexed fields, HostList and OSList in the WinGetInfo operator dataset. The script:

- Reads the source file line-by-line.
- Assigns host names (strHost) to indexed values in sequential HostList fields. The numbering starts with 0 and finishes at one less than the number of rows that are read from the source file.
- Assigns operating systems (strOS) to indexed values in sequentially numbered OSList fields. The numbering starts with 0 and finishes at one less than the number of rows that are read from the source file.

The script populates the oArgs.Item variable with the Parameters input area entries of the calling Run Script operator. The first entry populates oArgs.Item(0). The script assigns the number of rows it reads to the NumRowsRead operator dataset variable. The Wscript.sleep 30000 line pauses the operator for 30 seconds so the user can examine the folders and files created in the C2OSVD location. You would not include the Wscript.sleep 30000 line in a production script.

```
Dim oArgs
Dim oShell
Dim colProcessEnv
Dim objFSO
Dim objDir
Dim objFileIn
Dim objFileOut
Dim intCounter
Dim intExitCode
Dim strLine
Dim intSpacePos
Dim strHost
Dim strOS
on error resume next
set oArgs = WScript.Arguments
set oShell = WScript.CreateObject("WScript.Shell")
Set colProcessEnv = oShell.Environment("Process")
if oArgs.Count = 1 then          'must have the required argument
                                 to proceed normally,
                                 fails if more arguments are present
    strSourceFile = oArgs.Item(0)
    Set objFSO = CreateObject("Scripting.FileSystemObject")
    Set objDir = objFSO.CreateFolder(colProcessEnv("C2OSVD"))
    Set objDir = objFSO.CreateFolder(colProcessEnv("C2OSVD") & "\HostList")
    Set objDir = objFSO.CreateFolder(colProcessEnv("C2OSVD") & "\OSList")
    intCounter = 0
    Set objFileIn = objFSO.OpenTextFile(strSourceFile, 1)
    Do Until objFileIn.AtEndOfStream
        strLine = objFileIn.ReadLine
        intSpacePos = InStr(strLine, " ")
        strHost = Left(strLine, intSpacePos - 1)
        strOS = Right(strLine, Len(strLine) - intSpacePos)
        Set objFileOut = objFSO.CreateTextFile(colProcessEnv("C2OSVD") &
        "\HostList\" & intCounter)
        objFileOut.Write strHost
        objFileOut.Close
        Set objFileOut = objFSO.CreateTextFile(colProcessEnv("C2OSVD") & "\OSList\"
        & intCounter)
        objFileOut.Write strOS
        objFileOut.Close
        intCounter = intCounter + 1
    Loop
    objFileIn.close
    Set objFileOut = objFSO.CreateTextFile(colProcessEnv("C2OSVD") &
    "\NumRowsRead")
    objFileOut.Write intCounter
    objFileOut.Close
    intExitCode = 0
else
```

```
    'Wscript.echo "bad argument or required argument NOT present"
    intExitCode = 5
end if
Wscript.sleep 30000
on error goto 0
Wscript.Quit intExitCode
```

PerlScript Example: WinGetInfoPerl Script Operator

The WinGetInfoPerl operator runs PerlScript on a Windows touchpoint. In the example process, PerlScript runs on a Windows touchpoint, although it could also be run on a UNIX touchpoint.

Script (PerlScript)

The script creates two indexed fields, HostList and OSList in the WinGetInfoPerl operator dataset. The script:

- Reads the source file line-by-line.
- Assigns host names ($host) to indexed values in sequential HostList fields. The numbering starts with 0 and finishes at one less than the number of rows that are read from the source file.
- Assigns operating systems ($opsys) to indexed values in sequentially numbered OSList fields. The numbering starts with 0 and finishes at one less than the number of rows that are read from the source file.

The script populates the ARGV variable with the Parameters input area entries of the calling Run Script operator. The script assigns the number of rows it reads to the NumRowsRead operator dataset variable. The sleep 30 line pauses the operator for 30 seconds so the user can examine the folders and files created in the C2OSVD location. You would not include the sleep 30 line in a production script.

```
use strict;
my $filename = "";
print "  sample script to retrieve OS level data into variables within C2O\n\n";
my $numargs = @ARGV;
if ($numargs == 1) {
    $filename = shift @ARGV;
} else {
    print "enter path and filename to process:\n";
    chomp($filename = <STDIN>);
}
my $c2osvd = $ENV{'C2OSVD'};
mkdir $c2osvd;
mkdir $c2osvd . "/HostList";
mkdir $c2osvd . "/OSList";
open HANDLE, $filename or die "ERROR: unable to open $filename: $!\n";
my $counter = 0;
while (<HANDLE>) {
    my @fields = split;
    my $host = $fields[0];
    my $opsys = $fields[1];
    my $filename1 = $c2osvd . "/HostList/" . $counter;
    open HH, "> $filename1" or die "ERROR: unable to open $filename1: $!\n";
    print HH $host;
    close HH;
    my $filename2 = $c2osvd . "/OSList/" . $counter;
    open HH, "> $filename2" or die "ERROR: unable to open $filename2: $!\n";
    print HH $opsys;
    close HH;
    $counter++;
}
my $filename3 = $c2osvd . "/NumRowsRead";
open HH, "> $filename3" or die "ERROR: unable to open $filename3: $!\n";
print HH $counter;
close HH;
close HANDLE;
sleep 30;
```

Process Parameters

You can define parameters for process operators. The parameters can accept either literal strings or expressions. You can enter a value as a string without delimiting it in any way. Calculated parameters accept values as JavaScript expressions. Use single or double quotation marks to delimit literal strings in JavaScript.

Note: To help you identify CA Process Automation fields that do not accept expressions, their labels appear in italics.

Calculated parameters allow the following:

- Manipulation of module invocation results and other variables.
- Parameterization of operators.
- Definition of wait conditions based on Boolean expressions (preconditions and wait conditions). Wait conditions can be used to delay processing and synchronize the use of resources by different sequences of operators running simultaneously.

You can define parameters using dataset variables. Dataset variables are available to processes in the following contexts:

- In the CA Process Automation orchestrator context, datasets are referred to as *named datasets*. Named datasets define variables that are accessible to any process on the same orchestrator. A named dataset is accessed by specifying its full path name in an expression. To view or edit a named dataset, you double-click the dataset object in the Library Browser to open the object in the Dataset Designer.
- In the process context, there is the *process dataset*. The process dataset is available to any operator in a process. The process dataset is accessed by specifying the keyword process in an expression. You can edit variables in a process dataset in the Dataset palette in the Process Designer.
- For each operator in a process, there is an *operator dataset*. Variables in an operator dataset are available to the operator and to other operators in the same process. An operator dataset is accessed by specifying the operator name in an expression. You can edit variables in an operator dataset in the Dataset palette in the Process Designer.

More information:

Datasets (see page 189)
Calculated Parameters (see page 222)

Operator Properties

This section provides information about types of operator properties.

Literal Strings

To use a literal string value in a field that accepts an expression (its label is not in italics), enclose the string between delimiters. Use either single or double quotation marks. For example, you could type a literal string that specifies the path to a program to start a UNIX process as follows:

```
"/usr/smart/program"
```

The Escape Character in Literal Strings

You can use the escape character (\) in literal strings. Instead of being parsed by the CA Process Automation language interpreter, the character you enter after the escape character is interpreted literally. If a semantic action is attached to an escape character, the interpreter converts the action to its character equivalent rather than performing the semantic action.

For example, say that you want to include a double quotation mark character within a string delimited by double quotation marks. Precede your quotation character with the escape character, so that the parser does not interpret it as the string delimiter:

```
\"
```

To include the backslash character in a string, precede it with the escape character:

```
\\
```

More information:

String Data Type (see page 227)

Specify Paths in Literal Strings

When you use Microsoft Windows file nomenclature in a literal string to specify a path in an expression, backslashes must be escaped, as follows:

```
"C:\\IT PAM\\import\\script_ora1.bat"
```

In most cases, use *normalized* file names, with slash marks (/), even when specifying a path on a Microsoft Windows computer. For example:

```
"C:/IT PAM/import/script_ora1.bat"
```

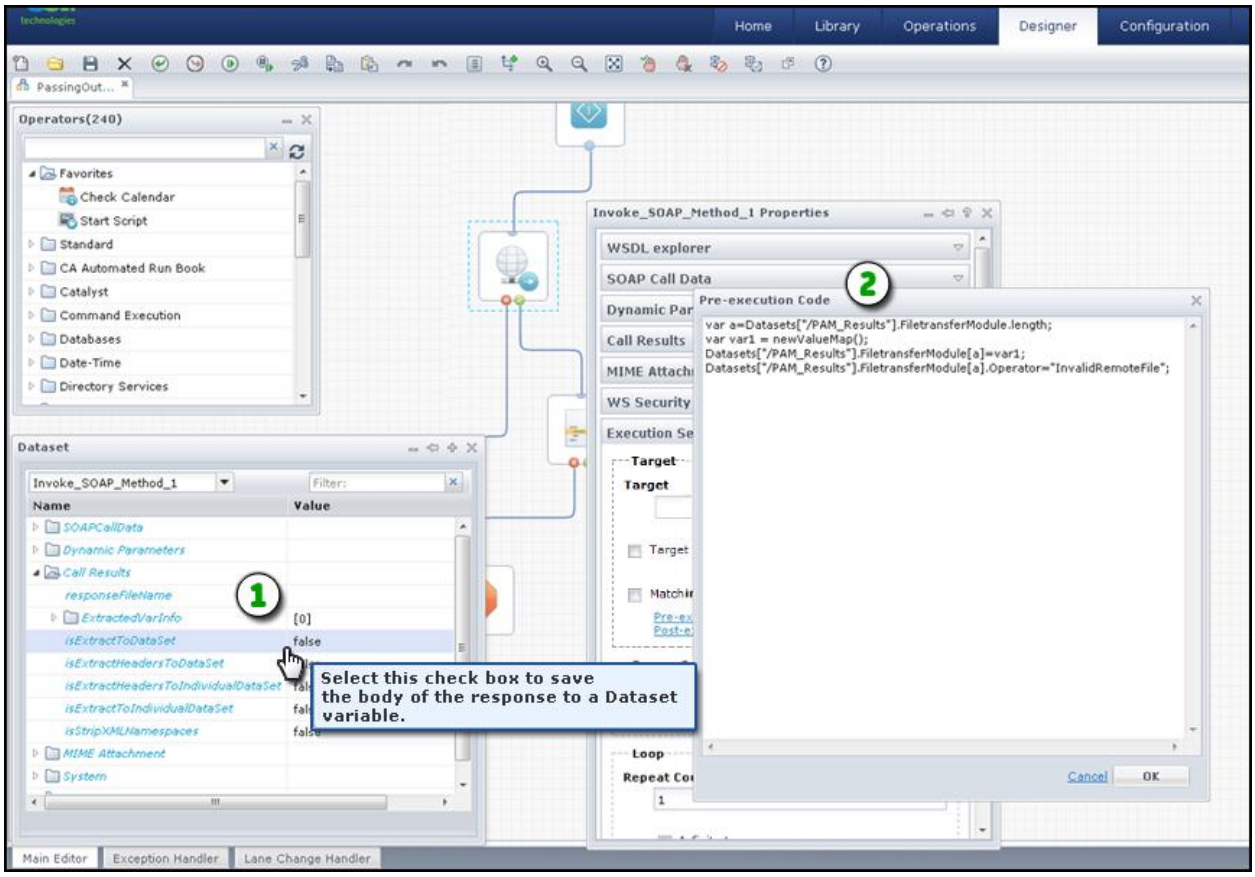
When you specify a path to a folder or object in a library, the root folder is represented by an initial slash mark, followed by the slash delimited folder hierarchy. For example:

```
"/Production/Processes/failover_process"
```

Dataset Variables in Parameters

In addition to literal strings, you can use dataset variables in a calculated expression. Variables in an expression are not enclosed between quotations marks. The name of the program in the following calculated expression includes variables and literal text:

```
"/usr/bin" + Datasets["/Application1/Settings"].ProgramName
```



Item:	Description:
1	**Dataset Variables and Values**: Hover over a dataset variable value to view a tooltip description for the variable if one is available.
2	**Reference Variables from Datasets in Pre-Execution Code**: Manually key in expressions that reference dataset variables.

When you want to use the output of one operator as the input for another operator, use the same variable reference by name. All input parameters are automatically converted to output dataset variables after the process finishes.

Relative Paths for Datasets

CA Process Automation can use either absolute or relative paths when accessing named datasets. Absolute paths are also known as full or fixed paths.

Example 1

Folder1 is under the root folder in the library. Folder1 contains two objects: *Process1* and *Dataset1*. You open *Process1*, double-click the Start Process operator, and locate the Process Name field in the Properties palette.

Rather than enter a value, you want to use or reference the value in a field that is called *ProcessName* in *Dataset1*. For the absolute path, you would specify:

```
Datasets["/Folder1/Dataset1"].ProcessName
```

You can also specify the path of Dataset1 relative to Process1. The same expression using a relative path is:

```
Datasets["Dataset1"].ProcessName
```

While CA Process Automation evaluates the relative path expression, it looks for Dataset1 in the same folder as Process1.

If you move Dataset1, the absolute path is no longer valid. To correct this situation, you would have to update it. However, as long as they are in the same folder, you can move Dataset1 and Process1 anywhere and the relative path is still valid.

Example 2

Similar to Example 1, you want to use a field in a dataset. This time, you want to use Dataset2, at the root level of the library. For the absolute path, you would specify:

```
Datasets["/Dataset2"].ProcessName.
```

The same expression using a relative path is:

```
Datasets["../Dataset2"].ProcessName.
```

This path expression tells the application to look in the folder which is the parent for Folder1 (the folder containing the process). Folder1 is the starting point. The code, "../Dataset2," literally says to go *up one level* in the folder hierarchy and look for Dataset2. In this case, the parent folder of Folder1 is the root folder and the application looks for Dataset2 there.

These concepts, summarized in the following two points, also apply to Linux/UNIX, Windows, and any environment that supports uniform naming conventions.

- A parent folder "/" exists.
- All other folders are children of the parent folder.

When these conditions exist, you can simplify complex expressions using relative paths. For example:

```
"../../"
```

Note: Relative or absolute paths can be used as expressions in any object.

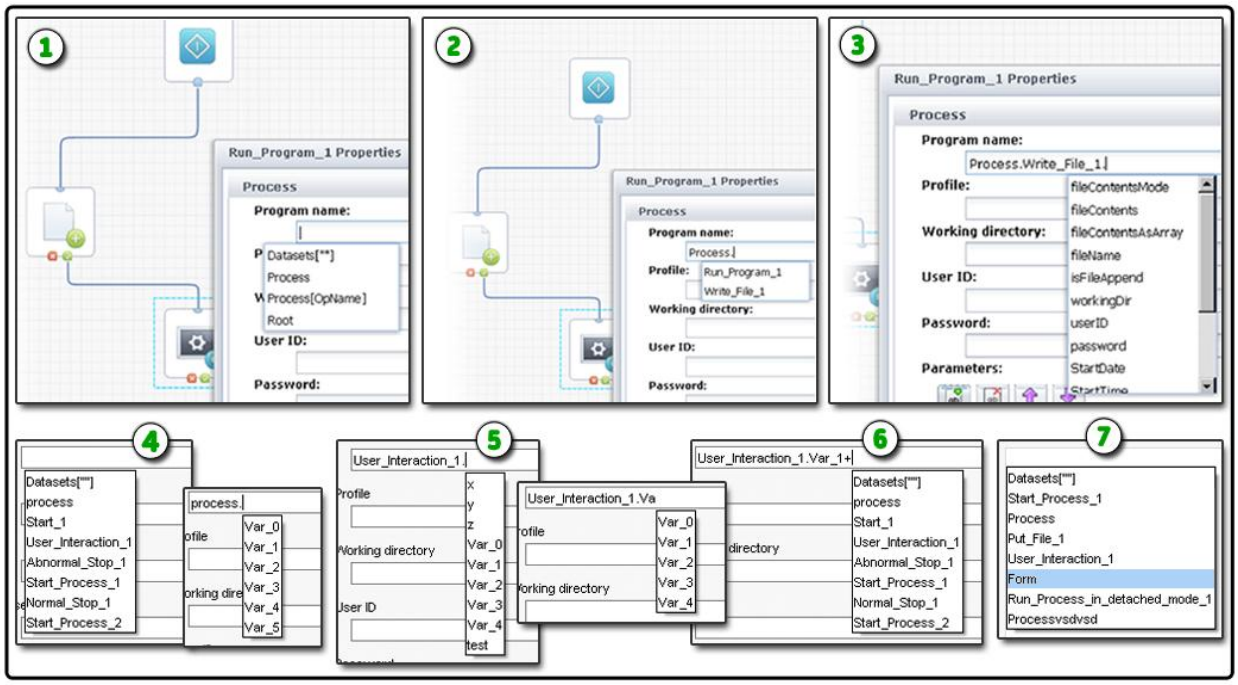
Dataset Variable Name Assistance

CA Process Automation datasets define and store groups of variables shared across process instances. CA Process Automation allows you to use the variables as input parameters in the execution of process instances.

After creating dataset variables for processes and operators, you might not remember variable names. Variables that are part of an operator and not something that you have defined are easy to forget. To assist you in referencing these process and operator dataset variables without going back and forth between operators, the application provides an in-context editing assistant. Known as *Dataset Variable Name Assistance* and invoked by pressing *Ctrl+Space,* this feature helps you:

- Identify dataset variables and apply them to any field that accepts expressions.
- Reduce your process development time
- Reduce errors in scope or syntax.

Most of the text fields that accept expressions as input support *Dataset Variable Name Assistance*. Refer to the following graphic for examples.



Item:	Description:
1	**Suggested Values for Run Program Operator**: In this example, the user has pressed **Ctrl+Space** to pop-up *Dataset Variable Name Assistance*. A list of values at the global and root levels appears.

Item:	Description:
2	**Suggested Operators**: After entering `Process.` a list of operators appears at the process level.
3	**Suggested Variables**: After specifying the process scope and operator, a list of variables appears at the operator level.
4	**Suggestions Based on Scope**: When you first enter the field, the suggestions are appropriate at that level or scope. After specifying the process parameter, the scope of suggestions is reduced to appropriate variables at the process level.
5	**Updates Based on User Input**: As you enter text, the application dynamically updates suggestions. In this example, typing Va reduces the possible choices to only the matching entries that begin with *Va*.
6	**Multiple Datasets**: The application dynamically updates the scope of the suggestions when you specify expressions that span more than one dataset.
7	**Smart Suggestions**: When appropriate, the application will include additional parameters based on scope and context. For example, a form operator may include the *Form* parameter and a Start Process operator may include the reserved word *Caller*. The application also automatically supports value map and array variables.

Use Dataset Variable Name Assistance

Use the *Dataset Variable Name Assistance* feature in various supported contexts throughout the application.

Follow these steps:

1. Open and check out the automation object to edit. For example, a process, form, or dataset.
2. Press **Tab** or click in a text field that accepts expressions.
3. Press **Ctrl + Space**.

   A list of suggested values appears.
4. Select the value to use. To select process or operator dataset variables, enter the following string value in the text field:

   `Process.`
5. Enter an operator name for operator-specific variable name assistance.
6. To filter data based on text input, begin typing or entering characters.

   The list of suggestions dynamically updates as you enter text.
7. Select the values to use.

Password Parameters

Characters entered in the password field show as asterisks (*). Passwords saved to a password type dataset field are encrypted. An expression in a calculated parameter can only assign the value of a password field to another password field.

Execution Settings

Execution Settings specify how and where to execute an operator. The Target and Timeout settings are available for operators in both processes and schedules. The Processing and Loop groups are only available for operators in processes.

Target Settings

The Target field in the Execution Settings section of the Properties palette specifies where the operator runs. Use the following guidelines when you specify a target in the Target field:

- If the target is an Orchestrator, enter its touchpoint. You can leave the Target field blank to target the Orchestrator that is running the process.

  Note: Do not specify the IP address of a computer hosting a clustered orchestrator.
- If the target is an agent, enter its touchpoint. If the touchpoint is mapped to multiple agents with the same priority, the exact execution target is selected for load balancing.
- If the target is a specific agent and the touchpoint is mapped to multiple agents, enter its agent ID.
- If the target has no agent but the target has a proxy touchpoint, enter its proxy touchpoint.
- If the target is a remote host that a host group references:
  - Select the Match target in Host Groups only (Enabled) drop-down menu option.
  - Select the Lookup DNS when matching target in Host Groups (Enabled) to identify the DNS if you are matching the target in the host group.
  - Select Target is a calculated expression. Create a dataset that references an IP address or FQDN of a host that meets the pattern criteria of a host group.
  - Enter the dataset reference to the IP address or FQDN in the Target field.

  Important! Use a dataset for the IP address or FQDN if the process with this operator is destined to be imported as a content package. Only a dataset can be modified in this scenario. That is, the Target field entry cannot be modified after import.

Operator Dataset Variables

The operator dataset contains variables that are associated with an operator. You can view, create, edit, or delete variables and their associated values at design time. The variables are available to the operator with which they are associated or any other operator in your process as soon as the process starts. After the operator runs, it automatically creates other variables in the operator dataset. Some variables are standard and define information such as the start time, stop time, and result. Other variables are information specific to each operator.

You can use the dot notation or the bracket notation with expressions to access an operator dataset variable from any operator in a process:

```
Process_name.Operator_name.field_name
Process_name[OpName_expression].field_name
Process_name[OpName_expression][field_name_expression]
```

Note: You can also use IconName in place of OpName.

The expressions return the name of the operator or variable, as indicated. The following syntax returns an element in an indexed field, where *n* is the element number:

```
field_name[n]
```

The pre-execution and post-execution code for an operator can use the OpName keyword to access the name of the current operator. To specify an operator dataset variable in the pre-execution or post-execution code of that same operator, use the following syntax:

```
Process[OpName].field_name
```

For example, use the following post-execution code statements:

- Assign the operator name and the value of the associated Result variable to the process dataset variables iName and iResult
- Create an operator dataset variable named World:

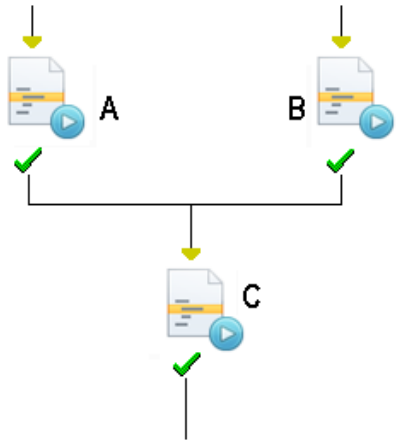
```
Process.iName = OpName
Process.iResult = Process[OpName].Result
Process[OpName].World = "Hello world!";
```

If you know the operator name at design time, you can use the literal operator name to reference the dataset variables in an expression.

Occasionally, however, you do not know the operator name at design time. For example:

- When you are editing pre-execution or post-execution actions for a custom operator object.
- When you must access the operator dataset variables in one of several operators, but you do not know which operator ran until run time.

You can use pre-execution or post-execution code in each operator to save its name to a variable, as with the iName variable in the previous example. The variable could be a process dataset variable or it could be an operator dataset variable that belongs to the operator that requires the name. For example, in the following illustration, the process runs either OperatorA or OperatorB before it reaches OperatorC. OperatorC can then use the operation name that OperatorA or OperatorB saved to a process variable to access dataset variables.



Using variables instead of fixed names makes code modular and interchangeable among operators in a process.

More information:

Create a Named Dataset Object (see page 191)
Calculated Parameters (see page 222)

Processing Properties Settings

Processing Properties define the pre-execution code and post-execution code.

Pre-Execution Code and Post-Execution Code

Pre-execution and post-execution JavaScript code is processed before and after an operator runs. Pre-execution code is typically used to set up loop variables or other variables that can be used as part of the operator. Post-execution code is typically used to process the results of an operator or to increase loops indexes.

The *OpName* keyword can be used to access the operator dataset. For example, the following statement inserts the operator name into a message and assigns the string to a new operator dataset variable named *operatorMsg*:

```
Process[OpName].operatorMsg = “Recovery Operator” + OpName + “restructuring main server at “+System[“Date”]+ “:” + System[“Time”];
```

Typically, you must include code that is closely associated with processing of your specific operator.

For unrelated code, a best practice is to add a separate Calculation operator to the process.

More information:

Specify Operator Dataset Variables (see page 244)

Set Operator Status

During the processing of pre- and post-execution code you have the option of specifying a value for *setOperatorStatus*. You can force the operator to either fail or pass.

To specify the success of the operator:

```
setOperatorStatus ("Success",Operation Result,"reason")
```

To specify the failure of the operator:

```
setOperatorStatus ("Failure",Operation Result,"reason")
```

Loop Settings

The Loop property specifies the number of times that an operator is repeated. When an operator is run in a loop, the exit conditions and the connecting links from the operator are evaluated only when the loop is terminated.

Loop settings have the following properties:

Repeat count

Specifies the number of times that an operator should be repeated. 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. The expression is evaluated after the operator has executed. 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.

Infinite Loop

Creates an infinite loop. The operator or process keeps repeating until either the process is interrupted or the loop is stopped from a different branch using a stop loop command link to the Loop operator.

Delay between iterations

CA Process Automation supports an inherent delay option for every operator that has a loop option. The *Delay between iterations* text field takes an expression. The expression is evaluated into an integer and the value is taken as delay in seconds. Before the next iteration is run, there is a delay as specified by the user after an iteration in the loop.

The minimum value for delay is zero. The default delay is zero seconds. CA Process Automation takes delay as zero seconds for all invalid inputs.

Timeout Settings

Timeout settings give the users ability to set a timeout as part of every operator. If the operator has not finished by the specified time defined in the Timeout settings, the execution takes a timeout exit port. Users still retain the choice to end the execution of the operator and take the timeout path or let the operator continue with the execution.

Timeout settings have the following properties:

No Timeout

Specifies that there is no timeout set for the operator (enabled by default).

To specify a timeout value, clear the *No Timeout* check-box.

Type

Specifies the type of timeout. Select one of the following timeout types.

Duration

Specifies the timeout duration in seconds.

Target Date

Specifies the timeout date (MM/DD/YYYY) and time (24 hours).

Duration/Target Date-Time

Defines the timeout duration or the target date for the operator.

Action

You can select a timeout action from the following:

Abandon

Specifies the flow is abandoned after the timeout of the operator. The following actions are performed:

- The operation executes in detached mode.
- Operator will timeout.
- Post-execution code is executed.
- Process flow is through the timeout branch.
- The delay operator is executed.

Abort

Specifies the flow is aborted after the timeout of the operator. The following actions are performed:

- The operator is aborted.
- Post-execution code is executed.
- Process flow is through the timeout branch.
- The delay operator is executed.

Continue

Specifies the flow continues after the timeout of the operator. The following actions are performed:

- The operator and the operation are in running state.
- Process flow is through the timeout branch.
- The delay operator is executed.
- Post-execution code is executed after the operator is executed.

Reset

Specifies the flow is reset after the timeout of the operator. The following actions are performed:

- The operator and the operation are in running state and are reset.
- Post-execution code is executed.
- Process flow is through the timeout branch.
- The delay operator is executed.

Calculated Parameters

Parameters in dialogs and properties pages that accept expressions are called *calculated parameters*. Values for calculated parameters must be entered as JavaScript expressions. You can use JavaScript expressions to set dataset values, perform calculations in the Interpreter Service Operators, as part of pre- and post-execution code, and to specify parameters wherever an expression is allowed. Most fields accept calculated parameters. Fields that do not accept expressions as input have italicized labels.

Expressions

An *expression* is any logical statement the application can evaluate to return a value. It can include any combination of the following types of data:

- integers (including long, double, and so forth)
- strings
- functions
- variables
- references to other operators
- JavaScript
- dates and times
- valuemaps
- literal values
- calculated values
- logical *And*, *Or*, and *Not* keywords or symbols (&&, ||, !)
- comparison operators (==, !=, <, >, <=, >=, <>)
- enclosing parentheses

Expressions are valid input for all fields, including JavaScript operators, functions, custom exit ports, and operator property fields, except for fields labeled in italics.

Reserved Words in Expressions

A number of words are reserved in CA Process Automation expressions. These include CA Process Automation reserved words, system functions, and JavaScript keywords. Do not use these words as identifiers (such as for variable or other object names) in expressions.

CA Process Automation Reserved Words

- Caller
- DateAdjust
- Process
- CurrentIndex
- FreeRes
- Size
- Datasets
- OpName
- System

CA Process Automation System Functions

See the *Content Designer Reference* for detailed information about each of the CA Process Automation system functions.

Reserved JavaScript keywords

- break
- do
- if
- switch
- var
- case
- else
- in
- this
- void
- catch
- false
- instanceof
- throw
- while
- continue
- finally
- new
- true
- with
- default
- for
- null
- try
- delete
- function
- return
- typeof

Data Types

CA Process Automation expressions support JavaScript data types. Variables or constants represent data.

Boolean Data Type

Boolean values have two possible values: true and false. JavaScript converts the true and false literals to 1 and 0 when necessary.

Boolean values are usually the result of comparison made in your JavaScript expressions. Boolean values are typically used in control structures. For example, the JavaScript if-then statement performs one action if it is true and a different action if it is false.

The following examples are all valid Boolean expressions:

```
Process.A == 1
!(Process.A == 1)
(Process.A != 1)
(Process.A == 1) && (Process.B > 0)
```

Date Data Type

The date type stores and returns dates from Dataset variables. The format of Date type can be specified as part of the data type. For example, you can specify that it represents a date as month then day or day then month.

Double Data Type

The double numeric data type can have a decimal point. The traditional syntax is used for real numbers. A real value is represented as the part of the number, followed by a decimal point and the fractional part of the number. This type can store real numbers from -1.7976931348623157E308 to 1.7976931348623157E308.

Floating-point literals can be represented using exponential notation—a real number followed by the letter e (or E), followed by an optional plus (+) or minus (-) sign, followed by an integral exponent, in the following format:

[digits][.digits][{E|e}[{+|-}]digits]

Examples

2.718

2345.789

7.748E-5

Integer Data Type

The 16-bit integer data type can be typed as literal values in an expression. You can exactly represent all integers from -2,147,483,647 to +2,147,483,647.

JavaObject Data Type

This data type lets you store Java objects in CA Process Automation.

All JavaObject variables are read-only. Their CurrentValue and Read-Only fields are disabled. You can only edit the following fields:

- Type
- Page
- Description
- Array

A JavaObject that is not empty shows its class type in the associated CurrentValue field. The CurrentValue field for an empty JavaObject is set to [JavaObject].

The actual JavaObject variable value is the serialized string version of the Java object, but CA Process Automation does not show this serialized string. Instead, it shows the Java class type of the object.

A manually created JavaObject is always empty because you cannot enter its value directly in CA Process Automation. JavaObject variables are typically saved into a dataset after a Run Java Code operator finishes running.

Long Data Type

The Long Data type is a 32-bit field that can be typed as literal values in an expression. You can exactly represent all integers from -9,223,372,036,854,775,808 to +9,223,372,036,854,775,808.

Password Data Type

The Password data type stores passwords in an encrypted format in Dataset variables. The value stored in a password type variable is not viewed by users.

Object Reference Data Type

The Object Reference data type stores the complete or reference path to an object in a Library. You can use it anywhere that an expression requires the path to an object. The Object Reference Filter on the object reference type constrains it to referencing one or more specified object types.

String Data Type

You can type strings as literal values in an expression. Functions and JavaScript operators in an expression also return strings. To distinguish between identifiers (such as variable names) and literal strings, enclose the literal strings in string delimiters. You can use either single quotation marks or double quotation marks as string delimiters. For example, to use a literal string for the name of a program to start a UNIX process, type it as follows:

```
"/usr/smart/program"
```

In text boxes marked with an asterisk (*), enter either an expression or a literal string between quotation marks. The asterisk indicates that CA Process Automation evaluates the contents of the text box as an expression. Do not use quotation marks to delimit expressions.

Text boxes that expect literal values are not labeled with an asterisk (*). In such cases, do not delimit strings with quotation marks.

JavaScript uses the backslash character (\) to escape literal strings. If a semantic action is attached to an escaped character, the character is converted to its character equivalent instead of performing the semantic action. For example, 'C:\\pam' is converted to 'C:\pam'.

CA Process Automation interprets any character that follows the escape character literally instead of parsing it through the language interpreter. For example, to include a double quotation mark in a string, precede it with the escape character, \". In this case, the parser does not interpret the double quotation mark as the string delimiter. To include the backslash in a string, precede it with the escape character, \\.

To include the same character that you use as your delimiter in your string, escape the character when do not use it as a delimiter. For example, the following string is delimited with single quotation marks. The string includes single quotation marks as escaped characters and double quotation marks as literal characters:

```
'Database engine can\'t find database \'CHECKERS\' on server "GAMES"'
```

The same string, delimited with double quotation marks, includes the single quotation marks as literal characters and the double quotation marks as escaped characters:

```
"Database engine can't find database 'CHECKERS' on server \"GAMES\""
```

The following table shows JavaScript escape sequences:

Sequence	Character Represented
\0	The NUL character (\u0000)
\b	Backspace (\u0008)

Sequence	Character Represented
\t	Horizontal tab (\u0009)
\n	Newline (\u000A)
\v	Vertical tab (\u000B)
\f	Form feed (\u000C)
\r	Carriage return (\u000D)
\"	Double quotation mark (\0022)
\'	Apostrophe or single quotation mark (\0u0027)
\\	The Latin-1 character specified by two hexadecimal digits NN

ValueMap Data Type

The ValueMap data type contains a collection of variables of various data types. You can use it to create a group of variables within a dataset. This type is also known as a record or a structure.

JavaScript Operators

You can use JavaScript operators to build string, integer, Boolean, and logical expressions from a combination of entities (integers, strings, functions, and datasets). The number of operands that they expect characterizes JavaScript operators. Most JavaScript operators are binary operators that combine two expressions into one more complex expression.

JavaScript also supports several unary operators that convert a single expression into one more complex expression. This section discusses the JavaScript operators that are most commonly used in CA Process Automation expressions.

Array and Object Access Operators

JavaScript uses dot (.) notation for arrays and object access. You can access elements of an array using square bracket notation ([ ]) and elements of an object using dot (.) notation. JavaScript treats dot and square bracket notation as operators.

Dot notation uses the following format:

`object.identifier`

The *identifier* operand can be the literal name of the property, method, or variable name (in a dataset), without single or double quotation marks. The operand cannot be a string or variable that contains a string.

Square bracket notation uses the following formats:

`array[expression] //`

The *array* operand refers to an array, and the [*expression*] operand evaluates to an integer value for an array index.

`object[expression] //`

The *object* operand refers to an object, and the [*expression*] operand evaluates to a string that names a property of the object.

Note: Unlike dot notation, where the second operand is an identifier, the [expression] operand is a string.

Square bracket notation allows access to array elements and object properties. Square bracket notation also allows access to object properties without restricting the *identifier* operand as dot notation does.

Assignment Operators

JavaScript provides the normal assignment operator and arithmetic assignment operators that provide shortcuts for common arithmetic operators.

Operator	Example	Equivalent
=	a = b	
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b

Arithmetic Operators

JavaScript uses the following operators to combine integer values:

Operator	Description
*	Multiplication
/	Division
+	Addition or unary plus
-	Subtraction or unary minus
%	Modulo
++	Increment
--	Decrement

Arithmetic calculations in an expression follow algebraic rules:

- When an expression contains multiple arithmetic operators, multiplication and division are calculated first, then subtraction and addition.
- When operators are of the same order, they are calculated from left to right.
- You can use parentheses to change the precedence. Calculations inside parentheses are evaluated first. If parentheses are nested, the most deeply nested calculation has precedence.

String Concatenation Operator

The interpreted language has the following operator for combining string values.

Operator	Description
+ (strings)	String concatenation

Use the string operator to combine, or concatenate, two or more character strings into a single character string. For example, the expression "ABCD" + "123" returns the concatenated string “ABCD123”.

Logical Operators

The interpreted language uses the following logical (or Boolean) operators to combine the outcomes of Boolean functions or operators.

Operator	Description
&&	Logical AND
||	Logical OR
!	Logical NOT

Logical operators return True or False. They recognize null, 0, "", or undefined as False and any other non-zero operand as True.

Equality and Comparison Operators

Comparison operators are used with strings and numeric data. Comparison operators evaluate to a Boolean value. They return True or False based on the outcome of the tested condition.

Operator	Description
==	Equal to
===	Identity
!=	Not equal to
!==	Non-identity
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
?:	Tertiary conditional operator The expression x1 ? x2 : x3 returns x2 when x1 is True or x3 when x1 is False.

Operator Precedence

The CA Process Automation interpreted language operators follow standard computational precedence rules, as shown in the following table. Operators at the same level of precedence are executed from left to right.

Precedence	Operator
1	. () [ ]
2	++ -- - (unary) + (unary) !
3	* / %
4	+ (addition) - (subtraction) + (string concatenation)
5	< <= > >=
6	== != === !==
7	&&
8	||
9	?:
10	= *= /= %= += -=

Keywords for Accessing Datasets

The following table describes the keywords that reference datasets in various contexts:

Dataset	Dataset Context	Description

Datasets	Named Dataset	Uses the following format to access a named dataset in a CA Process Automation Library:
		`Datasets[dataset_path].field_name`
		dataset_path
		A CA Process Automation expression that evaluates to the full path for a named dataset in the current Library.
		For example, the following path references a dataset named CxLinuxDev located in the Data subfolder of the Demo folder in the Library:
		`Datasets["/Demo/Data/]`

Dataset	Dataset Context	Description
Process	Dataset of a Process	Accesses a process dataset in the following format: Process.*field_name* or Process[*expression*] Process.*field_name* or Process[*expression*] **field_name** The dataset variable. **expression** A variable or other expression that returns the name of a field. For example: Process.x = 5; Process.fn = "x"; Process.y = Process[Process.fn]; A process dataset is defined in a process. Each time a process starts, it creates a copy of itself (called an *instance* of the process), including its process dataset. The original process object determines the initial values for the dataset. Changes to a dataset in a process instance do not affect the original.

Dataset	Dataset Context	Description
Caller	Process Dataset for a parent process when it is starting a child process	Passes values between processes in a call hierarchy when one process uses the Start Process operator (in attached, detached, or inline mode). The process dataset initialization code option of the Start Process operator specifies these assignments. For example, when ProcessA calling ProcessB needs to initialize fields in the ProcessB dataset, ProcessA specifies ProcessB in the process dataset initialization code. In this context, Caller refers to the dataset of the parent (ProcessA), and Process refers to the dataset of the child (ProcessB). In the context of the process dataset initiation code, the Process keyword is always required to reference a variable in the child process dataset. If you omit both the Process and Caller keywords on a variable name in the process dataset initiation script, CA Process Automation only looks for a calculation-scope variable. The product does not check for a similarly-named variable in either the parent or child dataset. For example, the following code fails if no calculation-scope X was previously created in the calculation context: Process.X = Caller.X; Process.Y = X + 100;

Dataset	Dataset Context	Description
none	The current calculation and, in some cases, the process dataset	If you omit the Process keyword on the left side of an assignment statement, CA Process Automation always creates or assigns a value in the scope of the current calculation (a calculation variable). A calculation variable exists as long as CA Process Automation is processing a calculation field. For example, the following code creates a calculation variable equal to the value 5: a = 5 If you omit the Process keyword in other contexts (such as on the right side of an assignment statement), CA Process Automation looks first for a calculation scope variable or a Process variable with the same name. For example: Process.a = 1 Process.b = 2 a = 5 x = a y = b CA Process Automation creates two variables in the process dataset (a=1 and b=2), and two calculation variables (x=5, y=2).
Process.operator_name or Process[expression]	Operator dataset in a process	Enables access to an operator dataset, where *operator_name* is a string that specifies the name of an operator in a process. For example: Process.y = Process.emailOp.subject Expression returns the name of an operator in a process, for example: Process.opName = “emailOp” Process.y = Process[Process.opName].subject **Notes:** - For information about specifying operator dataset fields and for a list of system generated fields in operator datasets, see Specify Operator Dataset Variables (see page 244). - For more information about specifying operator variables in operator properties settings, see Dataset Variables in Parameters (see page 210).

Dataset	Dataset Context	Description
System	System Dataset	Enables access to the system dataset. Fields in the system dataset represent CA Process Automation system information, such as the host name, date, and time. The system dataset is read-only. **Note:** For more information about fields in the system dataset, specifying system variables, and a list of system variables, see Specify System Dataset Variables (see page 245).
Root	Parent Dataset	Enables an inline child process to access the process dataset of a parent instance. You can use the keyword Root to access the Process dataset of the root instance. **Examples:** - ProcessA starts an inline process ProcessB. ProcessB starts another inline process ProcessC. ProcessC uses the keyword Root to access the dataset of ProcessA. - ProcessA starts ProcessB in non-inline mode and ProcessB starts ProcessC in inline mode. ProcessC only has access to the dataset of ProcessB (and not ProcessA) using the keyword Root.

Access Dataset Fields in Expressions

Dataset field values in an expression use either (or both) bracket ([*expression*]) or dot (.*field*) notation to specify a dataset or a field in a dataset.

Syntax for Specifying the Value of a Field

A value in a single unindexed field is accessed in an expression using dot notation:
dataset_reference.*field_name*

The *dataset_reference* parameter corresponds to one of the keywords previously described. The *field_name* parameter corresponds to a field name in the dataset. The following example returns the value of field Y in a process dataset:

```
Process.Y
```

Alternatively, a value can be accessed using bracket notation:
dataset_reference[*field_name_expression*]

The *field_name_expression* parameter is an expression that returns the name of a field in the referenced dataset. The following example returns the value of field Y in a process dataset:

```
Process["Y"]
```

The best practice is typically to use the bracket notation for an expression to specify a dataset or field name in a dataset.

Specify the Value of an Element in an Indexed Field

To access a value in an indexed field (array), use the following syntax:

dataset_reference.*indexed_field_name*[*index*]

The *indexed_field_name* parameter defines the field name in the dataset. *Index* is an integer that addresses an indexed element in the array.

To specify the *indexed_field_name* string with the bracket notation, use the following syntax. In the example, *indexed_field_name_expression* is a CA Process Automation expression that returns the *indexed_field_name* string.

dataset_reference[*indexed_field_name_expression*][*index*]

Like JavaScript arrays, CA Process Automation indexed fields are arrays of arrays instead of true arrays. To access an element in an array of arrays, use the [ ] operator twice. For example, for the two-dimensional indexed integer-field named integers, every list element integers[*x*] is itself an indexed list of integers. To access a specific integer in the indexed field, you would write the expression integers[*x*][*y*]. In general, for any indexed field of *n*-dimensions, you use the following syntax to access any data element:

dataset_reference.*indexed_field_name*[*index1*][*index2*] ... [*indexn*]

The *index* parameter is an integer or an expression that returns an integer. The parameter has a value from 0 (for the first value in an indexed list) to the length of the list minus 1. Evaluation of the index is circular, so when the *index* value exceeds the length of an indexed list, the following formula determines its value:

```
Actual-index = index % length-of-the-list;
```

In other words, for *n* elements in an indexed field, you get the following results:

- An index of *n* returns element 0
- An index of *n*+1 returns element 1
- An index of *n*-1 returns the last element

The following table illustrates the results of accessing elements of an indexed field in a process dataset:

Expression	Description
value = Process.X[2]	Refers to the third element of an indexed field X of the process dataset.
value = Process.X[18]	For an indexed field X with a size of 19, this expression refers to element 19 of X, the same as Process.X[18].
value = Process.X[Process.Y + 2]	An expression calculates the index.
value = Process.A[5][2]	Returns the value in a two-dimensional array. The third element in the sixth indexed list that the array defines addresses the array.

Access the Length of an Indexed Field

CA Process Automation supports the JavaScript length property for arrays and its own size property for accessing the length of an indexed field. The length property is read-only; the size property lets you change the number of elements in an indexed field. The length property uses either dot or bracket notation to return the number of elements in an indexed field:

dataset_reference.*indexed_field_name*.*length*

dataset_reference[*indexed_field_name_expression*].*length*

The size property works the same way, using either dot or bracket notation:

dataset_reference.*indexed_field_name*.*size*

dataset_reference[*indexed_field_name_expression*].*size*

Because an indexed field is a zero-based array, the length and size properties always return one more than the index for the last element in a field. Therefore, when length or size returns *n*, a field contains 0, 1, ..., *n*-1 indexed elements, and the index for the last element in the array is *n*-1.

Assigning a new value to the size property extends or truncates the number of elements in an indexed field. Decreasing the value for the size property removes elements from the upper end of an indexed field and deletes values stored in the deleted elements. The following code uses the size property to increase the length of an array X by one element, then assigns 25 to the new element:

```
Process.X.size = Process.X.size + 1;
Process.X[Process.X.size - 1] = 25;
```

For a multidimensional array, the size or length property returns the number of elements in an array address to which it is appended.

For example, for a two-dimensional array named matrix[a][b], the following syntax returns the size of the first dimension of matrix, with the containing elements 0...size1-1:

```
size1 = matrix.length
```

The following syntax returns the size of the second dimension of matrix, given first dimension element 2, with b containing elements 0...size2-1 when a = 2.containing elements 0...size1-1:

```
size2 = matrix[2].length
```

The following example addresses elements of a multidimensional indexed field by looping through all elements in a two-dimensional indexed field (an array of arrays) in the process dataset variable named matrix. The code assigns the value for each element to a one-dimensional indexed field in the process dataset variable named values:

```
var i; j; k=0;
for (i=0; i < Process.matrix.length; i++)
{
        for (j=0; j < Process.matrix[i].length; j++)
    {
        Process.values[k] = Process.matrix[i][j]
                k++
    }
}
```

Access Methods on an Indexed Field

Indexed fields support the following JavaScript array methods:

concat()

Concatenates elements to an array.

join()

Converts all array elements to strings and concatenates them.

pop()

Removes an item from the end of an array.

Note: If the array belongs to the operator dataset of another operator, do not use the pop method for JavaScript arrays to evaluate operator parameters.

push()

Pushes an item to the end of an array.

reverse()

Reverses the order of elements in an array.

shift()

Shifts an element from the beginning of an array.

slice()

Returns a subarray slice of an array.

sort()

Sorts elements of an array.

splice()

Inserts, deletes, or replaces array elements.

toLocaleString()

Converts an array to a localized string.

toString()

Converts an array to a string.

unshift()

Inserts elements at the beginning of an array.

For information about using these methods, see a JavaScript reference guide.

Specify Named Dataset Variables

Fields in dataset objects (named datasets) are identified in expressions using either dot (.string) or bracket ([expression]) notation:

```
Datasets[path_expression][variable_name_expression]
```

```
Datasets[path_expression].variable_name
```

path_expression

Represents any JavaScript expression that evaluates to a path descriptor for a dataset object in the current CA Process Automation Library. A path to any object in the Library starts with a forward slash (/) for the root element, followed by the slash-delimited folder hierarchy, and ends with the object name.

variable_name_expression

Represents an expression that returns the name of a field in the dataset object.

variable_name

Represents the actual name of a field in the dataset object.

Either of the following syntaxes is valid for referencing field Y in the named dataset Coordinates located in the /MathValues folder:

```
Datasets["/MathValues/Coordinates"].Y
```

```
Datasets["/MathValues/Coordinates"]["Y"]
```

More information:

Relative Paths for Datasets (see page 211)

Specify Process Dataset Variables

The process dataset contains variables that the developer defines or that CA Process Automation defines automatically when a process instance starts. The Process keyword is used to access variables in the process dataset. You can use either dot notation or bracket notation to specify a process variable in an expression:

```
Process.variable_name
Process[expression]
```

The expression can specify the variable name as a literal string in the format:

```
Process["variable_name"]
```

For example:

```
Process[“StartDate”]
```

```
Process.StartDate
```

CA Process Automation defines the following process system variables automatically when it runs a process:

CallerUser

The user ID that started the process instance. When a parent process uses a Start Process operator to start a child process, the parent process passes the CallerUser value forward. When a Start Process operator in a scheduled task starts another process, CallerUser is blank.

DisplayName

The name of the process object as seen in the library.

effectiveUser

The current owner of the process object.

EndDate

The date when this process instance ended, in the format:

```
MM/DD/YYYY
```

EndTime

The time when this process instance ended, in the format:

```
HH:MM:SS
```

InstanceName

The name of the original process object ending in a unique run-time object identifier that identifies each instance of a process. For example, 372 is appended to process_1, resulting in an InstanceName of process_1_372.

InstanceUUID

For internal use only.

ObjectID

Object identifier for internal use only.

ParentProcessROID

The unique run-time object identifier for the parent process that started the child process.

rootUUID

For internal use only.

RuntimeROID

A unique object identifier that the application appends to the process DisplayName to identify each process instance. For example, RuntimeROID 372 is appended to process_1, resulting in an InstanceName of process_1_372.

ScheduledStartTime

The date and time when the process was scheduled to start.

ServerName

The name of the server that is associated with the touchpoint.

ServerID

For internal use only.

StartDate

The date when this instance of the process was created, in the format:

`MM/DD/YYYY`

StartTime

The time when this instance of the process was created, in the format:

`HH:MM:SS`

TouchpointName

The name of the Orchestrator managing the running process.

Specify Operator Dataset Variables

Operator datasets contain variables that the developer defines at design time or that CA Process Automation defines at run time for a specific operator. Design-time variables are available immediately after process starts. Run-time variables are added when an operator runs.

The operator name in the local dataset at run time labels the operator datasets, so you can use dot (.) or bracket ([*expression*]) notation to specify an operator variable in an expression:

```
Operator.field_name

Process["Operator_name"].field_name

Process["Operator_name"]["field_name"]
```

The pre-execution and post-execution code for an operator can use the OpName keyword to access the name of the current operator. You can use this keyword to access or create an operator dataset. To specify an operator dataset variable in the pre-execution or post-execution code, use the following syntax:

```
Process[OpName].field_name
```

The following table lists common operator dataset system variables that CA Process Automation defines automatically. Additional variables can be defined for specific operators.

Operator Dataset Variable	Runtime Scope	Description
AgentName	during, after	Name of the machine associated with the touchpoint that runs the operator.
AgentID	during, after	For internal use only.
EndDate	after	The date when the operator finished running, in the format: `MM/DD/YYYY`
EndTime	after	The time when the operator stopped running, in the format: `HH:MM:SS`
Reason	after	A string that describes the result.
ResponseCode	after	A string that describes the result.

Operator Dataset Variable	Runtime Scope	Description
StartDate	during, after	The date when the operator started running, in the format: `MM/DD/YYYY`
StartTime	during, after	The time when the operator started running, in the format: `HH:MM:SS`
ServiceType	during, after	The CA Process Automation module that ran the operator.
TargetName	during, after	The name of the target (for example, Orchestrator).
TouchpointName	during, after	The name of the touchpoint that runs the operator.
UUID	during, after	For internal use only.

Specify System Dataset Variables

The system dataset for a process contains variables that return system information. The System keyword is used to access the system dataset. You can use either dot (.) or bracket ([*expression*]) notation to specify a system variable in an expression:

`System.`*`variable_name`*

`System[`*`expression`*`]`

The following syntax uses bracket notation to specify a system variable name with a literal string:

System["*variable_name*"]

The system dataset contains the following variables:

DATE

The current date, in the format MM/DD/YY.

DAY

The day of the month.

FIRSTDAYMONTH

The date of the first day of the current month in the format MM/DD/YY.

FIRSTDAYNEXTMONTH

The date of the first day of the next month, in the format MM/DD/YY.

FIRSTDAYPREVMONTH

The date of the first day of the previous month, in the format MM/DD/YY.

HOST

The name of the current host.

LASTDAYMONTH

The date of the last day of the current month, in the format MM/DD/YY.

LASTDAYNEXTMONTH

The date of the last day of the next month, in the format MM/DD/YY.

LASTDAYPREVMONTH

The date of the last day of the previous month, in the format MM/DD/YY.

MONTH

The current month, represented as a zero-based number (for example, this variable returns 0 for January).

TIME

The current time of day, in minutes (for example, this variable returns 600 for 10AM).

TIMES

The current time of day, in the format HHMM.

TOMORROW

The date of the day that follows the current date, in the format MM/DD/YY.

WEEK

The week of the month.

YEAR

The current year.

YESTERDAY

The date of the day that precedes the current day, in the format MM/DD/YY.

Statements

Expressions are JavaScript phrases that are evaluated to yield a value. JavaScript statements execute commands or combine one or more expressions to do things or yield values. A JavaScript program is a collection of statements.

This section briefly describes variable declaration and variable assignment, iterations, and loops that are commonly used in CA Process Automation calculations. The following table lists JavaScript statements, some of which are not documented in this section.

JavaScript Statements

Statement	Syntax	Description
break	break; break label_name:	Exit from a switch or iterative statement; or exit from the statement named by a label statement.
case	case expression:	Labels a statement within a switch statement.
continue	continue; continue label_name:	Restart the loop, or the loop named by a label statement.
default	default;	Label the default statement within a switch statement.
do/while	do statement while (expression)	Perform expressions in a while statement until an expression evaluates False.
empty	;	Do nothing.
for	for (initialize ; test ; increment) statement	Loop while a test is True.
for/in	for (variable_in_object) statement Loop through properties of an object.	See “The for/in Loop Statement."

Statement	Syntax	Description
function	function function_name(a1,a2,...an) {statements}	Declares a function. See “Include Common Resources in CA Process Automation Scripts” .
if/else	if (expression) statement1 else statement2	Execute conditionally. See “The if Statement.”
label	identifier: statement	Assign an identifier to a statement.
return	return[expression];	Return from a function or return a value from a function.
switch	switch (expression) { statements }	Multiway conditional branch to case or default statements. See “The switch Statement.”
throw	throw expression;	Throw an exception.
try	try { statements }	Catch an exception.
var	var name_1[=value1][, ..., name_n [=value_n]];	Declare and optionally initialize variables. See “Variable Declaration.”
while	while (expression) statement	Perform expressions in a while statement while an expression evaluates True. See “The while Loop Statement.”

More information:

Variable Declaration (see page 249)
The if Statement (see page 250)
The switch Statement (see page 252)
The while loop Statement (see page 253)
The for/in Loop Statement (see page 254)
Include Common Resources in CA Process Automation Scripts (see page 256)

Variable Declaration

The JavaScript var statement declares a JavaScript variable. Optionally, you can use the = assignment operator to initialize a variable at the same time that you create it. The JavaScript variable definition uses the following syntax:

```
var variable_name [= initial_value];
```

The following lines create variables but leave the initial values undefined until subsequent code assigns values to them:

```
var x
var s
```

You can initialize a variable as either an integer or a string. In the following example, x is initialized as an integer and s is initialized as an empty string.

```
var x = 0
var s = ""
```

You can create multiple variables in a single statement:

```
var i = 0, j = 0, k = 0
```

Variable Assignment

To assign values to dataset variables, use the = assignment operator. The variable assignment uses the following syntax:

```
[dataset_reference.]variable_name = expression;
```

The expression consists of any combination of functions, variables, values, and operators that returns a string or integer value. For example:

```
Process.S = "ABCDEF" + '_' + "123"
```

```
Datasets[“ThisDataset”].x = 18 * I
```

```
x = 18 * I
```

If you omit the dataset reference, JavaScript references the process dataset by default. If the variable does not exist in the process dataset, JavaScript creates a temporary variable. To create a process dataset variable, use the Process reference.

Reuse Variables

Parent–Child Process Variable Selection: Ability for a child process to query the variables initialized by the parent process

Conditional Statements

The CA Process Automation expressions recognize the JavaScript conditional statements. The if conditional selection evaluates a single Boolean condition while the else if conditional evaluates a series of Boolean conditions. CA Process Automation conditional statements expecting a Boolean value recognize 0 as False and any non-zero integer as True. CA Process Automation expressions also support the switch statement, which allows for multiple outcomes when evaluating a single variable.

The if Statement

The if conditional selection statement uses the following syntax:

if (Boolean_expression)

        statement

The Boolean_expression is any combination of functions, variables, values, and operators that returns a single True or False value. For example:

```
if (i <= 18) {
        y = 18 * I
        z = y * 56
}
```

The second form of the if conditional selection statement allows for two outcomes of the Boolean expression. It uses the following syntax:

```
if (Boolean_expression)
        statement1
else
        statement2
For example:
if (i <= 18)
        Process.Date = System.Date
else
        Process.Date = "2006/01/23"
```

The else if Statement

For multiple outcomes, you can nest if/else statements. However, the logic can become cumbersome to follow with too many nestings. You can therefore use the following construction for a series of if/else statements:

```
if (Boolean_expression_1)
        statement_1
else if (Boolean_expression_2)
        statement_2
else if (Boolean_expression_3)
        statement_3
...
else if (Boolean_expression_n)
        statement_n
else
        statement_else
```

The final else statement is optional. It merely specifies code to be executed if none of the Boolean expressions is True.

The switch Statement

The switch statement performs a multiway branch, useful when all branches of a conditional statement depend on the same variable. In this case, it is cumbersome to check the value of the same variable repeatedly using multiple if statements. The switch statement uses the following syntax to do the same thing more efficiently:

```
switch(variable)
{
        case value_1:
                statements
        break;
        case value_2:
                statements
        break
        ...
        case value_n:
                statements
        break
        default:
                statements
        break
}
```

The switch statement executes the code within the case statement that matches the current value of variable. If there is no match, the switch statement executes the default code or skips to the next statement if there is no default code. The break statements optionally delimit one case block of code from the next case. In the absence of a break statement, execution falls from one case to the next. This is a legal action, so be careful not to omit a break statement unless you actually intend for execution to fall through to the next case statement.

Iterative Statements

JavaScript has several iterative loop statements, a continue statement, and a break statement. The while and do-while loops perform one or more statements as long as some condition is True. The for and for loops perform one or more statements a specified number of times. The break statement exits an iterative statement. The continue statement restarts a loop in a new iteration.

The while loop Statement

The while loop has the following syntax:

```
while (Boolean_expression)
        statement
```

The while loop performs a sequence of statements as long as the Boolean expression tested at the start of the loop returns a True value. For example:

```
var n = 0
while (n < 10)
{
   Process.square[n] = n * n
        n++
}
```

The do/while Loop Statement

The do-while loop has the following syntax:

```
do
        statement
while (Boolean_expression);
```

The do-while loop is similar to the while loop except that it tests at the bottom of the loop rather than at the start of the loop. The while loop performs a sequence of statements as long as the Boolean expression returns a True value. For example:

```
var n = 0
do {
        Process.square[n] = n * n
} while (n++ < 10)
```

The for Loop Statement

The for loop performs a sequence of statements for a specified number of times. The for loop has the following syntax:

```
for (initialize ; test ; increment)
        statement
```

The for loop is similar to the while loop except that an initialization and increment is included in the loop syntax. Each iteration of the for loop increases the increment, performs the test, and performs the statement.

For example, given an indexed variable Process.square containing 35 values, you could use the following lines of code to set every value to the square of its index:

```
for (var i = 0; i < 34; i++)
        Process.square[i] = i * i
```

The for/in Loop Statement

The for/in loop performs a sequence of *statements* for all values of a specified *variable* in a specified *object*. The for/in loop has the following syntax:

```
for (variable in object)
    statement
```

The *variable* value is one of the following items:

- The name of a variable
- A var statement that declares a variable
- An array element
- An object property.

In other words, the *variable* value is equivalent to the left side of an assignment expression. The *object* value is the name of an object or an expression that evaluates to an object.

For example, to loop through elements of an indexed field in a dataset, define an index variable and specify the indexed field as the object.

```
for (var i in Process.square)
    Process.square[i] = i * i
```

The break Statement

The break statement can be used to exit out of a loop, as illustrated in the following lines of code.

```
var l = 0;
while (l < 10) {
        n = n++;
        if (n > 102)
                break;
}
```

The continue Statement

The continue statement can be used to skip to the next iteration of a loop. The following (rather trivial) example illustrates the use of the continue statement to assign even numbers to an indexed local (Process) variable.

```
var i = 0, j = 0

for (j=0; j < 102; j++)

{

        if (j%2) continue

        // following statement executed only for even values of j

        Process.evens[i] = j

        i++

        // following stops the loop when all elements of array are completed

        if (i >= Process.evens.Size) break

}
```

Specify System Paths in CA Process Automation Expressions

Calculations generally accept either UNIX or Microsoft Windows paths. The UNIX path works for locations on both UNIX and Microsoft Windows host systems. For example:

```
Process.Path = "/tmp/files/myfile"
```

The preceding example specifies the location on the current drive for a Microsoft Windows host or the Root for a UNIX host. Specify a network path as follows:

```
Process.NetPath = “//myhost/tmp/files/myfile.txt”
```

Include the drive specification in a path for a Microsoft Windows system as follows:

```
Process.Path = "C:/tmp/files/myfile.txt"
```

If a working directory (such as C:\tmp) is specified for a Microsoft Windows process, specify a path within the working directory without any leading slash character, as follows:

```
Process.Subdir = "files/myfile.txt"
```

If you use a Microsoft Windows path in a calculation, verify that you escape the backslash character so that the interpreter correctly evaluates it as a literal character, as follows:

```
Process.Path = "C:\\tmp\\files\\myfile.fm"
```

Include Common Resources in CA Process Automation Scripts

You can include previously defined scripts in a CA Process Automation script. This allows a script to read in and access saved functions at runtime. Use the include statement on any line of a script dialog to add a previously saved script to the file. The include statement uses the following syntax:

```
include(expression)
```

The expression argument can be any path that references an appropriate resource. Recognized paths include:

- A relative path, such as include("Scripts/functions.js"), specifies a common user resource (c2ouserresources) in the CA Process Automation Repository.

  Note: For more information on adding or managing resources in the CA Process Automation Repository see "Manage Common Resources" in the *Administration Guide.*

- A directory path, such as include("/scripts/functions.js"), specifies a script on the current drive for a Microsoft Windows host or the Root for a UNIX host.
- Including the drive letter in an explicit path, such as include("D:\\scripts\\functions.js"), specifies a script on a specified drive.
- A network path, such as include("//share/scripts/function.js"), specifies a script on a shared network resource.
- A URL, such as include("http://james:8080/itpam/scripts/functions.js"), specifies a path to a web resource.

Lines in an included script are added to a script as if they were typed in place of the include statement. Note that it is a best practice to include only necessary functions or other code instead of lengthy function libraries in an included script. Included scripts are compiled at runtime, so many unused lines of code unnecessarily increases the time required to run a script.

Comments in CA Process Automation Calculations

JavaScript comments are delimited in lines by the character pair //. The start of a comment always signifies the end of a logical line. A comment starts at the end of the logical line and terminates at the end of the physical line. Comments are ignored by the JavaScript language interpreter.

Chapter 7: Forms

CA Process Automation supports two main types of interactive form objects:

- Start Request Forms
- Interaction Request Forms

Design these forms at strategic points in your process to allow users to provide input and control the process.

Custom operators also include forms with pages and data fields that appear in the Properties palette of the Process Designer.

This section contains the following topics:

Start Request Forms (see page 257)
Interaction Request Forms (see page 259)
The Form Designer (see page 260)
Initialize Form Variables (see page 342)

Start Request Forms

The Start Request Form object enables you to create an interface that allows other users to launch a process and to provide input at startup in a structured manner. You design and maintain the layout and behavior of the form. Users fill out the form when prompted. For example, you can give someone at a Help Desk or in Human Resources the ability to provide information that influences how a related process starts.

You can group a series of related form elements on a specific page or in a specific section of the page. You can add any number of pages. You can use functions and events to get or set other field values. Design the form to gather all the information that is required from the user to start the process.

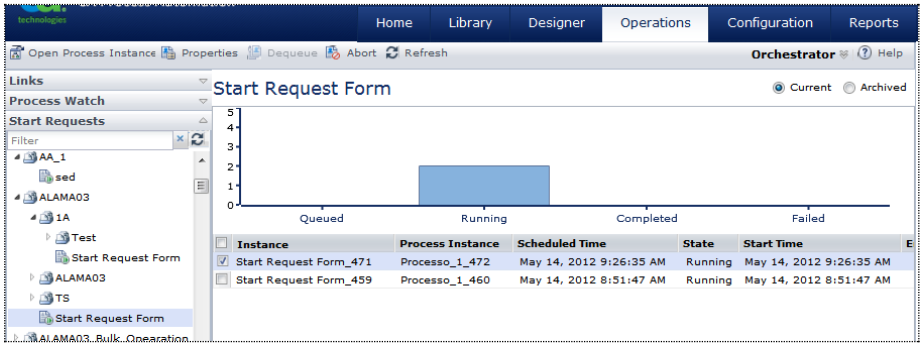
Monitor Start Request Form Instances and Process Instances

After you create and design a Start Request Form, it must start as part of another process, or you can manually start it. When a form starts, it results in a new form instance with a unique name consisting of the form name plus the form's runtime object ID. When a process starts, it results in a new process instance with a unique name consisting of the process name plus the process runtime object ID. Examples follow:

```
MyStartRequestForm_239
MyProcess_241
```

Follow these steps:

1. Click the Operations tab.
2. In the Operations pane, navigate to any of the following locations to view form and process instances:
   a. Expand Start Requests, expand folders, and click a Start Request Form.
   b. Expand Process Watch, expand folders, expand a Process Watch object, and click on an optional Start Request Form that you previously added to the object.
   c. Expand Links and click Start Requests.
3. In the Operations pane, navigate to one of the following locations to start a form and its associated process:
   a. Expand Start Requests, expand folders, and click a Start Request Form. Right-click the form and choose Start.
   b. Expand Process Watch, expand folders, expand a Process Watch object, and click on an optional Start Request Form that you previously added to the object. Right-click the form and choose Start.
4. Repeat step 2 to view both the form instance and the process instance.
5. To monitor the actual process instance, click a row and then click Open Process Instance in the toolbar.



Notes:

- The Process Instance column shows no data for any forms in the *Queued* state. Queued forms do not instantiate a process instance until the form starts *Running*.
- You can change the process instance name while the process is running by using Process.UserInstName. Click Refresh to view the new Process Instance name.
- The Process Instance column does not include any forms that were already running before an upgrade to the current version of CA Process Automation.

Interaction Request Forms

The Interaction Request Form enables you to create an interface that can be used during the execution of a process to interact with a user in a structured manner. The form is accessed in a web browser by a user who either administers processes within CA Process Automation or performs some other business objective.

Typical use cases for Interaction Request Forms include:

- Getting approval before you continue with a process or a path within a process.
- Letting a user select a course of action.
- Retrieving information that is only available from a person at runtime.
- Requesting manual actions (for example, physically connecting a server to a switch) to be performed and marked as completed before proceeding with a process.

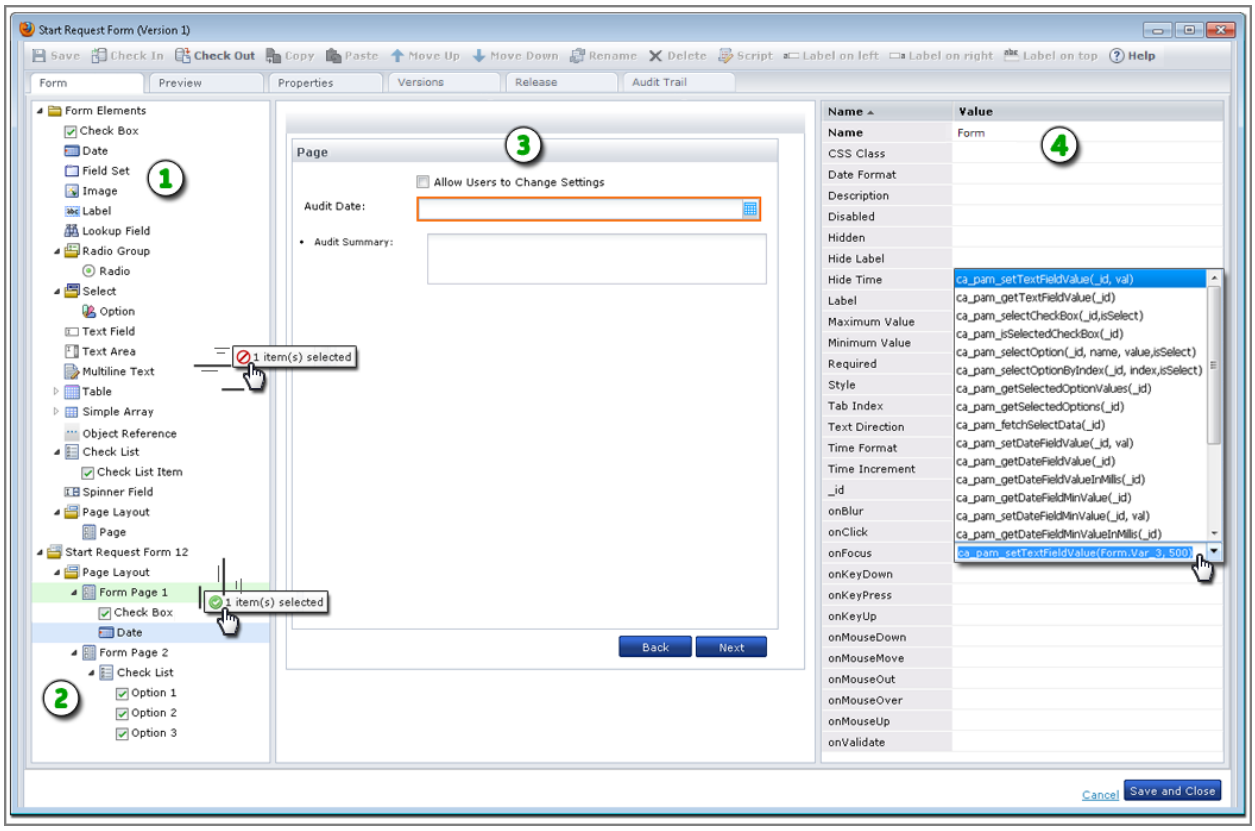
The Interaction Request Form object defines the pages, parameters, and other characteristics of the form. Parameters can be configured to display edit boxes, drop-down lists, list boxes, and check box lists on form pages. You can have multiple pages in the form, letting you group related parameters on separate pages. Users click the Next and Back buttons to step through the pages in an Interaction Request Form. You can add any number of pages to an Interaction Request Form to gather all the information required during the execution of a process that uses the form.

Interaction Request Forms are saved as separate objects in the automation library. After creating and checking in an Interaction Request Form, it can then be added to any process using the Assign User Task operator.

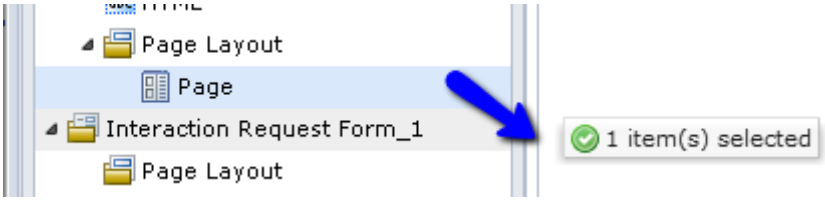
When a process executes the Assign User Task operator, the Interaction Request Form is listed as a pending user prompt in CA Process Automation. The User Interaction operator does not complete until an authorized user responds to the prompt by filling in and submitting the Interaction Request Form, or the timeout specified in the Assign User Task operator is reached.

The Form Designer

When you open a form from the Library Browser, the Form Designer opens. The Form Designer features a standard toolbar and series of tabs. Use the Form tab to design a form and use the Preview tab to review its appearance.

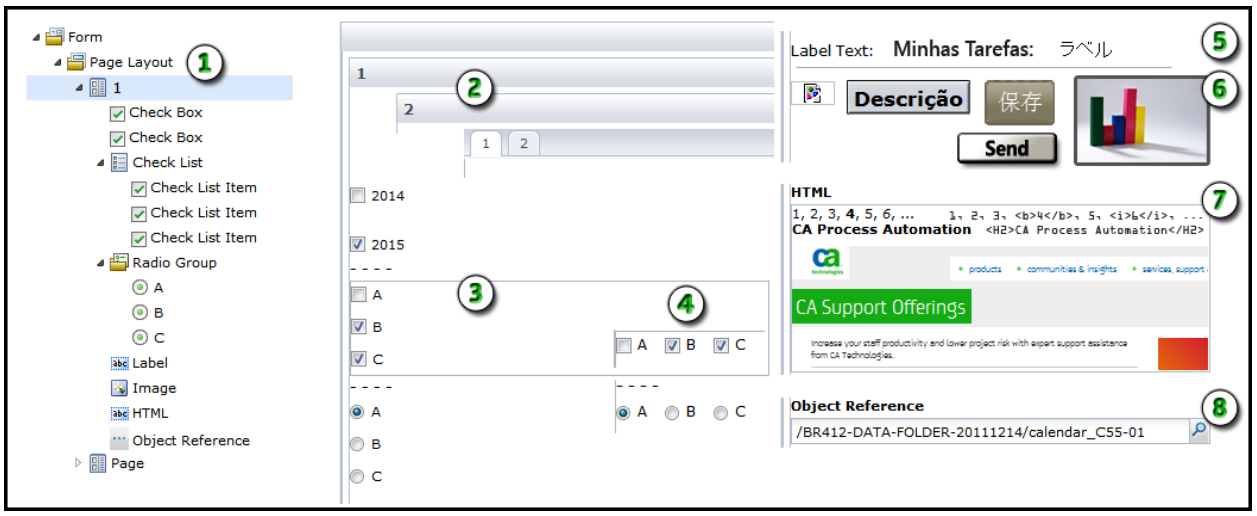


Item:	Description:
1	**Form Elements:** The top of this pane displays all the available types of controls.
2	**Form Structure:** The bottom of this pane shows the structure of your form. Drag and drop form elements to the pages of your form from here.

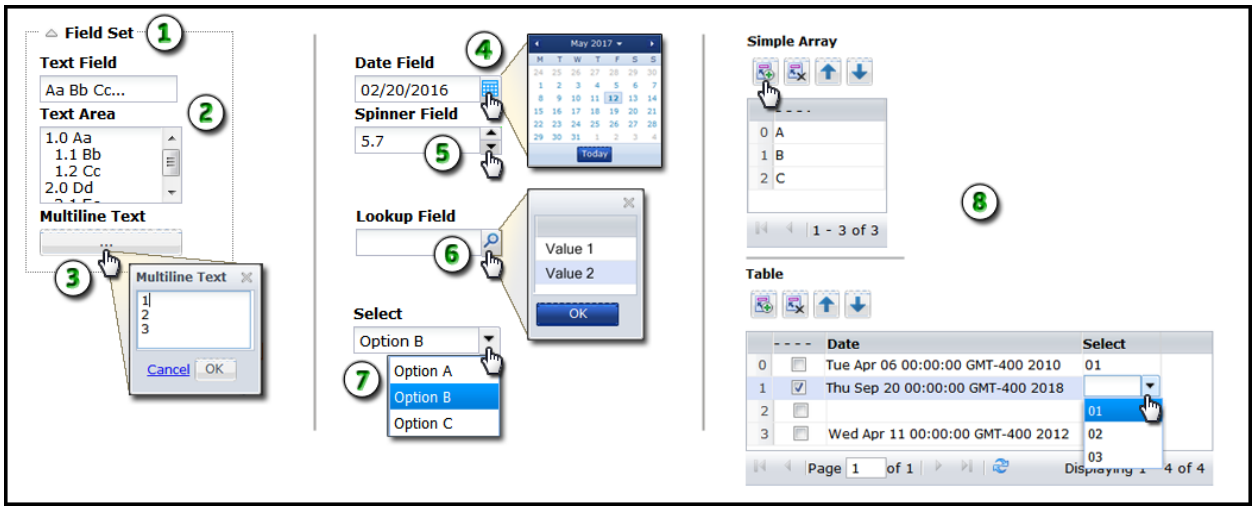
Item:	Description:
3	**Form Pages:** The layout for the pages of your form appears here. Click a control to edit its properties. For forms with multiple pages, click Back and Next to view other pages. Users can also click Next and Back to view the form pages. Drag and drop form elements to the page to design the forms. **Note**: The area that is available to drop a page onto another page is very small. We recommend dragging the page element to the bordering area in between the form elements and the form itself: 
4	**Property Pane:** Use this pane to view or edit the variables in the form elements. For example, set the Required property to true, change the Label that identifies a field, or specify a function for an event. The illustration shows the following arguments used in the onFocus event to set the value of a text field named Form.Var_3 to 500: `ca_pam_setTextFieldValue(Form.Var_3, 500)`

Form Elements

This topic presents basic examples of each form element type.



Item:	Description:
1	**Form Structure**: While you design the form, the bottom of the Form tab displays the arrangement of pages and form elements.
2	**Page Layout**: You can set the layout to display pages as cards or tabs. *Cards* appear in sequence when the user clicks Back or Next. *Tabs* let the user select any tab to view the associated page.
3	**Check Boxes and Radio Buttons**: Use a group of *check boxes* (check list items) when a user can select multiple related items. Use a group of *radio buttons* (option buttons) when a user can select only one related item. Use individual check boxes to control settings for unrelated items.
4	**Orientation**: You can set check list and radio group orientation to vertical or horizontal.
5	**Labels**: Use *labels* to identify specific fields or regions in the form.
6	**Images**: Use an *image* element to include a graphic. An image can display a logo, icon, status indicator, or button that users can click. When the form cannot locate an image, the product displays a broken link icon.
7	**HTML**: Use the *HTML* element to specify HTML code to render for the user.
8	**Object Reference**: Use an *object reference* to give users an easy way to select another object in the library browser. An object reference stores the path to an object in a library. For example, a user can specify a touchpoint on an Orchestrator and then run a process on the selected touchpoint. As the designer of the form, you can limit the available types of objects that an object reference allows.



Item:	Description:
1	**Field Set**: Use *field sets* to group related form elements. Users can expand and collapse field sets to focus on specific parts of a form, thereby avoiding clutter.
2	**Text Field and Text Area**: Use *text fields* to let users enter basic data such as names, addresses, email accounts, telephone numbers, and other details. Use *text areas* to let users enter multiple lines of text. Users can view the amount of text that you set in the height property on the form and scroll the remainder of the field.
3	**Multiline Text**: Use *multiline text* to display large amounts of text in a separate resizeable scrolling window. This element occupies a single line on the form. The form displays the element as a button with ellipsis (...) to indicate that the user can click to browse the full window.
4	**Date Field**: Use a *date field* to store a date. Users can enter a date or select a date from the integrated calendar control.
5	**Spinner**: Use a *spinner* to let the user adjust a value up or down in predefined increments.
6	**Lookup**: Use a *lookup element* to display a one-column table of values in a popup window. You can use a dataset or external data source to provide the values. A user can click a value and then click OK to both populate the lookup field and store the value.
7	**Select**: Use a *select* element to display a drop-down list of options.
8	**Simple Array and Table**: Use a *simple array* to store a single type of data in a table. Use a *table* to store multiple columns of data. These form elements include options for adding, deleting, and moving rows.

You can insert the following elements in a table:

- Check box
- Date
- Field set
- Image
- Label
- Lookup field
- Radio button (option button)
- Select field
- Text field
- Text area
- Multiline text
- Table
- Simple array
- Object reference
- Check list
- Spinner
- HTML
- Page

Form Element Properties

Note: Point to a form element attribute to view the element tool-tip description for assistance in completing the form.

Allow Adding of Rows

A Boolean (true or false) value for tables and simple arrays. When true, a button lets form users create rows in the table. When false, users cannot add rows to the table.

Allow Decimals

A Boolean (true or false) value for spinner fields. When true, users can enter numbers with decimals (such as 12.25 or 0.003). When false, users cannot enter decimal numbers in the field; only whole numbers.

Allow Deletion of Rows

A Boolean (true or false) value for tables and simple arrays. When true, a button lets form users remove rows from the table. When false, users cannot remove rows from the table.

Allow Negative Numbers

A Boolean (true or false) value for spinner fields. When true, users can enter numbers less than zero (such as -10). When false, users cannot enter negative numbers; only 0 or positive numbers.

Allow Reordering of Rows

A Boolean (true or false) value for tables and simple arrays. When true, two buttons let form users move entries up and down in the table. When false, users cannot move rows up or down in the table.

Calendar

A Boolean (true or false) value for object reference fields. When true, the Object Browser lookup dialog includes automation objects of the Calendar type. When false or empty, the Calendar type of automation object does not appear. The object properties of object reference fields filter specific types of automation objects when the user views the available options.

CheckBox Label

The text string or name that is applied to a single check box or a check box item in a group.

Created By

The name of the user or user account who created the form.

Created On

The date and time when the form was created.

Custom Icon

A Boolean (true or false) value for object reference fields. When true, the Object Browser lookup dialog includes automation objects of the Custom Icon type. When false or empty, the Custom Icon type of automation object does not appear. The object properties of object reference fields filter specific types of automation objects when the user views the available options.

Custom Operator

A Boolean (true or false) value for object reference fields. When true, the Object Browser lookup dialog includes automation objects of the custom operator type. When false or empty, the custom operator type of automation object does not appear. The object properties of object reference fields filter specific types of automation objects when the user views the available options.

Dataset

A Boolean (true or false) value for object reference fields. When true, the Object Browser lookup dialog includes automation objects of the Dataset type. When false or empty, the Dataset type of automation object does not appear. The object properties of object reference fields filter specific types of automation objects when the user views the available options.

Dataset Expression

A reference to a specific array variable in a dataset object in the library as in the following example:

```
Datasets["/MyFolder/MyForm/MyDatasetObject"].MyArray
```

Date Format

The preferred format for dates. For example, enter the value MMM dd, yyyy to format the date 05/01/2014 as May 01, 2014. Enter yyyyMMdd to display the same date as 20140501. The default date format is MM/dd/yyyy.

You can also store time in a Date field. For example, you can set the Date Format property to dd/MM/yyyy hh:mm:ss to display the date and time value 05/01/2014 10:17:43 AM.

Description

Information about the form element in addition to its name or short label.

Disabled

An optional setting that you can apply to a form element. The following behaviors characterize elements with the Disabled property set:

- The form element cannot receive focus.
- The form element appears dimmed.
- You cannot change the value of a disabled form element; however, you can copy the value to reuse it.

Disable a form element to apply business logic and prevent invalid data.

Editable

A Boolean (true or false) value for a field. When true, users can edit the field. When false, the data is read-only.

Height

The amount of vertical space the form element occupies. Specify a value (in pixels) for this property from the top to the bottom of the form element.

Hidden

A Boolean (true or false) value for a field.

True

Makes the field invisible when the form is rendered.

Hide Label

A Boolean (true or false) value for a field or table with a label. When true, the label is hidden. When false or empty, the label is visible.

_id

A unique read-only identifier for a specific instance of a form element. The _id appends the following items to the name of the form with a period (.) separator:

- The name of any parent object such as a table
- The name of the form element

For example, Form1 has two option button groups with the following _id properties:

```
Form1.rgName1
Form1.rgName2
```

A specific option button could have the following _id:

```
Form1.rgName2.RadioOptionA
```

A form element in a table named Table_2 could have the following _id:

```
Form_1.Table_2.Var_3
```

The _id for a form is the same as its Name property value.

Note: The _id is used in JavaScript functions. When you rename or move an element, be aware that you are also changing its _id.

Increment

For spinner fields, the value of an incremental adjustment, up or down. For example, if the field displays 6.55 and the increment is .02, one click up results in 6.57 and one click down results in 6.53.

Interaction Request Form

A Boolean (true or false) value for object reference fields. When true, the Object Browser lookup dialog includes automation objects of the Interaction Request Form type. When false or empty, automation objects of the Interaction Request Form type do not appear. The object properties of object reference fields filter specific types of automation objects when the user views the available options.

Invisible

Makes a field visible or invisible but read-only within the custom operator forms only.

True

Makes the field invisible within custom operators only.

Note: This attribute is valid only in custom operators. The Hidden attribute only hides a field, where as the Invisible attribute hides the field and also sets the field to read-only.

Within custom operators, the Invisible attribute created in CA Process Automation Release 3.x and the Hidden attribute created in CA Process Automation Release 4.1 is mapped to this attribute in CA Process Automation Release 4.2.

Label

The short descriptive name that is displayed to the user for a form element.

Label Align

A read-only property that indicates how field labels appear relative to the form elements that they describe. In the Form Designer, you can click toolbar buttons to align labels to the left, right, or the above elements of the selected form.

Label Width

The size (in pixels) of a line of label text. The long labels wrap to the next line.

Layout

A design-only property that determines how multiple pages on a form appear on the Form and Preview tabs.

Card

View one page at a time with Back and Next buttons to navigate pages in a sequence. This setting is how the pages of every form appear to form users. Card is the default Layout setting.

Tab

View the names of available pages in their own tabs. For convenience, designers can click a tab to navigate to the associated page in any order.

Maximum Length

The highest number of characters a user can enter in the field. For example, to require an 8-digit number in a specific field, set the Maximum Length and Minimum Length properties for the field to 8.

Maximum Rows

The highest number of entries that are allowed in a table.

Minimum Length

The lowest number of characters a user can enter in the field. For example, to require an 8-digit number in a specific field, set the Maximum Length and Minimum Length properties for the field to 8.

Minimum Rows

The lowest number of entries that are allowed in a table.

Modified By

The name of the user or user account who modified the form.

Modified On

The date on which the form object was last changed.

Name

A unique string that identifies a form element. The system assigns an initial name (such as Var_3) that you can modify value. Changing the name value also changes the _id value.

Form elements can have two separate names: A Name property and an internal name that identifies the form element in the Form Designer.

- Change the Name value to set the value of the _id variable that identifies the form element.
- Click Rename to change the internal name of an element that appears in the hierarchical form structure layout at design time. This internal name is the default value that is given to the Label of a form element. This label does appear to form users at run time.

Number Format

A string that defines the format for numeric input in the field. For example, enter $#.## to display $3.14.

Orientation

Specifies whether to arrange option groups and check boxes horizontally or vertically on the form. The default is vertical orientation.

Page Size

The number of rows to display on each page of a table or simple array.

Password

A Boolean (true or false) value for a text field, often used with Password fields. When true, user input is displayed as bullet characters to hide input from other users. When false or empty, user input is displayed exactly as typed.

Pattern

An input constraint or validation requirement for the values a user enters in a text field or text area. For example, set Pattern to [a-z] to require only lowercase alphabetic characters in the associated field.

Pattern Message

The on-screen alert or hint to display when a user entry does not match the pattern that is defined in the Pattern property.

Process

A Boolean (true or false) value for object reference fields. When true, the Object Browser lookup dialog includes automation objects of the Process type. When false or empty, automation objects of the Process type do not appear. The object properties of object reference fields filter specific types of automation objects when the user views the available options.

Process Watch

A Boolean (true or false) value for object reference fields. When true, the Object Browser lookup dialog includes automation objects of the Process Watch type. When false or empty, automation objects of the Process Watch type do not appear. The object properties of object reference fields filter specific types of automation objects when the user views the available options.

Render as HTML

A Boolean (true or false) value for multiline text fields. When true, the form interprets the content of the field as HTML code and the product displays the field much as a web browser would. For example, text that you tag with <H2> displays as a second-level heading and text that you tag with <b> displays as bold text.

Required

A Boolean (true or false) value that indicates whether the form element must contain a value or can remain empty.

Resource

A Boolean (true or false) value for object reference fields. When true, the Object Browser lookup dialog includes automation objects of the Resource type. When false or empty, automation objects of the Resource type do not appear. The object properties of object reference fields filter specific types of automation objects when the user views the available options.

Schedule

A Boolean (true or false) value for object reference fields. When true, the Object Browser lookup dialog includes automation objects of the Schedule type. When false or empty, automation objects of the Schedule type do not appear. The object properties of object reference fields filter specific types of automation objects when the user views the available options.

Start Request Form

A Boolean (true or false) value for object reference fields. When true, the Object Browser lookup dialog includes automation objects of the Start Request Form type. When false or empty, automation objects of the Start Request Form type do not appear. The object properties of object reference fields filter specific types of automation objects when the user views the available options.

Style

One or more statements (as in the following examples) that determine how text attributes in a field display to users. Format style attribute assignments in mixed case.

```
color:blue;
textAlign:center;
textDecoration:underline;
textTransform:uppercase
textIndent:30px;
fontStyle:italic;
fontFamily:"Courier";
fontSize:14px;
```

Tab Index

A number that controls the tab order of the form at design time and run time. The application does not enforce uniqueness. For example, you can use multiples of 5 or 10 as you set the tab order. Therefore, if you insert a new field in the layout between field 20 and field 25, you can assign its tab index to 22.

You can also leave the Tab Index value empty and click Move Up or Move Down in the toolbar to adjust the tab order. By default, the form tab order follows the layout from top to bottom.

Text Align

Specifies how an image file appears in the portion of the form layout that it occupies. Images can be aligned left or right, centered, or justified.

Text Direction

Specifies how characters appear in a field relative to the left and right borders of the field. Select **ltr** or leave the property empty to display text from left to right. ltr is the default value and the default text direction for ISO-8859 Latin I codesets. Select **rtl** to display text from right to left.

Touchpoint

A Boolean (true or false) value for object reference fields. When true, the Object Browser lookup dialog includes automation object types that are set to true. The user can browse touchpoints by domain, environment, and Orchestrator. When false or empty, only objects from the default touchpoint appear and the user cannot select another touchpoint.

Touchpoint Group

A Boolean (true or false) value for object reference fields. When true, the Object Browser lookup dialog includes automation object types that are set to true. The user can browse touchpoint groups by domain, environment, and Orchestrator. When false or empty, only objects from the default touchpoint group appear and the user cannot select another touchpoint group.

URL

For Image form elements, the URL path to an image file as in the following examples:

`https://www.<company_name>.com/images/logo1.png`

or

`http://my.intranet.site/Corporate%20Images/Big2.jpg`

Use images to display data, to indicate status, or to provide buttons that users can click.

Value

The stored data for a form element. All elements on a form store their values directly in their visual borders temporarily before committing to the database. Value appears as a property for two specific form elements:

- Each option for a Select field has its own value property. Use the property to store a string or number for each option. For example, although the user sees the Easy, Moderate, and Difficult options, the Value properties for these options could be 0.8, 1.0, and 1.2. These values could be used in additional calculations.
- An HTML form element displays a value property. Use the property to specify the actual text and code to store in the field. For example, enter the following data in the Value property for an HTML form element:

  ```
  1, 2, 3, <b>4</b>, 5, <i>6</i>, ... <H2>CA Process Automation</H2>
  ```

  The form user sees the following rendered data in the same field:

  1, 2, 3, **4**, 5, *6*, ...

  CA Process Automation

Width

The amount of horizontal space (in pixels) for the form element to occupy. Specify a value for this property from the left edge of the form element.

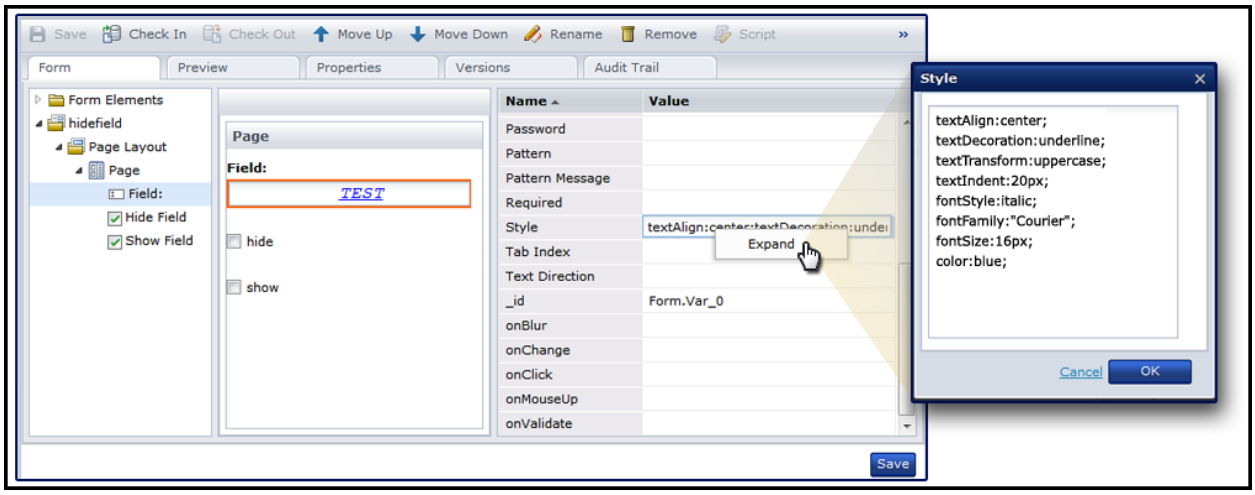
View Form Property Values in a Separate Window

When form properties values are not fully visible in the pane displaying the Name and Value settings, you can view them in a separate window.

Follow these steps:

1. Click the Library tab.
2. Open a form.
3. In the Form Designer:
   a. Drag a field to a page in the layout.
   b. Click the field listed in the form structure.
   c. Identify a property or event that can store multiple lines of text, such as Style or onMouseOver.
   d. Right-click the property or event value, then click Expand.

      The expanded property value appears in a separate window.



Note: When you expand an event, you can press Ctrl+Space to view code completion assistance suggestions. Press Ctrl+Alt to view a list of the functions.

Form Element Events

onBlur

Occurs when a form element loses focus. For example, a form contains a User Name field. A user establishes focus in the field by tabbing to it or clicking it. The onBlur event occurs when the user takes either of the following actions:

- The user clicks another object or clicks another window.
- The user presses the Tab key to navigate to the next field (for example, the Password field).

onChange

Occurs when a form element loses focus and the new value of a form element is different from its old value. For example, a form contains a field named Quantity with a value of 10. After the user changes the value to 15, the user does not move focus to another field. The onChange event occurs only after the user takes either of the following actions:

- The user clicks another object or clicks another window.
- The user presses the Tab key to navigate to the next field.

onClick

Occurs when a user clicks a form element. A valid click includes both the onMouseDown and onMouseUp events on the same object. This requirement helps prevent calling functions or other code accidentally because the mouse must remain on the clickable object.

onFocus

Occurs when a form element receives focus. To establish focus, a user tabs to or clicks a form element. You can also write scripts or code to establish focus in a form element.

onKeyDown

Occurs when a user first presses a key down (for example, when a user tabs to or clicks a Name field). A script that is associated with the onKeyDown event for the Name field alerts users when they attempt to type number keys.

onKeyPress

Occurs after a user presses a key down and continues to hold the key down. For example, the onKeyPress event occurs after a user tabs to or clicks a Select field called Name and presses some key representing an alphabet. A script that is associated with the onKeyPress event for the Name field cycles through names that match the pressed alphabetic key.

onKeyUp

Occurs when a user releases a key after pressing it down. For example, the onKeyUp event occurs when a user tabs to or clicks a Spinner field named Temperature. A script that is associated with the event increases or decreases the field value each time the user presses and releases a specific key.

onLoad

Occurs when the form first opens for the user to complete.

onLookup

Occurs when a user clicks Browse in a Lookup field to view values that are calculated (or "looked up" as directed by the script) based on other field values.

onMouseDown

Occurs when a user presses the left mouse button down on a form element.

onMouseMove

Occurs when a user moves the mouse pointer inside the boundary of a form element.

onMouseOut

Occurs when a user moves the mouse pointer outside the boundary of a form element.

onMouseOver

Occurs when a user moves the mouse pointer over a form element and the user stops moving the mouse.

onMouseUp

Occurs when a user releases the left mouse button after pressing the mouse button down on a form element.

onMouseWheel

Occurs when a user with a wheel-equipped mouse rolls the wheel forward or backwards to scroll a form element.

onSubmit

Occurs when the user submits the form. Any of the following actions can submit the form:

- The user clicks a Submit button.
- The user presses a specific key or combination of keys.
- The value of a field changes.
- The user reaches the end of the form in the tab order.

onValidate

Occurs when the associated code verifies a field value against business rules before one of the following actions occurs:

- The field value is stored in the data buffer.
- The field value is written to the database.

For example, the user tabs to or clicks a Serial Number field that must start with the letters SN and contain 10 numeric digits. Before the user can tab to the next field or click away from the field, the onValidate event and its associated code verify the data. You can alert the user if the serial number does not meet validation rules so they can adjust their entry.

You can use onValidate for custom validation of the field input. For example, to ensure that a field input is at least three characters in length, you can write a custom function in the Script dialog:

```
validateValue:function(_val) {
    if(_val.length < 3) {
    return "Please enter more than 3 characters for this field";
    } else {
    return null;
    }
```

In the Form Designer, include the onValidate attribute value for the text field on which to run the validation. For example:

```
ca_fd.js.validateValue(_val)
```

The custom function replaces the required parameter _val with the correct field value when the script runs.

If the validation script returns a null value, the field input passes validation. Otherwise, the field input fails validation and the script returns an error (for example, "Enter more than 3 characters for this field").

Form Element Functions

Many user interface events occur when a user interacts with a form. Events such as changing a field value (onChange) or positioning the mouse pointer over a table (onMouseOver) can initiate JavaScript functions. As a designer of a form, you can use form element functions to achieve the following and other goals:

- Dynamically control form elements based on events.
- Get data from a field or table.
- Set the data in a field or table.
- Use logic to assist users with navigating potentially complex options. For example, you can use functions to update a list of cities according to the Country a user selects.
- Assist users with minimizing data-entry errors.

This section lists predefined JavaScript functions that you can use in fields on start request and interaction request forms.

Form Elements

Form elements specify the information on a form that is to be submitted to a web site or service. An element can be of type text field, check box, password, radio button, submit button, and so on.

The following form elements are available in the Forms Designer.

General Functions for All Form Elements

These functions apply to all element types.

ca_pam_disableField(_id, isDisable)

Enables or disables a specified field. When a field is enabled, users can provide input or edit the values. Disabled fields are still visible but do not accept user input.

Input Parameters

***_id* (string)**

Specifies the unique identifier of the field to enable or disable. You can find this under the element properties. See Form Element Properties (see page 264) for more information.

***isDisable* (boolean)**

Specifies whether the field is to be disabled (true) or enabled (false).

Return Value

None.

Example

This example disables the specified checklist field, including all checklist items in it.

```
ca_pam_disableField('Form.ckListField27', true)
```

ca_pam_hideField(_id, isHide)

Shows or hides a specified field. Hidden fields are not visible to the user. The tab order of the form skips hidden fields. Although the hidden field can still hold a value, it is effectively removed from the form.

Input Parameters

***_id* (string)**

Specifies the unique identifier of the field to display or hide.

***hide* (boolean)**

Specifies whether the field is hidden (true) or displayed (false).

Return Value

None.

Example

Consider that a user fills an online application form with birth place details. If the cities listed under 'Birth City' field are only capital cities, then the user can select "Other City". Upon clicking "Other City", the Hidden attribute is set to false and a text field is visible. The user can now enter the city name.

This example hides the specified field.

```
ca_pam_hideField('Form.Var_0', true)
```

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folder, 02 Show and Hide Form Element: Show and Hide Form Element.

XML Parsing

ca_pam_showDataInTable(result, _id, tableHeader)

Displays data rows that result from a Lookup field function as a single-column table that has the specified header or title.

This function only displays the "result" passed to it. The result here represents an array of rows. The result can be either stored in a variable and passed to this function, or directly replaced with the function call that returns the result.

Input Parameters

***result* (object)**

Specifies a function call that results in the values displaying in a table.

***_id* (string)**

Specifies the unique identifier of the Lookup Field table in which to show the result set.

***tableHeader* (string)**

Specifies a text string to display at the top of the Lookup Field table on the form.

Return Value

No value is returned; this function only displays the table with the values included in it.

Example

```
ca_pam_showDataInTable(result,'Form.office','Office');
```

This example shows a table named Office that lists the results of a separate query to get office names. In the separate query, the user Region selection determines the offices to display.

The onLookup event for a Lookup Field named Office calls this function.

ca_pam_convertToSimpleArray (objectArray, fieldName)

Creates a simple array of strings from any of more complex JavaScript array of objects.

Input Parameters

objectArray (object)

Specifies a reference to an array of JavaScript objects containing multiple properties.

fieldName (string)

Defines the name of the property in objectArray from which to create the simple array.

Return Value

The array that the method creates is returned.

Example

```
        var location1 = new Object();
      location1.buildingCode = 10;
    location1.floorCode = 20;
       var location2 = new Object();
      location2.buildingCode = 100;
    location2.floorCode = 200;
var LocationArray = {location1,location2};

var  floorCodeArray = ca_pam_convertToSimpleArray (LocationArray,
'floorCode');
```

The output of floorCodeArray {20,200}.

ca_pam_convertToJavaScriptObject(valueObject)

Converts a process automation data type to a standard JavaScript object according to the input data type. If you provide a string as an input value, the method returns a string value.

Input Parameters

valueObject

Specifies the process automation data types. For example, Boolean, Date, and ValueMap.

Return Value

Returns a JavaScript Object based on the value of valueObject. If ValueMap is the input value, it returns a JavaScript object like a map. If Value array is the input parameter, the method returns a Javascript array.

Note: For the Date data type, the long value is returned, which is the canonical representation of the date. To convert it into a date object, use the following syntax:

```
new Date (ca_pam_convertToJavaScriptObject(valueObject))
```

Example

This example declares a variable (array) and sets it to the converted JavaScript object.

```
var array = ca_pam_convertToJavaScriptObject(valueObject);
```

Check Box

ca_pam_isSelectedCheckBox(_id)

Determines whether a check box is selected (true) or a check box is cleared (false).

Input Parameters

***_id* (string)**

Specifies the unique identifier of the Check Box field to evaluate.

Return Value

***Value* (boolean)**

True if the check box is selected, false otherwise.

Example

```
ca_pam_isSelectedCheckBox('Form.ckbxInsBuy')
```

The ckbxInsBuy check box label is Purchase Insurance. This example code returns "true" when the Purchase Insurance check box is selected. The code returns "false" when the check box is cleared.

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folder, 02 Show and Hide Form Element: Show and Hide Form Element.

ca_pam_selectCheckBox(_id, isSelect)

Selects or clears a specified check box.

Input Parameters

***_id* (string)**

Specifies the unique identifier of the Check Box field to select or clear.

***isSelect* (boolean)**

Specifies whether a field is selected (true) or a field is cleared (false).

Return Value

None.

Example

```
ca_pam_selectCheckBox('Form.ckbxInsBuy', true)
```

This example marks the *ckbxInsBuy* check box on the form.

- The *ckbxInsBuy* check box label is Purchase Insurance.
- The ca_pam_selectCheckBox function is useful if the user selects other options on the form that require the purchase of insurance.

Date

Except as noted in this section, all date functions return the date as a string in the format that you specify in the Date Format property.

ca_pam_getDateFieldMaxValue(_id)

Returns the value of the Date field Maximum Value property as a string.

Input Parameters

***_id* (string)**

The unique identifier of a Date field.

Return Value

The maximum value for a Date field as a string.

Example

This example returns the maximum value that is allowed for the *DateDeparture* field as a date string (for example, 05/20/2025).

```
ca_pam_getDateFieldMaxValue('Form.DateDeparture')
```

ca_pam_getDateFieldMaxValueInMillis(_id)

Returns the value of the Date field Maximum Value property, expressed as a long integer. This integer represents the number of milliseconds before or after January 1, 1970 00:00:00 UTC (known as the *UNIX Epoch*). You can pass the value in this field, as mentioned in the ca_pam_getDateFieldMaxValue(_id) (see page 284) method.

Input Parameters

***_id* (string)**

Defines the unique identifier of a Date field.

Return Value

The maximum value for a Date field, expressed as a long integer.

Example

```
ca_pam_getDateFieldMaxValueInMillis('Form.DateDeparture')
```

This example returns the maximum value that is allowed for the DateDeparture field as a numeric string measured in milliseconds. For a Date field with a maximum value of 05/25/2025, this function would return 1748188800000 (that is, 1,748,188,800,000 or 1.7 trillion milliseconds).

The ca_pam_getDateFieldMaxValueInMillis function returns a negative result for dates before 1970.

ca_pam_getDateFieldMinValue(_id)

Returns the value of the Date field Minimum Value property as a string. You can pass the value in this field, as mentioned in the ca_pam_getDateFieldMaxValue(_id) (see page 284) method.

Input Parameters

***_id* (string)**

Specifies the unique identifier of a Date field.

Return Value

The minimum value for a Date field as a string.

Example

This example returns the minimum value that is allowed for the *DateDeparture* field as a date string (for example, 05/05/2025).

```
ca_pam_getDateFieldMinValue('Form.DateDeparture')
```

ca_pam_getDateFieldMinValueInMillis(_id)

Returns the value of the Date field Minimum Value property, expressed as a long integer. This integer represents the number of milliseconds before or after January 1, 1970 00:00:00 UTC (known as the *UNIX Epoch*). You can pass the value in this field, as mentioned in the ca_pam_getDateFieldMaxValue(_id) (see page 284) method.

Input Parameters

***_id* (string)**

Specifies the unique identifier of a Date field.

Return Value

The minimum value for a Date field, expressed as a long integer.

Example

```
ca_pam_getDateFieldMinValueInMillis('Form.DateofBirth')
```

This example returns the minimum value that is allowed for the DateofBirth field as a numeric string measured in milliseconds. For a Date field with a minimum value of 01/01/1974, this function would return 126291600000 (that is, 126,291,600,000 or 126 billion milliseconds).

This function returns a negative result for dates before 1970.

ca_pam_getDateFieldValue(_id)

Returns the value of a Date field as a date string (for example, 05/05/2025).

Input Parameters

***_id* (string)**

Specifies the unique identifier of a Date field.

Return Value

Date value as a string.

Example

```
var LastDay =ca_pam_getDateFieldValue('Form.TripEndDate');
ca_pam_setDateFieldMaxValue('Form.DateDeparture',LastDay);
```

The first line of this example gets the TripEndDate (for example, 05/15/2014) and stores it in the LastDay variable.

The second line of this example sets the Maximum Value of the DateDeparture field to the value of the LastDay variable.

ca_pam_getDateFieldValueInMillis(_id)

Returns the value of a Date field as a long integer (for example, 61238000). This integer represents the number of milliseconds before or after January 1, 1970 00:00:00 UTC (known as the *UNIX Epoch*).

Input Parameters

***_id* (string)**

Defines the unique identifier of a Date field.

Return Value

Date value for a Date field that is expressed as a long integer.

Example

```
ca_pam_getDateFieldValueInMillis('Form.DateofBirth')
```

This example returns the value for the DateofBirth field as a numeric string measured in milliseconds. For a Date field with a value of 08/22/1965, this function would return -132307200000 (that is, -132,307,200,000 or -132 billion milliseconds).

The ca_pam_getDateFieldValueInMillis function returns a positive result for dates on or after January 1, 1970.

ca_pam_setDateFieldMaxValue(_id, val)

Sets the value of the Date field Maximum Value property. Form users cannot enter a date in the Date field that is later than the Maximum Value.

Input Parameters

***_id* (string)**

Specifies the unique identifier of a Date field.

val

Specifies the value to use as a maximum, expressed as a date string (for example, 05/20/2014).

Return Value

None.

Example

This example sets the Maximum Value property of the DateDeparture field to the date stored in the LastDay variable.

```
ca_pam_setDateFieldMaxValue('Form.DateDeparture',LastDay);
```

ca_pam_setDateFieldMinValue(_id, val)

Sets the value of the Date field Minimum Value property. Form users cannot enter a date in the Date field that is earlier than the Minimum Value.

Input Parameters

***_id* (string)**

Specifies the unique identifier of a Date field.

val

Specifies the value to use as a minimum, expressed as a date string (for example, 05/05/2012).

Return Value

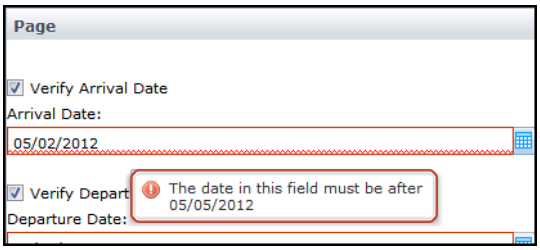
None.

Example

```
ca_pam_setDateFieldMinValue('Form.DateArrival',today);
```

This example sets the Minimum Value property of the DateArrival field to the date stored in the today variable.

- The product alerts form users if they enter a date earlier than the minimum value.



- Pop-up calendars on the form display invalid dates in gray and do not let users click invalid dates.

ca_pam_setDateFieldValue(_id, val)

Sets the value of a Date field to a date string (for example, 05/15/2014). The value is set only if it is specified in the correct format (that is, in the format specified in the Date Format property of the Date field). If the value of this property is blank, then the value is set in the Date Format specified in the User settings.

Note: Pass the date in the correct format; otherwise it is not set.

Input Parameters

_id

Defines the unique identifier of a Date field.

val

Defines a date value that you specify (for example, 05/05/2014).

Return Value

None.

Example

```
ca_pam_setDateFieldValue('Form.DateArrival','05/05/2014');
```

This example sets the DateArrival field to 05/05/2014.

Lookup Field

ca_pam_getTextFieldValue(_id)

Returns a value from a Text field.

Input Parameters

***_id* (string)**

Specifies the unique identifier of a Text field.

Return Value

Value from the Text field (string).

Example

This example returns the value of the top_vendors Text field.

```
ca_pam_getTextFieldValue('Form.top_vendors')
```

This example returns the value of the top_vendors Text field and sets the dynamic_field Text field to the returned value.

```
ca_pam_setTextFieldValue('Form.dynamic_field',
ca_pam_getTextFieldValue('Form.top_vendors'))
```

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folders, then:
   - 07 Populate Table RESTful WS: Populate Table RESTful WS.
   - 08 Populate Table SOAP WS: Populate Table SOAP WS.

ca_pam_setTextFieldValue(_id, val)

Sets the value of a Text field.

Input Parameters

***_id* (string)**

Specifies the unique identifier of the field for which to set a specific value.

***val* (string)**

Defines the new value for the Text field.

Return Value

None.

Example

This example sets the top_vendors Text field to "I vote for vendor 3 because:"

```
ca_pam_setTextFieldValue('Form.top_vendors', 'I vote for vendor 3 because: ')
```

To continue this example, use ca_pam_getTextFieldValue('Form.top_vendors') after users enter their reasons to submit their full statements.

```
ca_pam_setTextFieldValue('Form.dynamic_field', ca_pam_getTextFieldValue('Form.top_vendors'))
```

This example returns the value of the top_vendors field and sets the dynamic_field to that value.

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folder, then 08 Populate Table SOAP WS: Populate Table SOAP WS.

Radio Group

ca_pam_isSelectRadio (_id, radio_id)

Determines whether a specific radio option is selected.

Input Parameters

***_id* (string)**

Specifies the unique identifier of a Radio Group field.

***radio_id* (string)**

Specifies the unique identifier of a specific radio option button in a group.

Return Value

***Value* (boolean)**

True if the check button is selected, false otherwise.

Example

```
ca_pam_isSelectRadio('Form.Transportation',
'Form.Transportation.Option_01_Air')
```

Transportation is a group with two option buttons with the following IDs:

- Option_01_Air
- Option_02_Rail

This example returns "true" if the *Air* option button in the *Transportation* group is selected or "false" if any other option is selected.

ca_pam_selectRadio(_id, radio_id)

Selects a specific radio option button in a Radio Group field.

Input Parameters

***_id* (string)**

Specifies the unique identifier of a Radio Group field.

***radio_id* (string)**

Specifies the unique identifier of a radio option button in a Radio Group.

Return Value

None.

Example

```
ca_pam_selectRadio('Form.Transportation',
'Form.Transportation.Option_01_Air')
```

This example selects the *Air* option button in the *Transportation* radio group.

Transportation is a radio group with two option buttons with the following IDs:

- Option_01_Air
- Option_02_Rail

Select
ca_pam_getSelectedOptions(_id)

Returns the index value of the currently selected option in a Select field.

Input Parameters

***_id* (string)**

Specifies the unique identifier of the Select field for which to retrieve the option value.

Return Value

This function returns an array of selected options index.

In the case of a single Select drop-down list, the selected option index can be retrieved by accessing the first element of the array.

Example

```
ca_pam_getSelectedOptions('Form.VarFillerType')
```

This example retrieves the index value of the Filler Type that the user chose.

- *Filler Type* is a Select field with options for Rocks [0], Sand [1], and Water [2].
- The ca_pam_getSelectedOptions function returns the index value 1 when the user selects Sand from the Filler Type drop-down list.

ca_pam_getSelectedOptionValues(_id)

Returns the value of the currently selected option in a Select field.

Input Parameters

***_id* (string)**

Specifies the unique identifier of the field for which to retrieve the option value.

Return Value

This function returns an array of selected option value.

In the case of a single Select drop-down list, the selected option value can be retrieved by accessing the first element of the array.

Example

```
alert(ca_pam_getSelectedOptionValues('Form.VarFillerType')[0])
```

This example returns the option value number or text string for the Filler Type that the user chose.

- The alert command displays the option value result to the user in a pop-up dialog or message.
- *Filler Type* is a Select field with the following options, index values shown in brackets, and values:

  Rocks [0], $50

  Sand [1], $100

  Water [2], Call for details.
- The ca_pam_getSelectedOptionValues function returns the value of the option that is selected from the Filler Type drop-down list when the user clicks Update Order Form.

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folder, 06 Populate Dropdown DB: Populate Dropdown from DB.

ca_pam_selectOption(_id, name, value, isSelect)

Selects or clears an option in a Select field based on the option name and value. You can also use this function to prevent selection of a specific option.

This function can both select and unselect an option. If an option is not selected and you want to select the option programmatically, then isSelect is true. If an option is already selected and you want to unselect the option, then use isSelect is false.

Input Parameters

***_id* (string)**

Specifies the unique identifier of the Select field to set.

***name* (string)**

Specifies the name of the specific Select field option.

***value* (string)**

Defines the value of the specific Select field option. This value is the existing value of the option.

***isSelect* (boolean)**

Specifies whether the field is selected (true) or cleared (false).

Return Value

None.

Example

```
ca_pam_selectOption('Form.City', 'Regional Hub', 'Sydney', true);
```

This example sets the City field to the option named Regional Hub.

If multiple options are named Regional Hub, then this function applies the option with the specified *value*. This distinction is important. You can use ca_pam_getSelectedOptionValues(_id) to identify the specific *value* for a field option, even when multiple options have the same name. For example, users can continue to use a set of standard forms to direct business to the Regional Hub even if *value* changes.

ca_pam_selectOptionByIndex(_id, index, isSelect)

Selects or clears an option in a Select field based on the index.

Input Parameters

***_id* (string)**

Specifies the unique identifier of the select field to select or clear.

***index* (integer)**

Specifies the fixed identifier for an option in the field.

isSelect (boolean)

Specifies whether the Select field option is selected (true) or the option is cleared (false).

Return Value

None.

Examples

```
ca_pam_selectOptionByIndex('Form.VarFillerType', 2, true)
```

This example sets the current choice of *Filler Type* to the value at index position 2.

- *Filler Type* is a Select field with options for Rocks [0], Sand [1], and Water [2].
- *Economy Option* is a check box with this function in its onClick event.
- When the user selects the *Economy Option* check box, the ca_pam_selectOptionByIndex function sets the *Filler Type* field to the option at index position 2, *Water*.

Another example:

```
ca_pam_selectOptionByIndex('Form.VarFillerType', 2, false)
```

When *Filler Type* is already set to *Water*, this function clears the Filler Type field.

ca_pam_addValuesInSelectStore(_id, values)

Adds new options to the Select field with a simple set of values that you define in code.

Input Parameters

_id **(string)**

Specifies the unique identifier of a Select field.

values **(Javascript object)**

Specifies an array of objects where each object has two properties ("name" and "value") that are necessary to represent the Select field options. The "name" property for each object is represented as the key for the option added and "value" property would be the value. If you have an array of names and values, then you could also use the ca_pam_createSelectStore function to create the "values" object (as shown in the example).

Return Value

None.

Example

This example uses the user selection of West or North in a separate Region field to update icons available in the City field dynamically.

```
if('West'==regionChoice)
  var cityOptionNames =["New York","Rio De Janeiro","Mexico City"]
    var cityOptionValues =["West_01","West_02","West_03"];
if('North'==regionChoice)
  var cityOptionNames =["Madrid","Moscow","Copenhagen"]
    var cityOptionValues =["North_04","North_05","North_06"];
ca_pam_addValuesInSelectStore('Form1.City',
ca_pam_createSelectStore(cityOptionNames ,cityOptionValues ))
```

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folder. The function can be found in the following forms:
   - 04 Populate Dropdown Dataset: Populate Dropdown Dataset.
   - 06 Populate Dropdown DB: Populate Dropdown from DB.

ca_pam_clearSelectStore(_id)

Clears all options from the Select field.

Input Parameters

***_id* (string)**

Specifies the unique identifier of the Select field to clear.

Return Value

None.

Example

This example clears all of the existing options in the City field on Form1. This can be used to reinitialize the select field and populate fresh options using the ca_pam_addValuesInSelectStore().

```
ca_pam_clearSelectStore(Form1.City)
```

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folder, then 06 Populate Dropdown DB: Populate Dropdown from DB.

ca_pam_createSelectStore(nameArray, valueArray)

Used to create a set of options for a Select field.

Input Parameters

***nameArray* (array)**

Defines an array of options names that are displayed to the user.

***valueArray* (array)**

Optionally defines an array of option values which corresponds to each item in the nameArray. If omitted, nameArray is used to define both names and values.

Return Value

JavaScript object that represents store options and their values.

Example

This example sets the available options for the City field to the specified names and associated values.

```
var cityOptionNames = ["Los Angeles", "New York", "New Carolina"];
var cityOptionValues = ["LA","NY","NC"];
var newStore = ca_pam_createSelectStore(cityOptionNames
,cityOptionValues )
```

Now the newStore variable can be passed to a Select Form element using the following method:

```
ca_pam_addValuesInSelectStore('Form.City',newStore);
```

ca_pam_createSelectStoreFromSQLResult(resultFromSQLQuery, nameColumnID, valueColumnID)

Creates a set of options for a Select field directly from the result of an SQL query.

Input Parameters

***resultFromSQLQuery* (object)**

Defines the data retrieved by a SQL statement.

***nameColumnID* (array)**

Defines the name of the column in the SQL result that is used as names in the Select field.

***valueColumnID* (array)**

Optionally defines the name of the column in the SQL result that is used as values in the Select field. If omitted, the column that is defined by nameColumnID is used for both names and values.

Return Value

The array that the function builds is returned.

Example

This example uses the result of a query of an external data source to set the options for the City field. The query runs when the user selects an option from the Region field.

```
ca_pam_addValuesInSelectStore('Form1.City',ca_pam_createSelectS
toreFromSQLResult(result,'txtRegion'));
```

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folder, then 06 Populate Dropdown DB: Populate Dropdown from DB.

Text Field

ca_pam_getTextFieldValue(_id)

Returns a value from a Text field.

Input Parameters

***_id* (string)**

Specifies the unique identifier of a Text field.

Return Value

Value from the Text field (string).

Example

This example returns the value of the top_vendors Text field.

```
ca_pam_getTextFieldValue('Form.top_vendors')
```

This example returns the value of the top_vendors Text field and sets the dynamic_field Text field to the returned value.

```
ca_pam_setTextFieldValue('Form.dynamic_field',
ca_pam_getTextFieldValue('Form.top_vendors'))
```

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folders, then:
   - 07 Populate Table RESTful WS: Populate Table RESTful WS.
   - 08 Populate Table SOAP WS: Populate Table SOAP WS.

ca_pam_setTextFieldValue(_id, val)

Sets the value of a Text field.

Input Parameters

***_id* (string)**

Specifies the unique identifier of the field for which to set a specific value.

***val* (string)**

Defines the new value for the Text field.

Return Value

None.

Example

This example sets the top_vendors Text field to "I vote for vendor 3 because:"

```
ca_pam_setTextFieldValue('Form.top_vendors', 'I vote for vendor 3 because: ')
```

To continue this example, use ca_pam_getTextFieldValue('Form.top_vendors') after users enter their reasons to submit their full statements.

```
ca_pam_setTextFieldValue('Form.dynamic_field', ca_pam_getTextFieldValue('Form.top_vendors'))
```

This example returns the value of the top_vendors field and sets the dynamic_field to that value.

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folder, then 08 Populate Table SOAP WS: Populate Table SOAP WS.

Text Area

ca_pam_getTextFieldValue(_id)

Returns a value from a Text field.

Input Parameters

***_id* (string)**

Specifies the unique identifier of a Text field.

Return Value

Value from the Text field (string).

Example

This example returns the value of the top_vendors Text field.

```
ca_pam_getTextFieldValue('Form.top_vendors')
```

This example returns the value of the top_vendors Text field and sets the dynamic_field Text field to the returned value.

```
ca_pam_setTextFieldValue('Form.dynamic_field',
ca_pam_getTextFieldValue('Form.top_vendors'))
```

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folders, then:
   - 07 Populate Table RESTful WS: Populate Table RESTful WS.
   - 08 Populate Table SOAP WS: Populate Table SOAP WS.

ca_pam_setTextFieldValue(_id, val)

Sets the value of a Text field.

Input Parameters

***_id* (string)**

Specifies the unique identifier of the field for which to set a specific value.

***val* (string)**

Defines the new value for the Text field.

Return Value

None.

Example

This example sets the top_vendors Text field to "I vote for vendor 3 because:"

```
ca_pam_setTextFieldValue('Form.top_vendors', 'I vote for vendor 3 because: ')
```

To continue this example, use ca_pam_getTextFieldValue('Form.top_vendors') after users enter their reasons to submit their full statements.

```
ca_pam_setTextFieldValue('Form.dynamic_field', ca_pam_getTextFieldValue('Form.top_vendors'))
```

This example returns the value of the top_vendors field and sets the dynamic_field to that value.

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folder, then 08 Populate Table SOAP WS: Populate Table SOAP WS.

Multi-Line Text

ca_pam_getTextFieldValue(_id)

Returns a value from a Text field.

Input Parameters

***_id* (string)**

Specifies the unique identifier of a Text field.

Return Value

Value from the Text field (string).

Example

This example returns the value of the top_vendors Text field.

```
ca_pam_getTextFieldValue('Form.top_vendors')
```

This example returns the value of the top_vendors Text field and sets the dynamic_field Text field to the returned value.

```
ca_pam_setTextFieldValue('Form.dynamic_field',
ca_pam_getTextFieldValue('Form.top_vendors'))
```

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folders, then:
   - 07 Populate Table RESTful WS: Populate Table RESTful WS.
   - 08 Populate Table SOAP WS: Populate Table SOAP WS.

ca_pam_setTextFieldValue(_id, val)

Sets the value of a Text field.

Input Parameters

***_id* (string)**

Specifies the unique identifier of the field for which to set a specific value.

***val* (string)**

Defines the new value for the Text field.

Return Value

None.

Example

This example sets the top_vendors Text field to "I vote for vendor 3 because:"

```
ca_pam_setTextFieldValue('Form.top_vendors', 'I vote for vendor 3 because: ')
```

To continue this example, use ca_pam_getTextFieldValue('Form.top_vendors') after users enter their reasons to submit their full statements.

```
ca_pam_setTextFieldValue('Form.dynamic_field', ca_pam_getTextFieldValue('Form.top_vendors'))
```

This example returns the value of the top_vendors field and sets the dynamic_field to that value.

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folder, then 08 Populate Table SOAP WS: Populate Table SOAP WS.

Table

ca_pam_clearTableData(_id, startIndex, endIndex)

Deletes one or more rows of data from a table.

Input Parameters

_id **(string)**

Specifies the unique identifier of a Table form element.

startIndex **(integer)**

Specifies the numeric index of the first row of table data to delete.

endIndex **(integer)**

Specifies the numeric index of the last row of table data to delete.

Return Value

None.

Example

```
ca_pam_clearTableData('Form.employeeList',0,ca_pam_getTableRowCount('Form.employeeList')-1);
```

This example deletes all rows in the employeeList table.

- The range to delete begins with row 0 and ends with the last row.
- To calculate the last row, the ca_pam_clearTableData function gets the current row count for the table from the ca_pam_getTableRowCount function and subtracts 1.

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folder, then:
   - 07 Populate Table RESTful WS: Populate Table RESTful WS.
   - 08 Populate Table SOAP WS: Populate Table SOAP WS.

ca_pam_getTableData(_id, startIndex, endIndex)

Returns the data of the table from the start index to the end index as provided in the method name. This function returns an array of JavaScript objects where each element in the array represents one row in the table. To retrieve the value for a particular column, you can access the value using column Name property from the JavaScript Object corresponding to the row.

Input Parameters

***_id* (string)**

Specifies the unique identifier of a Table form element.

***startIndex* (integer)**

Specifies the numeric index of the first row of the table from which to return data.

***endIndex* (integer)**

Specifies the numeric index of the last row of the table from which to return data. If you specify an index greater than the last available row, the function fails.

Return Value

Returns an array that is represented as a JavaScript object.

Important! After the method returns the table data, the variable names used to access the columns must be in lowercase, regardless of how you define the Name property for each column field in the Forms Designer.

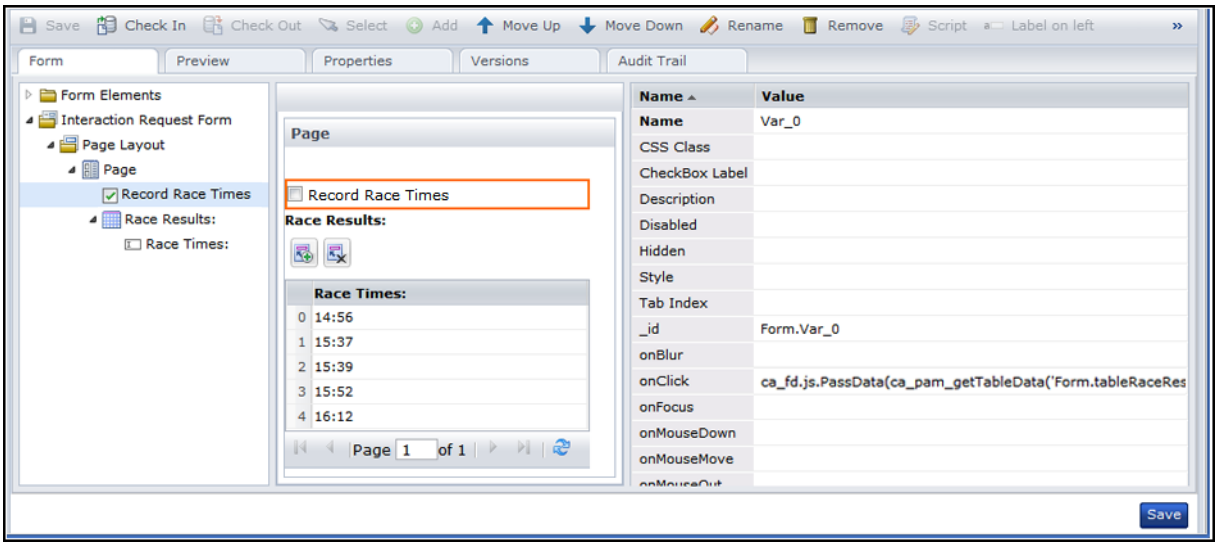
Example

```
ca_fd.js.PassData(ca_pam_getTableData('Form.tableRaceResults',0
,4))
```

This example uses the following form script to pass the top five finishing times in a race from a Table form element to a custom database. The Record Race Times check box onClick event initiates function calls to get the table data, pass the data, and record the results.

```
{
 PassData : function(result) {
   for(i=0;i<result.length;i++)
   {
    var firstRow = result[i];
    alert(firstRow.var_0);
   }
 }
}
```

The following illustration shows the form as it appears at design time.



ca_pam_getTableRowCount(_id)

Returns the total number of rows of data from a table.

Input Parameters

***_id* (string)**

Specifies the unique identifier of a Table Form element.

Return Value

Returns a number of rows as an integer.

Example

```
ca_pam_getTableRowCount('Form.employeeList')
```

This example returns the row count of the employee list Table Form element.

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folders, then:
   - 07 Populate Table RESTful WS: Populate Table RESTful WS.
   - 08 Populate Table SOAP WS: Populate Table SOAP WS.

ca_pam_setTableData(_id, values)

Sets the data in a table.

Input Parameters

***_id* (string)**

Specifies the unique identifier of a Table Form element.

***values* (object)**

Specifies the data to display in the rows and columns of the table. This parameter only accepts the result of the SQL query.

Return Value

None.

Example

```
ca_pam_setTableData('Form.employeeList',result);
```

This example sets the employeeList table to the values that the result variable returns.

ca_pam_setTableDataFromJSObject(_id, values)

Populates a table with the JavaScript object array.

Input Parameters

_id **(string)**

Specifies the table ID.

values **(object)**

Specifies the JavaScript array containing a list of objects. The objects properties must be same as the column name to populate the data.

Return Value

None.

Example

This example shows how to populate a table from a JavaScript object array. The table ID is Form.name and has two columns as firstName and lastName.

```
var tableData = new Array();
var data1 = new Object();
data1.firstName = "firstName1";
data1.lastName = "lastName1";
tableData[0] = data1;

var data2 = new Object();
data2.firstName = "firstName2";
data2.lastName = "lastName2";
tableData[1] = data2;
ca_pam_setTableDataFromJSObject('Form.names',tableData );
```

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folders, then:
   - 07 Populate Table RESTful WS: Populate Table RESTful WS.
   - 08 Populate Table SOAP WS: Populate Table SOAP WS.

Simple Array

ca_pam_clearTableData(_id, startIndex, endIndex)

Deletes one or more rows of data from a table.

Input Parameters

***_id* (string)**

Specifies the unique identifier of a Table form element.

***startIndex* (integer)**

Specifies the numeric index of the first row of table data to delete.

***endIndex* (integer)**

Specifies the numeric index of the last row of table data to delete.

Return Value

None.

Example

```
ca_pam_clearTableData('Form.employeeList',0,ca_pam_getTableRowC
ount('Form.employeeList')-1);
```

This example deletes all rows in the employeeList table.

- The range to delete begins with row 0 and ends with the last row.
- To calculate the last row, the ca_pam_clearTableData function gets the current row count for the table from the ca_pam_getTableRowCount function and subtracts 1.

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folder, then:
   - 07 Populate Table RESTful WS: Populate Table RESTful WS.
   - 08 Populate Table SOAP WS: Populate Table SOAP WS.

ca_pam_getTableData(_id, startIndex, endIndex)

Returns the data of the table from the start index to the end index as provided in the method name. This function returns an array of JavaScript objects where each element in the array represents one row in the table. To retrieve the value for a particular column, you can access the value using column Name property from the JavaScript Object corresponding to the row.

Input Parameters

***_id* (string)**

Specifies the unique identifier of a Table form element.

***startIndex* (integer)**

Specifies the numeric index of the first row of the table from which to return data.

***endIndex* (integer)**

Specifies the numeric index of the last row of the table from which to return data. If you specify an index greater than the last available row, the function fails.

Return Value

Returns an array that is represented as a JavaScript object.

Important! After the method returns the table data, the variable names used to access the columns must be in lowercase, regardless of how you define the Name property for each column field in the Forms Designer.

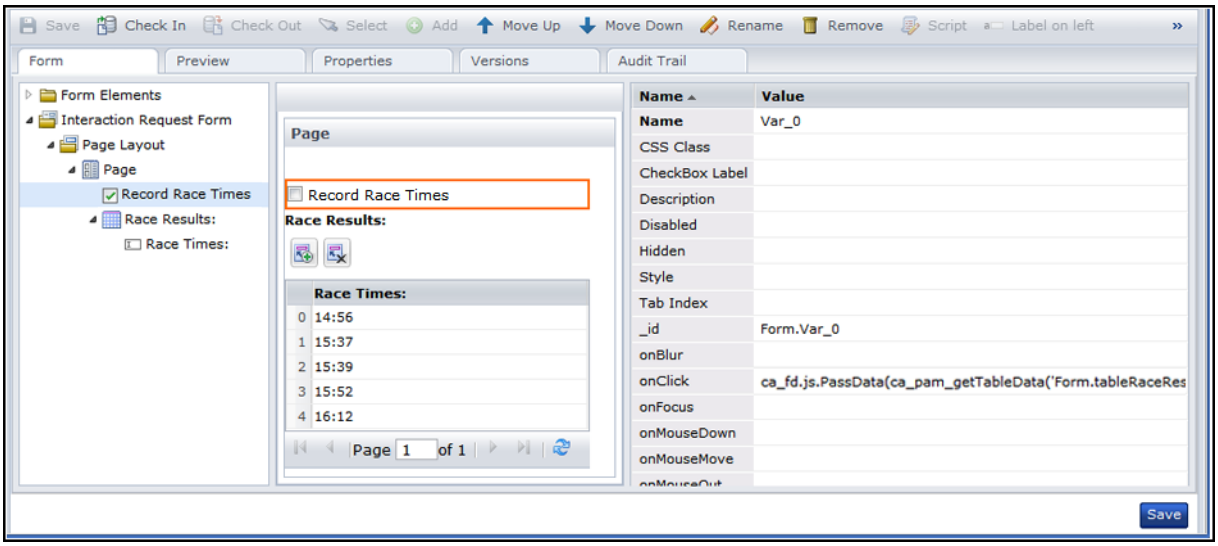
Example

```
ca_fd.js.PassData(ca_pam_getTableData('Form.tableRaceResults',0
,4))
```

This example uses the following form script to pass the top five finishing times in a race from a Table form element to a custom database. The Record Race Times check box onClick event initiates function calls to get the table data, pass the data, and record the results.

```
{
 PassData : function(result) {
   for(i=0;i<result.length;i++)
   {
    var firstRow = result[i];
    alert(firstRow.var_0);
   }
 }
}
```

The following illustration shows the form as it appears at design time.



ca_pam_getTableRowCount(_id)

Returns the total number of rows of data from a table.

Input Parameters

***_id* (string)**

Specifies the unique identifier of a Table Form element.

Return Value

Returns a number of rows as an integer.

Example

```
ca_pam_getTableRowCount('Form.employeeList')
```

This example returns the row count of the employee list Table Form element.

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folders, then:
   - 07 Populate Table RESTful WS: Populate Table RESTful WS.
   - 08 Populate Table SOAP WS: Populate Table SOAP WS.

ca_pam_setTableData(_id, values)

Sets the data in a table.

Input Parameters

***_id* (string)**

Specifies the unique identifier of a Table Form element.

***values* (object)**

Specifies the data to display in the rows and columns of the table. This parameter only accepts the result of the SQL query.

Return Value

None.

Example

```
ca_pam_setTableData('Form.employeeList',result);
```

This example sets the employeeList table to the values that the result variable returns.

ca_pam_setTableDataFromJSObject(_id, values)

Populates a table with the JavaScript object array.

Input Parameters

***_id* (string)**

Specifies the table ID.

***values* (object)**

Specifies the JavaScript array containing a list of objects. The objects properties must be same as the column name to populate the data.

Return Value

None.

Example

This example shows how to populate a table from a JavaScript object array. The table ID is Form.name and has two columns as firstName and lastName.

```
var tableData = new Array();
var data1 = new Object();
data1.firstName = "firstName1";
data1.lastName = "lastName1";
tableData[0] = data1;

var data2 = new Object();
data2.firstName = "firstName2";
data2.lastName = "lastName2";
tableData[1] = data2;
ca_pam_setTableDataFromJSObject('Form.names',tableData );
```

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folders, then:
   - 07 Populate Table RESTful WS: Populate Table RESTful WS.
   - 08 Populate Table SOAP WS: Populate Table SOAP WS.

Object Reference

ca_pam_getTextFieldValue(_id)

Returns a value from a Text field.

Input Parameters

***_id* (string)**

Specifies the unique identifier of a Text field.

Return Value

Value from the Text field (string).

Example

This example returns the value of the top_vendors Text field.

```
ca_pam_getTextFieldValue('Form.top_vendors')
```

This example returns the value of the top_vendors Text field and sets the dynamic_field Text field to the returned value.

```
ca_pam_setTextFieldValue('Form.dynamic_field',
ca_pam_getTextFieldValue('Form.top_vendors'))
```

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folders, then:
   - 07 Populate Table RESTful WS: Populate Table RESTful WS.
   - 08 Populate Table SOAP WS: Populate Table SOAP WS.

ca_pam_setTextFieldValue(_id, val)

Sets the value of a Text field.

Input Parameters

***_id* (string)**

Specifies the unique identifier of the field for which to set a specific value.

***val* (string)**

Defines the new value for the Text field.

Return Value

None.

Example

This example sets the top_vendors Text field to "I vote for vendor 3 because:"

```
ca_pam_setTextFieldValue('Form.top_vendors', 'I vote for vendor 3 because: ')
```

To continue this example, use ca_pam_getTextFieldValue('Form.top_vendors') after users enter their reasons to submit their full statements.

```
ca_pam_setTextFieldValue('Form.dynamic_field', ca_pam_getTextFieldValue('Form.top_vendors'))
```

This example returns the value of the top_vendors field and sets the dynamic_field to that value.

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folder, then 08 Populate Table SOAP WS: Populate Table SOAP WS.

Data Sources

ca_pam_getDatasetData(dataSetExpression,callBack)

Retrieves the value of a variable from a CA Process Automation dataset.

Note: In CA Process Automation releases before 04.1.00, this method was named ca_pam_getDataFromDatasetQuery.

Input Parameters

***datasetExpression* (string)**

Defines a string, expression, or function call that results in an absolute path to a dataset. This parameter includes the Datasets keyword, the library path to a dataset object, and the ValueMap array variable name.

```
Datasets["/BR412-DATA-FOLDER-20111214/Folder_FORMS_20120328
/dataset_forms_demo/My_Dataset55"].MyVal
```

***callBack* (object)**

Defines a reference to the function object to call after retrieving the data, including its onSuccess and onFailure methods.

Return Value

When the function retrieves the data, either the onSuccess(result) or onFailure(caught) callBack function runs. The function itself does not return any value.

Example

```
ca_pam_getDatasetData(ca_pam_getTextFieldValue('Form.TxtFieldWi
thDatasetPath'),callBack)
```

The example code contains the following information:

- The TxtFieldWithDatasetPath Text field stores the dataset expression.
- The getTextFieldValue function gets the value of the dataset expression.
- The getDataUsingDatasetExpression function gets the dataset values and calls the callBack function to determine what happens next.
- To continue with this example, you can use the following functions in the callBack.onSuccess declaration to apply the data to a table or Lookup field, respectively:

```
ca_pam_setTableData('Form.employeeList',result);
ca_pam_showDataInTable(result,'Form.Results','Dataset
results');
```

ca_pam_getSQLData(driverName,connectionURL,userName,password,query,callBack)

Directs SQL to retrieve data from an external data source and store that data in a variable (named callBack) that you define.

Note: In CA Process Automation releases before 04.1.00, this method was named ca_pam_getDataFromSQLQuery.

Input Parameters

***driverName* (string)**

Defines the class name for the installed SQL driver that you are using (for example, com.mysql.jdbc.driver).

***connectionURL* (string)**

Defines the URL of the database application to query as in the following example:

```
jdbc:mysql://myPC-xp.myCompany.com:CA Portal/<path> or
https://<server>:CA Portal/
```

***userName* (string)**

Defines a user name or login credentials for a predefined user account with sufficient permissions to run the query.

***password* (string)**

Defines the password that is associated with the specified user name.

***query* (string)**

Defines a specific SQL query statement or a reference to an SQL query statement. The following code represents a sample SQL query:

```
"select * from employeedatatable where empName like
'%"+ca_pam_getTextFieldValue('Form.empName')+"%'";
```

***callBack* (object)**

Defines a reference to the function object to call after retrieving the data, including its onSuccess and onFailure methods.

Return Value

When the function retrieves the data, either the onSuccess(result) or onFailure(caught) callBack function runs. The function itself does not return any value.

Example

```
ca_pam_getSQLData(ca_fd.js.driverName
(),ca_fd.js.connectionURL(),ca_fd.js.userName(),ca_fd.js.passwo
rd(),ca_fd.js.queryEmployeeName(),callBack);
```

This example queries an external data source and stores the resulting data in the callBack variable.

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folder, then 06 Populate Dropdown DB: Populate Dropdown from DB.

JSON Parsing

ca_pam_convertJSONToJSObject(jsonString)

Use this method to create a JavaScript object from a JSON string. The ca_pam_convertJSONToJSObject method supports quotation marks notation. Quotation marks notation is an alternate way to access any property from a JavaScript object when the property name is not a valid JavaScript identifier. For example, to access the id property from the book object, quotation marks notation syntax is book["id"].

Input Parameters

***jsonString* (string)**

Defines the JSON string to convert to a JavaScript object.

Return Value

Returns the JavaScript object.

Example

The example shows a JSON string:

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

The following sample script accesses the id field value:

```
var parsedJSON = ca_pam_convertJSONToJSObject(jsonString);
var menuObj = parsedJSON.menu.;
var idValue = menuObj.id.;
```

The following sample script accesses the onclick element in the second element of the menuitem array:

```
var parsedJSON = ca_pam_convertJSONToJSObject(jsonString);
var menuObj = parsedJSON.menu;
var popUpObj = menuObj .popup;
var menuItemArray = popUpObj .menuitem.;
var secondElementOfMenuItemArray = menuItemArray[1];
var onclickElement = secondElementOfMenuItemArray.onclick;
```

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folder, then 07 Populate Table RESTful WS: Populate Table RESTful WS.

REST Methods

ca_pam_getRESTData(url, doNotValidateCert, headers, callBack)

Uses the HTTP get method to start the REST service. You can use this method with the following signatures:

```
ca_pam_getRESTData(url, callBack);
```

```
ca_pam_getRESTData(url, doNotValidateCert, callBack);
```

Input Parameters

***url* (string)**

Defines the URL of the HTTP request. The URL starts with http:// or https://.

***doNotValidateCert* (boolean)**

Specifies whether a valid SSL certificate is found. This field is relevant when querying an HTTPS URL.

- false - Validates the SSL certificate and fails the operation if the certificate is invalid.
- true - Accepts the SSL certificate even if it is invalid and continues to make the HTTP call.

***headers (*object)**

Defines a list of key/value pairs that sets headers in the request.

***callBack* (object)**

Defines a reference to the function object to call after retrieving the data, including its onSuccess and onFailure methods.

Return Value

When the function retrieves the data, either the onSuccess(result) or onFailure(caught) callBack function runs. The function itself does not return any value.

Example

You can find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folder, then 07 Populate Table RESTful WS: Populate Table RESTful WS.

ca_pam_getRESTDataAuth(url, userName, password, doNotValidateCert, headers, callBack)

Starts the REST service at a specified URL that requires authentication. This method supports only Basic HTTP authentication. You can use this method with the following signatures:

`ca_pam_getRESTDataAuth (url, userName, password, callBack)`

`ca_pam_getRESTDataAuth (url, userName, password, doNotValidateCert, callBack)`

Input Parameters

***url* (string)**

Defines the URL of the HTTP request. The URL starts with http:// or https://.

***username* (string)**

Defines the user name for which to authenticate the specified URL.

***password* (string)**

Defines the password associated with the specified user name.

***doNotValidateCert* (boolean)**

Specifies whether a valid SSL certificate is found. This field is relevant when querying an HTTPS URL.

- false - Validates the SSL certificate and fails the operation if the certificate is invalid.
- true - Accepts the SSL certificate even if it is invalid and continues to make the HTTP call.

***headers* (object)**

Defines a list of key/value pairs that sets headers in the request.

***callBack* (object)**

Defines a reference to the function object to call after retrieving the data, including its onSuccess and onFailure methods.

Return Value

When the function retrieves the data, either the onSuccess(result) or onFailure(caught) callBack function runs. The function itself does not return any value.

ca_pam_postRESTData(url, body, contentType, doNotValidateCert, headers, callBack)

Uses the HTTP post method to start the REST service. You can use this method with the following signatures:

```
ca_pam_postRESTData (url, callBack)
ca_pam_postRESTData (url, body, callBack)
ca_pam_postRESTData (url, body, contentType, callBack)
ca_pam_postRESTData (url, body, contentType, doNotValidateCert,
callBack)
```

Input Parameters

***url* (string)**

Defines the URL of the HTTP request. The URL starts with http:// or https://.

***body* (string)**

Defines the data to send in the HTTP request.

***contentType* (string)**

Defines the type of content that comprises the HTTP request body. This value is sent as a header (content-type) in the HTTP request.

***doNotValidateCert* (boolean)**

Specifies whether a valid SSL certificate is found. This field is relevant when querying an HTTPS URL.

- false - Validates the SSL certificate and fails the operation if the certificate is invalid.
- true - Accepts the SSL certificate even if it is invalid and continues to make the HTTP call.

***headers* (object)**

Defines a list of key/value pairs that sets headers in the request.

***callBack* (object)**

Defines a reference to the function object to call after retrieving the data, including its onSuccess and onFailure methods.

Return Value

When the function retrieves the data, either the onSuccess(result) or onFailure(caught) callBack function runs. The function itself does not return any value.

ca_pam_postRESTDataAuth(url, userName, password, body, contentType, doNotValidateCert, headers, callBack)

Starts the REST service at a specified URL that requires authentication. The server calls the REST Methods asynchronously.

The onSuccess(result) or the onFailure(caught) function of the callback object runs as appropriate after the data is retrieved from the server. The result can be any data format that the REST service returns. The method supports only Basic HTTP authentication. You can use this method with the following signatures:

```
ca_pam_postRESTDataAuth(url, userName, password, callBack)
ca_pam_postRESTDataAuth(url, userName, password, body, callBack)
ca_pam_postRESTDataAuth(url, userName, password, body, contentType,
callBack)
ca_pam_postRESTDataAuth(url, userName, password, body, contentType,
doNotValidateCert, callBack)
```

Input Parameters

***url* (string)**

Defines the URL of the HTTP request. The URL starts with http:// or https://.

***username* (string)**

Defines the user name for which to authenticate the specified URL.

***password* (string)**

Defines the password that is associated with the specified user name.

***body* (string)**

Defines the data to send in the HTTP request.

***contentType* (string)**

Defines the type of content that comprises the HTTP request body. This value is sent as a header (content-type) in the HTTP request.

***doNotValidateCert* (boolean)**

Specifies whether a valid SSL certificate is found. This field is relevant when querying an HTTPS URL.

- false - Validates the SSL certificate and fails the operation if the certificate is invalid.
- true - Accepts the SSL certificate even if it is invalid and continues to make the HTTP call.

***headers* (object)**

Defines a list of key/value pairs that sets headers in the request.

***callBack* (object)**

Defines a reference to the function object to call after retrieving the data, including its onSuccess and onFailure methods.

Return Value

When the function retrieves the data, either the onSuccess(result) or onFailure(caught) callBack function runs. The function itself does not return any value.

Example

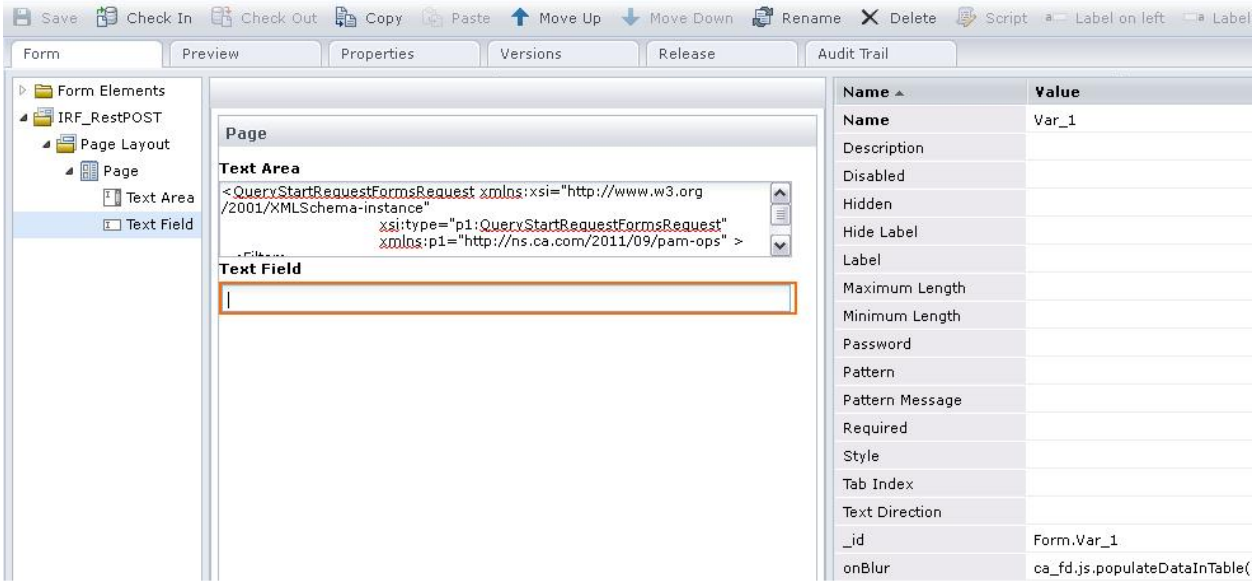
You can find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folder, then 07 Populate Table RESTful WS: Populate Table RESTful WS.

Example: REST POST Method

To retrieve all the start request forms in the CA Process Automation library, call the PAM REST API contained in the CA Catalyst Container.

1. Create an interaction request form and design it as the following illustration shows:



2. Add the Keyword field to the interaction request form.
3. Add the following text in the Script section of the interaction request form:

```
{
  sample : function()
  {
    var callBack = new Object();
    callBack.onSuccess = function(result)
      {
            alert(result);
      }

      callBack.onFailure = function(caught)
            {
              alert(caught);
            }

var headers = new Object();
var contentType="application/xml";
ca_pam_getDataFromRESTPostHTTPAuthentication('https://<hostName>:<PortNumber>
/node/rest/CA:00074:01/_ops/QueryStartRequests',useName
password,<requestBody>,contentType,true,headers,callBack);
  }
}
```

The Request body is as follows:

```
<QueryStartRequestFormsRequest
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                xsi:type="p1:QueryStartRequestFormsRequest"
                                xmlns:p1="http://ns.ca.com/2011/09/pam-ops" >
  <Filter>
    <LookUpPath>/</LookUpPath>
    <IsRecursive>true</IsRecursive>
  </Filter>
</QueryStartRequestFormsRequest>
```

4. Provide the ca_fd.js.sample() value in the Keyword field onBlur event.
5. Click Save.

   The script runs according to the Keyword field onBlur event. The start request form is retrieved according to the specified query body and the REST request made.

SOAP Methods

ca_pam_getSOAPData(serviceURL, methodName, inlineText, soapVersion, stripXMLNamespacesFromResponse, callBack)

Makes a Web service call and converts the resulting XML into a JavaScript object.

This method is an overloaded method. ca_pam_getSOAPData can be invoked without providing the SOAP version and/or stripXMLNamespacesFromResponse parameter. If you do not provide these parameters, then the SOAP version is considered SOAP_1_1, and stripXMLNamespacesFromResponse is considered true.

The following are the overloaded method signatures:

- ca_pam_getSOAPData (serviceURL,methodName, inlineText,callBack)
- ca_pam_getSOAPData (serviceURL,methodName, inlineText,soapVersion, callBack)

Input Parameters

***serviceURL* (string)**

Specifies the URL for the SOAP service. The URL is typically accessed over HTTP or HTTPS. The URL is an entry point for one or more methods.

***methodName* (string)**

Specifies the method or function to run. The function passes the method to the SOAP service as a MIME SOAPAction header.

***inlineText* (string)**

Specifies the source for the SOAP service input request. This parameter includes an XML message, which can include a SOAP envelope.

***soapVersion* (string)**

Specifies the version of the SOAP server on which the call is made. Possible values include SOAP_1_1 or SOAP_1_2. The default is SOAP _1_1.

***stripXMLNamespacesFromResponse* (boolean)**

Specifies whether the response Name Spaces are removed from the XML response (true). The default is true.

***callBack* (object)**

Defines a reference to the function object to call after retrieving the data, including its onSuccess and onFailure methods.

The ca_pam_getSOAPData function calls the callBack method asynchronously on the server. When the function retrieves the data, either the onSuccess(result) or onFailure(caught) callBack function runs. If the function returns a SOAP fault, it calls the onFault(faultString) method and the result is an XML string.

If you do not provide the SOAP version and stripXMLNameSpacesFromResponse parameters, their default values are substituted and the response is returned to the callBack object in the onSuccess(result).

Return Value

When the function retrieves the data, either the onSuccess(result) or onFailure(caught) callBack function runs. If the function returns a SOAP fault, it calls the onFault(faultString) method and the result is an XML string. The function itself does not return any value.

Note: You can find this function in the out-of-the-box content in CA Process Automation.

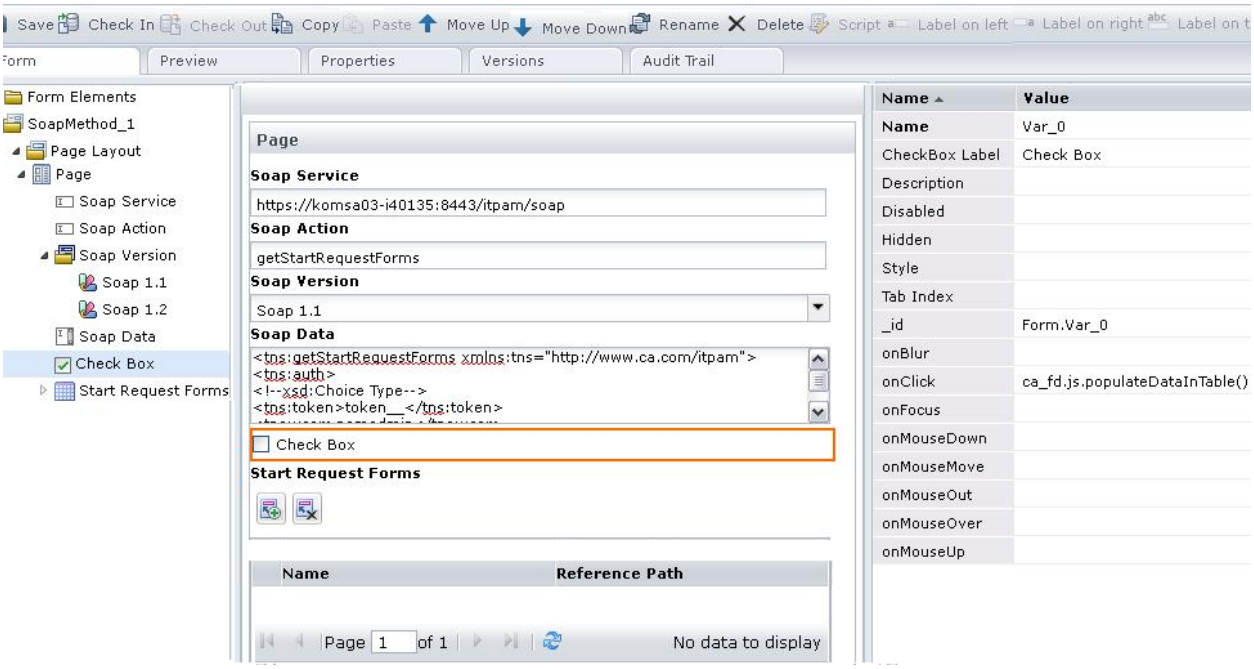
1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folder, then 08 Populate Table SOAP WS: Populate Table SOAP WS.

Example: SOAP Method

This example uses the CA Process Automation getStartRequestForms web service method to populate a table with all Start Request Forms in the Library . The getStartRequestForms web service method returns all the start request forms to a specified folder.

Note: This method uses the ca_pam_convertXMLToJSObject(xmlString, elementTagName) method (described later in this section) to create a JavaScript object from XML.

1. Create an interaction request form and design it as the following illustration shows:



2. Add the following fields to the interaction request form:

   soapService

   Set this field to the following URL:

   ```
   http://hostname:portNumber/itpam/soap
   ```

   soapAction

   Set this field to getStartRequestForms.

   soapData

   Populate this text area with the following code:

   ```
   <tns:getStartRequestForms xmlns:tns="http://www.ca.com/itpam">
    <tns:auth>
     <!--xsd:Choice Type-->
     <tns:token>token__</tns:token>
     <tns:user>pamadmin</tns:user>
     <tns:password>pamadmin</tns:password>
    </tns:auth>
    <tns:filter>
     <tns:lookUpPath isRecursive="true">/</tns:lookUpPath>
    </tns:filter>
   </tns:getStartRequestForms>
   ```

   retrieveSRF

   Set the onClick attribute of this check box to the ca_fd.js.retreiveSRFAndPopulateTable() value.

3. Create a table named srfs.
4. Add name and refPath text fields as columns to the srfs table.
5. Add the following text in the Script section of the form:

   ```
   {
     retreiveSRFAndPopulateTable: function()
       {
          var callBack = new Object();
          callBack.onSuccess = function(result)
            {
               var srfResult = ca_pam_ convertXMLToJSObject
   (result,'startRequest');
               var tableArray = new Array();
               for( i=0;i<srfResult .length;i++)
                 {
                    var object = new Object();
                    object.name = srfResult [i]["name"];
                    object.refPath = srfResult [i]["refPath"];
                    tableArray[i] = object ;
                 }
   ```

```
ca_pam_clearTableData('Form.srfs',0,ca_pam_getTableRowCount('Form.srfs')-1);
            ca_pam_setTableDataFromJSObject('Form.srfs',tableArray);
          }
        callBack.onFailure = function(caught)
          {
            alert(caught);
          }

ca_pam_getSOAPData(ca_pam_getTextFieldValue('Form.soapService'),ca_pam_getTex
tFieldValue('Form.soapAction'),ca_pam_getTextFieldValue('Form.soapData'),'SOA
P_1_1',true,callBack);
    }
}
```

6. Click Save.
7. To have CA Process Automation verify that the srfs table is populated dynamically with data, select the Check Box.

The script runs when the checkbox value is changed. The script populates the srfs table dynamically with the start request forms returned by the SOAP call. The getSOAPData method is used to make a SOAP query to retrieve the data and the convertXMLToJSObject is used to convert the response XML into a JavaScript object. The JavaScript object can then be used to populate the table dynamically by using the ca_pam_setTableDataFromJSObject method.

XML Parsing

ca_pam_convertXMLToJSObject(xmlString, elementTagName)

Creates a JavaScript object from XML.

Note: If the XML element is namespace aware and has a namespace prefix, use quotes notation to access it.

Input Parameters

***xmlString* (string)**

Defines the XML string to convert to a JavaScript object.

***elementTagName* (string)**

(Optional) Returns the element that has the specified tag name. If you do not provide the elementTagName, the method returns the root element.

Return Value

Returns a value of Object type.

Example

Consider the following XML to convert it to a JavaScript object:

```
<?xml version="1.0"?>
<catalog>
   <book id="bk101">
      <author>Gambardella, Matthew</author>
      <title>XML Developer”s Guide</title>
      <genre>Computer</genre>
      <price>44.95</price>
      <publish_date>2000-10-01</publish_date>
      <description>An in-depth look at creating applications
      with XML.</description>
   </book>
</catalog>
```

This example converts the preceding XML data to a JavaScript object.

- The following syntax converts the XML document to a JavaScript object:

  ```
  var parsedXML = ca_pam_convertXMLToJSObject(xmlString);
  ```

- You can use either of the following lines to access the book element:

  ```
  var bookObj = parsedXML . book[0];
  var bookObj = parsedXML [“book”][0];
  ```

- You can use either of the following lines to retrieve the id attribute value of the book element:

  ```
  var idVal = bookObj.id;
  var idVal = bookObj[“id”];
  ```

- You can use either of the following lines to access the description value of the book element:

```
var bookDesription = bookObj. description[0]._text;
var bookDesription = bookObj[“description”][0][“_text”];
```

Note: If an element has an attribute and an element with the same name, access them as an array. In the array, the first member (index 0) is always the attribute and the elements follow.

Example

Consider the following XML to convert it to a JavaScript object:

```
<?xml version="1.0"?>
<categories>
    <category name="Weather">
        <name part="1">Sunny</name>
        <name part="2">Rainy</name>
    </category>
    <category name="Climate">
        <name part="1">Wet</name>
        <name part="2">Dry</name>
    </category>
</categories>
```

This example converts the preceding XML data to a JavaScript object.

- Use the following syntax to retrieve the first category object from categories XML represented as the xmlString:

```
var parsedXML= ca_pam_convertXMLToJSObject(xmlString);
var firstCatagory= parsedXML.category[0];
```

- Use the following syntax to access the name attribute:

```
var bookName = firstCatagory.name[0];
```

- Use the following syntax to access the value of first name element:

```
var firstBookName = firstCatagory.name[1]._text;
```

Example

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <Header/>
   <Body>
         <getStartRequestFormsResponse
xmlns="http://www.ca.com/itpam">
               <startRequests>
                     <startRequest name="RahulSrf"
refPath="/Folder/">
                           <description/>
                     </startRequest>
                     <startRequest name="Start Request Form"
refPath="/RestSupport/">
                           <description/>
                     </startRequest>
               </startRequests>
         </getStartRequestFormsResponse>
   </Body>
</Envelope>
```

This example shows the usage of the ca_pam_convertXMLToJSObject(xmlString, elementTagName) method with the elementTagName parameter.

The following script retrieves all the <startRequest> elements from XML:

```
var srfResult =
ca_pam_convertXMLToJavaScriptObject(xmlString,'startRequest');
```

srfResult is an array of all the startRequest elements.

Example

```
<?xml version="1.0"?>
<note _text="abc">
    <_text>Tove</_text>
</note>
```

This example shows a scenario in which the XML has _text as an element or attribute. In this case, the method creates an array with the name "_text".

- The first element of the array returns the text node in the element.
- The second element of the array returns the attribute node (with the name "_text") of the element.
- Other elements of the array return the child element nodes (with the name "_text") of the element.

The following script accesses the values:

```
var parsedXML= ca_pam_convertXMLToJavaScriptObject(xmlString);
var textNode= parsedXML["_text"][0];
var attributeValue= parsedXML["_text"][1];
var childTextNodeValue= parsedXML["_text"][2]["_text"];
```

The script has the following result:

```
textNode="";
```

(...as there is no scalar value with the node.)

```
attributeValue="abc";
childTextNodeValue="Tove"
```

You can also find this function in the out-of-the-box content in CA Process Automation.

1. On the Home page, click Browse Out-of-the-Box Content.
2. Navigate to the User Interaction Forms folder, then 08 Populate Table SOAP WS: Populate Table SOAP WS.

Create a Simple Form with Basic Functions

In this example, you are developing a form for a company that operates in three primary regions (California, Pennsylvania, and North Carolina). In each state, the company has a primary and secondary office location that the name of the city identifies. The stakeholders responsible for implementing company policy want the form to meet the following business requirements:

- All business that originates in California must be routed by default to the Anaheim office.
- All business that originates in Pennsylvania must be routed by default to the Philadelphia office.
- All business that originates in North Carolina must be routed by default to the Raleigh office.
- All business that originates in any other state must be routed by default to the Pittsburgh office.
- The user has the option of specifying another state or city.

Example: A Basic Form

Follow these steps:

1. Click the Library tab.
2. Create an Interaction Request Form object or locate an existing one to use for this sample procedure.
3. Double-click the form object.

   The Interaction Request Form dialog, or Forms Designer, appears.
4. If the form object is not already checked out, click Check Out.
5. In the Form Elements pane, expand the following entries:
   a. Form Elements
   b. Your form (for example, Interaction_Request_Form_4)
   c. Page Layout
   d. Page

6. Click Page.
7. Click Rename in the toolbar
8. Enter the name *Location:* and click OK.
9. Drag a Select field from the Form pane to the Location page.
10. Repeat Step 9.

    Two Select fields appear under your Location page layout.
11. Complete the following actions in the Form pane:
    a. Drag three Select Options to the first Select field.
    b. Drag six Select Options to the second Select field.
12. Click each of the following form elements and then click Rename to rename each object as indicated:
    a. Rename the first Select field to *State*.
        - Rename the first option to *CA*.
        - Rename the second option to *PA*.
        - Rename the third option to *NC*.
    b. Rename the second Select field to *City*.
        - Rename the first option to *Anaheim*.
        - Rename the second option to *Los Angeles*.
        - Rename the third option to *Pittsburgh*.
        - Rename the fourth option to *Philadelphia*.
        - Rename the fifth option to *Charlotte*.
        - Rename the sixth option to *Raleigh*.

13. Click each of the following form elements and make the following property settings in the Properties (Name and Value) pane.

    a. For State:

    - Set Name to *State*.
    - Set the onChange event value to ca_fd.js.selectRegion().
    - Set the value of the CA option to *California*.
    - Set the value of the PA option to *Pennsylvania*.
    - Set the value of the NC option to *North Carolina*.

    b. For City:

    - Set Name to *City*.
    - Set the value property for each city option to the city name. For example, set Anaheim to *Anaheim*, Los Angeles to *Los Angeles*, and so on.

14. In the Form Elements pane, click the name of your form and then click Script in the toolbar.

    The Script dialog opens.

15. Copy and paste the following code in the Script editor (see page 134):

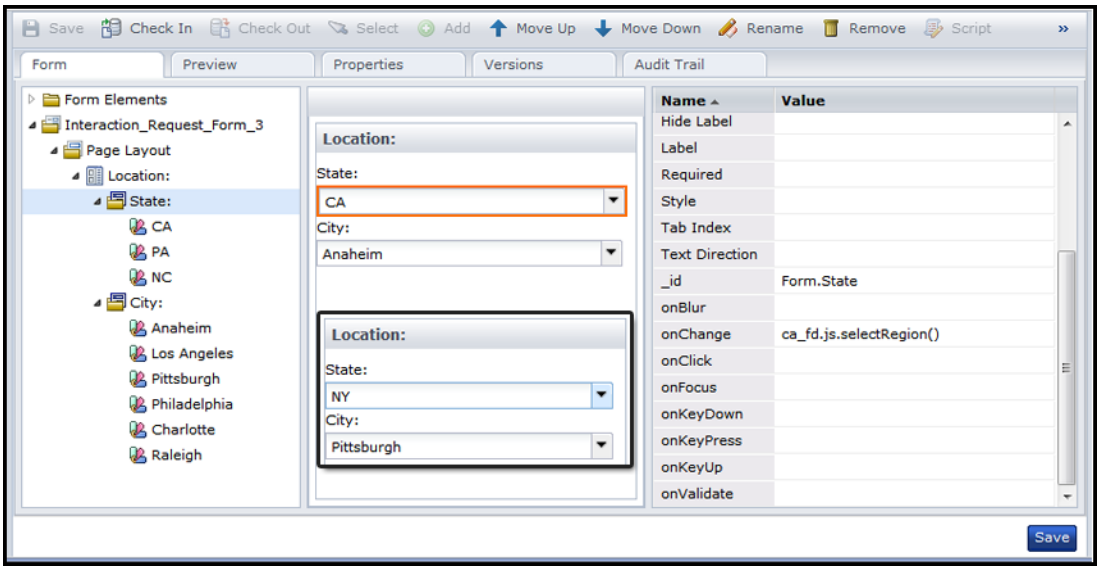
```
{
 selectRegion: function()
  {
    var selectedState = ca_pam_getSelectedOptionValues('Form.State')[0];

    if(selectedState == 'North Carolina')
     {
      ca_pam_selectOption('Form.City', 'Raleigh', 'Raleigh', true);
     }
    else if(selectedState == 'California')
     {
      ca_pam_selectOption('Form.City', 'Anaheim', 'Anaheim', true);
     }
    else if(selectedState == 'Pennsylvania')
     {
      ca_pam_selectOption('Form.City', 'Philadelphia', 'Philadelphia', true);
     }
    else
     {
      ca_pam_selectOption('Form.City', 'Pittsburgh', 'Pittsburgh', true);
     }
  }
}
```

16. Click Save. The Script dialog closes.

17. Click Save in the Forms Designer.

18. Click the Preview tab.
19. Confirm that your results are similar to the sample results and values in the following illustration and table:



State:	Default City:
CA	Anaheim
PA	Philadelphia
NC	Raleigh
Any other state. For example, NY.	Pittsburgh

Initialize Form Variables

After designing an interaction request form, you can set it to a specific Assign User Task operator in a process. You can also add code to initialize form fields at runtime.

Follow these steps:

1. Click the Designer tab.
2. In the Process Designer, open a process or create one.
3. In the Operators palette Process Control group, drag an Assign User Task operator to your process.
4. Double-click the Assign User Task operator to open its Properties palette.
5. In the Properties palette:
   a. Expand User Task.
   b. In the Interaction Request Form field, click the lookup button to browse for a form. Click OK.
   c. Click the Form Data Initialization Code field to expand it.
6. In the Form Data Initialization Code window, initialize any form variables. For the following examples, myTextField is the _id of the form element that you want to initialize.
   - For simple data types, enter:

     ```
     Form.myTextField='welcome'.
     ```

   - If the simple field is inside a ValueMap or field set, enter:

     ```
     Form.value_map= newValueMap();
     Form.valuemap.myTextField="welcome";
     ```

   - If the simple field is inside a complex value map in a valueMap, enter:

     ```
     Form.value_map.value_map_nested= newValueMap();
     Form.value_map.value_map_nested.text_field_nested="test";
     ```

7. In the Form Data Initialization Code window, click OK.

Chapter 8: Resources

A *Resource* object is a model that represents elements of your system architecture. Use them to synchronize independent processes that rely on common infrastructure elements and to quantify and control access to the specific IT entities. Include multiple resources that represent related entities in a single Resource object.

Use Resource objects to:

- Balance the processing load across all processes running on a touchpoint.
- Synchronize the execution of processes that cannot execute in parallel.
- Implement environment level locks that simultaneously enable or disable multiple resources.
- Strategically manage processes and systems with common security rights.

You group resources because they are related to each other in some way. Examples include shared databases, transmission links, simultaneous access to a limited number of software licenses, concurrent processes on a touchpoint, numeric quotas, and other resources. After measuring performance, you can allocate system resources to processes required by mission critical tasks. You can limit the number of simultaneous FTP connections used by CA Process Automation. You can use resources to start a successor process *only* after an antecedent process has released a resource. Resources can also be used to represent and control access to a particular IT environment entity such as a log file that receives updates from multiple processes.

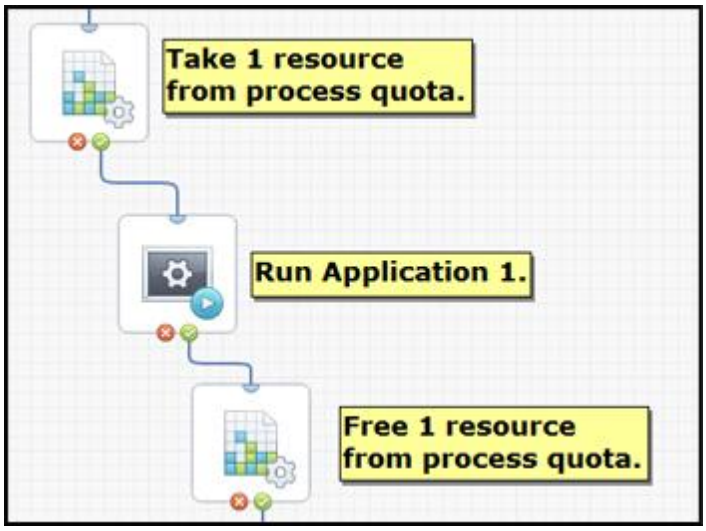
This section contains the following topics:

How Resources Work (see page 343)
Create a Resource Object (see page 345)
Edit a Resource Object (see page 346)
Monitor and Edit Resources (see page 348)
Add a Manage Resources Operator to a Process (see page 349)
Define Resource Actions (see page 350)
Check for and Respond to Unavailable Resources (see page 351)

How Resources Work

The *Manage Resources* operator is in the Process Control operator group. A process can use the Manage Resources operator to take available units from a specified resource. If all units of the specified resource are taken (in the same or different processes), the Manage Resources operator delays processing along that branch until the resource has free units.

The following design shows a process taking one unit from the *process quota* resource before running an application. If there are available units in the process quota resource, processing continues to *Run Application 1*. If there are no available resource units, processing does not start until either units become available or the operator times out. After the Run Application 1 operator ends, the resource unit is freed back to the process quota resource, and processing continues with subsequent operators in the process.



Resources let you define the number of units available and how many units are consumed. A resource consists of a maximum number of units, the current value of available units, and a flag indicating whether the resource is locked.

You can take the following actions with resources:

- Take a specified quantity of units of an available resource. Enter a higher value in the Used field.
- Free a specified quantity of units of an available resource. Enter a lower value in the Used field.
- Lock a resource
- Unlock a resource

A Manage Resources operator can consume or free any specified number of resources. As a developer or administrator, you can use Manage Resources operators to tune the load balancing on a particular touchpoint. You can also lock a resource to prevent consumption of resource units by any other process.

The following constraints apply to resources:

- The maximum number of units of any resource is an arbitrary value that you can calibrate and fine-tune for your system requirements. The number of resources that a process uses is also arbitrary. The maximum limit is 9,999; however, let performance and architecture be your guide. Allocate resource units to processes to best suit your implementation requirements.
- The currently used value of resource units is always less than or equal to the maximum value of the resource.
- A resource-dependent process must wait until its specified number of units is available.
- Operators cannot consume units from a locked resource.
- Operators cannot lock a resource that another process has locked.

Create a Resource Object

Create and define as many *Resource* objects as required in each orchestrator. The Resource operator performs resource operations in a process. The Resource operator uses the current versions of specified Resource objects. Any modifications made to the Resource object in its current state are immediately available to the Resource operator upon check-in.

Follow these steps:

1. Click the Library tab.
2. In the left pane, click a folder, and select New and then Resources.

   A new resource appears.
3. Click the resource name and rename it to a more meaningful name.
4. Edit the Resource object.

Note: You can also create Resource objects dynamically using code.

Edit a Resource Object

Edit a Resource object to manage individual resources within it. You can also manage versions, view properties, and examine the object's history.

Add individual resource entries for applications, connections, or other instances you want to control. This enables you to:

- Set a maximum number of instances that can be run at any moment.
- Track the number of instances or units running concurrently.
- Track the number of available instances or free units that can be started at any time.

Follow these steps:

1. Click the Library tab.
2. Double-click a Resource object.

   The Resources dialog appears. The Resources tab opens by default.
3. Click Add to add an individual resource.
4. Review any of the values in the columns of the Resources tab. Click in the editable cells to enter new values.

   Name

   Lists the names of individual resources in a Resource object.

   Amount

   Lists the total number of units assigned to a resource. A unit is an arbitrary number that serves as a quota in a process.

   Used

   Indicates the number of assigned units.

   Free

   Indicates the number of unassigned units. Defined by the formula:

   ```
   Free = Amount - Used
   ```

   State

   Specifies whether a resource is locked or unlocked. Click the lock icon in this column to toggle the locked or unlocked state for a resource. You can also use a Resource operator to lock a resource programmatically in a schedule or process. Other Manage Resource operators cannot lock, unlock, take, or return resource units for a locked resource until the lock is released. A lock allows a process or schedule to monopolize a resource while it processes operators.

   % Usage

Position your mouse over this visual indicator to view the numeric percentage of the resource currently being utilized. Any remaining portion is free.

Description

Provides a description of the resource. The Description column allows you to enter text that describes a resource.

5. Click Check In or Save and Close.

Notes:

- In the *Amount* field, specify the quantity for the resource. Quotas for operators in processes are drawn from this number. The quantity is an arbitrary value that is not by itself related to units of any actual computer or system resource (such as CPU, memory, or bandwidth). You can use it to apportion a resource to processes in whatever manner that you require. There are no rules about the quantity of a resource. You might specify an amount of *1*, so only one instance of a CPU-intensive operator can be run by any process at any given time.
- Dataset variables can be used to set resource use, so usage can be fine-tuned on a touchpoint without opening and configuring processes that consume a resource. For example, if you set the amount to *100*, you could change a usage variable from *10*, to *20*, to *50*, or even to *100* to accommodate demands on a touchpoint.
- The *Used* and *Free* columns show how many units of a resource are currently consumed and available. You can enter a value in the Used field between 0 and the total number of units shown in the *Amount* field. More commonly, a Resource operator changes these settings programmatically in a process or schedule.

Monitor and Edit Resources

You can monitor and edit the resources in a Resource object on the Operations page. The Operations page allows you to modify the current version of a Resource object. Your changes are automatically applied to any Manage Resource operators using the Resource object.

Follow these steps:

1. Click the Operations tab.
2. In the Links pane:
   a. Expand Process Watch or Resources.
   b. Expand the list of folders or search for a specific process watch or resource object.
   c. Select a resource object.

      The current version of the Resource object appears.
3. Select a resource and then make any of the following changes.
   a. In the toolbar, click Lock to lock the resource. Click Unlock to unlock a resource. You can also click the lock icon in the State column to toggle this setting.
   b. Click Clear to release any used resources and reset free resources to the maximum amount available.
   c. Click the Amount, Used, and Description fields to edit their values.
   d. Click Refresh to view the latest system data.

      Your changes are available to Manage Resource operators.
4. To add or edit resources in a Manage Resource operator in a Schedule:
   a. Expand Links and click Active or Global Schedules.
   b. In the list of schedules, double-click a schedule.
   c. In a Manage Resources item of the schedule, click Properties.
   d. Click the Specific tab.
   e. Click Check Out.
   f. Click Add, Edit, or Delete to configure the resources.
   g. Click Check In or click Save and Close.

Add a Manage Resources Operator to a Process

After creating a resource object with one or more resources within it, add a Resource operator to a process. For example, place one Resource operator before and another one after other operators to balance load. The first Resource operator uses or takes resources and the second operator frees or gives back units, making them available to other waiting processes.

Follow these steps:

1. Open a process in the Process Designer.
2. Open the Process Control operator palette for your resource.
3. Drag and drop the Manage Resources operator to a location in the process.
4. Define the entry and exit links. The Manage Resources operator has four possible exit links:
   - **Completed** is processed when the operator succeeds. The *Result* variable is set to *1* and the *Reason* variable is set to *COMPLETED*.
   - **Failed** is processed when the Interpreter module is unable to complete the operator successfully. The *Result* variable is set to *-1* and the *Reason* variable is set to *FAILED*.
   - **Timeout** is processed if the resource operator is not completed within an optional specified time-out interval. The *Result* variable is set to *1* and the *Reason* variable is set to *TIMEOUT*.
   - **Custom Result** is processed when execution settings determine the result. The *Result* variable is set to 0 and the *Reason* variable is set to *CUSTOM*.
5. Double-click the Resource operator to configure options.

   The Dataset and Resource Properties palettes appear.

Define Resource Actions

You can set the action you want each Manage Resources operator to take. Possible actions include taking and freeing units or locking and unlocking resources.

Follow these steps:

1. In the Process Designer, double-click a Manage Resources operator.
2. In the Properties palette, click the **Add** button. You can create multiple actions for multiple resources within one resource operator.

   The Action Properties dialog appears.
3. In the Resources object field, specify the resource object that you want to use. You can either enter the full path to the object in the automation library or click the browse button to locate the object.
4. (Optional) Click Open to view or edit the resources in the object.
5. In the Resource name field, enter the name of the resource or an expression.

   Note: Both the Resource Path and Resource Name fields accept expressions. Enclose any literal strings between double quotation marks.
6. In the Action field, select the action that you want the resource to perform from the drop-down list:

   Take Units

   Takes the number of resource units specified in the *Amount* field.

   Free Units

   Makes the number of resources specified in the *Amount* field available.

   Lock Resource

   Locks the resource so other resource operators cannot take resource units or lock the resource. Actions can still free resource units that were taken before a resource was locked, but the freed units will not be available until after the resource is unlocked.

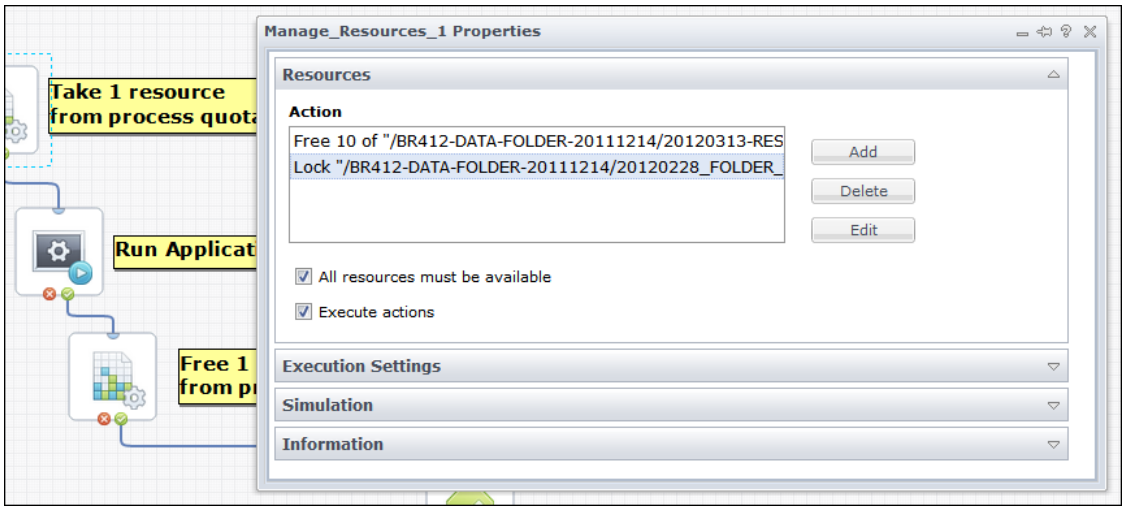
   Unlock Resource

   Unlocks a locked resource.
7. When taking or freeing resource units, specify a quantity in the Amount field.
8. Click Save and Close.

   The new action is added to the Action list.

   Notes: To remove an action from the Action list, click the action, and then click the **Delete** button. To edit an existing action, click the action and then click the **Edit** button. To view a selected action in a process that is not checked out, click the **View** button.

The following graphic shows two examples.



Check for and Respond to Unavailable Resources

The following properties determine what resources are executed by the operator and how it responds when resources are unavailable:

- *Time-out* field
- *All resources must be available* check box
- *Execute actions* check box

Specify a Time-Out Interval

Sometimes an action attempts to take more units of a resource than are available or tries to lock an already locked resource. In these situations, set the *Time-out* interval to determine how long the operator waits for resources to become available. After the time-out interval expires, the operator checks one final time if the resource is available before it times out. After a Resources operator times out, the process module processes the time-out exit link on the operator.

Follow these steps:

1. Open a process.
2. Drag a Manage Resources operator into the process.
3. Double-click the Manage Resources operator to view its properties.

4. In the Execution Settings palette, configure the options available in the Timeout group.

   No Timeout

   When checked, the operator waits indefinitely until all resources including all the actions listed under *Actions* can be executed and are available.

   When clear, a Duration or Target Date timeout setting is applied.

   Type

   Select Duration or Target Date. The operator waits before timing out. For example, to specify a time-out interval, select *Duration* and enter the number of seconds. When you specify a time-out duration of *0 seconds,* the operator does not wait. If resources are unavailable, it immediately times out. The operator succeeds only if resources are immediately available.

   Duration/Target Date and Time

   Specify the numeric quantity of seconds or a fixed date and time to serve as the timeout period.

   Action

   Select Continue, Reset, Abort, or Abandon. If you choose Abort, the operator processes the *Failed* exit link.

5. Save the process.

Specify Resource Availability and Action Settings

The following properties determine which resources the operator runs. They also determine how the operator responds when resources are unavailable. Set the *All resources must be available* check box to determine how CA Process Automation handles resource availability. Set the *Execute Actions* option to determine how CA Process Automation behaves with respect to resource availability and predefined actions.

Follow these steps:

1. Open a process.
2. Drag a Manage Resources operator into the process.
3. Double-click the Manage Resources operator to view its properties.

4. Set the *All resources must be available* check box:

   Checked

   Specifies that all resources that the actions listed in the *Action* field require must be available before any action is applied. The operator succeeds only if all the resources become available within the *Timeout* setting constraints.

   Clear

   Allows the operator to complete only those actions for which resources are available. The operator then succeeds if one or more of the listed actions is successfully executed within the *Timeout* setting constraints.

5. Set the *Execute Actions* check box:

   Checked

   The operator runs all the actions, if it can.

   Clear

   The operator does not run any actions. If resources are available within the *Timeout* constraints, the operator runs the *Successful* exit link without performing any actions.

6. Save the process.

Note: These settings can be used with a resource that is set to enable or disable a whole set of processes. Before they start their tasks, those processes check that there is no lock on the resource 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
- A process that monitors some internal or external condition in a loop

Check for Resource Availability without Executing Actions

Set the *Execute Actions* option to determine how CA Process Automation behaves with respect to resource availability and predefined actions.

Chapter 9: Calendars, Schedules, Tasks, and Triggers

In general, any process can be scheduled by a *Run Process* task in a *Schedule* object according to valid dates defined in a *Calendar* object. A *task* in a schedule specifies a selected operator to run on a specified touchpoint. *Triggers* allow external applications to start a process.

This section describes calendars, schedules, tasks, and triggers.

This section contains the following topics:

Calendars (see page 355)
Schedules (see page 370)
Task Management (see page 378)
Administer Triggers (see page 381)

Calendars

Calendars define rules for dates that are applied to tasks so that they run or do not run when you expect. For example, you can create a Calendar object named *LastOpenDayofMonth* and use it to schedule complete backups and monthly reports on the last available day of each month.

After defining *Calendar* objects, use them in *Schedule* objects to determine when tasks run. The scheduling of tasks or processes on certain days requires a Schedule object. Schedules coordinate times for tasks or processes with the valid dates defined by a calendar.

You can create multiple calendars and associate any single calendar with any number of scheduled tasks. The separation of calendars from schedules allows you to define common rules for dates that can be reused in many scheduling contexts. To change the rules (for example, those representing closed days) for all the tasks that use (include or exclude) a calendar, edit the calendar describing those dates. Schedules automatically apply any changes you make to a calendar.

Create a Calendar Object

You can create a Calendar object in any folder in the Library Browser.

Follow these steps:

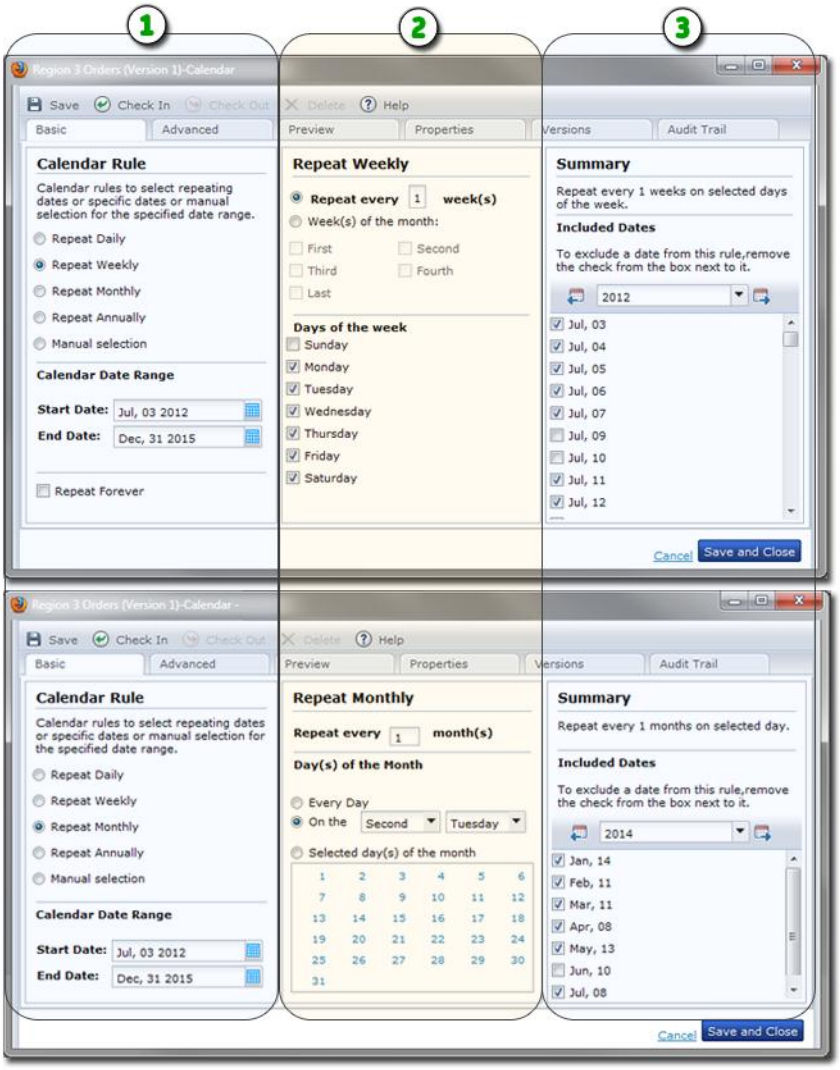
1. Click the Library tab.
2. Click a folder.
3. In the toolbar, click New, select Object, and then choose Calendar.

   A new Calendar object appears.
4. Click the Calendar name to rename it.
5. Double-click the calendar to edit it.

   The Calendar Designer opens.
6. Define calendar rules that the application uses to build a calendar of included and excluded dates. Choose one of the following:
   - Click the Basic tab to define basic rules.
   - Click the Advanced tab to define advanced rules.

The Basic Calendar Designer

The Calendar Designer opens when you edit a Calendar object. Use the Basic tab to define rules that generate included dates. You can then exclude certain dates. Another approach is to create a calendar specifically for excluded dates that you want to apply and manage separately.



Item:	Description:
1	**Calendar Rule Recurrence**: Select an option representing the recurrence pattern for the calendar dates that you want to define. Options include Daily, Weekly, Monthly, Annually, or your own manually selected dates. You can also define a date range or set the rule to repeat indefinitely.

Item:	Description:
2	**Detailed Settings**: This portion of the Basic tab bases its appearance on your Calendar Rule selection. For example, a weekly recurrence results in detailed settings for the days of the week and weeks of the month. As another example, a monthly recurrence shows settings for specific days of the month.
3	**Summary of Included and Excluded Dates**: This area lists all the days in your calendar rule. Clear the check box from a date to exclude it from the calendar.

The Advanced Calendar Designer

Advanced calendar rules are hierarchical tree structures. The Advanced tab for a Calendar object always shows three sets of calendar rules:

- Manually Included Dates
- Manually Excluded Dates
- Calendar Rule

These three root elements for a calendar rule are fixed. These three sets cannot be deleted. Define rules by building date conditions in the root elements. A calendar rule combines date operators (such as day of the month, month of the year, or week of the year) with logical operators (Intersection, Union, Exclusion, and Like). The Calendar Rule is actually the root Union operator for all Calendar rules.

Start building a calendar rule by dragging a rule arranged by icon under All Rules to the Calendar Rule set. You can also manually include or exclude dates.

After you place a rule under Calendar Rules, you configure its properties.

Properties for a date condition object in a calendar rule also have properties that you can expand to configure the date condition, such as the Month Interval properties.

For example, if the Month Interval properties sets Step to “3”, it specifies that every third month in a year is valid. Because the interval starts in January and ends in December, this three-month interval repeats throughout the year. This rule defines a condition for performing tasks each trimester.

The following task example shows this condition in a rule and adds a condition that specifies the fifth day of every month. The Intersection operator behaves as a logical And to combine these two conditions in a single rule that specifies the fifth day of every trimester.

Example: Create a rule specifying the fifth day of every trimester

1. Drag Intersection operator to Calendar Rule.

2. Drag Month Interval under Intersection.
3. Click the Month Interval operator. Define a Month Interval from January to December with the Step set to 3.
4. Drag a Day Interval operator under Intersection and specify an interval from 5 to 5 with the Step set to 1.
5. To save your changes to the working version of the Calendar object, click Save and Close.
6. To test the Calendar click the Preview tab.

You can continue to add additional operators and conditions to define a rule further. The logical hierarchy defines the order in which conditions are applied.

In terms of a Boolean equation, you can picture a Calendar object as enclosing parentheses around and applying Boolean constraints to subordinate operators and conditions.

- The Union operator applies a Boolean OR to subordinate operators and conditions.
- The Intersection operator applies a Boolean AND to subordinate operators and conditions.
- The Exclusion operator applies a Boolean NOT to an excluded condition.
- The Like operator inserts another calendar rule into an equation, so you can think of it as inserting a user-created Boolean function. A Calendar rule or combination of Calendar rules can handle virtually any scheduling problem.

Note that depending on the needs of a particular scheduling problem, there are multiple ways to build calendar rules. You can also define a set of dates in a separate calendar. You can use a Like operator to specify the calendar object, and add it directly under the Calendar Rule. You can also use a Union, Intersection, or Exclusion operator to include a Like operator in a calendar rule.

Calendar Rule Logical Operators

Use the four logical set operators to include and exclude dates in calendar rules. The *Union*, *Intersection*, and *Exclusion* operators provide a way to include and exclude dates represented by basic date conditions or by branching combinations of conditions and operators. The following list describes each logical operator:

Union

Indicates that one or more of the linked conditions must be satisfied for the combined condition to be satisfied.

Place one or more branches or basic conditions under this icon.

Intersection

Indicates that the linked conditions must all be satisfied for the combined condition to be satisfied.

Place one or more branches or basic conditions under this icon.

Exclusion

Indicates a basic condition or a branch to be excluded from a rule.

All dates are excluded that are not otherwise selected. It is therefore only useful to exclude days when they are selected by another part of the rule. For example, no purpose is served by excluding Tuesdays unless they are defined as valid days by other conditions and operators in a rule. So if a condition specifies the work week (Monday through Friday) as valid days, you could use the Exclusion operator to exclude Tuesdays from this set.

Expand the Exclusion operator to show the Included and Excluded branches.

Click a branch and then add a condition or operator to define included or excluded dates.This operator has two sets of branched arguments:

Included: One or more basic conditions or branches that represent dates to be included in the rule.

Excluded: One or more basic conditions or branches representing dates to be excluded from the dates defined by the Included set.

Like

Use the Like operator to use an existing set of dates defined by another calendar object in your rule. This operator has the following parameters:

Calendar Name: The name of the referenced calendar.

Delta: Shifts the valid dates defined by the referenced Calendar by the specified number of days. Enter a negative number to move the dates earlier, or a positive number to move the dates later.

Open Days: When checked, indicates that the delta or shift only applies to open days.

For example, a new calendar rule could reference another calendar specifying backup days with a delta of 1. The resulting condition in the new calendar rule specifies the day immediately following backup days.

Calendar Rule Date Operators

This section describes the elementary conditions on dates and their parameters. The conditions can be placed in the rule pane of the Calendar Designer. To select a date in any operator's properties, click the calendar icon to open the calendar viewer and select a date.

Dates List

Specifies individual dates. For example:

- March 1, 2014
- July 15, 2014
- September 23, 2015

Parameters

- A list of dates with years.
- To add a date, click the Add Date button.
- To delete a date in the list, select the date and then click the Delete Date button.
- Click the Move Up and Move Down buttons to reorder dates in the list.

Date Interval

Specifies a regular daily, weekly, or monthly interval in a range of dates from beginning to end.

For example, every week from March 1, 2015 to July 1, 2016.

Parameters

- **Beginning**: The starting date for the range.
- **End**: The ending date for the range.
- **Repeat Forever**: Check this box to ignore the End date and extend the interval indefinitely.
- **Step**: Indicates the quantity of units (days, weeks, or months) in each interval. For example, an interval with a unit of week and default Step of 1 occurs once in week 1, again in week 2, and a third time in week 3. When Step is set to 3, the interval occurs once in weeks 1 to 3, again in weeks 4 to 6, and a third time in weeks 7 to 9.
- **Unit**: Specifies the recurrence frequency or interval. Select Day, Week, or Month.

Date Without Year List

Specifies a list of explicit anniversary dates.

This condition is commonly used to specify holidays that fall on the same day every year. Examples include January 1st and December 25th.

Parameters

- A list of dates without years.
- To add a date, click the Add Parameter (+) button. Click the browse (...) button on the new parameter to add open the Select Date calendar control and select a date.
- To delete a date in the list, select the date and then click the Delete Parameter (x) button. You can click the Move Up and Move Down buttons to reorder dates in the list.

Date Without Year Interval

Specifies an anniversary interval of dates without a year.

For example, from March 21st to June 20th (for Spring).

Parameters

- **Beginning**: The starting date without a year for the interval.
- **End**: The ending date without a year for the interval.

Year Interval

Specifies an interval of years.

You can specify leap years by starting an interval on a leap year and specifying a step of 4 (such as 2000 to 2024 with a step of 4).

Parameters

- **Beginning**: The starting year for the interval.
- **End**: The ending year for the interval.
- **Step**: The number of years from one valid year to the next valid year.

Month Interval

Specifies one or more months in the year.

The first semester is specified with a range from 1 to 6 with a step of 1. The second semester is specified with a range from 7 to 12 with a step of 1.

Parameters

- **Beginning**: The starting month for the interval.
- **End**: The ending month for the interval.
- **Step**: The number of months from one valid month to the next valid month.

Week of the Month Interval

Specifies one or more weeks in the month.

CA Process Automation implements ISO standards for partial weeks. A week which intersects with a given month is considered to be part of the month if the Thursday of that week falls in the month.

For example, if June 1st is a Friday, the First week of the month starts on June 4th. If June 1st is a Wednesday, the first week of the month starts on May 30th.

It is possible to have the “first Monday of the month” not be “Monday of the first week of the month.” To define the former, it is simpler to combine “Day of the month” and “Day of the Week” conditions.

Parameters

Beginning: The starting week for the interval.

End: The ending week for the interval.

Step: The number of weeks from one valid week to the next valid week.

Reverse: Counting starts with the last week of the month and goes backwards.

Week of the Year Interval

Specifies one or more weeks in the year.

CA Process Automation implement ISO standards for partial weeks. A week which intersects with a given year is considered to be part of the year if the Thursday of that week falls in the year.

For example, if January 1st is a Friday, the First week of the year starts on January 4th. If January 1st is a Wednesday, the first week of the year starts on December 30th of the previous year.

It is therefore possible to have the “first Monday of the year” not be “Monday of the first week of the year.” To define the former, it is simpler to combine “Day of the year” and “Day of the Week” conditions.

Parameters

Beginning: The starting week for the interval.

End: The ending week for the interval.

Step: The number of weeks from one valid week to the next valid week.

Reverse: Counting starts with the last week of the year and goes backwards.

Day Interval

Specifies an interval of valid days (between 1 to 31) in a month with a starting day, an ending day, and a step.

You can also specify that the iteration start from the end of the month or that only open days are counted in each step. Open days are those days not specified by a condition or rule that closes or excludes dates.

For example, the last day of the month is specified by the interval beginning and ending with 1 with Reverse selected. The last weekday of the month would be specified when the Open check box is also selected and a Weekday Interval specifying Monday through Friday is added with an And operator.

Parameters

Beginning: The starting day for the interval.

End: The ending day for the interval.

Step: The number of days from one valid day to the next valid day.

Reverse: Counting in steps starts with the last day of the month and goes backwards.

Open Days: Counting in steps includes only open days when days are closed by a condition or rule.

Day of the Year Interval

Specifies an interval of valid days (between 1 and 366) in a year with a starting day, an ending day, and a step. The day 366 is valid on leap years.

You can also specify that the iteration start from the end of the year or that only open days are counted in each step. Open days are those days not specified by a condition or rule that closes or excludes dates.

For example, you can specify winter as the interval from December 21st to March 20th.

Or for a slightly more complicated example, to specify every 10th day throughout the entire year, you could use a range from 1 to 365 (or 366 for a leap) with a step of 1. You could specify the last ten open days of the year with a starting day of 1, an ending day of 10, with Reverse and Open selected.

Parameters

Beginning: The starting day for the interval.

End: The ending day for the interval.

Step: The number of days from one valid day to the next valid day.

Reverse: Counting in steps starts with the last day of the year and goes backwards.

Open: Counting in steps includes only open days.

Day of the Week Interval

Specifies one or more days of the week (from Monday through Sunday) as an interval with a starting day, an ending day, and a step.

For example, weekends are specified by the interval beginning on Saturday and ending on Sunday with a step of 1.

Parameters

Beginning: The starting day for the interval

End: The ending day for the interval.

Step: The number of days from one valid day to the next valid day.

Weekday of the Month

Specifies a weekday in an indexed week of a particular month. The week is indexed from either the beginning or the end of the month.

Parameters

Weekday: Specifies the day of the week.

Month: Specifies the month for which the week day is applicable.

Week Index: Specifies the index of the week for which the week day would be applicable. (Value can be 1 to 5 because in any month there cannot be more than 5 weeks)

Reverse: If you select this check box, the counting for the week index starts from the last week.

For example, if you select Monday as a weekday, September as a month, and 3 as a Week Index: in September, the third Monday is included in the calendar. If you selected the reverse check box, in September, the third Monday from the last is included in the calendar.

Weekday of the Year

Specifies a weekday in an indexed week of the year. The week is indexed from either the beginning or the end of the year.

Parameters

Weekday: Specifies the day of the week.

Week Index: Specifies the index of the week for which the week day is applicable. (Value can be 1 to 53 because in a year there cannot be more than 53 weeks)

Reverse: If you select this check box, the week index counting starts from the last week.

For example, if you select Monday as a weekday, 43 as the Week Index, the forty third Monday of the year is included in the calendar. If you selected the reverse check box then the forty third Monday from the last week is included in the calendar.

Add and Remove Calendar Dates Manually

You may sometimes require dates in a Calendar object that are not easily specified by a calendar rule. Similarly, a rule can include dates that for some reason you do not want in a calendar. You can use the Manually Included Dates and Manually Excluded Dates in the Selected Calendar Rules pane to add or remove selected dates manually.

To add or remove dates

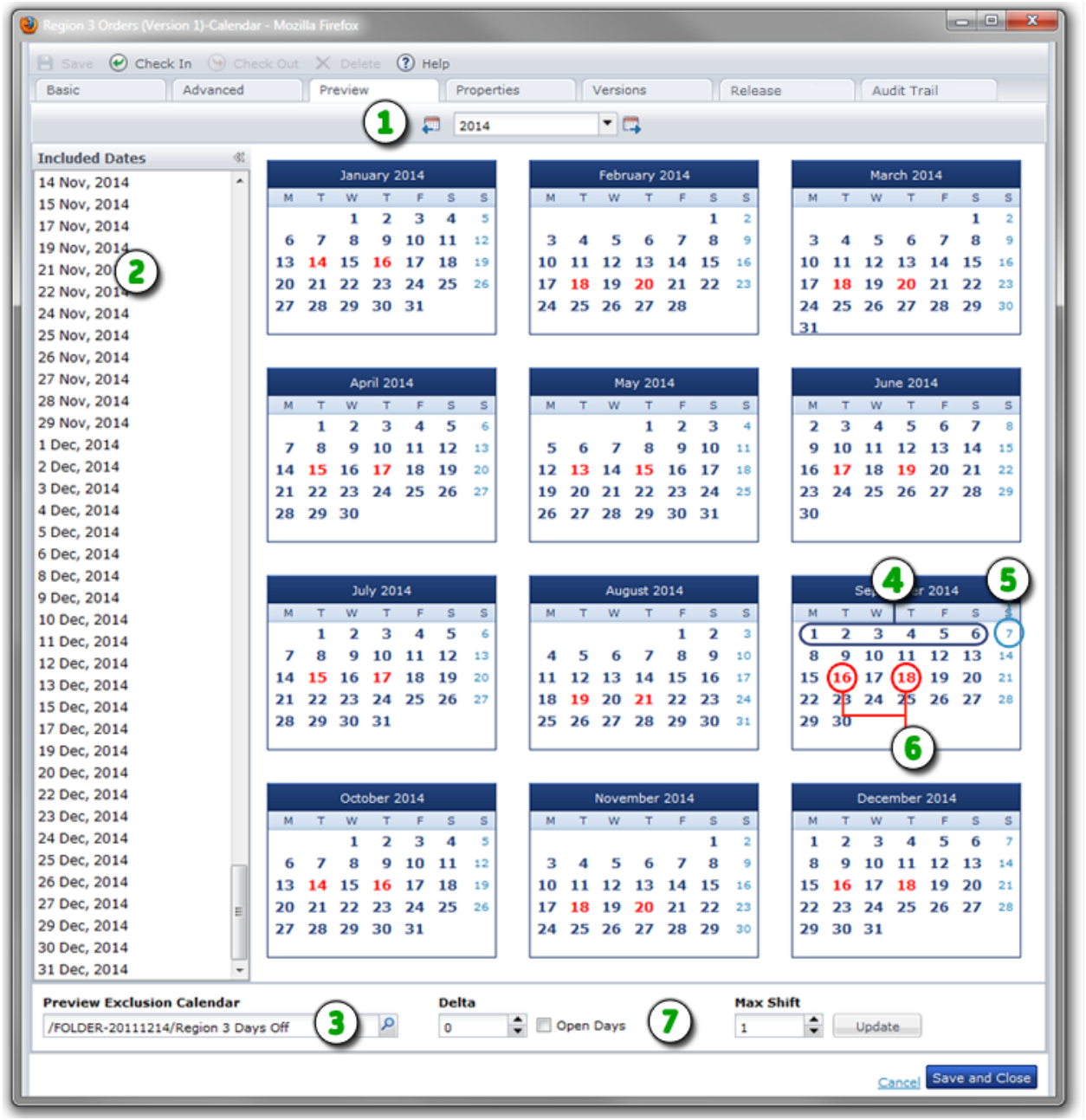
1. Open a calendar.
2. Click the Advanced tab.
3. Expand Manually Included Dates or Manually Excluded Dates.
4. In the month viewer in the Properties pane, right-click a selected date and click one of the include or exclude date commands on the shortcut menu.
   - To include specific dates for a particular year, click the Include Full Date Command.
   - To include anniversary dates for all years, click the Include Anniversary Date Command.
   - To exclude specific dates for a particular year, click the Exclude Full Date Command.
   - To exclude anniversary dates for all years, click the Exclude Anniversary Date Command.

   The Include Full Date command is only available when excluded dates are selected in the pane. The Exclude Full Date command is only available when included dates are selected in the pane.

The Calendar Designer: Preview Tab

Use the Preview tab to inspect the dates to include and exclude in a calendar.

You can also compare the dates in one calendar with dates in a second calendar. For example, define a standard work calendar that omits the holidays or vacation days defined in another *exclusion calendar*. You can preview how the exclusion calendar would affect the standard work calendar as you edit it.



Item:	Description:
1	**Preview Tab and Year**: After defining a calendar on the Basic or Advanced tab, click the Preview tab to view the dates. Click Previous Year, Next Year, or select a year.
2	**Included Dates**: This pane displays all the dates included in your calendar rule settings.
3	**Preview Exclusion Calendar**: (Optional) Select a separate calendar to display conflicts in bold red in the calendar preview.
4	**Included Dates**: The calendar preview displays dates included in your calendar with bold dark blue numbers.
5	**Excluded Dates**: The calendar preview displays dates that are manually or automatically omitted from the calendar rules with light blue numbers.
6	**Conflicting Dates**: The calendar preview displays dates that overlap or conflict with the dates defined by an optional exclusion calendar with bold red numbers.
7	**Conflict Resolution Fields**: Use the Delta field to specify the number of days an eligible date is shifted when it falls on an omitted or excluded date. A negative Delta value shifts forward (earlier), and a positive value shifts backward (later). When this value is zero (the default), the eligible date, normally included in the calendar rule, is marked in bold red and omitted. Select the Open Days check box to count only included days when shifting the schedule to avoid an excluded or omitted date. Open days are days that are not specified by a condition or rule that omits or excludes dates. If you clear the Open Days check box, a shifted date potentially can fall on another excluded or omitted day. Use the Max Shift field to define the maximum number of shifts or adjustments to allow if repeated shifts fall on closed days.

Exclude Calendars

Closed days are those days on which a group of scheduled tasks cannot be performed. Closed days can be specified in a calendar (for example, weekends are implicitly closed when a rule specifies weekdays) or in a separate vacation calendar. A vacation calendar is created with rules specifying valid dates, like any other calendar. Specifying a calendar as a vacation or exclude calendar closes out dates that would otherwise be defined as valid dates for performing tasks.

For example, certain tasks cannot be performed on company holidays. In this event, you create a calendar that specifies all company holidays. Then, for each task in a schedule you want to skip on company holidays, specify the company holiday calendar as the *Exclude Calendar*. The company holidays are then closed days for those tasks.

Schedules

Schedule objects configure when process or operator tasks run. Specify valid days in the schedule or by reference to previously defined *Calendar* objects. Schedule objects let you group, coordinate, and schedule when tasks run relative to organizational or architectural elements of an enterprise. For example:

- Applications
- Ownership
- Monitoring
- Maintenance
- Functional processes

Schedule objects specify:

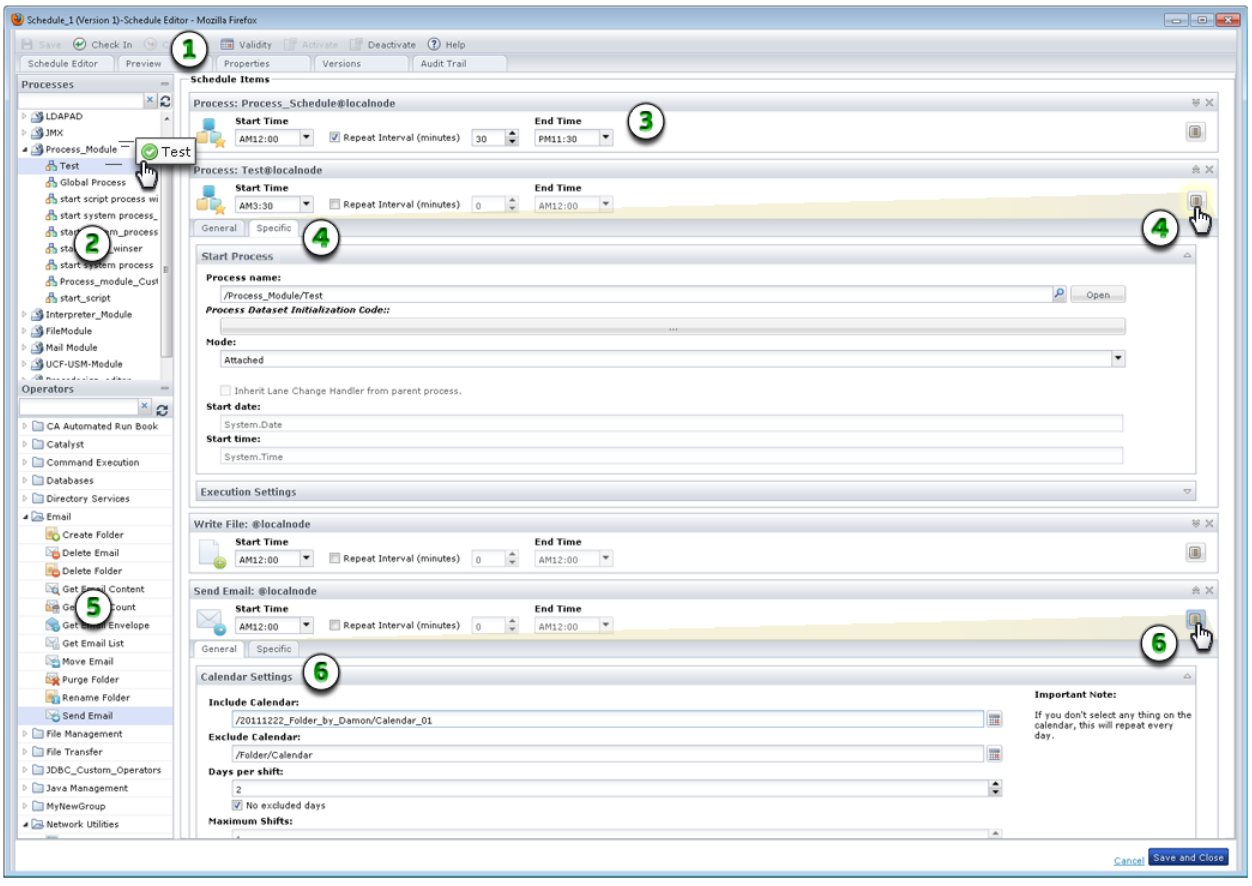
- The tasks (processes and operators) to run
- The time of day when each task starts
- The repeat interval for multiple occurrences of a task
- The days when tasks start (using a calendar, by specifying explicit days, or a combination of both)
- The days when tasks are not permitted to start (using vacation calendars and by specifying excluded days)
- The time of day when each task ends

You can create a schedule with or without specifying a calendar. Processes and other tasks that run every day or on specified days do not require a calendar object. To schedule dates using calendar rules:

1. Create calendar objects.
2. Specify the calendars in schedule objects.

Calendars define rules that specify valid dates for performing tasks and closed days on which the tasks cannot run. Schedule objects associate operationally-related tasks with a calendar and specify the times when the tasks run on the valid days the calendar rules define.

Equation 2: This graphic highlights the functions of the Schedules page.



Item:	Description:
1	**Schedule Editor**: When you open a schedule from the Library Browser, the Schedule dialog appears. Use the toolbar to activate, set validity, check in, or save the schedule. Click any of the tabs.

Item:	Description:
2	**Processes Pane**: Select the processes to include and drag them to the Schedule Items list.
3	**Scheduled Process**: Set the duration and frequency for running the process in a single day from 12:00 a.m. to 11:45 p.m.
4	**Process Properties**: Click Properties to view process properties on the General and Specific tabs.
5	**Operators Pane**: Select the operators to include and drag them to the Schedule Items list.
6	**Operator Properties**: Click Properties to view operator properties on the General and Specific tabs.

Create a Schedule Object

To create any automation object in CA Process Automation refer to Create an Object (see page 61).

Schedule Process and Operator Tasks

You can configure a process or operator to run as a single task or a series of tasks in a schedule. The difference between specifying an operator as a task in a schedule and specifying an operator in a process is that the scheduled operator starts at a scheduled time rather than as a step in a process. You can also schedule any process to start by using a Start Process operator in a schedule.

Follow these steps:

1. In the Library, double-click a Schedule object.
2. In the Schedule Editor, identify the processes and operators you want to schedule.
3. Expand the Processes or Operators pane and drag any available process or operator to the Schedule Items list. You can also right-click a process or operator and choose Add.
4. For each item, complete the following fields:

   Start Time

   The starting time for a task to begin running on scheduled days.

   Repeat Interval (minutes)

   Indicates whether a task runs repeatedly between the start and end times, and if so, how frequently. For example, every 2 minutes, or every 120 minutes (2 hours). Each time the task is repeated, a new instance of the task is created. Specify the number of minutes from one start to the next in the adjacent (minutes) field. For example, the value *120* in the minutes box repeats a task every two hours.

   Make sure that the End Time is later than the last time that you want the task to repeat. For example, you configure a task as follows:

   - Start Time is 00:00
   - Repeat Interval check box is selected
   - The (minutes) field is set to 120 minutes
   - End Time is 16:00

   These settings schedule the process or operator to start for the first time at 12:00 a.m., repeat every two hours, and run for the last time before 4:00 p.m.

   End Time

   For a repeating task, the time that the task stops repeating on any scheduled day.
5. At this point, you have scheduled a task to repeat at the specified interval every day from the specified start time to the specified end time. Continue with the remaining steps only if you need to make detailed changes to schedule dates or properties.
6. Expand an item group or click Properties.

The General and Specific tabs appear.

7. On the General tab:
   a. Expand Calendar Settings and select from the following fields:
      - **Only Manually Selected**: Only consider manually scheduled dates. When no Calendar is specified in a scheduled item, the item is considered to be scheduled every day, except when this option is selected. When this option is selected, run dates need to be explicitly scheduled under Manually Included or Manually Excluded or both.
      - **Include Calendar**: A Calendar object that schedules dates to run the task.
      - **Exclude Calendar**: A vacation Calendar object that specifies closed days on which tasks are not run. The full path of the Calendar used to specify closed days (those on which a task may not be scheduled). There are no closed days when no vacation Calendar is specified here.
      - **Days per shift**: Activates rules that shift the dates that tasks are run when a date specified by the Calendar object falls on a closed date. 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 the a scheduled date to avoid a closed date.
      - **Maximum Shifts**: When a task is rescheduled because the original scheduled date falls on a closed day, it is possible that the new date also falls on a closed date. This parameter defines the maximum number of shifts that are allowed.This situation does not occur when the No Excluded Days check box is selected.
   b. Expand **Manually Included Dates** to list individual dates for inclusion in the schedule.
   c. Expand **Manually Excluded Dates** to list individual dates to exclude, even to exclude dates from previously specified *include* calendars.
   d. Expand **Task Name** to enter a more meaningful name for the task.
8. Click the Specific tab and expand the groups of fields that vary by operator or process. For a process, the Start Process and Execution Settings groups appear. For an operator, Execution Settings and other parameters appear. Configure the fields.
9. To set the valid date range for the entire schedule, click Validity. Specify a date far in the future such as 12/31/2050 to continue evaluating dates indefinitely or until the schedule is manually deactivated.
10. To delete a task, click X along the right edge.
11. Click Activate to initiate your scheduled items.
12. Click Check In and then close the Schedule dialog.

13. Monitor scheduled items on the Operations tab.

Preview All Occurrences of a Scheduled Task

You can preview the scheduled task occurrences for a specific day. When you configure a task to repeat on a specific day, the application tracks each occurrence of the task. For example, if a task repeats every 10 minutes for half an hour, the preview shows three occurrences. Use this procedure to plan for a future date or to view the results of tasks that were scheduled on a specific past date.

Follow these steps:

1. On the Schedule dialog, click the Schedule Editor tab to configure the list of scheduled items for the entire period of validity.
2. Click the Preview tab. If your task repeats, multiple occurrences on the same day appear on the Preview tab.
3. On the Preview tab:
   a. In the Preview Date field, select a date in the valid range for the schedule.
   b. From the All Nodes drop-down list, select All Nodes or a specific Orchestrator touchpoint.
   c. Select the Current or Archived options.
   d. Click Refresh.

   The application evaluates calendar rules and the validity period that is associated with tasks before it lists the appropriate occurrences.

   - When you select a past date, the Preview tab includes tasks that were started, their state, their start time, and their end time.
   - When you select the current date, the Preview tab indicates whether occurrences have started and their start time, state, and end time.
   - When you select a future date, the Preview tab includes all occurrences for the selected date if the schedule is active on the selected touchpoint.
4. Double-click an occurrence to view a read-only Properties pane that shows the task configuration settings.
5. Click the Schedule Editor tab to make changes to the schedule.
6. Click Validity to set the valid date range for the entire schedule.
7. Click Activate to initiate your scheduled items.
8. Click Check In and then close the Schedule dialog.
9. Monitor scheduled items on the Operations tab.

Using Schedules

You must activate and check in a schedule to use it. You can activate a schedule on the particular orchestrator touchpoint on which it resides. When a schedule is active on multiple touchpoints, it behaves as a separate instance on each touchpoint. This allows you to schedule the same tasks on multiple computers simultaneously. Examples of these types of tasks include log tidying, software installations, updates, and file backups.

Note: When you run a schedule, the schedule and any operators in the schedule use only the checked-in copies of objects they reference.

Monitor Active Schedules

After you activate a schedule, you can monitor it using the Active Schedules link on the Operations page.

Follow these steps:

1. Click the Operations tab.
2. On the Operations page, expand the Links pane.
3. Click Active Schedules.
4. In the toolbar, select an orchestrator and environment and then click Refresh.
5. In the Active Schedules table, double-click a schedule.

   The Schedule dialog opens.
6. On the Schedule dialog:
   a. Edit the scheduled items. See Schedule Process and Operator Tasks (see page 373).
   b. In the toolbar, click Activate to enable the schedule.
   c. In the toolbar, click Deactivate to disable the schedule. You can also deactivate a schedule on the Operations page.

   Note: Your assigned permissions determine whether you can list, open, or edit a specific schedule object. A content administrator or automation object owner can change permissions on an automation object.
7. Click Check In or Save and Close.

Monitor All Occurrences of All Scheduled Tasks

You can monitor the scheduled occurrences of all tasks for a specific day. When you configure a task to repeat multiple times daily for multiple days, CA Process Automation tracks each occurrence of the task. For example, if a task repeats every 10 minutes for half an hour (3 occurrences) every day for 1 year, CA Process Automation tracks 1,095 occurrences. Use this procedure to plan for a future date or to view the results of all tasks that were scheduled for a specific past date.

Follow these steps:

1. Click the Operations tab.
2. Expand the Links pane on the Operations page.
3. Click Global Schedules.
4. From the toolbar, select an Orchestrator and environment, then click Refresh.
5. In the Global Schedules table:
   a. In the Preview Date field, select a date in the valid range for the schedule.
   b. From the Nodes drop-down list, select All Nodes or a specific Orchestrator touchpoint.
   c. Select the Current or Archived option.
   d. Click Refresh.

   CA Process Automation evaluates calendar rules and the validity period that is associated with tasks before displaying the appropriate occurrences in the list. If your task repeats, multiple occurrences on the same day are displayed.

   - When you select a past date, the list includes tasks that were started, their state, their start time, and their end time.
   - When you select the current date, the list indicates whether occurrences have started and their start time, state, and end time.
   - When you select a future date, the list includes all occurrences for the selected date if the schedule is active on the selected touchpoint.
6. To view a read-only Properties window that shows the configuration settings for the scheduled task, double-click the appropriate occurrence.
7. In the toolbar:
   - Click Dataset to view the process or operator dataset.
   - Click On Hold or Release Hold to hold or release the occurrence.
   - Click Reset to restart an occurrence.

Task Management

You can manage CA Process Automation tasks by designing user interaction forms to enable users to control tasks or provide custom input. On the Operations page Task List or the Home page My Tasks list, right-click a task to:

Reply

Modify the process in some way using a form in the Reply dialog. For example, you can change field parameters or values before clicking Finish to complete the task.

Take

Temporarily assign yourself complete ownership and responsibility for the pending task.

Return

For tasks with a status of Taken only, releases your exclusive ownership of the task. The task is returned to its designated assignee or delegate users or groups.

Delegate

Assign the task to a secondary user or group, known as a delegate.

Transfer

Assign the task to a different assignee.

Open Process Instance

View the task in the context of the parent process instance and review the process design.

Refresh

Update the task list with the latest changes from all users and system activity.

Properties

View more information about the task including its description, due date, status, and its assignees and delegates.

Assign a Task to a User

To create a task, create a process using the *Assign User Task* operator, then start it.

You can specify the following attributes when creating a task:

1. Open a process in the Process Designer.
2. In the Operators palette, expand Process Control or search for the Assign User Task operator.
3. Drag the Assign User Task operator to your process.

4. Double-click the Assign User Task operator.
5. In the Properties palette:
   a. Expand *Assignees*, and enter the users and groups to assign to this task.
   b. Expand *Transfer/Delegate Filters* to permit delegation of the task, which is restricted to the specified users or groups.
   c. Expand *User Task* and complete the following fields:

      Title

      The title of the task.

      Description

      A description for the task.

      Interaction Request Form

      The library path to the interaction request form.

      Form Data Initialization Code

      You can write JavaScript to populate this field.

      Show approval page

      A check box that specifies if an approval screen must be displayed at the end of the interaction request form. The user working on the task can approve or reject the task using the approval page.

   d. Expand the following common operator property groups to specify any designer information about the Assign User Task operator:
      - Execution Settings
      - Simulation
      - Information
6. In the Designer toolbar, click Save.

The Task List

You can work with tasks on either the Home page or the Operations page. The Home page displays only the Status, Title, Description, and Due Date fields in the convenient *My Tasks* table. Use the Task List on the Operations page to view the most detailed information about tasks.

Tasks originate from processes that include an Assign User Task operator and an Interaction Request Form. You can sort the Task List in ascending/descending order by clicking the column headers. Use the *Status* column to determine if a task is pending, completed, approved, rejected, or taken.

On the Operations page, you can filter the task list to show the following:

- Only your tasks (*My Tasks*, the tasks that are assigned to the current user)
- Only the tasks assigned to any *groups* that you belong to
- *All* tasks

The Task List on the Operations page displays the following field columns for each task:

- Task ID
- Title
- Description
- Start Time
- Due Date
- Completion Date
- Status
- Assignees
- Delegates

Administer Triggers

You can control processes with external applications using any of the following methods:

- Triggers
- Web services (SOAP)
- Command line utility
- Scripts

SOAP calls are recommended over triggers because Web services are more robust. Applications that cannot make SOAP calls can use triggers as an alternative.

Triggers allow external applications to start a process in CA Process Automation. A trigger invokes the CA Process Automation process that is defined in XML content or in an SNMP trap. The XML content can be delivered to the configured file location or to the configured email address. SNMP trap content can be sent in an OID matching a configured regular expression. CA Process Automation listens for incoming SNMP traps on the configured SNMP trap port, 162 by default.

Whenever you start a process, begin an operation such as run the Start Process operator, or use a trigger or SOAP call, you are acting on behalf of some user or owner. For triggers or SOAP calls, information about the content owner is in the payload or messages. This information determines the versions of automation objects that are run:

- If you check out a process and then run, call, or trigger it (you are both the content owner and initiator), CA Process Automation uses your private checked-out version.
- Otherwise, CA Process Automation uses the current versions of the automation objects. This includes processes that are not checked out or checked out by another user.

You can run and verify your own checked-out version before checking the objects back in or making them current.

Controlling Processes from an External Application with SOAP Calls

The CA Process Automation Orchestrator exposes Web services that allow external applications to start and control processes in a library. SOAP calls require valid XML. The Web services methods and parameters exposed are described in the WSDL. For details, retrieve the WSDL from the appropriate domain URL, depending on whether CA Process Automation supports secure communication and is clustered. In the following examples, *load_balancer_hostname* is the host name or IP address for the Apache load balancer.

- Secure and unclustered:
  `https://<DomainOrchestrator_hostname>:8443/itpam/soap?wsdl`
- Unsecure and unclustered:
  `http://<DomainOrchestrator_hostname>:8080/itpam/soap?wsdl`
- Secure and clustered:
  `https://<load_balancer_hostname>:<Apache_secure_port>/itpam/soap?wsdl`

  Note: The secure port of Apache is typically 443.
- Unsecure and clustered:
  `http://<load_balancer_hostname>:<Apache_unsecure_port>/itpam/soap?wsdl`

  Note: The unsecure port of Apache is typically 80.

For sample scripts that use SOAP calls to the CA Process Automation Orchestrator to start processes, navigate to the following folder:

`<install_dir>/server/c2o/.c2orepository/public/scripts/trigger`

Note: The <install_dir> path is typically C:\Program Files\CA\PAM.

The Java subfolder contains a Java-based tool and all the resources the tool requires to start CA Process Automation processes remotely using SOAP. The path to the Java subfolder follows:

`<install_dir>/server/c2o/.c2orepository/public/scripts/trigger/java`

How File and Mail Triggers Work

This topic provides a description of the processing sequence for triggers. It uses file and mail triggers as examples.

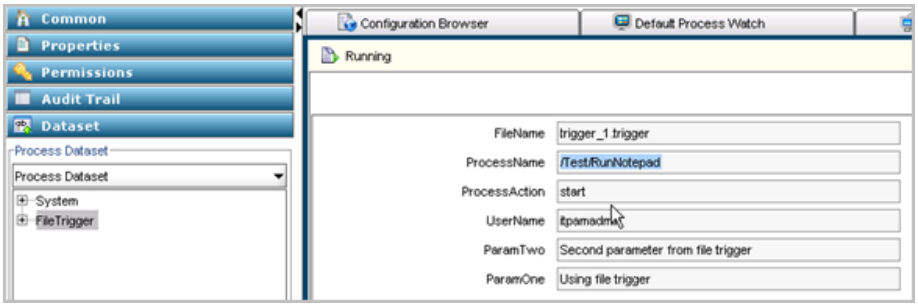
File and Mail Trigger Example:

1. At the configured frequency, CA Process Automation searches for new content in the configured folder and the configured email account.
2. If a new file object or mail object is found, CA Process Automation attempts to run the process, based on the XML content.

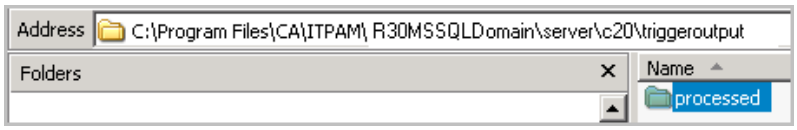
   An illustration of valid XML content for the file trigger follows:

```
<c2oflow version="1.0">
    <flow name="/Test/RunNotepad" action="start"> <!-- Full path of the process -->
    <auth>
        <user>itpamadmin</user>       <!-- ITPAM Username -->
        <password>itpamdemo</password>  <!-- ITPAM Password -->
    </auth>
    <options>
        <!-- Optional parameters for delayed execution
        <startDate></startDate>
        <startTime></startTime>
        -->
    </options>
    <params>
        <!-- Process initialization parameters, if needed -->
        <param name="ParamOne">Using file trigger</param>
        <param name="ParamTwo">Second parameter from file trigger</param>
    </params>
    </flow>
</c2oflow>
```

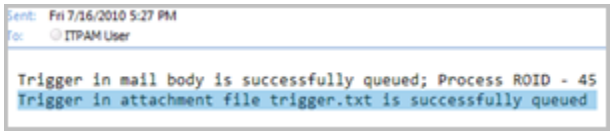
3. When the trigger executes the process specified in the trigger instance, the process dataset is populated with the values contained in the XML. The following example demonstrates how the values in the XML content are used to populate the file trigger process dataset.



4. The results are posted to the processed folder in the configured path.



5. Emails received at the configured email account are processed in much the same way as XML content received in files. In addition confirmation emails are sent, where the content states whether the XML content was found in the body of the email or in an attachment. The following example shows both messages:



Monitor the FileTrigger Dataset of a Process Started by a File Trigger

When valid XML content in a file triggers a process instance, you can monitor the file trigger process dataset in the Process Watch. Use the following field descriptions to interpret the displayed values.

FileName

The name of the file with the content that triggered the process.

<additional_parameters>

Additional parameters that are passed under the <params> tag in the triggering XML content of the file.

Monitor the SMTP Dataset of a Process Started by a Mail Trigger

When valid XML content in email triggers a process instance, you can monitor the SMTP process dataset with Process Watch. Use the following field descriptions to interpret the displayed values.

SenderAddress

The email address of the account from which the triggering email is sent.

SentDate

The date and time when the email was sent.

ReceivedDate

The date and time when the server received the email.

MailSubject

Subject of the triggering mail.

MessageNumber

Message number of the triggering mail at the time when the process was triggered.

Note: This number can change for the same mail, if messages are deleted or moved from the Inbox.

MessageID

Unique ID of the mail in the server.

MailBody

The body of the email message in these cases:

- When valid XML content in an attachment triggers the process.
- When the default trigger process is started, that is, when no valid XML content is found in either the email body or attachment.

Note: This string value is truncated to the first 64k characters in the mail body.

MailAttachments

A ValueMap array type variable which holds ValueMaps with the following information about the attachments:

a. contentType: Attachment content type.

b. contentID: Attachment contentID if present.

c. fileURL: URL from which attachment can be viewed or downloaded.

d. name: Name of the attachment.

e. attachmentID: Unique ID for this attachment. This ID can be passed to JavaScript system functions.

Note: See the *Content Designer Reference* for details about the JavaScript system functions.

<additional parameters>

These parameters, passed under the <params> tag in the triggering XML content, exist only when valid XML content in the mail body or attachment starts the CA Process Automation process.

XML Content Format for File and Mail Triggers

External applications that use file or mail triggers to start CA Process Automation processes must create input in a valid XML format. XML content can be written to the body of an email or sent as an attachment. If the XML is copied to the email body, it can contain no more than what is required to trigger a process. For file triggers, the triggering file must include the entire content.

An example of valid XML format follows:

```
<c2oflow version="1.0">
        <flow name="/Test/RunNotepad" action="start">  <!-- Full path of the process -->
        CA AuthMinder
                <user>pamadmin</user>         <!-- CA Process Automation Username  -->
                <password>pamadmin</password> <!-- CA Process Automation Password  -->
        </auth>
        <options>                       <!-- Optional parameters  for delayed execution -->
                <startDate></startDate>  <!—Start Date in [MM/dd/yyyy] format  -->
                <startTime></startTime>  <!-- Start Time in [HH:mm] format; HH in 24 hrs  -->
        </options>
        <params>                      <!-- Process initialization parameters, if needed  -->
                <param name="ParamOne">Using file trigger</param>
                <param name="ParamTwo">Second parameter from file trigger</param>
        </params>
        </flow>
</c2oflow>
```

SNMP Trap Input Considerations

CA Process Automation supports SNMPv1 and SNMPv2 traps; however, it does not process SNMPv3 traps. When a network device or an enterprise application sends an SNMPv1 or SNMPv2 trap that CA Process Automation detects on the configured port, CA Process Automation processes the content.

Change the SNMP Traps Listener Port

By default, CA Process Automation listens on port 162 for SNMP traps designed to start CA Process Automation processes. If you have closed port 162 at your site and configured an alternative port, change the CA Process Automation configuration for this port in the OasisConfig.properties file. Then restart the Orchestrator service.

You can change the port on which CA Process Automation listens for SNMP traps.

Follow these steps:

1. Log on to the server on which the Domain Orchestrator is configured.
2. Navigate to the following folder or directory:

   install_dir/server/c2o/.config/
3. Open the OasisConfig.properties file.
4. Change the value in the following line from 162 to the port number you are using for SNMP traps.

   `oasis.snmptrigger.service.port=162`
5. Save the file.
6. Restart the Orchestrator service.
   a. Stop the Orchestrator.
   b. Start the Orchestrator.

   As soon as the service restarts, CA Process Automation begins listening on the port you configured. CA Process Automation listens for new SNMP traps that meet the criteria configured in the SNMP trigger.

Monitor the SNMP Dataset of a Process Started by an SNMP Trap Trigger

When an SNMP trap triggers a process instance, you can monitor the SNMP process dataset with Process Watch. Use the following field descriptions to interpret the displayed values.

SenderAddress

IP address of the source.

AgentIPAddress

IP address of the SNMP agent, if available in the trap.

SNMPVersion

Version of the SNMP trap.

ErrorIndex

Error Index of the trap.

AgentUptime

Uptime of the agent sending the trap.

EnterpriseOID

Object identifier (OID) of the managed object that generated the SNMP trap.

PayloadOIDs

Object IDs present in the payload of the trap. The payload object IDs represent a CA Process Automation string array variable.

PayloadValues

Values in the payload that correspond to the values in *PayloadOIDs*. This data is also a CA Process Automation string array variable.

Note: If there are several filters, the first match is processed.

Chapter 10: Running, Testing, and Debugging Processes

This chapter describes how to run, test, and debug processes interactively during development. The same methods can be used to run processes in a production environment.

When you want to run a process, you can initiate it using any of the following methods:

- Manual process initiation
- Start Request Form
- Call from other processes
- Trigger using external applications, FTP, SOAP calls, SNMP traps, or SMTP (email) messages

The Workflow module on an orchestrator runs processes. When you start a process on an orchestrator, the Workflow module creates and runs a copy of the process object in the orchestrator automation library. This running copy is an *instance* of the process. The Workflow module creates a separate instance of a process each time you start a process (or another process or application starts a process).

You can open, view, and work with an instance of a process while it is running or after it finishes. Changes made to an instance of a process affect only that instance and do not affect the original process object stored in an automation library.

When a process starts, it connects to the correct agent or orchestrator modules on managed network computers. A process performs its designated operator functionality, tests conditions, and exercises dependencies. When error conditions arise, a process performs corrective actions and notifies operators and administrators when necessary. An administrator can use the Application Monitor to monitor running processes and to perform corrective actions.

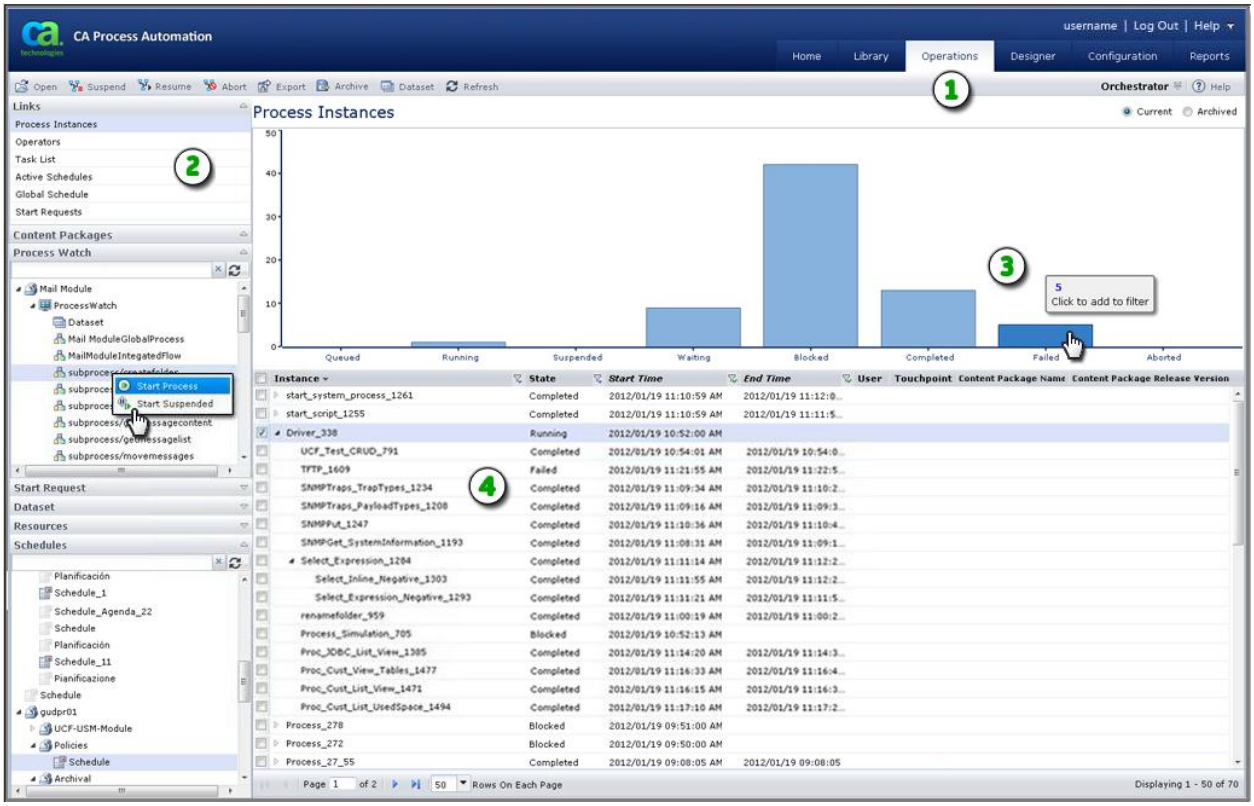
This section contains the following topics:

The Operations Page (see page 390)
Execution Rules (see page 396)
Runtime Security (see page 397)
Exception Handling (see page 399)
Run Processes Interactively (see page 404)
Process States (see page 408)
Debug a Process (see page 408)
Control a Process Branch (see page 412)
Simulate Processing of Operators (see page 415)

The Operations Page

The Operations page has the following functions:

- Monitors selected tasks and system elements that CA Process Automation manages
- Displays the properties of imported content packages under the Content Packages palette
- Displays all process activities for a selected automation library, including:
  - Process instances
  - Schedules
  - Module invocations
  - Datasets
  - User prompts
  - Resources



Item:	Description:

Item:	Description:
1	**Operations Tab**: Click this tab to navigate to the Operations page, a high-level automation dashboard. From the toolbar, you can select an Orchestrator, open a process instance, and control the process instance.
2	**Links Pane**: Select a link to view the associated items. For example, click Process Instances to view all processes by state. Expand a group to browse folders and select an instance. Some objects include a shortcut menu of commands that you can right-click to access. For example, the illustration shows that the user right-clicked a Process Watch object subprocess and clicked Start Process in the shortcut menu.
3	**Chart Area**: Point to a bar in the chart area to display the number of items that match the associated state. Click a bar to display only items that match the associated state.
4	**Operations Table or List**: Depending on your selection from the Links pane, this area displays the resulting data in a table or list.

Filters for Process Instances

You can apply the following filters to a Touchpoint Manager shortcut in a process watch object to define the objects that the shortcut displays:

All Instances

Displays all the process instances in the current Orchestrator. All instances display only if all filters are cleared or all filters are selected.

Queued

Displays all instances that are in a queued state and waiting to run.

Running

Displays running process instances on the touchpoint. This filter does not list Waiting, Suspended, or Breakpoint suspended instances.

Suspended

Displays process instances that a user or the application on the touchpoint currently have suspended.

Waiting

Displays process instances that have an inactive run state, such as processes where all active operators are waiting for an external asynchronous event. The event could be a user interaction, target date and time, or other long-running operation.

Blocked

Displays process instances that are blocked because there is no available path. A blocked process instance can complete after a user suspends the process and provides a valid path. The blocked state reflects the following situations:

- The process instance requires user intervention.
- A process instance cannot proceed because of an unexpected condition.

Completed

Displays all process instances on the touchpoint that completed without issues.

Failed

Displays all process instances on the touchpoint that ended without completing.

Aborted

Displays all process instances on the touchpoint that ended abnormally. To identify processes that have issues quickly, examine this folder. To troubleshoot the process, open the instance that failed.

Filter Objects Displayed by a Shortcut

The Process Watch includes a shortcut for each of the filters. You can use the following procedure to add filters to the Process Watch.

Follow these steps:

1. Open the Process Watch object: select a Library folder and double-click the Process Watch object.
2. On the Process Watch tab, select a process from a Library folder in the left pane, then click Add New in the toolbar.

   The selected object is added as a shortcut in the Process Watch.
3. Expand the process object shortcut and select the filters.
4. Save the changes.

To view the object instance for the selected filters, click the Operations tab and expand the Process Watch palette. You can select the new shortcut from the location you created the shortcut in. Corresponding instances are displayed for the process object, depending on the filters that you specified.

Process Watch Objects

A Process Watch object provides an easy way to monitor the status of other automation objects. You create Process Watch objects in the Library Browser. You can add objects (technically pointers to objects) from multiple domains, touchpoints, orchestrators, and other libraries. When you view the Process Watch object on the Operations page, it displays the current state of the monitored objects.

Create a New Process Watch Object

Define a new Process Watch object in the Library Browser and monitor the status of each object that it includes on the Operations page.

Note: To set rights and perform other object tasks, the best practice is to organize the objects in folders. Do not create objects at the root level because you cannot manage such objects as a group.

Follow these steps:

1. Click the Library tab.
2. In the Library folders pane
   a. Select an orchestrator in the design or production environment.
   b. Select a folder.
3. In the toolbar, click New and select Process Watch.

   A new process watch object appears in the selected folder.
4. In the Name field, enter a name for the process watch.
5. Double-click the new process watch.

   The Process Watch opens in a browser window.
6. In the left pane
   a. Select the domain, environment, orchestrator, touchpoint, host group, and folder for the object you want to add to the process watch.
   b. Select the object that you want to include in the process watch.
7. In the toolbar, click Add New.

   The selected object is added to the process watch. The name, object type, reference path, mode, and description of the selected object appears.
8. If you added a process, select one or more States that you want to display in process watch, for example, queued, suspended, and blocked.
9. Click Save.

Monitor Objects from within a Process Watch

You can open an instance of a process shown in the Process Watch window Details pane. Opening a process instance lets you closely monitor or edit the execution of the object instance. Your changes to an object instance affect only the execution of that instance; they do not change the object definition in the Library.

Note: A CA Process Automation user must have sufficient permissions to view or edit an object in the Process Watch window.

To save changes to an object instance, right-click the process instance in Process Watch, and click Export.

Note: For more information about working with process instances in Process Watch, see the *Content Administrator Guide*.

Extended Relative Path Support

An automation solution developed in CA Process Automation consists of various automation object types that can include processes, datasets, start request forms, and interaction request forms.

CA Process Automation lets the users move an automation solution from any level throughout the library hierarchy. For example, users can move objects from one folder to another folder or they can move the automation objects to a different Domain library. When you move the automation solutions through a library, the root path of the objects changes. To move an automation solution without breaking or changing the relationship between the objects despite the root folder change, use the Extended relative path option.

Note: CA Process Automation provides Relative and Absolute options in the drop-down list of the MODE column for the object shortcut added in Process Watch and Package.

Content Package Objects

The content package automation object bundles the CA Process Automation objects that the content designer exported into a different environment. The content designer adds the automation objects in a folder and exports the folder as a content package. When you view the content package object on the Operations page, it displays the current state of the imported objects.

More Information

Content Package Objects (see page 432)

Monitor Objects from within a Process Watch

You can open an instance of an object from the Content Packages window Details pane. Opening an object lets you closely monitor the object instance while the instance runs. Your changes to an object instance affect only how that instance runs; the changes do not modify the object definition in the Library.

Note: A CA Process Automation user must have sufficient permissions to view or edit an object in the Content Package window.

You can perform the following actions to an object in a content package:

- Start or suspend a process
- Start a start request form
- Make a custom operator available or unavailable
- Activate or deactivate a schedule

Execution Rules

Operators in a process can have multiple entry and exit links. An entry link serves as an execution order and invokes the operator. Each exit link corresponds to a particular outcome of the operator. Operators have predefined exit links (such as Aborted, Completed, Failed, or Successful). Some operators also allow you to use a Boolean expression to define a custom exit link based on the results and the value of variables accessible to operators in a process.

The execution rules of a process are as follows:

- Start operators in the main flow can have an entry link. If it has an entry link, it cannot have an exit link and act as a Reset operator (for example, used to reset a complete process).
- Stop operators have no exit and complete the execution of a process.
- All exit links with Boolean conditions evaluated as True are enabled and lead to activation of subsequent exit operators. Default exit links are mutually exclusive with one another. All custom links for which the Boolean expression evaluates to true are enabled and lead to subsequent exit operators.
- Operators (other than recapitalized operators in a looped branch of a process) are processed only once during execution of a branch of a process. When a link from a completed operator leads to an operator that has already been activated, then the activated operator is not processed a second time. After activation, the processed link is unavailable to subsequent processing of a process.

  Some operators support looped processing, in which the Workflow module executes the operator either a specified number of times or indefinitely. The exit conditions and the connecting links from the operator are evaluated only when the loop is terminated. The Loop operator further allows you to apply looped processing and its exit conditions to an embedded sequence of operators.
- Break links interrupt execution of a loop in operators that support the use of looped processing.

Runtime Security

The optional *Runtime Security* feature, when enabled, helps verify the identity of the user who is running the secure process or schedule. The user for any process is either the owner or the one calling the process. The user for any schedule is always the owner. The *caller user* identity is the user identity that starts a process, schedule, or operator.

Runtime security enforcement is used when a process starts, regardless of how the process is invoked. For example, runtime security enforcement applies to child processes started by parent processes.

See Specify Runtime Security Properties (see page 68).

You can also configure an operator in a process to run in the context of the user who called it by checking the *Run as Caller User* check box listed under Execution Settings. Marking this option indicates that you want the operator to run as if the user who started the process was in control. Operator settings override process property settings, if different.

Properties Affecting Security of Running Processes

Only the process owner or environment content administrators can set *Runtime Security*. Two process properties impact runtime security for instances of this process:

- Runtime Security
- Run as Owner

Runtime Security

Specifies whether to enforce runtime security for this process. Runtime security can be enabled or disabled either explicitly or through inheritance. When set explicitly, changes to inherited settings have no impact.

Inherit from Orchestrator

Applies the same setting that is currently configured on the orchestrator. *Enable Runtime Security* can be selected or cleared on the Policies tab of the parent orchestrator.

Enable

Indicates you want to enable *Runtime Security*. When a user attempts to start an instance of this process, CA Process Automation examines the setting for *Run As Owner* for the user.

- If *Run As Owner* is selected, CA Process Automation determines the user currently set as owner and starts the process under the identity of the owner. If this process calls another process, that process runs under the identity of the owner of the parent process.

  Note: This setting can be overridden at the operator level if *Run as caller user* is selected.

- If *Run As Owner* is cleared, CA Process Automation examines permissions for the user that is attempting to start an instance of the process. If that user has start rights, CA Process Automation allows the process instance to start under the caller user identity. If this process invokes another process set as caller user, CA Process Automation checks start rights for the child process.

Disable

Indicates you want to disable *Runtime Security*. The *Run As Owner* check box is disabled.

Run As Owner

This check box is enabled only if *Runtime Security* is enabled either explicitly or through inheritance.

Selected

Specifies that all instances of the current process can run under the identity of the owner (run as owner). When an authorized user starts the process, the owner gains access to child processes and other objects. Access by the owner can include objects that the caller user, who launched the instance, is not permitted to access. Only the process owner or environment content administrator can set this property.

Cleared

Specifies that start permission is verified at runtime for the caller user that attempts to start the process instance .

Guidelines for Setting Runtime Security for a Process

At startup, a process instance can assume one of the following identities:

- The caller user, that is, the user who started the process instance.
- The process owner.

When configuring runtime security at the process level, consider the following guidelines.

Your Objective:	Required Configuration:
Run the process as the caller user. Enforce runtime security rights with the identity of the user who starts the process instance.	■ Select *Enable* in the Runtime Security field. ■ Clear *Run as Owner*.
Run the process as owner. Enforce runtime security by running process instances under the identity of the owner, regardless of who starts it.	■ Select *Enable* in the Runtime Security field. ■ Select *Run as Owner*.
Disable validating and enforcing process ownership at runtime.	Select *Disable* in the Runtime Security field. This option helps ensure backward compatibility for existing processes.

More information:

Determine When to Select Run as Owner (see page 67)

Exception Handling

Exception handling allows you to define sequences of operators for predefined exceptions on operators in a process, such as Failure, Abort, or Unexpected outcome. You can also create a default sequence of operators to perform for any exceptions lacking an explicit sequence. While the Workflow module processes an exception, it pauses execution of any other operators in the process.

Exception handling uses priorities when evaluating exit conditions on an operator. The following table lists the exception types:

Priority	Exception Type	Occurs When
1	System Exception	There is an incorrect touchpoint name, an unreachable agent, or any type of communications failure.

Priority	Exception Type	Occurs When
2	Unidentified Response	There is no exit link for a particular exit condition.
3	Aborted	An operator aborts or a user aborts an operator.
4	Timeout	An operation times out and there is no path defined from the timeout port to the main flow.

When a process operator experiences an exception, the Workflow module takes the following actions:

- Suspends processing of the process after executing the current operators.
- Tries to match and run an exception in the following order:

Priority	Matches	Action
1	Exception handler defined in the process object.	Runs the exception handler defined in the process object.
2	Exception handler defined on the default process object for the orchestrator running the Workflow module.	Runs the exception handler defined in the default process object for the orchestrator.
3	None	Ignores the exception. The Workflow module continues processing the process.

Create Exception Handlers

Exception handlers let you create sequences of operators for the following predefined exceptions in a process:

Aborted

Occurs on a user-specified or operator abort.

System Error

Occurs with any type of communication failure. For example, when the process contains an incorrect touchpoint name or it refers to an agent that is not running.

Timeout

Occurs when both of the following circumstances are true:

- The operator times out before it finishes.
- The operator is configured to take the timeout path and end or continue with a result. The exception handler defines the timeout path.

Unidentified Response

Occurs when no output connector corresponds to the response.

Follow these steps:

1. Click the Designer tab.
2. Click Open, navigate to the folder with the process to open, select the process, and click Open.
3. Select the Exception Handler tab.
4. Expand the Standard folder in the Operators palette and drag the Exception operator to the process.
5. Expand the Exception operator properties dialog.
6. Expand Information and type a name in the Name field.

   Note: The best practice is to name operators in a default exception handler with a prefix so they do not match operator names in the process that loads the default exception handler.
7. Expand Exception Occurred.
8. Select an exception type from the drop-down list.
9. From the palette, drag more operators to the process that completes the rule for the exception. Link the operators in the execution sequence.

   Note: If you finish the sequence without adding any Stop operators, the main process resumes. Optionally, you can stop the process for one or more paths in the Exception Handler.
10. In the toolbar, click Save.

The new exception rule is added.

As part of exception handling, you can reset the operator and then continue the process. You can also select to ignore an exception and continue with the process. To ignore an exception, set the operator in simulate mode and continue with the process. The Reset operator resides on the Common palette. You can use the Reset operator in the process pane, exception handler, and lane change handler.

Follow these steps:

1. Right-click Add, Reset to add a Reset operator.

   The text box displays an entry with a drop-down list from which to select one of the operators in the current process. You can add multiple operator names.

2. To manipulate an operator name, click Delete, Move Up, and Move Down as appropriate.

3. Enter an expression that resolves to a string (for example, operator name) or a list of values (for example, operator names) at run time. Take this action instead of selecting an operator name from the drop-down list.

4. Select or clear Continue with Result.

   Selected:

   Makes the EndCondition option available to select either Successful or Unsuccessful.

   Successful:

   If an error condition is met at run time, CA Process Automation assumes that the selected operators are successful. It continues with the rest of the process flow.

   Unsuccessful:

   If an error condition is met at run time, CA Process Automation assumes that the selected operators failed. It continues with the rest of the process flow.

   Cleared:

   If an error condition is met at run time, CA Process Automation resets the selected operators, then continues with the process flow.

5. (Optional) To ignore an exception and continue the process, add a Reset operator in the exception handler mode to ignore the exception:

   `exceptionStart-<operator-name>.Source`

Note: When you add a field to the Reset operator Operators List, the process Loop operator names appear in the drop-down list of the new field. The Reset operator resets all operators in the Loop operator and resets the Loop operator to the first iteration. After the reset, the Loop operator restarts from the first iteration. Because the Loop operator does not support simulation, the Reset operator *always* resets a Loop operator. The Loop operator resets and the following field values are ignored:

- Continue with Result

- End condition
- Pre-execution code
- Post-execution code

Run Processes Interactively

You can start an instance of a process immediately or in suspended mode.

When a process starts immediately, an instance of the process is created, loaded it to memory, and immediately starts processing operators. If a process is started in suspended mode, the instance of the process is loaded into memory, but it does not start processing.

You can put a shortcut to a process in a Process Watch object. If you are an authorized user, you can start and monitor the process from within the Process Watch object. In the Operations tab's Process Watch palette, right-click the process and click Start.

You can start the current version of a process by accessing the process object in the Library Browser. You can also start a process while you are editing it in the Process Designer. While you are editing a process, you can check in changes and can start the current version of the process without leaving the Process Designer.

When a process starts, CA Process Automation creates a copy or *instance* of it in the automation library. Changes to an instance do not affect the base definition of the process. You can access the base definition through the Library Browser. Process instances are monitored through a Process Watch. To monitor a process instance, you can open the Process Watch in the Operations tab's Process Watch palette. You can open the actual Process Watch automation object through the Library Browser.

More information:

The Operations Page (see page 390)

Start a Process from the Library

You can start a process from the Library tab. Starting a process immediately lets you perform a task in a production environment.

Follow these steps:

1. Click the Library tab.
2. Click Orchestrator and select the appropriate *Orchestrator:environment* combination.
3. Navigate to the folder that contains the process to start.
4. Take one of the following actions:
   - Right-click the process and select Start Process.
   - Select the process and select Start Process from the More Actions drop-down list on the toolbar.

   The Monitor Process Instance prompt opens.
5. Take one of the following actions:
   - Click Yes to open a new window to monitor the running instance of the process.
   - Click No to run an instance of the process. The process is not displayed.

   The process starts immediately.

Start a Process as Suspended from the Library

You can start an instance of a process in a suspended state to achieve any of the following objectives:

- Insert breakpoints.
- Set parameters.
- Make other changes before the process runs.
- Monitor or control the execution of a process.
- Debug the sequence of steps in the process.

Follow these steps:

1. Click the Library tab.
2. Click Orchestrator and select the appropriate *Orchestrator:environment*.
3. Navigate to the folder that contains the process to start in suspended state.
4. Take one of the following actions:
   - Right-click the process and select Start Suspended.
   - Select the process and select Start Suspended from the More Actions drop-down list on the toolbar.

   The Monitor Process Instance prompt opens.
5. Take one of the following actions:
   - Click Yes to open the Designer tab with the debug toolbar. You can begin working with the suspended instance immediately.
   - Click No to load the process into the Operations tab. The Designer tab does not open. You can later navigate to the instance in the Process Watch palette in the Operations tab. To continue, right-click that instance and select Start Suspended.

More information:

Debug a Process (see page 408)

Start a Process While Editing

While you are editing a process object, you can start the current version without leaving the Process Designer. The Start and Start Suspended commands are available on the File menu. Start and Start Suspended buttons are also available on the Process Designer toolbar.

The Process Designer Start and Start Suspended commands create an instance of the current process in memory, like starting a process in the Library Browser. If you click Start, the instance of the corresponding process is created and starts executing the process immediately. If you click Start Suspended, the instance of the corresponding process is created but does not start execution of the process.

Both the Start and Start Suspended commands prompt for the touchpoint on which to run the process, then prompt whether you want to monitor execution of the process. If you monitor execution of the process, CA Process Automation opens a separate Process Designer window to work with the new instance in debug mode.

Open an Instance of a Process

The Process Watch object and Operations page let you view instances of processes on an orchestrator. You can:

- Recover and restart the processes that are suspended after an incident.
- Assess the values of dataset variables and the status of operators in running, suspended, or ended processes.

You can create Process Watch objects or use the Operations page to monitor and edit instances of processes. Click the Operations tab to monitor all instances of processes on an orchestrator. You can also set filters to monitor only selected objects.

More information:

The Operations Page (see page 390)

Process States

The Process Designer periodically updates the current state of the process. Color-coded icons are used to indicate the state of every operator. You can edit the process while it is running or suspended. After a process has completed or aborted you can no longer change the instance.

Debug a Process

When you open an instance of a process, the Process Designer helps you monitor the status and debug a process. Debug buttons are available on the Process Designer toolbar. When processing is suspended, you can edit the process and change parameter values in operators.

A process can be suspended in several ways:

- When an instance is started in a suspended state.
- When you click the Suspend Process button.
- When a process ends and you click the Switch Process Status on Completion button on the Process Designer toolbar. The process is suspended but the status appears as Blocked.
- When there are no valid operators left on any branch of a running process. The process is suspended but the status appears as *Blocked*.

Suspend a Process

When a process is in the suspended state, you can do any of the following:

- Change whether a process is unloaded after completion
- Reset the process
- Reset operators in a process
- Add or remove breakpoints
- Modify the process
- Click the **Resume Process** button to continue processing
- Abort the process

Modifying a process in a suspended state lets you work on an unanticipated issue, and then resume automated execution while still tracking any changes that you have made. Switch to a Process Watch to export a modified instance of a process and permanently save any changes made at runtime.

To suspend execution of an instance of a process while working in debug mode, click the **Suspend Process** button on the toolbar of the Process Designer.

The execution of the process instance stops. No further dependencies are examined until you resume execution of the process. You can edit a process in a suspended state. Any modifications to operator parameters or other elements of the process affect only that instance of the process.

To restart execution in a suspended instance of a process, click the **Resume Process** button. The **Resume Process** button restarts a suspended process from where it stopped executing unless it is reset. If an instance of a suspended process is reset, execution restarts from the beginning of the process.

Change whether Processes are Unloaded on Completion

When running a process in debug mode, the Workflow module typically does not unload the process instance when it reaches a Stop operator. This allows you to modify and restart the process.

To force the Workflow module to unload a process when it reaches a Stop operator, click Keep State on the Control menu to clear the check mark next to the command. You can also use the Switch Process Status on Completion button on the toolbar to switch this command on or off. The toolbar button remains inactive while the command is toggled on.

Set and Remove Breakpoints in a Process

You can use breakpoints to identify errors. Breakpoints help you check variable values and operator parameters. Set a breakpoint on an operator to interrupt a process immediately before the operator starts. You can then set parameter values and examine processing of an operator as it occurs.

When a breakpoint is set, the entire process is suspended when it reaches the operator with the breakpoint. An exclamation point (!) symbol appears near the operator that has suspended the process.

You can set and remove breakpoints in a process object or in a suspended process instance. Breakpoints you set in the original process object definition automatically appear in any instances of that process.

Follow these steps:

1. Open a process in the Process Designer.
2. Select one or more operators in the process.
3. On the toolbar, click Set Breakpoints.

   The breakpoint symbol appears next to the selected operator.
4. To remove existing breakpoints, select one or more operators, and click the Remove Breakpoints button on the Debug toolbar.

Debug a Java Process

The Java connector uses *Apache Log4j* to capture the connector's log messages. When troubleshooting an issue with a Java process, debug it by enabling and then reviewing the log files. The log messages captured at the DEBUG level are very detailed and should help system engineers define the root cause of an issue.

Note: The paths to the log4j.xml and c2o.log files change when running the Java module on a CA Process Automation agent.

Follow these steps:

1. Locate the log4j.xml file at the following path:
   `CA Process Automation_Installation_path\Domain\server\c2o\conf\log4j.xml`

2. Set the Java module's log4j threshold level to DEBUG.

   An example showing the specific section and line (shown in bold) of the log4j.xml file follows:

```
<!-- A size based file rolling appender for C2O and JXTA Logs-->
  <appender name="C2OFILE"
class="org.jboss.logging.appender.RollingFileAppender">
    <errorHandler class="org.jboss.logging.util.OnlyOnceErrorHandler"/>
    <param name="File" value="${jboss.server.home.dir}/log/c2o.log"/>
    <param name="Threshold" value="DEBUG"/>
    <param name="Append" value="true"/>
    <param name="MaxFileSize" value="50000KB"/>
    <param name="MaxBackupIndex" value="3"/>
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%d %-5p [%c] [%15.15t] %m%n"/>
    </layout>
  </appender>
```

3. Insert the following section in the log4j.xml file:

```
<category name="com.optinuity.c2o.servicegroup.javaobject">
  <priority value="DEBUG" />
</category>
```

4. Open the Java connector's messages captured in the c2o.log file located at the following path:
   `CA Process Automation_Installation_path\Domain\server\c2o\log\c2o.log`

Reset a Process

You can reset a suspended instance of a process to restart execution. All variables, parameters, and operators in a suspended instance of a process are reset to their initial states, with one exception. The application does not reset user-defined parameters.

If you resume execution of a process after resetting it, processing restarts at the Start operator. You can reset a process during testing or debugging to rerun chains of operators. In production, you can reset a process following an incident. You can modify a reset process before restarting it to avoid redoing some tasks or to perform some additional tasks. For example, you can:

- Set or remove breakpoints
- Use a different process
- Set operator parameter settings

Abort a Process

To stop an instance of a process, click the Abort Process button on the toolbar of the Process Designer.

Note: You cannot modify a process instance after executing the abort command.

Control a Process Branch

The following commands allow you to control individual branches of a process without affecting the entire process.

- enable and disable operators
- abort execution of operators
- reset individual operators and resume execution of operators in a process.

These commands appear in the Process Designer toolbar for both process designs and instances.

Disable Operators or Deactivate Branches

You can use the Disable Operators command to disable an operator and temporarily disable part of a process without otherwise modifying the process. Disabling an operator stops execution of the operator and all subsequent operators in a branch of a process. An operator is inaccessible to a process when every path to it from any Start operator passes through a deactivated operator. The Disable Operators command is available when editing either the definition object for a process or an instance of a process.

To disable an operator in a process

1. Open either a process object or an instance of a process.

   Note: For a running instance of a process, suspend execution before proceeding to the next step of the procedure.

2. Select one or more operators that you want to disable.
3. Click Disable Operators on the Process Designer toolbar to disable an operator.

   The disabled symbol appears next to the operator.

Note: To enable disabled operators, select one or more disabled operators, and click Enable Operators on the toolbar.

You can also deactivate an entire branch by right-clicking a link and selecting Disable Link from the menu.

More information:

Simulate Processing of Operators (see page 415)

Abort an Operator

The Abort Operators command aborts processing of one or more selected operators that are in a *Running* state. An aborted operator enters a *Failed* state. Processing does not continue on an aborted branch of a process, but you can still continue on other branches. If you defined an exception handler, it is executed to handle an abort exception.

Follow these steps:

1. Open a running instance of a process.
2. Select the running operator that you want to abort.
3. Click the Abort Operators button on the Process Designer toolbar:

   Processing for the operator is aborted.

Reset Operators in a Process

The Reset Operators command resets selected operators in a suspended process to their initial states as if they had not been executed. This command is typically used to allow operators in an instance of a process to run again without resetting the entire process. You can modify the operators with different parameter settings, or modify the process in other ways.

Follow these steps:

1. Open a suspended instance of a process.

   If necessary, suspend execution of a running instance.

2. Select one or more operators that you want to reset.
3. Click the Reset Operators button on the Process Designer toolbar.

   The operators are reset.

   Any Loop operators are reset as follows:

   - Resets all operators inside the Loop operator.
   - Resets the Loop operator to its first iteration.

   After the reset, the Loop operator will restart from the first iteration.

Resume Execution of a Suspended Process

The Resume Process command resumes processing of operators you have reset in a suspended instance of a process.

To resume execution of a suspended process

1. Open a suspended instance of a process.
2. Click the Resume Process button on the Process Designer toolbar.

   The process is resumed.

Simulate Processing of Operators

To bypass an operator while executing subsequent operators in a process, you can *simulate* execution of the operator, rather than disable it. You can simulate processing of selected operators or an entire process. You can use the process simulation properties for every operator in a process. You can also override the process settings for individual process operators.

Simulation mode is often used during development to verify the flow within a process or to verify the synchronization of interdependent processes. In the production environment, the local mode is often used to skip individual tasks to accommodate transient conditions in the system, such as an unavailable agent, orchestrator, or a suspended application. The distant simulation mode is used mostly in the development phase to verify the configuration and parameters of operators in a process. The distant simulation mode can also be used to run simulated tasks in production periodically, for the sole purpose of verifying dataset field values.

Simulation mode controls the results and duration of an operator in a process. You can use it during development to avoid executing an operator while testing operators in subsequent branches of a process. Simulation is useful in a production environment when an operator should not run. The operators do not run when the processes or resources are unavailable. In either case, simulation of an operator lets you run a process without changing entry or exit links on the operator. In other words, you are able to avoid running the operator temporarily while preserving the structure and dependencies in your process.

Simulate Processing of a Selected Operator

You can simulate the processing of a selected operator.

Follow these steps:

1. Open process in the Process Designer.
2. Open the operator properties.
3. Configure the Simulation properties for any operator.
4. To activate simulation, specify either a Local or Distant simulation type.
   - Local: The operator is not processed. CA Process Automation does not call the associated module or verify the module parameters. The simulated operator returns the result and processes the link specified by the End condition option.
   - Distant: The Workflow module calls the associated module. The module verifies 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 simulated operator returns the result and processes the link specified by the End condition option.

   The simulation symbol appears next to an operator when either the local or distant simulation is activated:

5. (Optional) Select the Evaluate Pre-Execution and post-Execution Code check box to prevent side effects generated by ignoring preconditions or post-execution code. Select this option to evaluate the operator's preconditions and execute its post-execution code.
6. Set the End condition parameter. It specifies the simulated outcome of an operator. The End condition determines which exit link is processed after the simulated processing of the operator. Setting this parameter determines subsequent processing of dependent branches in the process. This setting determines the value returned by the *Result* variable of an operator. The *Successful* or *Completed* setting returns a positive value, which activates a Successful exit link on an operator. The *Failed* setting returns zero or a negative value, which activates a Failure or Abort exit link. The *Custom Result* setting allows you to return an arbitrary value, typically used to test a custom exit link on an operator.
7. (Optional) Specify a delay in seconds to simulate the duration of the task. Change the Delay field from the default value of zero if the time typically required to execute an operator could affect the behavior or outcome of other operators in the process.

More information:

Pre-Execution Code and Post-Execution Code (see page 218)

Simulate Processing of an Entire Process

You can simulate processing for all steps of a process. The process simulation settings apply to all operators in the process that are not configured to override the simulation settings for the process.

Follow these steps:

1. Open the process and view its properties.
2. On the Properties pane Simulation tab, configure the default simulation settings for all operators in the process:
   - Mode
   - Delay
   - End Condition
   - Evaluate Pre-execution and Post-execution Code

Except for the option to override the process settings, these settings are the same as described for configuring an operator.

Chapter 11: Generating and Using Process Documentation

You can generate documentation of your automated processes. Others can print the process documentation that you generate.

This section contains the following topics:

About Process Documentation (see page 419)
Generate Process Documentation (see page 420)

About Process Documentation

You can generate process definition details and graphics with annotations. The output format is PDF.

Process documentation includes details at the following levels:

- Process-level details
- Operator-level details for each operator used in the process flow

The following list summarizes the details that are generated at each level:

Process-level details

Process documentation generates the following details for a selected process:

- Process image
- Process name
- User-created parameters in the process dataset
- Library path
- Owner
- Process version
- Release
- Checked out by
- Creation date
- Last modified date
- Orchestrator where the process resides
- Process tags

Operator-level details

Process documentation generates the following details for each operator that a selected process uses:

- Bitmap of the icon that represents the operator
- User-created parameters in the operator dataset
- Operator name
- Operator description
- Operator type (that is, the operator category)
- Operator parameters
- Exit ports
- Execution Settings
  - Pre-execution code
  - Post-execution code
  - Loop
  - Timeout
  - Target

Generate Process Documentation

You can generate process documentation at any time. For example, use this feature to record the status of a process before you transition it to production.

Follow these steps:

1. Open the process from the Library tab.
2. In the Designer tab, click Process Documentation toolbar button.

   The Process Documentation Options dialog opens.

3. (Optional) Select options to include in the documentation.
   - Process Dataset
   - Operator Information
   - Pre-/Post-Execution Code
   - Operator Properties
   - Operator Dataset
   - Exception Handler (if selected, CA Process Automation generates documentation and an image file for any operators in the Exception Handler)
   - Lane Change Handler (if selected, CA Process Automation generates documentation and an image file for any operators in the Lane Change Handler)
4. Click OK.

   A dialog prompts you whether to open or save the file.
5. Click open to view the contents of the Zip file, or click Save to save the Zip file to a location of your choice.

   The Zip file contains:
   - A PDF file that contains the process documentation.
   - The full-size images contained in the process documentation. The PDF contains scaled images, so they may not be legible if the process is very large.
   - An XML file that contains the raw data used to generate the documentation in XML format. This file lets you transform the XML into other formats.

Chapter 12: Release Objects to Another Environment

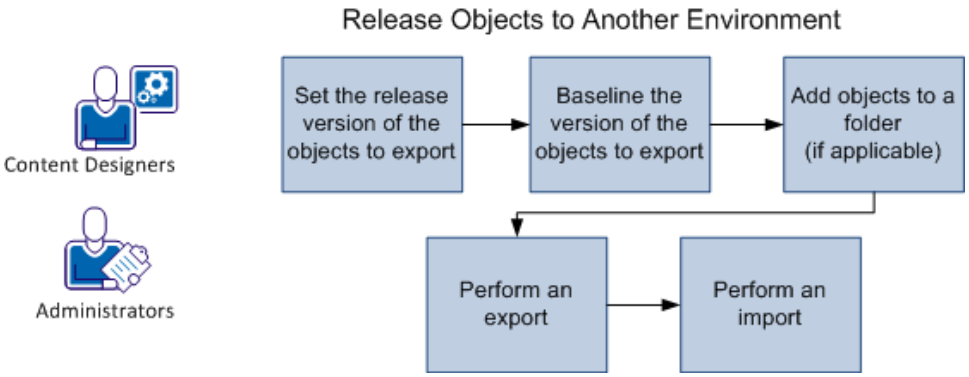
After you design and test a new process, you can release the process with the objects it uses. Typically, you release a new process from a design environment Orchestrator library to a production environment Orchestrator library. You can export objects themselves, in a folder, in a package, or in a content package (see page 429). However, to export related objects in a single container and attach a release version to it, you must export those objects as a content package.

If you used packages in CA Process Automation 04.1.00, migrate those objects to a new content package.

The design environment and the production environment are in the same Domain. If the functionality is designed on one location to deploy in a different location, you can implement the product in two separate Domains. You can import the XML file for an object, folder, package, or content package to an Orchestrator in another environment in a different Domain.

Note: When you export objects to import to an Orchestrator in the same environment, you can omit baselining objects and setting release versions.

An administrator or content designer can perform this process. Typically, an administrator exports a folder as a content package from the default environment and then imports it to a production environment. The following illustration describes the process:



Follow these steps:

1. Set the release version of objects to export. (see page 425)
2. Baseline the version of objects to export (if applicable) (see page 426).
3. Add objects to a folder (if applicable) (see page 434).
4. Perform an export (see page 430).
5. Perform an import (see page 444).

Release Versions

A *release* occurs when you deploy a process and its components from a design environment to a production environment. When you select components to package for export, you select one version of each component. You can enter a release-specific value in the Release Version field of the version of each folder and object you plan to package for export. This practice is required when you export objects in a folder as a content package.

After you set a release version for each object, you can add a corresponding version to the folder to export as a content package.

Content packages begin as folders in the initial CA Process Automation environment. When you export a folder as a content package and you import it to a new environment, the product baselines the imported object versions in the content package. Users cannot modify the Release Version data of imported objects.

View Release Version Information

The Versions tab displays information about the release versions that are set for the selected object.

Follow these steps:

1. Click the Library tab.
2. Select an object in the main window and click Properties.
3. The Properties pane appears in the bottom portion of the window.
4. Click the Version tab.
5. View the displayed data under Release Version.

   The Release Version identifies the specific version of an object that was imported, or that you want to export, then import, to a production environment. You can set this identifier on any individual object.

The product locks this field in the following circumstances:

- The object was imported as part of a content package.
- The object was imported from a package (in release 4.1.00) with its release version in nonmodifiable mode.

Note: See the *User Interface Reference* for detailed information about the Versions tab.

Set the Release Version of Objects to Export

The Versions tab displays information about the release version(s) that are set for the selected object. You can set the release version when you are ready to deploy an object to a production environment. An object's release version typically identifies its specific version for deployment. Through the Release Version field, for example, you can set the release version attribute for a process and each associated object.

When you are ready to deploy an object to a production environment:

- You can export a single object.
- You can export all of the objects in a selected folder, including its subfolders.
- You can export the objects as a content package. In this case, release versions are required on every folder that you export as a content package, and all of the objects included in the folder. Once the content package is imported, the release version cannot be modified in the new environment unless a copy of the object is made.

Note: See What is the Difference Between Exporting an Object, a Folder, a Content Package, and a Package (see page 429) for more information.

In each case, the exported file contains the value of the release version attribute of each object being exported.

You can set the release version in the following ways:

From the Library

Follow these steps:

1. Click the Library tab.
2. Select a folder from the left pane and then a single object from the list in the right pane.
3. Right-click the object and then select Properties.
4. From the Properties pane, click the Versions tab.

   Note: See the tooltips or the *User Interface Reference* for field descriptions and examples.
5. Right-click the applicable version and then select Set Release Version.
6. Enter an appropriate value in the Release Version field.

From the Versions Tab in the Object Editor

Follow these steps:

1. Click the Library tab.
2. Double-click an object.

   The current version of the object opens in the object editor.
3. Click the Versions tab.

   Note: The process editor does not contain a Versions tab.
4. Right-click the version of the object to deploy and select Set Release Version.

   The Release Version field opens.
5. Enter a release version for the object, and then press Enter.

While Checking in an Object

See Check In an Object. (see page 79)

Baseline the Release Version of the Objects

A content designer baselines an object to create a snapshot version of it in the design environment. Users cannot change the baseline version of an object but can use it as the basis for a new version. A content designer can baseline objects at any time, but typically baseline the version of the object that is targeted for export. To prepare for export, content designers can both baseline and set the release version on the export version in the same session.

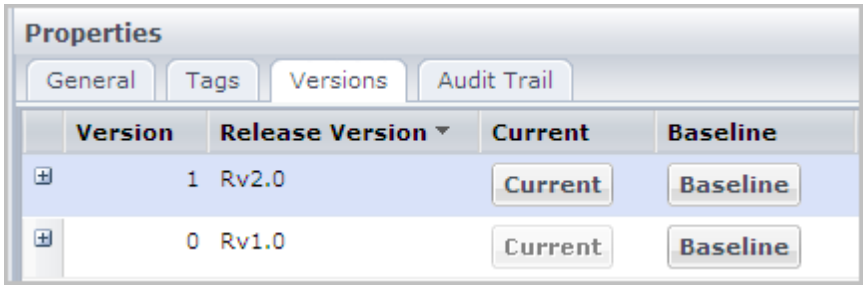
Note: The version of an object that you baseline for exporting in a folder as a content package can be the current version or another version. Typically, it is the current version that is checked it.

Follow these steps:

1. Click the Library tab.
2. Click Orchestrator and select the appropriate *Orchestrator:environment*.
3. Navigate to the folder with a target object and verify that the State field for the object indicates Checked In. (If it does not, check in the object before continuing.)
4. Select the target object and click Properties.
5. Click the Versions tab.
6. If the Baseline button is enabled on the version to release, click Baseline.

   In the following example, the Baseline button for Rv2.0 is enabled.

Equation 3: The Properties of all objects has a Versions tab. Here you can, optionally, set a release version, set one version as current, and baseline one or more listed versions.



7. Close the Properties pane.

Set the Release Version of Content Package as Current

The Versions tab displays information about the release versions that are available for a content package. The release version typically identifies a specific package version to deploy in a production environment.

After you import the content package to the new environment, you can use the Versions tab to set the content package release version attribute.

Follow these steps:

1. Click the Library tab.
2. Right-click a content package, and then select Properties.
3. On the Properties pane, click the Versions tab.
4. To set the content package as the current release version, select it and click Current.
5. Click Yes on the warning message.

   The Library browser shows the current version of the content package.

Exporting and Importing

To move the following items from the Library on one Orchestrator to the Library on another Orchestrator, use the export and import processes:

- Objects
- Folders
- Packages
- Content packages

For example, you can copy from a development environment to a production environment or copy from one Orchestrator to another Orchestrator in the same environment. The export process creates an XML file that the import process uses. After you export objects, you import them on the destination Orchestrator.

Consider the following points before you export and import:

- The export file preserves complete information, including hierarchies and paths, for each exported object. During export, you can either save the full absolute folder hierarchy from the root folder, or only the relative paths starting in the selected folder.
- When you import an xml file that was exported as Absolute or Relative paths to an automation library, CA Process Automation reconstructs the path information (absolute or relative) in the file from the exported folder structure.
- When you export the root folder, the product exports the entire automation library. Be aware of size constraints before you export the root folder.
- You can open an export file in any XML text editor.
- You cannot modify the release version of an exported content package. The export process encrypts the XML export file to prevent modifications.
- When you export a folder or object outside of a content package, the object itself determines the state of the imported object. For example, if you export a nonmodifiable process outside of a content package, then the process remains nonmodifiable after import.

Exporting

The following topics explain the export process.

What is the Difference Between Exporting an Object, a Folder, a Content Package, or a Package?

CA Process Automation lets you export objects and folders in various ways. Every export option creates an XML file that you can import to a different environment.

You can export the following items:

An object by itself

You can only export an object in a modifiable mode. You can add a release version to the object, but it is not required. After you import an object to a new environment, you can modify the object directly.

A folder and all of the subfolders and objects in it

Exporting a folder exports the folder and its contents, including any subfolders. The contents can include any object available in the CA Process Automation Library.

You can export a folder in a modifiable mode so that you can modify the folder and its contents directly in a new environment. The product does not require objects that you export through a folder to have a release version.

The product preserves the path to the folder and the hierarchical structure of objects and subordinate folders in the export file.

A folder as a content package

A folder becomes a content package upon export and you cannot modify it in the new environment. The product preserves the path to the folder and the hierarchical structure of objects and subordinate folders in the export file.

You can add release information to a folder to export as a content package. The release version attribute is required for all folders, but you can add other relevant information about the folder or the objects in it.

It is good practice to export a folder as a content package when you are transitioning a new process with its components from the design environment to the production environment. The recipients in the production environment cannot modify the contents of the content package.

Important! If you do not set the release version of the content package or the child objects in the folder, the product aborts the export task.

A package

The package object is only supported for backward compatibility in CA Process Automation 04.2.00. You cannot reference packages in interaction request forms, start request forms, custom operators, process watches, or datasets. You can still export a 04.1.00 package, but you cannot modify the package after you import it to a new environment. You can, however, modify the package objects.

To bundle similar objects and export them to a new environment, use a content package instead.

Packages that you migrate from release 04.1.00 to release 04.2.00 export the same way as an object.

Note: You cannot export multiple folders or automation objects simultaneously. Instead, create a folder and then assemble all the objects to export as a single content package. You cannot export a folder that already contains a content package as a content package.

Perform an Export

To move objects, folders, packages, and content packages from the Library on one Orchestrator to the Library on another Orchestrator, use the export process. The Orchestrator can be on the same environment or a different environment.

Export an Object

Note: Packages that you migrate from release 04.1.00 to release 04.2.00 export in the same way as an object.

Follow these steps:

1. Click the Library tab.
2. Click Orchestrator, then select the appropriate *Orchestrator*:*environment*.

3. Navigate to the object to export, right-click the object, and then select Export.
4. To specify the paths, select one of the following options:

   Export, Absolute Paths

   Includes the full path for the selected object.

   Export, Relative Paths

   Includes the path relative to the folder that contains the selected object.
5. To save the XML file, click Save on the File Download dialog.

   The default file name is object-name.xml.
6. Navigate to the location on your local drive at which to save the XML file.
7. Define the name with which to save the file.

   For example, append _RP to the file name to denote a relative path or _AP to denote an absolute path.

   *object-name*_RP.xml

   *object-name*_AP.xml
8. Click Save.

   The product exports the object.

Export a Folder

Follow these steps:

1. Click the Library tab.
2. Click Orchestrator, then select the appropriate *Orchestrator*:*environment*.
3. Navigate to the object or folder to export, right-click the folder, and then select one of the following options:

   Export

   Exports a folder or package in a modifiable mode.

   Export As Content Package (folders only)

   Exports a folder as a content package in a nonmodifiable mode.

4. To specify the export paths, select one of the following options:

   Export, Absolute Paths

   Includes the full path for the selected folder.

   Export, Relative Paths

   Includes the path relative to the folder that contains the selected folder.

   Important! If the release version of the content package or its child objects is not set in the folder, the product aborts the export task.

5. To save the XML file, click Save on the File Download dialog.

   The default file name is folder-name.xml.

6. Navigate to the location on your local drive at which to save the XML file.

   Define the name with which to save the file. For example, append _RP to the file name to denote a relative path or _AP to denote an absolute path.

   *folder-name*_RP.xml

   *folder-name*_AP.xml

7. Click Save.

   CA Process Automation exports the folder and its contents.

More Information

Export Relative Paths or Export Absolute Paths (see page 436)

Content Package Objects

Complex processes can use various objects including operators, icons, resources, datasets, and other processes. They can be distributed among multiple folders with unrelated objects in an automation library. This complexity presents the following potential issues:

- A user can have difficulty working with only the objects that are required for a deployment or subsequent update.
- A developer working with different objects in different folders can have difficulty tracking or finding specific objects.
- An administrator exporting updated objects for deployment to a production environment can forget a required object.

To export all of the related automation objects, you can assemble them in a folder and export that folder as a content package to another CA Process Automation environment. Content packages are useful because they let you bundle related objects in a single container and then assign a release version to them. Then when the objects are imported into a new production environment, they cannot be modified unless the imported objects are copied. You cannot edit or delete any of the objects in the content package. This safeguards those objects from being altered in a potentially harmful way. In other words, you can always roll back to the original content package if you want to reverse any unwanted changes that are made to the objects. Since the release versions for the objects are nonmodifiable, content designers in the source design environment can identify the exported and imported object versions if there is a need to troubleshoot.

In comparison, if you export objects by themselves or through a folder, they are modifiable in a new environment. Any objects that are imported through by themselves or through a folder (instead of a content package) are not required to have a release version. These objects can be modified directly. It is good practice to export a folder as a content package when you are transitioning a new process with its components from the design environment to the production environment.

Note: You cannot export multiple folders or automation objects at a time. Create a folder and then assemble all the objects to be exported as a single content package. You cannot export a folder as a content package that already contains a content package *or* a package.

You can view imported content packages in the Library.

- When you expand a content package in the right-hand pane, its child objects (that are part of the current release) display underneath the content package.
- You can view the latest release information for a folder or content package in their properties. Right-click the object, select Properties, then click the Versions tab. The Audit Trail tab records any changes made to the folder or content package.

You can also view all the imported content packages in the Content Packages palette of the Operations tab.

- If you click the content package in the left-hand pane, its properties display in the right-hand pane.
- If you click the individual objects it contains in the left-hand pane, the object properties display in the right-hand pane.
- If the content package contains a process, the right-hand side displays a graphical view of the instances.

More Information

Release Versions (see page 424)

Create a Content Package

Add Objects to a Folder

Before you export a folder as a content package, populate the folder with objects.

A CA Process Automation release 04.02.00 content package requires all of the objects to export to be in a single folder. In release 04.1.00, you could add objects as shortcuts from different folders to a package for export.

After you add an object, you can use either the Version or the Release Version parameter to set the object version. If you customized the Release Version parameter, set the object version in the Release Version column.

You can set the release version attribute before you add a selected version of each object to a folder. After you add objects to the folder, set the content package release version.

Follow these steps:

1. Create a folder.
2. To open the folder, double-click it in the Library Browser folder pane.
3. In the toolbar, click New, and then select an object type.

   CA Process Automation creates an object.
4. Provide a unique name for the object.

Note: You can also cut and paste an object from another folder to your new content package folder.

Add Release Information for a Folder

Adding release information to a folder is a useful way to provide relevant information about the folder and any of the objects it contains.

You can define the folder release in the Release tab under the folder properties. The ReleaseVersion attribute value that you define for a folder in the Release tab lets you manage the version of a folder, which is especially useful if you want to export that folder as a content package. ReleaseVersion is the one required attribute about the folder. You cannot delete or edit the ReleaseVersion attribute but you can insert the appropriate release version in the Value column.

Note: You can view the latest release information for a folder in the folder properties. Right-click the folder, select Properties, then click the Release tab. The Audit Trail tab records any changes made to the folder.

Follow these steps:

1. Click the Library tab.
2. Right-click the folder in the right pane, and select Properties.

3. From the Properties pane, click the Release tab to view the folder properties.
4. Click the Value field.

   The Value box displays.
5. Enter a value for the ReleaseVersion attribute and then click OK.
6. Click the Add Property link to define an additional folder property, and click Delete Property to remove a user-defined folder property.

Once the release version has been assigned, you can export the folder as a content package.

More Information

Export a Folder (see page 431)

Call Content Package Objects

When any object calls another object in CA Process Automation, the product retrieves the referenced object from the database unless the object is deleted. For example, one process can call another process, or a process can refer to a specific dataset. If the referenced object has been deleted, the product displays an error. Otherwise, the product returns the object.

However, calling an object that is in a content package has another restriction. If any object references another object in a content package, the product verifies that it resides in the *current* release version of the content package. If the object is not in the current release version of the content package (or has been deleted) the product returns an error.

The product requires the most current version of an object in a content package in the following instances:

- You search for a custom operator from the Operator Browser.
- One calendar calls another calendar as a preview exclusion calendar (under the Preview Exclusion Calendar field, located on the Preview tab of the calendar designer). If the selected calendar is not part of the current release version of the content package, its dates are not excluded when that calendar is applied with the Preview Exclusion Calendar field.
- A custom operator calls another object through an object reference that is not part of the current release version of the content package.
- A dataset that the Start Process operator uses calls another object through an object reference. The object reference is not in the current release version of the content package.
- A process uses the Check Calendar operator to call a calender that is not part of the current release version of the content package.

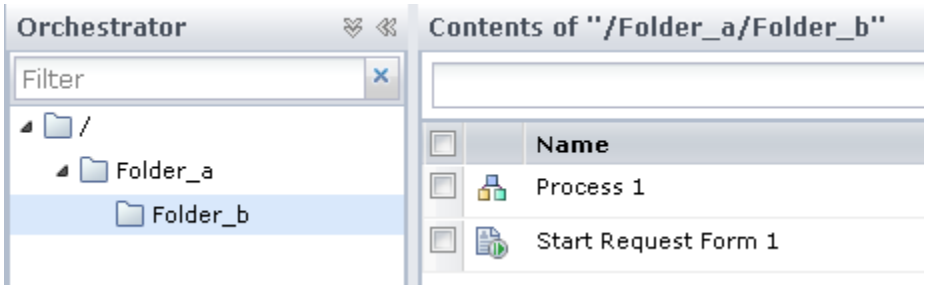
- A process watch calls another object that is not part of the current release version of the content package. In this instance, the Process Watch palette does not display that process watch.
- A process calls an interaction request form that is not in the current release version of the content package. In this instance, that process fails.
- A process calls another process that is not in the current release version of the content package. In this instance, that process fails.
- A schedule calls another process that is not in the current release version of the content package. In this instance, that process does not run.
- A start request form calls another object through an object reference that is not in the current release version of the content package.
- An interaction request form (used in the Assign User Task operator and called in a Start Process operator) calls another object through an object reference that is not a part of the current release version of the content package.

Export Relative Paths or Export Absolute Paths

When you initiate an export, you can select either Export Absolute Paths or Export Relative Paths. The option that you select affects the folder structure in which the selected object appears in the destination Orchestrator library.

Example: Content Package Export

Consider the following example. Under the root folder, the source Orchestrator library is Folder_a contains Folder_b, which contains Process 1 and Start Request Form 1.



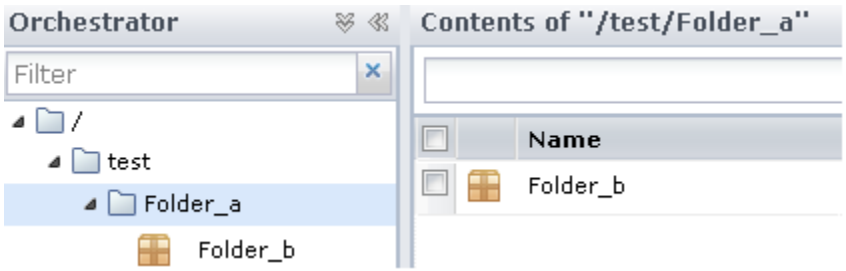
Case 1:

Assume the following criteria:

- You export Folder_b (including Process 1 and Start Request Form 1) as a content package with an absolute path.
- The importing user selects the /test folder as the destination in the new environment.

After the import, the content package appears in the following folder structure:

`/test/Folder_a/Folder_b`



In the hierarchy, test and Folder_a are folders, but Folder_b is now a content package.

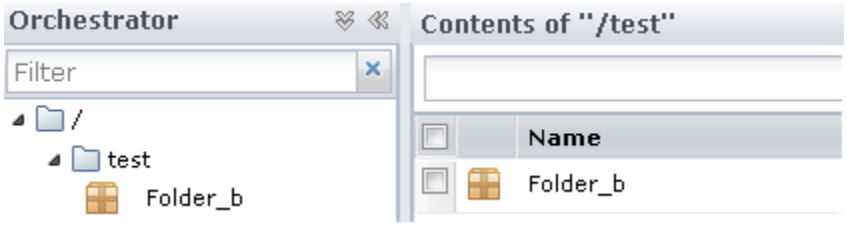
Case 2:

Assume the following criteria:

- You export Folder_b with a relative path.
- The importing user selects the /test folder as the destination.

After the import, Content Pack appears in the following folder structure:

`/test/Folder_b`

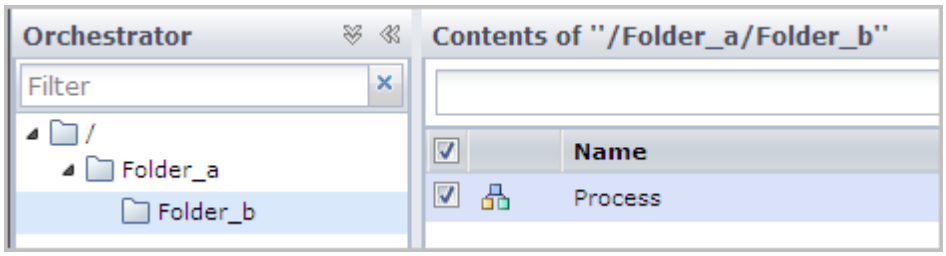


Example: Folder Export

Assume the following criteria:

- The source folder structure is root, Folder_a, Folder_b.
- The object Process resides in Folder_b.

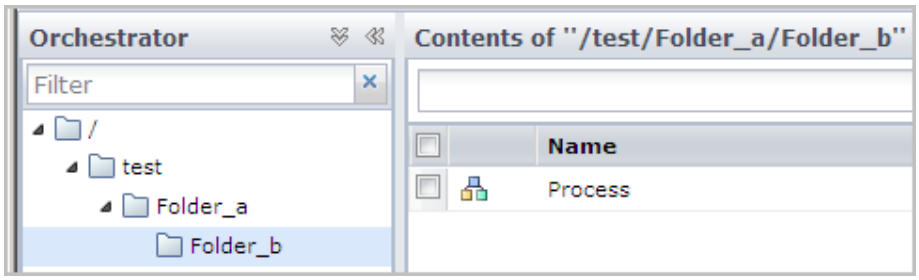
`/Folder_a/Folder_b/Process`



Case 1:

Export Folder_b with absolute paths. Import the exported content to the /test folder. After the import, the structure in the destination is:

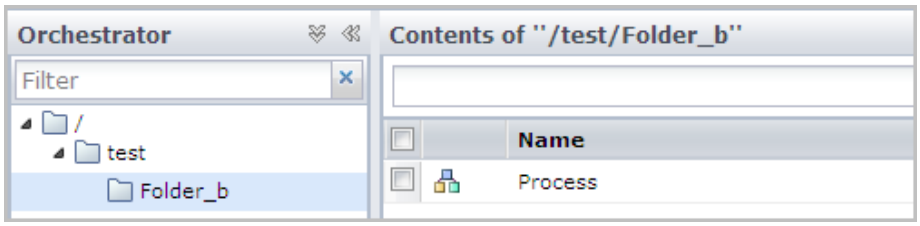
`/test/Folder_a/Folder_b/Process`



Case 2:

Export Folder_b with relative paths. Import the exported content to the /test folder. After the import, the structure in the destination is:

`/test/Folder_b/Process`

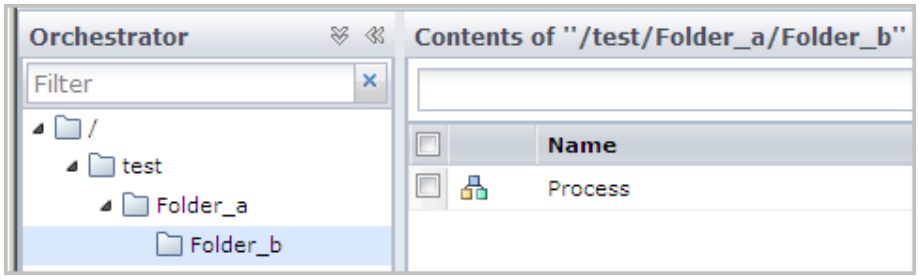


Example: Object Export

Case 1:

Export the object Process with absolute paths. Import the exported content to the /test folder. After the import, the structure in the destination is:

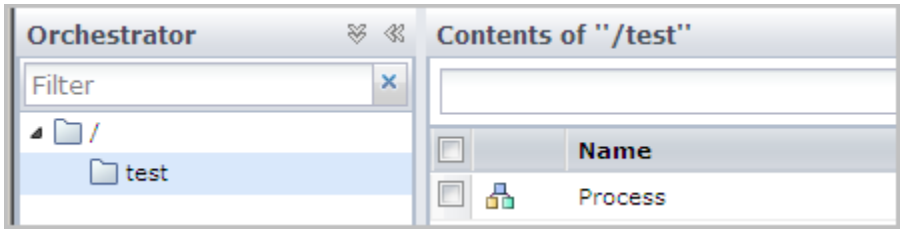
```
/test/Folder_a/Folder_b/Process
```



Case 2:

Export the object Process with relative paths. Import the exported content to the /test folder. After the import, the structure in the destination is:

```
/test/Process
```



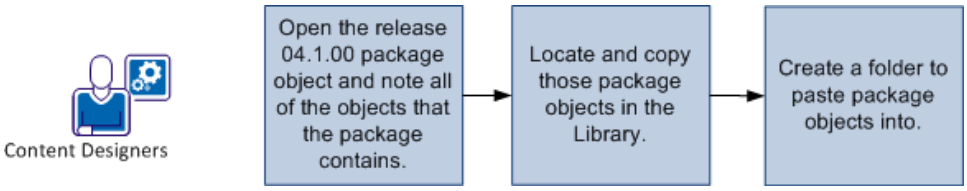
How to Migrate Objects From a Package to a New Folder to Prepare for Export

The content package is an object that is introduced in CA Process Automation 04.2.00. In CA Process Automation 04.1.00, objects were contained in a package for exporting from one environment into a different environment. The package object is only supported for backward compatibility in 04.2.00. You cannot reference packages in interaction request forms, start request forms, custom operators, process watches, or datasets. You can still export a 04.1.00 package, but you cannot modify the package once it is imported into a new environment. You can, however, modify its objects.

If you want to bundle similar objects together and export them as one *non-modifiable* release into a new environment, you must use the content package. CA Process Automation creates the content package once a folder is exported (as a content package) from one environment and imported into another environment. Once imported, the content package is then viewable in the Library Browser and the Operations tab in the Content Packages palette, as indicated by the content package icon .

This scenario describes the required steps to migrate content from a package object into a new folder, if you want to continue to move content as one non-modifiable release between environments.

The following diagram depicts the tasks that content designers perform when migrating objects from a package to a new folder to prepare for export:



Follow these steps:

1. Open the release 04.1.00 package object and note all of the objects that the package contains (see page 441).
2. Locate and copy those package objects in the Library (see page 441).
3. Create a folder to paste package objects into (see page 442).

Open the 04.1.00 Package Object and Note All of the Objects that the Package Contains

The package object that you created in release 04.1.00 is indicated by a blue package icon:

Follow these steps:

1. Open the Library Browser.
2. Locate and open the package object from release 04.1.00.
3. Make a note of all of the objects that are contained within the package.

Locate Those Package Objects in the Library

Once you identify the objects within the package, locate them within the Library, copy them, and paste them into a new folder.

You can open and scroll through known folders in the Library folders or search for folders by name for those objects that the package contains. Additionally, you can search for objects with Basic Search using a single criterion or an Advanced Search using multiple criteria. The Search then finds objects meeting your search criteria within the folder structure you specify.

Follow these steps:

1. Click the Library tab.
2. Navigate the folder tree to the folder containing the object. As necessary, you can query for folders by name from the search field atop the Orchestrator pane.
3. (Optional) Perform a search for the object. From the selected folder, perform one of the following search types:
   - Use a Basic search if querying with a single search criterion, such as object type or owner.
   - Use an Advanced search if querying with multiple criteria, such as object state or modification date.
4. Right-click each object that the search returns and select Copy.

Create a Folder to Paste Package Objects Into

Once you locate an object in the Library that the release 04.1.00 package contains, paste them into a new folder to export them into a new environment.

All of the objects you want to export must be located under one folder. In release 04.1.00, you could add objects as shortcuts from different folders into a package for export. This behavior is not possible with a content package. All of the objects must be housed under one folder only for export.

Follow these steps:

1. In the Library Browser folder pane, select the folder where you want to create a folder.
2. In the toolbar click New, and then click Folder.

   The new folder appears.
3. Rename the folder.
4. Right-click the folder, and then select Paste.

Repeat the process for each object that the package contains until the new folder contains all of the release 04.1.00 content. You can edit the objects once they are pasted in the new folder.

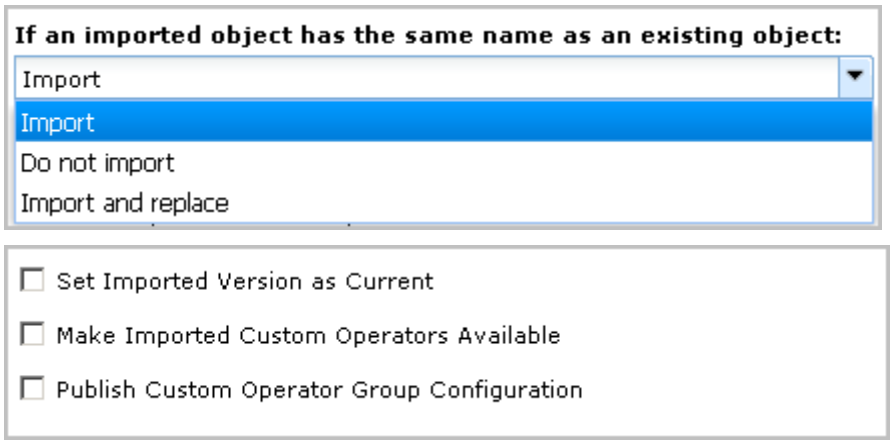
You content migration is now complete and you can export the folder into a new environment.

Importing

The topics in this section explain the import process.

How to Set Import Options

CA Process Automation provides you with some flexibility in how to import objects.



If your import includes custom operator, select Make Imported Custom Operators Available.

If the custom operators are new and they belong to a new custom group, take the action appropriate to your environment.

- Do not select Publish Custom Operator Group Configuration if the import environment is in the same Domain as the export environment. In this case, the custom operator group configuration is already published.
- Select Publish Custom Operator Group Configuration if the import environment is in a different Domain than the export environment

Consider the import content when you configure Set Imported Version as Current and select how to handle duplicate names.

- To activate the imported objects, with the ability to revert to a previous version of an imported object, if needed:
  - Select: Import
  - Select: Set Imported Version as Current

  Note: These options are best when you are importing an upgrade release version and all operator targets are set to hosts in the import environment. You can expect to be notified of duplicate names because objects of the last release are located in the destination folder.

- To import without activating the upgraded objects, where the previous version retains its current version status:
  - Select: Import
  - Clear: Set Imported Version as Current

  Note: These options are best when the import includes operators that target hosts that are not yet defined with their touchpoint name in the import environment. With this setting, you can make objects current after ensuring that the process targets are available in the import environment.
- To defer the import of any object with a duplicate name and to opt for making the objects current manually:
  - Select: Do not import
  - Clear: Set Imported Version as Current
  - **Note:** These options are best when you are importing new objects into a populated folder. These options avoid making an import object a new version of an object with the same name but a different function. These options also let you make the objects current after you test and verify their use in the new environment.

    If you receive alerts, consider these actions:
    - Record the duplicate names in the alert message and inform an administrator in the source environment. Perhaps those objects can be renamed and exported again.
    - Import again, but import to an empty folder.
- To activate the imported objects without the ability to revert the action for objects with duplicate names:
  - Select: Import and replace
  - Select: Set Imported Version as Current
  - **Note:** These options are best when you are reimporting fixes to objects in the destination folder. In this case, you would never need to revert to the replaced version.

Perform an Import

You can import an object, a folder, a package, or a content package that was exported. The export process creates the XML file that the import process uses.

Note: If you exported a folder as a content package, the product imports the exported folder as a nonmodifiable content package in the import folder. You cannot cut and delete from a content package. You can copy the automation objects from a content package and paste them to another folder to modify them. You can view all the imported content packages in the Operations tab Content Packages palette.

Follow these steps:

1. Click the Library tab.
2. Click Orchestrator, then select the appropriate *Orchestrator*:*environment*.
3. Navigate to the destination folder.
4. Right-click the destination folder, and then select Import.
5. On the Import dialog, browse to the XML file you exported and then click Open.

   Note: Ensure that the XML file you want to import is no larger than 250 MB.
6. Specify how to import an object that has the same name as an existing object in the same path.

   Note: If you import to an empty folder, there can be no conflicts with existing names.

   Import

   Processes the imported object as a new version of the existing object. Select this option if the purpose of the import is an upgrade and you want to keep the history of previous versions. If the imported object has the same release version, the product overwrites the existing release version with the release version of the imported object.

   Do Not Import

   Stops the object import and keeps the existing object. The import process lists the objects that have conflicting names. If there are conflicts, import to an empty folder. Alternatively, you can rename the object in the source environment and then repeat the export and import. This option is a good choice when the objects you are importing are new objects instead of new versions of existing objects.

   Import and Replace

   Delete the existing object and import the new version of the object as version 0.

7. Complete your import specifications according to the following guidelines:

   Set Imported Version as Current

   Specifies whether to set the imported version as Current. This option applies if you selected Import and Replace in Step 6.

   - **Selected:** Sets the imported version as the current version.
   - **Cleared:** Leaves as current the existing version that is marked as Current. Clear this option if the previous version of this object is active.

     Note: If you search in a folder hierarchy with a content package, the product returns only objects in the current release version of the content package. The product also returns objects that reside in the folder hierarchy but are not in the content package.

   Make Imported Custom Operators Available

   Specifies whether to set the imported custom operators to Available. You cannot use custom operators until they are made available.

   - **Selected:** Sets imported custom operators as available. Select this option so users do not have to make imported custom operators available individually.
   - **Cleared:** Import custom operators without changing their availability.

   Publish Custom Operator Group Configuration

   Specifies whether to publish the custom operator groups. Publish when importing the object, folder, or package into a different Domain than the one from which the file was exported. Publication publishes the custom operator groups to the Domain-level Modules tab, where every environment also contains the groups that inherit from the Domain.

   - **Selected:** Publishes the custom operator groups. This setting applies when you import custom operator groups to a different Domain than the Domain in which the custom operator groups were designed.
   - **Cleared:** Does not publish the custom operator groups. This setting applies when imported custom operator group configurations were published to the current Domain and environments when the you defined the custom operator group.

   Note: If you do not have Group_Config_Admin rights, this option is disabled.

8. Click Submit.
9. Click OK on the successful import confirmation message.

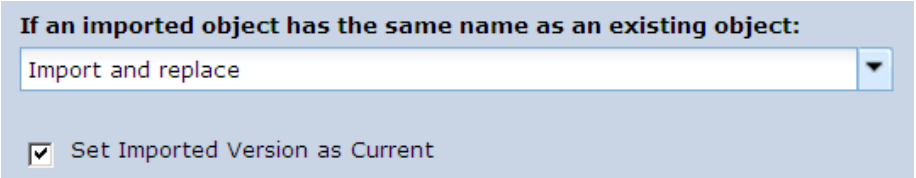
Release Version and Baseline Status of Imported Objects

The state of the Release Version attribute (modifiable or nonmodifiable) and the Baseline status of each imported object depend on the following conditions:

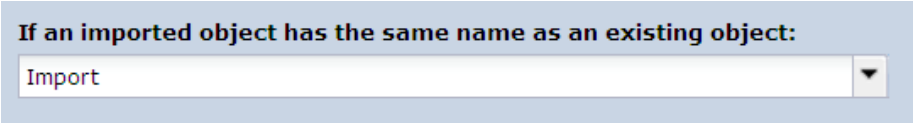
- Whether the import environment already contains existing versions of the objects being imported.
- If the objects exist, whether objects are imported with the option to import as a new version and keep the existing object versions.

The following import options are available for handling existing objects for which the import is a new version:

- Delete the existing object and treat the imported object version as version 0.



- Keep the existing object and treat the imported object version as a new version.

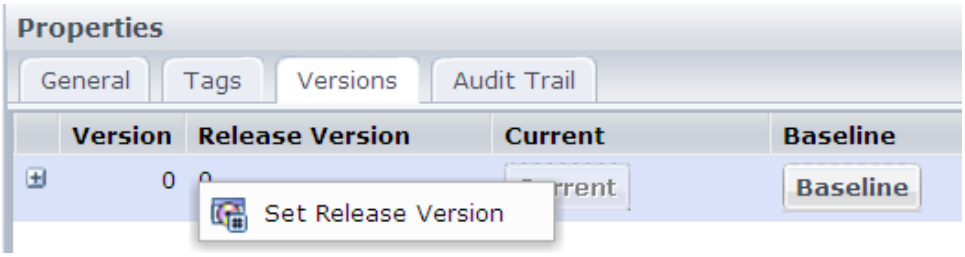


Case 1: The imported versions are the only versions of the objects in the import environment.

The imported version is the only version of the object in the following situations:

- The object did not exist in the import environment before import.
- The object existed, but the imported object replaced the existing object with the same name.

Consider the properties of an object that you create. Version 0 of the object is marked as current, is not baselined, and has a modifiable Release Version attribute.



- When the XML file release version is nonmodifiable, the imported object Release Version attribute state is nonmodifiable and the new version is baselined.
- When the XML file release version is modifiable, the imported object Release Version attribute state is modifiable and the new version is not baselined.

The following illustration summarizes the cases:

State of Object Release Versions After Import
New--No Existing Versions

Release Versions In XML File	Import	Release Versions After Import	New Version
Nonmodifiable		Nonmodifiable	Baselined
Modifiable		Modifiable	Not Baselined

Case 2: The imported versions are the new versions of existing objects in the import environment.

Consider the case where you import a new version of an existing object, where the Release Version attribute of the existing object is nonmodifiable.

- When the XML file release version is nonmodifiable, the imported object Release Version attribute state is nonmodifiable and the new version is baselined.
- When the XML file release version is modifiable, the imported object Release Version attribute state is nonmodifiable and the new version is not baselined.

Consider the case where you import a new version of an existing object, where the Release Version attribute of the existing object is modifiable.

- When the XML file release version is nonmodifiable, the imported object Release Version attribute state is nonmodifiable and the new version is baselined.
- When the XML file release version is modifiable, the imported object Release Version attribute state is modifiable and the new version is not baselined.

The following illustration summarizes these cases:

State of Object Release Versions Before and After Import
Import as New Version and Keep Existing Versions

Release Versions Before Import	Release Versions In XML File	Import	Release Versions After Import	New Version
Nonmodifiable	Nonmodifiable		Nonmodifiable	Baselined
Nonmodifiable	Modifiable		Nonmodifiable	Not Baselined
Modifiable	Nonmodifiable		Nonmodifiable	Baselined
Modifiable	Modifiable		Modifiable	Not Baselined

Values Maintained in the Initial Version of a Copied Object

The *current version* of an object is the subject of a copy and paste action. That is, the new copy has one version, the current version of the original object. The Release Version value and whether it is modifiable are maintained in the initial version of the object copy. If at import the current version of the original object was baselined, the object copy is also baselined.

Appendix A: Format Specifiers for Dates

CA Process Automation uses standard Java date and time format pattern strings. Within date and time pattern strings, unquoted letters from 'A' to 'Z' and from 'a' to 'z' are interpreted as pattern letters representing the components of a date or time string. Text can be quoted using single quotes (') or double quotes (") to avoid interpretation. "''" represents a single quote. All other characters are not interpreted; they are copied into the output string during formatting or matched against the input string during parsing.

Letter	Date or Time Component	Presentation	Examples
G	Era designator	Text	AD
y	Year	Year	2009; 09
M	Month in year	Month	July; 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day of week	Text	Tuesday; Tue
a	AM/PM marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in AM/PM (0-11)	Number	0
h	Hour in AM/PM (1-12)	Number	12
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978
z	Time zone	General time zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800

Patterns are usually repeated; their number determines the exact presentation as shown in the following table:

Presentation	Formatting
Text	If the number of pattern letters is 4 or more, the full form is used; otherwise a short or abbreviated form is accepted, independent of the number of pattern letters.
Number	The number of pattern letters is the minimum number of digits, and shorter numbers are zero-padded to this amount. During parsing the number of pattern letters is ignored unless it is needed to separate two adjacent fields.
Year	If the number of pattern letters is 2, the year is truncated to 2 digits; otherwise it is interpreted as a number. During parsing, if the number of pattern letters is more than 2, the year is interpreted literally, regardless of the number of digits. So using the pattern "MM/dd/yyyy", "01/11/12" parses to Jan 11, 12 A.D.
Month	If the number of pattern letters is 3 or more, the month is interpreted as text; otherwise, it is interpreted as a number.
General Time Zone	Time zones are interpreted as text if they have names. For time zones representing a GMT offset value, the following syntax is used: ■ GMTOffsetTimeZone: ■ GMT Sign Hours : Minutes ■ Sign is either + or - ■ Hours must be between 0 and 23, and one of the following formats: Digit Digit Digit ■ Minutes must be between 00 and 59 and in the following format: Digit Digit ■ Digit is one of the following: 0 1 2 3 4 5 6 7 8 9 The format is locale independent and must be taken from the Basic Latin block of the Unicode standard.

Presentation	Formatting
RFC 822 time zone	A four digit time zone format is used: ■ RFC822TimeZone: Sing TwoDigitHours Minutes ■ TwoDigitHours must be between 00 and 23. Other definitions are as for general time zones.

SimpleDateFormat also supports localized date and time pattern strings. In these strings, the pattern letters described above may be replaced with other, locale dependent, and pattern letters. SimpleDateFormat does not deal with the localization of text other than the pattern letters; that is up to the client of the class.

The following examples show how date and time patterns are interpreted in the US English locale. The given date and time are 2009-07-04 12:08:56 in the US Pacific time zone.

Date and Time Pattern	Result
"yyyy.MM.dd G 'at' HH:mm:ss z"	2009.07.04 AD at 12:08:56 PDT
"EEE, MMM d, ''yy"	Wed, Jul 4, '09
"h:mm a"	12:08 PM
"hh 'o''clock' a, zzzz"	12 o'clock PM, Pacific Daylight Time
"K:mm a, z"	0:08 PM, PDT
"yyyyy.MMMMM.dd GGG hh:mm aaa"	02009.July.04 AD 12:08 PM
"EEE, d MMM yyyy HH:mm:ss Z"	Wed, 4 Jul 2009 12:08:56 -0700
"yyMMddHHmmssZ"	010704120856-0700
"yyyy-MM-dd'T'HH:mm:ss.SSSZ"	2009-07-04T12:08:56.235-0700

Appendix B: Using Masks to Specify Patterns in Strings

Masks are used for operator parameters that monitor messages and monitor file contents. The masks that CA Process Automation uses are JavaScript regular expressions that describe sets of strings without actually listing their contents.

You can use masks to determine whether a specific pattern is matched and to extract parts of the matched string. CA Process Automation puts extracted substrings in variables that you can use to specify parameter values for other operators. Operators that use masks include the Directory Entries operator and extraction macros. You can also use masks to specify permissible strings such as date formats for text input fields.

This section contains the following topics:

Mask Syntax (see page 455)
Sample Mask (see page 458)

Mask Syntax

A regular expression consists of a sequence of characters that must correspond to ones that appear in matched strings. Characters shown in the following table serve as regular expression operators

Regular Expression Operator	Description	Example
.	The dot operator represents any character except an end of line character.	The expression a.b matches any string of three characters starting with “a” and ending with “b”.

Regular Expression Operator	Description	Example
\	The escape enables the use of the operator as literal characters instead of being recognized as operators. If a mask is specified as a literal string in a CA Process Automation expression, the escape operator must itself be escaped to avoid having the escape character stripped by the CA Process Automation expression interpreter before the mask is passed to the regular expression parser.	The escaped character \\ matches the backslash character \. The escaped operator | matches the vertical bar. For a literal string in a CA Process Automation expression, "\\?" matches the question mark.
\n	End of line escape character sequence. If a mask is specified as a literal string in a CA Process Automation expression, the end of line escape character sequence must itself be escaped to avoid having the escape character stripped off by the CA Process Automation expression interpreter before the mask is passed to the regular expression parser.	The expression first line\nsecond line matches strings where “first line” appears at the end of a line and “second line” appears at the start of the next line. If the same expression is specified by a literal string in a CA Process Automation expression, the operator is escaped as follows: "first line\\nsecond line"
|	Alternative. This operator creates an alternative between characters or groups. It applies to the character or groups immediately adjacent to it on its left or right.	(a|b) matches the character “a” or the character “b”. (abcd)|(efgh) matches the string “abcd” or the string “efgh”. (ab|Bc) matches the string “abc” or the string “aBc”.
( )	Group. This operator groups characters in a sub-expression that applies to alternative and repetition operators.	(abcd)|(efgh) matches the string “abcd” or the string “efgh”. (abcde)* matches zero, one or several successive occurrences of the string “abcde”, whereas abcde* matches strings starting with “abcd” followed by an zero or more “e” characters.

Regular Expression Operator	Description	Example
[ ]	Used to define a class of characters by inclusion. It is composed of a list of individual characters and intervals (ranges).	This construct matches a single character. [0-9] matches one digit. [aeiou] matches one vowel. [a-cx-z9] matches a letter among the first three and the last three characters of the alphabet or the digit 9. [0-9]+ matches an integer. [^] Used to define a class of characters by exclusion. This construct matches a single character. [^0-9] matches any character except a digit. [^aeiou] matches any character except a vowel.
*	Repetition. Defines zero or more occurrences of the character or group that appears next to the operator. The series can be empty. a* matches zero or more “a” characters.	(123)* matches zero or more repetitions of the string “123”. 123* matches the digits 12 followed by zero or more “3” characters.
+	Repetition. Defines one or more occurrences of the character or group that appears next to the operator. The series cannot be empty.	a+ matches one or more “a” characters. (ab)+ matches a series one or more repetitions of the string “ab”. [a-z , |.] matches a series of words in small letters, separated by blanks, commas, or periods.
?	Used to indicate that the character or group preceding the operator is optional. ab?c matches the strings “abc” and “ac”.	a(X|Y)?b matches strings “ab”, “aXb”, and “aYb”.
{n}	Used to define a precise number of repetitions of the character or group preceding the operator.	[0-9]{,2} matches a number made of exactly two digits. (ab){2} matches the string “abab”. abc{2} matches the string “abcc”. (a{2})* matches the strings made of an even number of “a” characters.
{n,m}	Used to specify that the number of occurrences of the character or group preceding the operator is between n and m.	[0-9]{2,3} matches a number made of two or three digits. (ab){1,2} matches the string “ab” or “abab”.

Regular Expression Operator	Description	Example
{n,}	Used to specify that the number of occurrences of the character or group preceding the operator is n or more.	0-9]{2,} matches a number made of at least two digits.
{,n}	Used to specify that the number of occurrences of the character or group preceding the operator is n or fewer.	[0-9]{,2} matches numbers equal to or less than 99, including the empty string.

Sample Mask

The following pattern allows precise filtering of the output of the DIR command in the Windows command shell:

([0-9]{2}/){2}[0-9]{2} *[0-9]{2}:[0-9]{2}(a|p) *[0-9,]+ [_~0-9a-zA-Z]*(\.[_0-9a-zA-Z]*)?

The following lines match this pattern:

10/26/05 07:03p 134,217,723 pagefile.sys

10/26/05 07:03p 1,024 testfile

The preceding lines are made up of the following elements:

- ([0-9]{2}/) represents character strings composed of two digits and a slash mark (/), as in “10/”.
- ([0-9]{2}/){2}[0-9]{2} builds on the previous element and specifies that the two-digit sequence can be repeated 3 times, each sequence being separated from the next by a slash mark, as in “10/26/05”.
- *[0-9]{2}:[0-9]{2}(a|p) matches any number of spaces followed by the time in the format hh:mm followed by “a” for AM or “p” for PM.
- *[0-9,]+ matches any number of spaces followed by any number of digits and commas for the file size.
- [_~0-9a-zA-Z]*(\.[_~0-9a-zA-Z]*)? represents a sequence of alphanumerical characters, underscores (_) and tildes (~), optionally followed by a period and another sequence. Because the period (.) is the dot operator, it is necessary to escape it using the escape character (\) when we intend to match it as a character.

The pattern in a mask may differ depending on the exact characters to be matched. For example, [\._~0-9a-zA-Z]* instead of [_~0-9a-zA-Z]*(\.[_~0-9a-zA-Z]*)? could also match the file names in our example outputs, but without imposing any restraints on the number of periods (.) in the file name.

Appendix C: How Targets of an Operator are Processed

This appendix describes how CA Process Automation processes operator targets.

This section contains the following topics:

How Targets for an Operator Can Be Specified (see page 459)
Processing a Target Specified as an IP Address or FQDN (see page 460)
Processing a Target Specified as the ID of an Agent or Orchestrator (see page 463)
Use Case: Track Recovered Processes through Logs (see page 465)
Operators Auto Recovery Example (see page 465)

How Targets for an Operator Can Be Specified

The target of an operator is the host on which the operator runs. A host that can be the target of an operator must be configured with a touchpoint, a proxy touchpoint, or a host group. When determining the run settings for an operator, content designers typically base target host selection on how that target is configured.

Specify the target as indicated in the following list when the appropriate conditions are met:

Touchpoint

The target host has an Orchestrator or agent installed. When you specify a touchpoint as the target for an operator, and that touchpoint is enabled and associated with an active orchestrator or agent, CA Process Automation executes the operator on the specified touchpoint. If a targeted touchpoint is associated with only one agent or with an orchestrator, each execution targets the same host. If a targeted touchpoint is associated with multiple agents with the same priority, each execution could target a different agent on a different host.

Touchpoint Group

The operator is to run on all of the hosts that are associated with touchpoints in the specified touchpoint group.

<AgentID>

Exception to touchpoint: Specify the same AgentID for consecutive operators when the process requires the operator to run on the same host as the previous operator. Specify a specific AgentID when the process requires the operator to run on the same host each time it is run. A test run of the operator displays the AgentID string in the system dataset. You copy the AgentID string from the dataset into the Target field.

Proxy Touchpoint

The target host is specified as the remote host in a proxy touchpoint configuration. Typically, this host does not have an installed agent or an Orchestrator.

IP address or FQDN

The target host has an IP address or FQDN that can be matched to a pattern in a configured host group. Typically, this host does not have an installed agent or an Orchestrator.

Note: The host name can be used, but FQDN is preferred.

Important! Unexpected results can occur if you specify the target with a touchpoint that is named the same as a host in the environment. Unexpected results also occur if the agent associated with that touchpoint is inactive when you run the operator. In this case, CA Process Automation does not run the operator on the touchpoint. Instead, CA Process Automation tries to resolve the target (intended to be the touchpoint) to the name of a host in the environment. If it finds such a host name, CA Process Automation searches for another touchpoint, Orchestrator, proxy touchpoint, or host group that runs its operators on this host. CA Process Automation runs the operator on the first occurrence it finds.

Processing a Target Specified as an IP Address or FQDN

When you use an IP address or FQDN to specify the target host of an operator, CA Process Automation repeatedly queries the Domain Name Server (DNS). If you specify an FQDN as the target, the first search returns the IP addresses defined on the host with that FQDN. If you specify an IP address, the first search returns the host FQDN and IP addresses associated with that FQDN. Each subsequent search tests against all of the host specifiers that are retrieved in the initial search.

CA Process Automation looks for all of the ways that the target host is configured in the current CA Process Automation environment. A host group defines remote hosts with subnet and host name patterns. Therefore, a host group can include hosts with Orchestrators or agents that are mapped to touchpoints. A host group can also include individual hosts that are mapped to a proxy touchpoint. A host that is identified with an IP address or FQDN could be processed with a touchpoint name or proxy touchpoint name.

Some searches find any active Orchestrators or agents that are defined on the host that have enabled touchpoints. Another search finds any enabled proxy touchpoint on an active agent that is mapped to the host. A search finds enabled host groups on an active agent with host name patterns or IP address subnets that match the FQDN or IP address. When search criteria are met, these queries delay the start of the operator run. CA Process Automation runs the operator on the first found enabled touchpoint, proxy touchpoint, or host group that is running on an active Orchestrator or agent. The touchpoint, proxy touchpoint, or host group must belong to the current environment. When none of the search criteria are met, the operator fails and an error message is generated.

The Target area of the Execution Settings tab for operators contains a "Match target in Host Groups only" check box. Select this check box in the following cases:

- When specifying an IP address within a subnet that is specified in a host group.
- When specifying a host name that matches a pattern that is specified in a host group.

When you select the Look Up DNS Check box, CA Process Automation resolves the target name by doing a lookup in DNS. The lookup is for a record that matches a reference by a host group.

Consider the case where you clear this check box but the target IP address or FQDN exists. The following processing determines the host on which to run the operator:

1. The application searches the DNS for all of the identifiers for the specified target. If the target is specified as an FQDN, CA Process Automation searches for all the IP addresses for that FQDN. If the target is an IP address, CA Process Automation searches for the FQDN and for other IP addresses for that FQDN.
   - If the connection to the DNS fails, the application writes an error to the log file and exits.
   - If the query returns the FQDN and other IP addresses, the application uses the specified IP address and this result set for all subsequent searches.
   - If the query returns one or more IP addresses, the application uses the specified FQDN and the IP addresses in all subsequent searches.
2. The application searches for an active Orchestrator with an enabled touchpoint (current environment) that is installed on a host in the Step 1 search results.
   - If found, the application runs the operator on that Orchestrator touchpoint.
   - If not found, the application continues processing.

     Note: The application does not detect clustered Orchestrators. Hosts with clustered Orchestrators do not appear in the domain configuration.

3. The application searches for an active agent with an enabled touchpoint (current environment) that is installed on a host in the Step 1 search results.
   - If the host has an active agent with one enabled touchpoint, CA Process Automation uses that touchpoint to run the operator.
   - If the host has an active agent with multiple enabled touchpoints, CA Process Automation uses one of the enabled touchpoints to run the operator.
   - If not found, the application continues processing.
4. The application searches for an active agent that is connected to the remote target host. This agent must have an enabled proxy touchpoint or host group in the current environment. For host groups, the queries use pattern matching. The host group queries find whether the referenced subnet or host name pattern includes a host identifier in the Step 1 result set. For proxy touchpoints, the application searches for an exact match to any of the host identifiers that are retrieved in Step 1.
   - If the only match is a remote host referenced by a proxy touchpoint, the application runs the operator on that remote host.
   - If the only match is a remote host referenced by a host group, the application runs the operator on that remote host.
   - If the application finds multiple matches, it runs the operator on the first remote host found that is referenced by either a proxy touchpoint or a host group.
   - If there are no matches to the search criteria, the application continues processing.

5. The application determines why the operator did not run. Then, the application fails the operator with a failure message.

   - Condition: None of the identifiers that are retrieved in Step 1 belongs to any host in the current environment.

     Results: The application fails the operator with the following message:

     ```
     Message could not be posted to the node.
     ```

   - Condition: One or more identifiers that are retrieved in Step 1 belong to a host where all Orchestrators and agents are inactive.

     Results: The operator fails with the following message:

     ```
     Message could not be posted to the node.
     ```

   - Condition: One or more Step 1 identifiers belong to a host with all disabled touchpoints or is referenced with only disabled proxy touchpoints or host groups.

     Results: The application returns a disabled touchpoint or host group in the TouchpointName field of the operator dataset system output variables. The application fails the operator with the following error message:

     ```
     Node is disabled.
     ```

Note: Consider the exception where the user specifies a host name or IP address in the Target field of an *Invoke Java* operator. CA Process Automation looks first for a matching agent host. If successful, the Invoke Java operator runs on the agent host. If the search fails, the application looks for a matching Orchestrator host. If found, it attempts to run but the Invoke Java operator fails because it cannot run on an Orchestrator.

More information:

Target Settings (see page 215)

Processing a Target Specified as the ID of an Agent or Orchestrator

Suppose you want an operator to target the same host every time it runs. In such a case, you can specify the AgentID data as a target. Unique AgentID data is set during the installation of each orchestrator and each agent. At runtime, CA Process Automation retrieves the ID for the orchestrator or agent and saves it to the *AgentID* field in the system dataset.

After a test execution of an operator, you can copy the AgentID string that is displayed in the system dataset into the *Target* field.

When you need two consecutive operators to run on the same agent host, specify the same AgentID as the target for both operators. You can use AgentID as an alternative target for any given touchpoint.

Important! Use of AgentID refers to the local host where the agent or orchestrator is installed. When you specify AgentID, CA Process Automation does not use any proxy touchpoint or host group running on the agent to run on remote hosts. Consider the case where *Operator_1* executes on a proxy touchpoint that is configured on an agent with a specified AgentID. Do not use this AgentID as the target for *Operator_2*. If the AgentID is specified, CA Process Automation does not execute on that proxy touchpoint for *Operator_2*.

CA Process Automation fails the operator when encountering any of the following conditions:

- The target agent or orchestrator is inactive. CA Process Automation posts the following message:

  ```
  Message could not be posted to the node.
  ```

- All touchpoints that are mapped to the target are disabled. CA Process Automation posts the following message with one of the disabled touchpoints:

  ```
  Node is disabled.
  ```

Note: The Invoke Java operator can target only agents. The Invoke Java operator cannot run on an Orchestrator. An ID in the Target field of the Invoke Java operator must resolve to an agent (not an Orchestrator), or the operator fails.

Use Case: Track Recovered Processes through Logs

The following scenario is an alternative to opening many processes to identify the ones that were automatically recovered. In this scenario, you examine the log files.

Assumptions:

- An agent is mapped to two touchpoints.
- Over 1000 processes are running concurrently.
- Operators Auto Recovery is configured for the touchpoints.

Scenario:

1. The agent suddenly becomes inactive.
2. While the agent is inactive, 300 of the 1000 processes try to run operators on the touchpoints.
3. The 300 processes go into BLOCKED state.
4. The agent becomes active.
5. Operators recovery, which is initiated automatically, recovers the 300 processes.
6. Later you notice that 1000 processes finished running. You open one of them to examine its logs and learn that it was automatically recovered.
7. Complete one of the following actions to find out which of the 1000 processes that finished running finished after being recovered:
   - Open each of the 1000 processes to determine which were automatically recovered.
   - Examine the log file.

   Note: The next topic describes the preferred alternative, examining the log file.

Operators Auto Recovery Example

If Operators Auto Recovery is selected (the default), you do not have to initiate the recovery. Each Orchestrator in the environment detects that the agent or Orchestrator is active and runs the recovery. Each Orchestrator runs recovery on any enabled touchpoint, host group, or Orchestrator that is mapped to the agent or Orchestrator in the environment.

When an agent is mapped to many touchpoints, the Orchestrators run the recovery on all enabled touchpoints that are mapped to the agent. When an agent or Orchestrator becomes active, the Domain Orchestrator broadcasts the changes to the other Orchestrators. In a clustered setup, only the master node runs the recovery.

Example: Automatic Recovery

Consider the following scenario, in which an agent is mapped to the following touchpoints and host groups in an environment:

- TP_user01-w500,
- TP_user01-w500_2
- TP_user01-w500_3
- TP_manyAgents
- HG_user01-w500

Assume that all touchpoints and host groups are enabled and that only the following touchpoints and host groups are set to Operators Auto Recovery:

- TP_user01-w500
- TP_user01-w500_3
- HG_user01-w500

When the agent becomes active, each Orchestrator in the environment tries to run the automatic recovery on the three touchpoints.

Assume there are two Orchestrators in the environment:

Orchestrator1 log file contains:

- Message to signal the agent that became active again:

```
2010-06-28 19:22:59,984 DEBUG
[com.optinuity.c2o.c2oserver.ServerManager] [aListenerThread] Checking whether
to initiate Touchpoint/Orchestrator Recovery of 'System_Error' operators in
'Waiting' or 'Running' or 'Blocked' Processes (with auto recovery flag set), that
ran against Agent/Server ID: f3492322-5517-4a21-8a19-92838ccb3f65
```

- Orchestrator mapped the agent to HG_user01-w500:

```
2011-03-28 19:22:59,984 INFO
[com.optinuity.c2o.c2oserver.ServerManager] [aListenerThread] Initiating
Touchpoint/Orchestrator: HG_user01-w500 Recovery... Recovering 'System_Error'
Operators that ran against: HG_user01-w500 and belong to 'Waiting' or 'Running'
or 'Blocked' Processes (with auto recovery flag set).
2011-03-28 19:22:59,984 DEBUG
[com.optinuity.c2o.workflowengine.C2oWorkFlowEngine] [aListenerThread]
recoverSystemErrorProcesses: Retrieve the processes (with enabled auto recovery
flag) that are in 'Waiting' or ‘Running’ or 'Blocked' states and contain operators
in 'System Error' when ran against: HG_user01-w500
```

- Orchestrator signals 0 processes to be recovered for HG_user01-w500:

```
2011-03-28 19:23:00,000 WARN
[com.optinuity.c2o.workflowengine.C2oWorkFlowEngine] [aListenerThread]
recoverSystemErrorProcesses: Retrieved - 0 processes to recover for
HG_user01-w500 recovery.
2011-03-28 19:23:00,000 INFO
[com.optinuity.c2o.c2oserver.ServerManager] [aListenerThread] Found no
Processes to recover for Touchpoint/Orchestrator: HG_user01-w500 Recovery.
```

- Orchestrator mapped the agent to TP_user01-w500:

```
2011-03-28 19:23:00,000 INFO
[com.optinuity.c2o.c2oserver.ServerManager] [aListenerThread] Initiating
Touchpoint/Orchestrator: TP_user01-w500 Recovery... Recovering 'System_Error'
Operators that ran against: TP_user01-w500 and belong to 'Waiting' or 'Running'
or 'Blocked' Processes (with auto recovery flag set).
2011-03-28 19:23:00,000 DEBUG
[com.optinuity.c2o.workflowengine.C2oWorkFlowEngine] [aListenerThread]
recoverSystemErrorProcesses: Retrieve the processes (with enabled auto recovery
flag) that are in 'Waiting' or ‘Running’ or 'Blocked' states and contain operators
in 'System Error' when ran against: TP_user01-w500.
```

- Orchestrator signals 1 process to be recovered for TP_user01-w500:

```
2011-03-28 19:23:00,015 WARN
[com.optinuity.c2o.workflowengine.C2oWorkFlowEngine] [aListenerThread]
recoverSystemErrorProcesses: Retrieved - 1 processes to recover for
TP_user01-w500 recovery.
```

- Orchestrator queues recovery request for the 1 process (ROID: 2889) to be recovered for TP_user01-w500:

```
2011-03-28 19:23:00,015 WARN
[com.optinuity.c2o.workflowengine.C2oWorkFlowEngine] [aListenerThread]
recoverSystemErrorProcesses: Queueing recovery of process with ROID - 2889 for
TP_user01-w500 recovery.
2011-03-28 19:23:00,015 DEBUG
[com.optinuity.c2o.workflowengine.C2oWorkFlowEngine] [aListenerThread]
recoverSystemErrorProcesses: Special Response for process recovery. ROID is 2889
UUID is 3fe95f08-a347-4d6d-a0a3-b3639836e130
2011-03-28 19:23:00,015 DEBUG
[com.optinuity.c2o.workflowengine.C2oWorkFlowEngine] [aListenerThread]
recoverSystemErrorProcesses: Special Response for process recovery was posted.
ROID is 2889 UUID is 3fe95f08-a347-4d6d-a0a3-b3639836e130
2011-03-28 19:23:00,015 INFO
[com.optinuity.c2o.c2oserver.ServerManager] [aListenerThread] Queued requests
for Touchpoint: TP_user01-w500 Recovery... Recovering Processes with the
following ROIDs: 2889
```

- Orchestrator signals that Touchpoint TP_user01-w500_2 is mapped to the agent but it is not set to automatic recovery:

```
2011-03-28 19:23:00,015 DEBUG
[com.optinuity.c2o.c2oserver.ServerManager] [aListenerThread] Touchpoint:
TP_user01-w500_2 mapped to Agent: f3492322-5517-4a21-8a19-92838ccb3f65 is not
set to Automatic Recovery of 'System_Error' operators in 'Waiting' or ‘Running’
or 'Blocked' Processes.
```

- Orchestrator mapped the agent to TP_user01-w500_3:

```
2011-03-28 19:23:00,015 INFO
[com.optinuity.c2o.c2oserver.ServerManager] [aListenerThread] Initiating
Touchpoint/Orchestrator: TP_user01-w500_3 Recovery. Recovering 'System_Error'
Operators that ran against: TP_user01-w500_3 and belong to 'Waiting' or 'Running'
or 'Blocked' Processes (with auto recovery flag set).
2011-03-28 19:23:00,015 DEBUG
[com.optinuity.c2o.workflowengine.C2oWorkFlowEngine] [aListenerThread]
recoverSystemErrorProcesses: Retrieve the processes (with enabled auto recovery
flag) that are in 'Waiting' or ‘Running’ or 'Blocked' states and contain operators
in 'System Error' when ran against:: TP_user01-w500_3
```

- Orchestrator signals 0 processes to be recovered for TP_user01-w500_3:

```
2011-03-28 19:23:00,015 WARN
[com.optinuity.c2o.workflowengine.C2oWorkFlowEngine] [aListenerThread]
recoverSystemErrorProcesses: Retrieved - 0 processes to recover for
TP_user01-w500_3 recovery.
2011-03-28 19:23:00,031 INFO
[com.optinuity.c2o.c2oserver.ServerManager] [aListenerThread] Found no
Processes to recover for Touchpoint/Orchestrator: TP_user01-w500_3 Recovery.
```

- Orchestrator signals that Touchpoint TP_manyAgents is mapped to the agent but it is not set to automatic recovery:

```
2011-03-28 19:23:00,031 DEBUG
[com.optinuity.c2o.c2oserver.ServerManager] [aListenerThread] Touchpoint:
TP_manyAgents mapped to Agent: f3492322-5517-4a21-8a19-92838ccb3f65 is not set
to Automatic Recovery  of 'System_Error' operators in 'Waiting' or ‘Running’ or
'Blocked' Processes
```

- Orchestrator picks up the recovery request from the queue for process 2889:

```
2011-03-28 19:23:30,015 INFO
[com.optinuity.c2o.workflowengine.FlowManagerCache] [onPool Worker-3]
Registering flow manager: 2889
2011-03-28 19:23:30,015 INFO
[com.optinuity.c2o.workflowengine.WorkflowManager] [onPool Worker-3] Handling
Response: [$$TouchpointRecovery&&]. flow id is 2889
2011-03-28 19:23:30,171 INFO
[com.optinuity.c2o.workflowengine.WorkflowManager] [onPool Worker-3] Start
Recovery of Process with ROID:2889, for TP_user01-w500 Recovery.
```

- Orchestrator resets the operator(s) with System_Error when ran against TP_user01-w500 within the process 2889:

```
2011-03-28 19:23:30,171 INFO
[com.optinuity.c2o.workflowengine.WorkflowManager] [onPool Worker-3] Resetting
System_Error Operators in Process with ROID: 2889 for TP_user01-w500 Recovery.
2011-03-28 19:23:30,203 INFO
[com.optinuity.c2o.workflowengine.WorkflowManager] [onPool Worker-3] Resetting
Operator with ROID: 2912 in Process with ROID: 2889 for TP_user01-w500 Recovery.
```

- Orchestrator resumes the entire process 2889 after resetting the appropriate operators:

```
2011-03-28 19:23:30,343 INFO
[com.optinuity.c2o.workflowengine.WorkflowManager] [onPool Worker-3] Resuming
Process with ROID: 2889 for TP_user01-w500 Recovery.
2011-03-28 19:23:30,343 INFO
[com.optinuity.c2o.workflowengine.WorkflowManager] [onPool Worker-3] Resuming
WorkFlow - ROID is 2889
2011-03-28 19:23:30,703 INFO
[com.optinuity.c2o.workflowengine.WorkflowManager] [onPool Worker-3] Done
resuming Process with ROID: 2889 for TP_user01-w500 Recovery.
```

  Process 2889 completes successfully. Because the recovery process started automatically, the process logs indicate that the engine initiated the recovery. For manual recovery, logs include the name of the user who started the recovery.

Orchestrator2 (OrchestratorD610b) log file contains:

- The following for OrchestratorD610b is similar to Orchestrator:

```
2011-03-28 19:23:41,444 INFO
[com.optinuity.c2o.c2oserver.ServerManager] [aListenerThread] Initiating
Touchpoint/Orchestrator: HG_user01-w500 Recovery. Recovering 'System_Error'
Operators that ran against: HG_user01-w500 and belong to 'Waiting' or 'Running'
or 'Blocked' Processes (with auto recovery flag set).
2011-03-28 19:23:41,444 WARN
[com.optinuity.c2o.workflowengine.C2oWorkFlowEngine] [aListenerThread]
recoverSystemErrorProcesses: Retrieved - 0 processes to recover for
HG_user01-w500 recovery.
2011-03-28 19:23:41,444 INFO
[com.optinuity.c2o.c2oserver.ServerManager] [aListenerThread] Found no
Processes to recover for Touchpoint/Orchestrator: HG_user01-w500 Recovery...
2011-03-28 19:23:41,444 INFO
[com.optinuity.c2o.c2oserver.ServerManager] [aListenerThread] Initiating
Touchpoint/Orchestrator: TP_user01-w500 Recovery... Recovering 'System_Error'
Operators that ran against: TP_user01-w500 and belong to 'Waiting' or 'Running'
or 'Blocked' Processes (with auto recovery flag set).
2011-03-28 19:23:41,444 WARN
[com.optinuity.c2o.workflowengine.C2oWorkFlowEngine] [aListenerThread]
recoverSystemErrorProcesses: Retrieved - 0 processes to recover for
TP_user01-w500 recovery.
2011-03-28 19:23:41,444 INFO
[com.optinuity.c2o.c2oserver.ServerManager] [aListenerThread] Found no
Processes to recover for Touchpoint/Orchestrator: TP_user01-w500 Recovery...
2011-03-28 19:23:41,444 INFO
[com.optinuity.c2o.c2oserver.ServerManager] [aListenerThread] Initiating
Touchpoint/Orchestrator: TP_user01-w500_3 Recovery... Recovering 'System_Error'
Operators that ran against: TP_user01-w500_3 and belong to 'Waiting' or 'Running'
or 'Blocked' Processes (with auto recovery flag set).
```

- OrchestratorD610b finds 1 process to recover for TP_user01-W500_3 (process ROID: 541) and queues its recovery request:

```
2011-03-28 19:23:41,444 WARN
[com.optinuity.c2o.workflowengine.C2oWorkFlowEngine] [aListenerThread]
recoverSystemErrorProcesses: Retrieved - 1 processes to recover for
TP_user01-w500_3 recovery.
2011-03-28 19:23:41,444 WARN
[com.optinuity.c2o.workflowengine.C2oWorkFlowEngine] [aListenerThread]
recoverSystemErrorProcesses: Queueing recovery of process with ROID - 541 for
TP_user01-w500_3 recovery.
2011-03-28 19:23:41,444 INFO
[com.optinuity.c2o.c2oserver.ServerManager] [aListenerThread] Queued requests
for Touchpoint: TP_user01-w500_3 Recovery... Recovering Processes with the
following ROIDs: 541
2011-03-28 19:23:41,444 WARN
[com.optinuity.c2o.workflowengine.C2oWorkFlowEngine] [aListenerThread]
recoverSystemErrorProcesses: Retrieved - 1 processes to recover for
TP_user01-w500_3 recovery.
2011-03-28 19:23:41,444 WARN
[com.optinuity.c2o.workflowengine.C2oWorkFlowEngine] [aListenerThread]
recoverSystemErrorProcesses: Queueing recovery of process with ROID - 541 for
TP_user01-w500_3 recovery.
2011-03-28 19:23:41,444 INFO
[com.optinuity.c2o.c2oserver.ServerManager] [aListenerThread] Queued requests
for Touchpoint: TP_user01-w500_3 Recovery... Recovering Processes with the
following ROIDs: 541
```

- OrchestratorD610b picks up the recovery request, resets the Operators with System_Error, which ran against TP_user01-w500_3, in the process and resumes the process:

```
2011-03-28 19:24:11,461 INFO
[com.optinuity.c2o.workflowengine.WorkflowManager] [nPool Worker-63] Handling
Response: [$$TouchpointRecovery&&]. flow id is 541
2011-03-28 19:24:11,804 INFO
[com.optinuity.c2o.workflowengine.WorkflowManager] [nPool Worker-63] Start
Recovery of Process with ROID:541, for TP_user01-w500_3 Recovery.
2011-03-28 19:24:11,820 INFO
[com.optinuity.c2o.workflowengine.WorkflowManager] [nPool Worker-63] Resetting
System_Error Operators in Process with ROID: 541 for TP_user01-w500_3 Recovery.
2011-03-28 19:24:11,883 INFO
[com.optinuity.c2o.workflowengine.WorkflowManager] [nPool Worker-63] Resetting
Operator with ROID: 564 in Process with ROID: 541 for TP_user01-w500_3 Recovery.
2011-03-28 19:24:12,039 INFO
[com.optinuity.c2o.workflowengine.C2OSvcIconInstanceRefObject] [nPool
Worker-63] Inside Update ConnectorTraversed, Current/destination is
Start_System_Process_1 Source is Start_1
2011-03-28 19:24:12,117 INFO
[com.optinuity.c2o.workflowengine.WorkflowManager] [nPool Worker-63] Resuming
Process with ROID: 541 for TP_user01-w500_3 Recovery.
2011-03-28 19:24:12,117 INFO
[com.optinuity.c2o.workflowengine.WorkflowManager] [nPool Worker-63] Resuming
WorkFlow - ROID is 541
2011-03-28 19:24:12,133 INFO
[com.optinuity.c2o.workflowengine.C2OSvcIconInstanceRefObject] [nPool
Worker-63] Inside Update ConnectorTraversed, Current/destination is
Start_System_Process_1 Source is Start_1
2011-03-28 19:24:12,648 INFO
[com.optinuity.c2o.workflowengine.WorkflowManager] [nPool Worker-63] Done
resuming Process with ROID: 541 for TP_user01-w500_3 Recovery.
```

Index

A

agenda
  Operator support in, • 372
  see schedules • 370
AgentID
  how processed • 463
  when to use • 459
archival policy
  archival policy, defining manually • 64
attached mode
  calling Processes recursively in, • 120
automation object
  baselining for export • 426
  checking in • 79
  checking out • 78
  copying • 83
  moving • 394
  opening • 77
  ownership • 64
  packaging • 432
  relative path support • 394
  versioning • 71

C

calendar
  building, rule • 358
  creating, object • 356
content package
  use in testing and deployment • 27
custom icon
  implementing for a custom operator • 185
  implementing for an operator in a process • 184
custom operator
  invisible parameter • 158
  pre and post execution code • 165
  protected • 175
custom operator group
  assign to custom operator • 167
  define • 167

D

dataset
  accessing by keywords • 232
  creating • 191
  reading operating system value into, • 200
  returning variables to parent Process • 120
  types • 191
  using, in expressions • 236
  variable name assistance • 213
date and time format
  specifying • 451
Default Process Watch
  starting • 407
detached mode
  starting Process instance in, • 120
documentation
  auto generating • 420
  roles in generating • 29

E

exception handling
  adding rule • 401
  types and priorities • 399
expressions
  data types • 224
  expressions, using Datasets in, • 21
  in JavaScript statements • 247
  masks • 455
  reserved words • 222
  system paths in, • 255

F

File trigger
  example • 382
  monitoring dataset for • 384
  XML content for, • 386

I

icon
  creating • 178
indexed field
  creating • 197
interaction request forms
  defined • 259

J

Javascript
  arithmetic operators • 230

operators • 228

L

Library Browser
customize view • 53
search • 53
user interface • 37
link
breaking • 105
logging on
<Global> in CA EEM • 51
after browsing to CA Process Automation • 32

M

Mail trigger
example • 382
monitoring dataset for • 384
XML content for • 386
masks
example • 458
syntax • 455
multi-tenancy
creating policy for, • 133

O

OasisConfig properties
oasis.snmptrigger.service.port • 387
Operator recovery
automated example • 465
operators
adding to a Process • 96
assigning values to, • 208
configuring • 144
controlling execution • 412
defining parameters for, • 202
JavaScript • 228
linking • 100
logical and derivation • 97
looping • 105
simulating processing • 415
using in Agendas • 372
using in Calendars • 360
Out-of-the-Box Content
browse • 50

P

package • 432
adding objects to • 434
password
changing in CA EEM • 51
process
aborting • 412
adding comments • 127
breakpoints • 409
debugging • 408
deployment • 27
execution rules • 396
generating documentation about, • 420
generating documentation for, • 420
icons representing states • 408
initializing • 134
inline • 120
Process, defining run duration for • 69
resetting • 412
starting a child • 117
starting directly • 405
starting while editing • 407
suspending • 408
terminating • 99
Process Control module
Process version run by • 126

Q

quotation marks pair
as Operator parameter notation • 209

R

regular expressions
masks • 455
strings in Dataset • 196
Release Versions
setting • 425
requirements
analyzing • 14
resources
checking for availability • 353
creating • 345
using quotas • 343
Return on Investment (ROI)
Return on Investment (ROI), Specifying Properties • 66
Runtime Security
defined • 397
Runtime Security, setting properties • 68
setting process properties • 399

S

scripts
comments • 256
including common resources in • 256
security, application
setting for Automation Objects • 64
setting as Current or • 76
SNMP trigger
changing the listening port • 387
input • 386
monitoring Dataset for • 388
SOAP
use in starting Processes from external applications • 382
statements, JavaScript
conditional • 250
iterative • 252
types • 247
swim lanes
adding • 125
processing • 123

T

tags
tags, assign to objects • 63
target of Operator
critieria for selecting • 459
how Agent ID or Operator ID is processed • 463
how FQDN is processed • 460
how IP address is processed • 460
settings • 215
timeout
setting for a Resource Operator • 351
trigger
alternative • 382

V

variable
assignment • 249
declaration • 249
view or change for an Object • 63