CA Process Automation

Content Designer Guide

Service Pack 04.0.01

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.

Documentation Changes

The following documentation updates have been made since the last release of this documentation:

Forms

The following topics have been added or enhanced to support form designers:

- Monitor Start Request Form Instances and Process Instances (see page 244)
- Form Elements (see page 248)
- Form Element Properties (see page 251)
- View Form Properties in a Separate Window (see page 259)
- Form Element Events (see page 260)
- Form Element Functions (see page 262)
- Create a Simple Form with Basic Functions (see page 283)
- Initialize Form Variables (see page 287)

Processes

The following topics have been added or enhanced to support process designers:

- Operator Status Icons (see page 164)
- Invisible Parameter Option (see page 151)

Datasets

The following topics have been added or enhanced to support developers working with datasets:

- Define Dataset Pages and Variables (v4) (see page 177)
- Modify a Dataset (see page 183)
- View a Dataset Expression (see page 184)

General

The following topics have been added or enhanced to support the general user experience:

- Configure User Settings (see page 35)
- Web Browsers (see page 53)

Contents

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

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 16)

Identify and characterize the automation processes required at your organization.

Phase II: Design and Implementation (see page 19)

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 29)

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 56) 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:

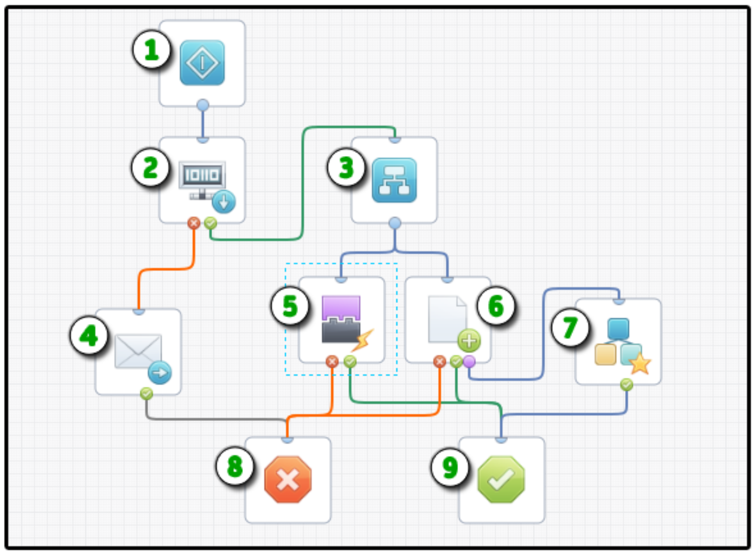
- [ ] Identify the automation objects and process operators to implement the process.
- [ ] Configure all operators and logical steps, including all tasks, parameters, outcomes, resources, and dependencies. Define any required runtime configuration information (such as user names, passwords, and file 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 out a process step by step, 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 dependency 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 be used to indicate the first step in an independent branch.
2	**Get SNMP Variable**: This operator returns the value of an SNMP variable.
3	**Or Operator**: *Or* and *And* operators control steps according to logic. For example, do step A *or* do step B.
4	**Send Email**: Use this operator to notify other users by email.

Item:	Description:
5	**External Interfaces**: You can use operators that are included with connecting modules to interface with other applications. In this example, a UCF Storage Volume Update operator interfaces with CA Catalyst.
6	**Write File**: This operator writes a dataset variable to a file.
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 execution of all branches in a process. The Stop Failure operator 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 also stops execution of all branches in a process. The Stop Success operator 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 33)
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 *CA Process Automation Reference Guide*.

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:

Access Dataset Fields in Expressions (see page 223)
Start Request Forms (see page 243)

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 289)

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 create your own *Custom Operator* objects that are based 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 settings for a task**: You can preconfigure custom operators to perform certain 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 also define packages to support common enterprise applications.

You can modify your own custom operator and the original operator does not change.

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 335).
- **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 301).
- Scheduling tasks with schedule objects is described in Schedules (see page 316).

More information:

Calendars (see page 301)

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 243)

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 327).

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:

Process Watch Objects (see page 336)

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 *PAMUsers* 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

Deployment of your solution to production includes the following activities:

- ☐ Assemble the related automation objects for this solution into a folder, multiple folders, or a package spanning multiple folders.
- ☐ Test, debug, and verify all content for this particular solution or project.
- ☐ Export the solution from the development environment.
- ☐ Import the solution to a production environment.
- ☐ Provide instructions and required training for users to activate schedules and use forms to start processes on demand.

More information:

Copy Objects Using Export and Import (see page 78)

Assemble the Solution

Assemble the related automation objects for this solution in a folder, folders, or package spanning multiple folders. Use a package when your solution spans multiple objects and folders.

Test Components

Testing is an important part of deploying the package or folder that represents the processes you want to automate. Verify that all components of the solution work before it is allowed to become operational in the production environment.

CA Process Automation has several features that assist in testing processes:

- To have full control over a process and test it during runtime, start the process in a *suspended* state. For example, open the suspended process in a process watch and observe it as it runs. You can start, pause, restart, and stop each process instance.
- Set breakpoints to pause and debug steps in a process.
- *Simulation* mode allows you to define specific outcomes for a step. You get the simulated results of each step without performing any real actions.

Testing takes place on two different levels. On one level, you test the automation objects and processes. On another level, you test the deployment of the automation content. For example, the deployment as a whole can pass all tests and can be operational while a specific form is still being redesigned and tested. Many terms reflect this difference including unit testing versus solution testing, internal release candidate build versus a customer proof-of-concept build, or development versus production. Periodically retest rarely executed mission-critical processes such as system failovers to verify that the processes and users stay fully current and functional in the production environment. Periodic testing prevents incidents from becoming full-blown crises.

More information:

Run Processes Interactively (see page 348)
Debug a Process (see page 352)
Simulate Processing of Operators (see page 359)

Export a Package

After testing a solution, export the folder, folders, or package of related automation content to an XML file. Include all required automation objects and folders from the development environment. Production administrators deploy the solution by importing this XML file into the production environment.

Specify Import Instructions

To help facilitate the deployment of the 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 Start Processes On Demand Using a Form

Chapter 2: Getting Started

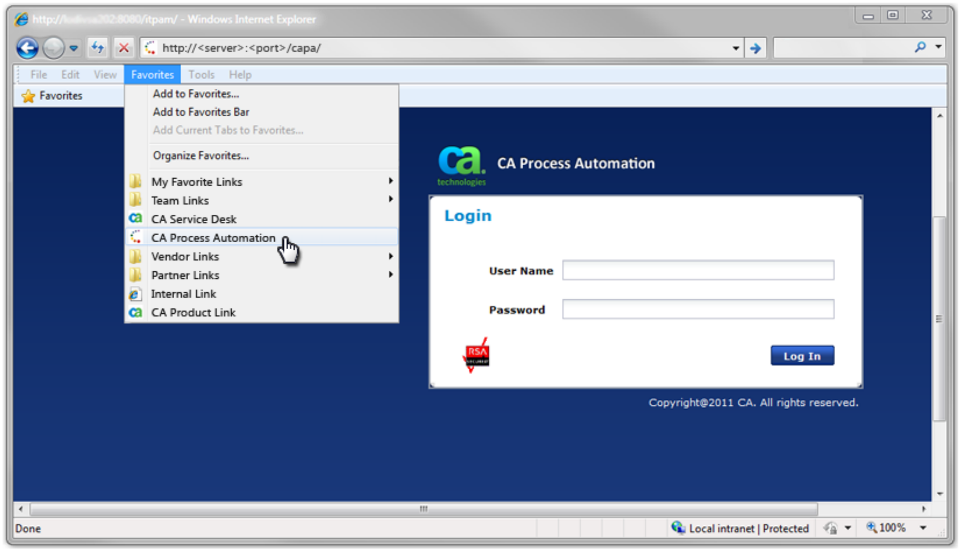
This section orients you to CA Process Automation and provides basic information about logging in and becoming familiar with the user interface.

This section contains the following topics:

Log In to CA Process Automation

After an administrator has installed and configured the CA Process Automation web interface, all users can begin working with the application. Application administrators can use the Configuration tab to perform additional installation and configuration steps. Content architects and designers can create objects and design processes. And users can start process forms or run reports.

Use the CA Process Automation user interface in a web browser for all your development and testing tasks. For security, you are prompted to log in.



Follow these steps:

1. Open a supported web browser.
2. Enter the URL for your CA Process Automation deployment. Configure the URL in one of the following ways, depending on whether the CA Process Automation orchestrator was installed with secure communication enabled over SSL. You can enter either the hostname or the IP address of the CA Process Automation domain orchestrator.

   ■ For secure communication, use the following syntax:
   `https://<hostname_or_IPaddress>:8443/itpam`

   ■ For nonsecure communication, use the following syntax:
   `http://<hostname_or_IPaddress>:8080/itpam`

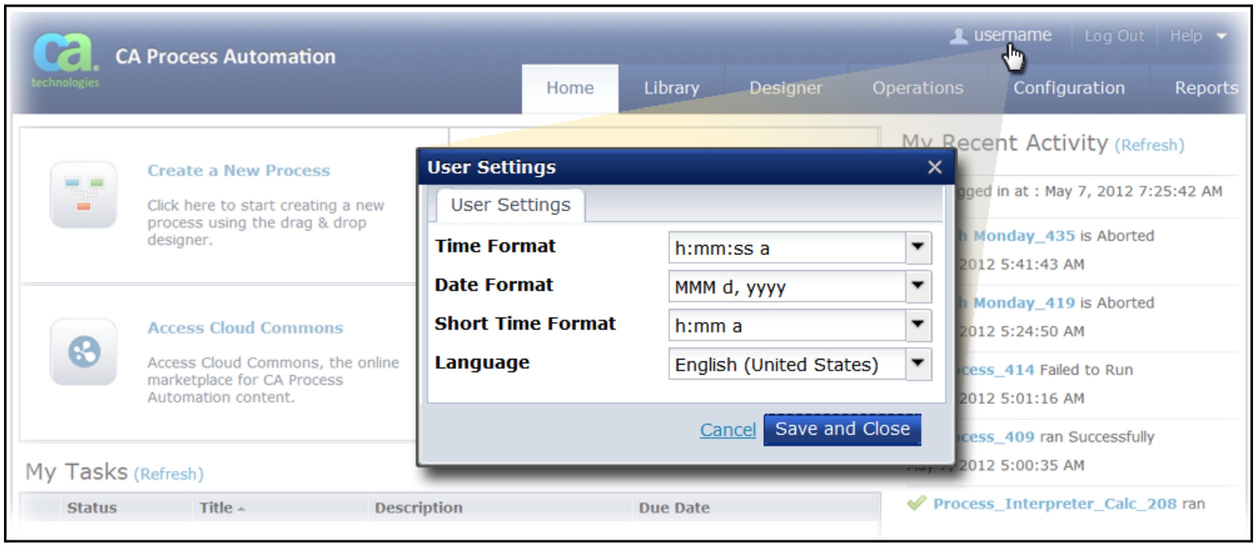
   For example, you could click a predefined link (*Bookmark* or *Favorite*) for CA Process Automation that is mapped to a server by hostname and port.

   The CA Process Automation login page appears.
3. Enter your user name and password and click Log In.

The CA Process Automation interface appears. If your login fails, verify the domain orchestrator and the CA EEM directory server (for CA Process Automation) are running on the host computer.

Configure User Settings

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



Time Format	h:mm:ss a
Date Format	MMM d, yyyy
Short Time Format	h:mm a
Language	English (United States)

Follow these steps:

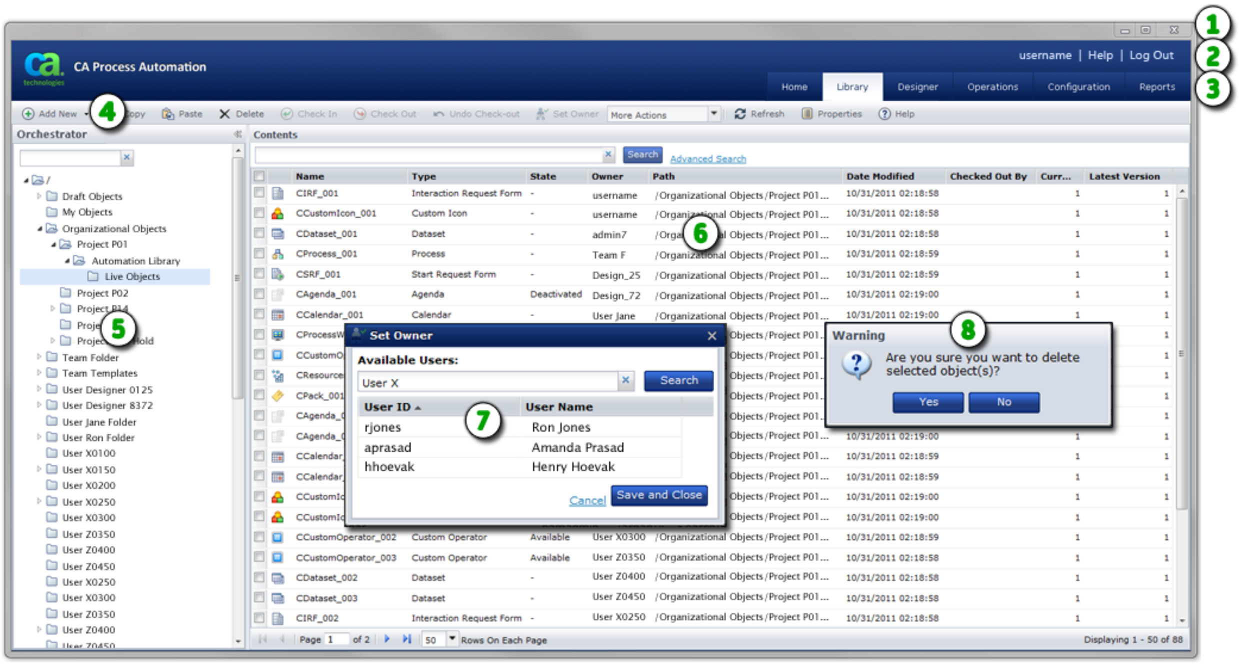
1. From the list of links at the top of the main page, click your username.
2. On the User Settings dialog, indicate your display preferences for the date and time formats, and language.
3. Click Save and Close.

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 that has 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 appear throughout the application making it easy to use. For example, sort a list of entries or configure which columns appear using the same basic steps.



Name	Type	State	Owner	Path	Date Modified	Checked Out By	Curr...	Latest Version
CIRF_001	Interaction Request Form	-	username	/Organizational Objects/Project P01...	10/31/2011 02:18:58		1	1
CCustomIcon_001	Custom Icon	-	username	/Organizational Objects/Project P01...	10/31/2011 02:18:58		1	1
CDataset_001	Dataset	-	admin7	/Organizational Objects/Project P01...	10/31/2011 02:18:58		1	1
CProcess_001	Process	-	Team F	/Organizational Objects/Project P01...	10/31/2011 02:18:59		1	1
CSRF_001	Start Request Form	-	Design_25	/Organizational Objects/Project P01...	10/31/2011 02:18:59		1	1
CAgenda_001	Agenda	Deactivated	Design_72	/Organizational Objects/Project P01...	10/31/2011 02:19:00		1	1
CCalendar_001	Calendar	-	User Jane	/Organizational Objects/Project P01...	10/31/2011 02:19:00		1	1
CProcessW				Objects/Project P01...				1
CCustomO				Objects/Project P01...				1
CResource				Objects/Project P01...				1
CPack_001				Objects/Project P01...				1
CAgenda_0				Objects/Project P01...				1
CAgenda_0				Objects/Project P01...	10/31/2011 02:19:00		1	1
CCalendar_				Objects/Project P01...	10/31/2011 02:18:59		1	1
CCalendar_				Objects/Project P01...	10/31/2011 02:18:59		1	1
CCustomIc				Objects/Project P01...	10/31/2011 02:19:00		1	1
CCustomIc				Objects/Project P01...	10/31/2011 02:19:00		1	1
CCustomOperator_002	Custom Operator	Available	User X0300	/Organizational Objects/Project P01...	10/31/2011 02:18:59		1	1
CCustomOperator_003	Custom Operator	Available	User Z0350	/Organizational Objects/Project P01...	10/31/2011 02:18:58		1	1
CDataset_002	Dataset	-	User Z0400	/Organizational Objects/Project P01...	10/31/2011 02:18:58		1	1
CDataset_003	Dataset	-	User Z0450	/Organizational Objects/Project P01...	10/31/2011 02:18:58		1	1
CIRF_002	Interaction Request Form	-	User X0250	/Organizational Objects/Project P01...	10/31/2011 02:18:58		1	1

Page 1 of 2 | 50 Rows On Each Page | Displaying 1 - 50 of 88

Set Owner

Available Users: User X

User ID	User Name
rjones	Ron Jones
aprasad	Amanda Prasad
hhoevak	Henry Hoevak

Cancel | Save and Close

Warning

Are you sure you want to delete selected object(s)?

Yes | No

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 example, Minimize, Maximize, Restore, and Close. Your browser also displays its own menus, toolbars, panes, and search areas. Browser features can sometimes be used to supplement built-in CA Process Automation features. For example, you can refresh a page or you can adjust the view magnification (Zoom) using your browser.
2	**Links**: CA Process Automation provides a series of the most common application links including User Settings, Help, and Log Out. Individual pages will include appropriate links to related content.

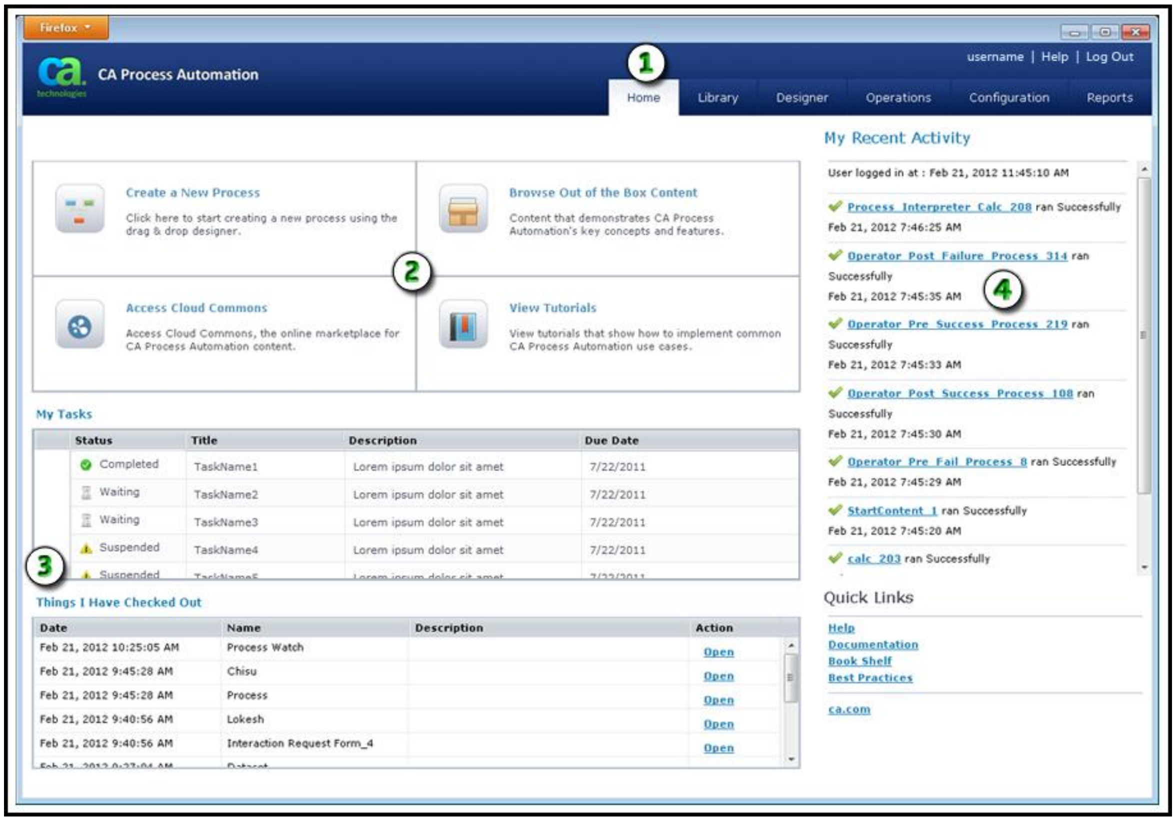
Item:	Description:
3	**Main Application Tabs**: Click a tab to focus on a specific section of CA Process Automation. In the sample image, the Library tab is selected. The application displays folders and objects in the Library Browser.
4	**Toolbar**: Pages and dialogs can sometimes display their own toolbars with an appropriate set of tool buttons and icons.
5	**Pane**: A pane provides a way to divide a window or page. In this example, the Library Browser pane appears at left. This pane features a filter input field and an expandable folder hierarchy. The detailed information for an entry you select in the pane is displayed on the main page.
6	**Main Page**: The main area of a page displays the essential information that you need. This data is often presented in a table, list, form, design canvas, or chart. This area can sometimes also be further divided using palettes, tabs, panes, or other visual controls.
7	**Dialog**: When you click certain buttons, apply actions, or issue commands, the application often presents dialog boxes to collect additional input from you.
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 you with necessary feedback. However, some messages can display critical warnings that are designed to protect your data. Error messages provide useful information that can be combined with log file data to troubleshoot problems.

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.



Create a New Process
Click here to start creating a new process using the drag & drop designer.

Browse Out of the Box Content
Content that demonstrates CA Process Automation's key concepts and features.

Access Cloud Commons
Access Cloud Commons, the online marketplace for CA Process Automation content.

View Tutorials
View tutorials that show how to implement common CA Process Automation use cases.

My Tasks

Status	Title	Description	Due Date
✓ Completed	TaskName1	Lorem ipsum dolor sit amet	7/22/2011
Waiting	TaskName2	Lorem ipsum dolor sit amet	7/22/2011
Waiting	TaskName3	Lorem ipsum dolor sit amet	7/22/2011
⚠ Suspended	TaskName4	Lorem ipsum dolor sit amet	7/22/2011
⚠ Suspended			

Things I Have Checked Out

Date	Name	Description	Action
Feb 21, 2012 10:25:05 AM	Process Watch		Open
Feb 21, 2012 9:45:28 AM	Chisu		Open
Feb 21, 2012 9:45:28 AM	Process		Open
Feb 21, 2012 9:40:56 AM	Lokesh		Open
Feb 21, 2012 9:40:56 AM	Interaction Request Form_4		Open

My Recent Activity

User logged in at : Feb 21, 2012 11:45:10 AM

✓ Process_Interpreter_Calc_208 ran Successfully
Feb 21, 2012 7:46:25 AM

✓ Operator_Post_Failure_Process_314 ran Successfully
Feb 21, 2012 7:45:35 AM

✓ Operator_Pre_Success_Process_219 ran Successfully
Feb 21, 2012 7:45:33 AM

✓ Operator_Post_Success_Process_108 ran Successfully
Feb 21, 2012 7:45:30 AM

✓ Operator_Pre_Fail_Process_8 ran Successfully
Feb 21, 2012 7:45:29 AM

✓ StartContent_1 ran Successfully
Feb 21, 2012 7:45:20 AM

✓ calc_203 ran Successfully

Quick Links

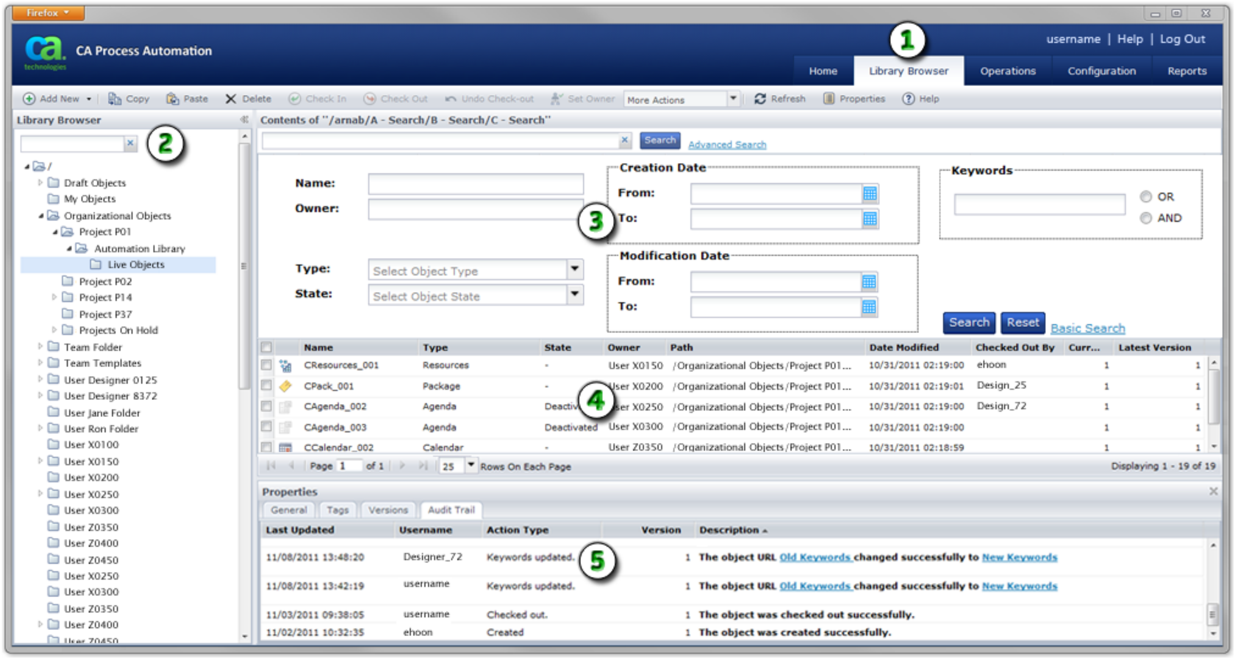
Help
Documentation
Book Shelf
Best Practices

ca.com

Item:	Description:
1	**Home Tab**: Click the Home tab to view the Home page. When you log in to CA Process Automation the Home page appears.
2	**Quick Buttons**: CA Process Automation provides a series of options to help you save time and begin working. A series of Quick Links is also listed in the lower right corner of the page. Click View Tutorials, browse sample content, design a new process, refer to the online help, or learn about Best Practices.
3	**Tasks and Automation Objects**: CA Process Automation displays convenient tables listing your tasks as well as the automation objects you have checked-out (working versions of objects).
4	**Recent Activity**: A list of recent process activity appears sorted by date and time. Click a link to open the process instance.

Library Browser

If you are assigned the role of a content administrator, architect, or designer, then take some time to become familiar with the Library Browser. Use this part of the application as a central repository to create, edit, and manage automation objects in a hierarchy of folders. Use the Library Browser for version control on each object, including object check-in and check-out.



/
- Draft Objects
- My Objects
- Organizational Objects
  - Project P01
    - Automation Library
      - Live Objects
  - Project P02
  - Project P14
  - Project P37
  - Projects On Hold
- Team Folder
- Team Templates
- User Designer 0125
- User Designer 8372
- User Jane Folder
- User Ron Folder
- User X0100
- User X0150
- User X0200
- User X0250
- User X0300
- User Z0350
- User Z0400
- User Z0450
- User X0250
- User X0300
- User Z0350
- User Z0400
- User Z0450

Name	Type	State	Owner	Path	Date Modified	Checked Out By	Curr...	Latest Version
CResources_001	Resources	-	User X0150	/Organizational Objects/Project P01...	10/31/2011 02:19:00	ehoon	1	1
CPack_001	Package	-	User X0200	/Organizational Objects/Project P01...	10/31/2011 02:19:01	Design_25	1	1
CAgenda_002	Agenda	Deactiv	ser X0250	/Organizational Objects/Project P01...	10/31/2011 02:19:00	Design_72	1	1
CAgenda_003	Agenda	Deactivated	User X0300	/Organizational Objects/Project P01...	10/31/2011 02:19:00		1	1
CCalendar_002	Calendar	-	User Z0350	/Organizational Objects/Project P01...	10/31/2011 02:18:59		1	1

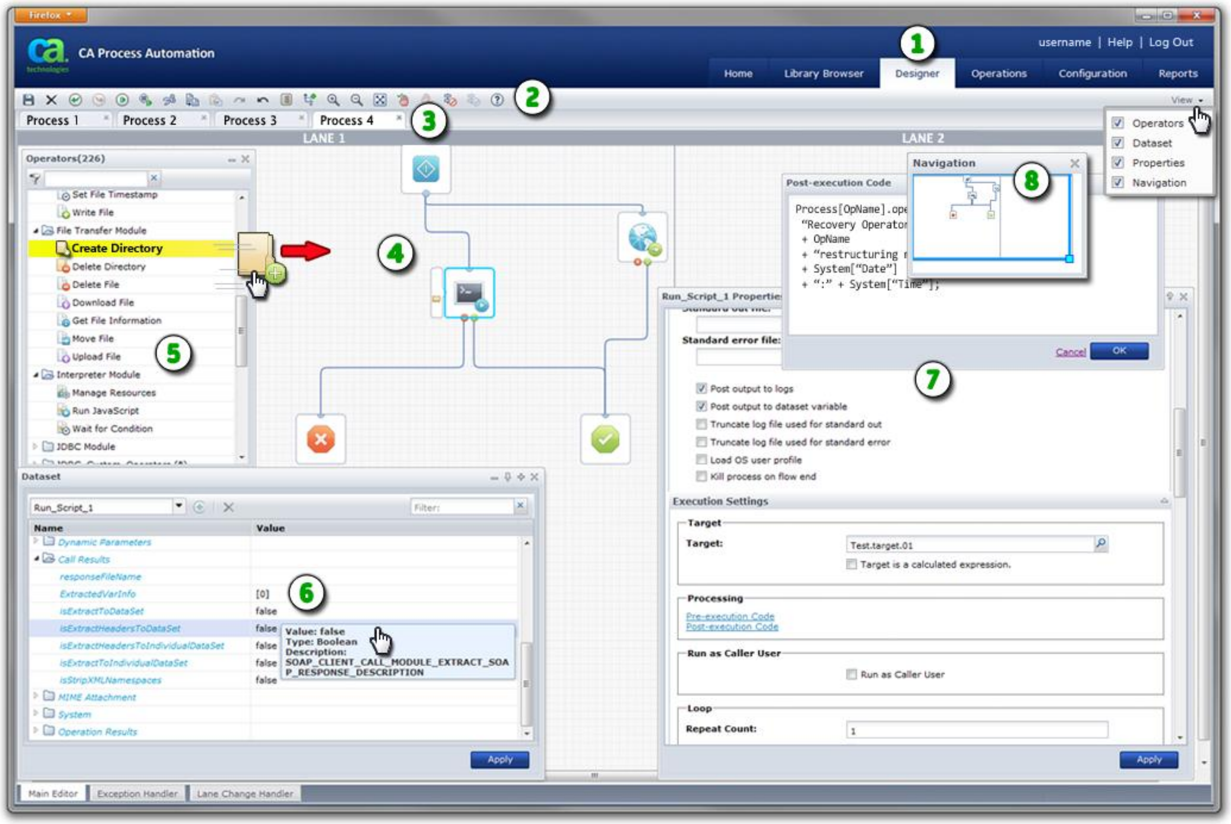
Last Updated	Username	Action Type	Version	Description
11/08/2011 13:48:20	Designer_72	Keywords updated.	1	**The object URL** Old Keywords **changed successfully to** New Keywords
11/08/2011 13:42:19	username	Keywords updated.	1	**The object URL** Old Keywords **changed successfully to** New Keywords
11/03/2011 09:38:05	username	Checked out.	1	**The object was checked out successfully.**
11/02/2011 10:32:35	ehoon	Created	1	**The object was created successfully.**

Item:	Description:
1	**Library Tab**: Click this tab to navigate to the Library Browser to view, create, edit, or remove automation objects and folders.
2	**Folder Hierarchy**: CA Process Automation displays the folders that you can access in this resizeable pane. You have the option of applying a filter to view only matching folders.
3	**Search Area**: Perform a basic or advanced search using your own input criteria. For example, locate one or more objects by name, type, or keyword.
4	**Automation Objects**: Filter and sort the list of automation objects in this table.
5	**Properties**: This tabbed panel displays the general data, tags, versions, and audit trail for each object.

Designer

When you are ready to design, edit, and even test a process, use this page for all your work. The Designer supports all of the following process details:

- the basic flow of operators
- 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



Name	Value
responseFileName	
ExtractedVarInfo	[0]
isExtractToDataSet	false
isExtractHeadersToDataSet	false
isExtractHeadersToIndividualDataSet	false
isExtractToIndividualDataSet	false
isStripXMLNamespaces	false

Value: false
Type: Boolean
Description:
SOAP_CLIENT_CALL_MODULE_EXTRACT_SOAP_RESPONSE_DESCRIPTION

```
Process[OpName].ope
"Recovery Operator
+ OpName
+ "restructuring r
+ System["Date"]
+ ":" + System["Time"];
```

Standard error file:

- [✓] Post output to logs
- [✓] Post output to dataset variable
- [ ] Truncate log file used for standard out
- [ ] Truncate log file used for standard error
- [ ] Load OS user profile
- [ ] Kill process on flow end

Target: Test.target.01
- [ ] Target is a calculated expression.

Processing: Pre-execution Code, Post-execution Code

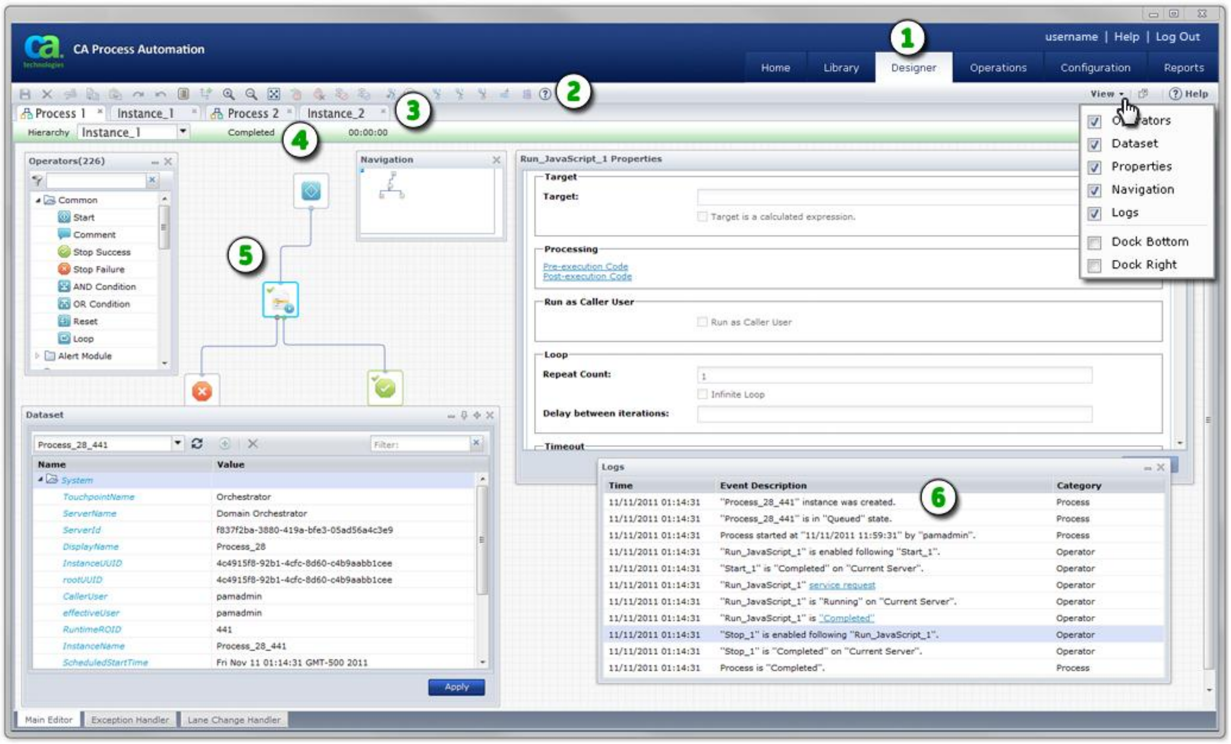
Run as Caller User
- [ ] Run as Caller User

Loop
Repeat Count: 1

Item:	Description:
1	**Designer Tab**: When you open a new process or edit an existing one from the Library Browser, this tab appears. If you do not see this tab, 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 additional links and windows to manage the properties for an operator. In this example, the Run Script operator's properties appear. The user has also added code in the Post-execution Code dialog.
8	**Navigation Palette**: Use this palette to navigate to specific regions inside 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.



Name	Value
System	
TouchpointName	Orchestrator
ServerName	Domain Orchestrator
ServerId	f837f2ba-3880-419a-bfe3-05ad56a4c3e9
DisplayName	Process_28
InstanceUUID	4c4915f8-92b1-4cfc-8d60-c4b9aabb1cee
rootUUID	4c4915f8-92b1-4cfc-8d60-c4b9aabb1cee
CallerUser	pamadmin
effectiveUser	pamadmin
RuntimeROID	441
InstanceName	Process_28_441
ScheduledStartTime	Fri Nov 11 01:14:31 GMT-500 2011

Time	Event Description	Category
11/11/2011 01:14:31	"Process_28_441" instance was created.	Process
11/11/2011 01:14:31	"Process_28_441" is in "Queued" state.	Process
11/11/2011 01:14:31	Process started at "11/11/2011 11:59:31" by "pamadmin".	Process
11/11/2011 01:14:31	"Run_JavaScript_1" is enabled following "Start_1".	Operator
11/11/2011 01:14:31	"Start_1" is "Completed" on "Current Server".	Operator
11/11/2011 01:14:31	"Run_JavaScript_1" service request	Operator
11/11/2011 01:14:31	"Run_JavaScript_1" is "Running" on "Current Server".	Operator
11/11/2011 01:14:31	"Run_JavaScript_1" is "Completed"	Operator
11/11/2011 01:14:31	"Stop_1" is enabled following "Run_JavaScript_1".	Operator
11/11/2011 01:14:31	"Stop_1" is "Completed" on "Current Server".	Operator
11/11/2011 01:14:31	Process is "Completed".	Process

Item:	Description:
1	**Designer Tab**: When you open an existing process instance from another page in the application, this tab appears. If you do not see this tab, it means 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. Instance names are also assigned a unique process ID number 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 sub-processes.

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.

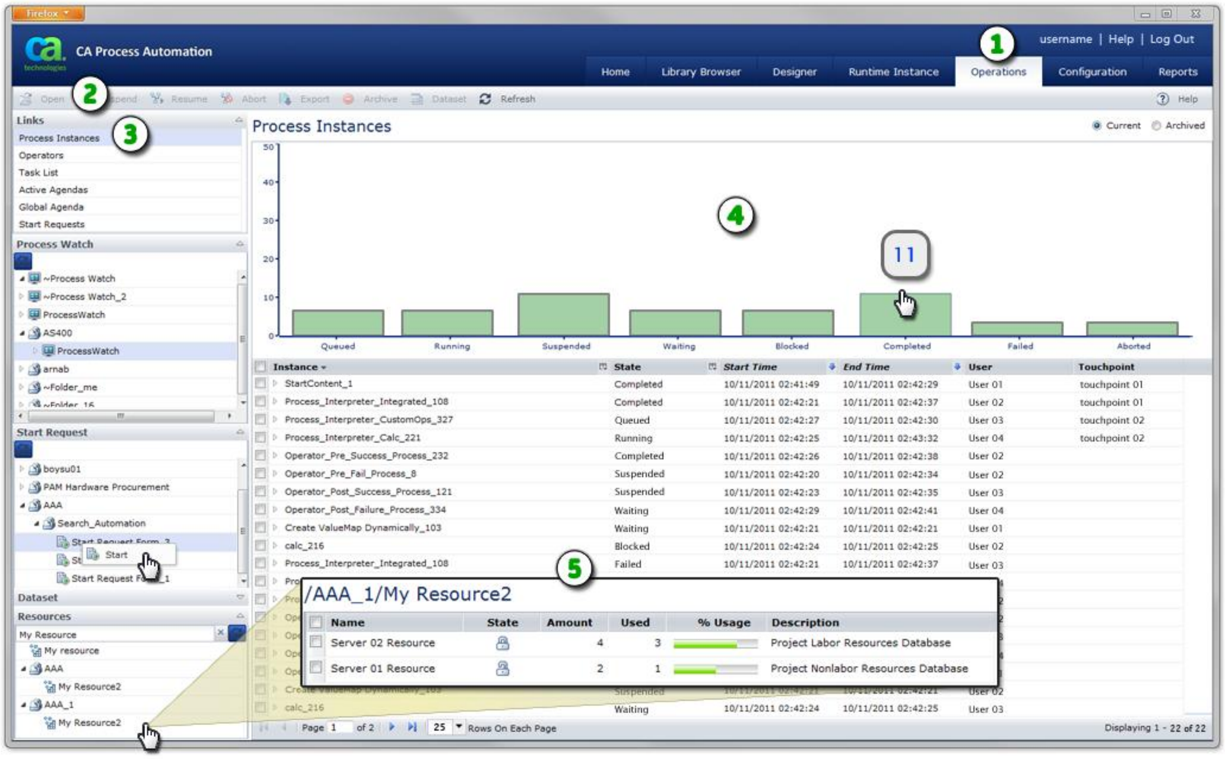
Operations

Use this page to view a dashboard showing the following key elements:

- Links to operators, processes, and other objects
- Process Watch
- Start Request Forms
- Datasets
- Resources

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

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



Instance	State	Start Time	End Time	User	Touchpoint
StartContent_1	Completed	10/11/2011 02:41:49	10/11/2011 02:42:29	User 01	touchpoint 01
Process_Interpreter_Integrated_108	Completed	10/11/2011 02:42:21	10/11/2011 02:42:37	User 02	touchpoint 01
Process_Interpreter_CustomOps_327	Queued	10/11/2011 02:42:27	10/11/2011 02:42:30	User 03	touchpoint 02
Process_Interpreter_Calc_221	Running	10/11/2011 02:42:25	10/11/2011 02:43:32	User 04	touchpoint 02
Operator_Pre_Success_Process_232	Completed	10/11/2011 02:42:26	10/11/2011 02:42:38	User 02	
Operator_Pre_Fail_Process_8	Suspended	10/11/2011 02:42:20	10/11/2011 02:42:34	User 02	
Operator_Post_Success_Process_121	Suspended	10/11/2011 02:42:23	10/11/2011 02:42:35	User 03	
Operator_Post_Failure_Process_334	Waiting	10/11/2011 02:42:29	10/11/2011 02:42:41	User 04	
Create ValueMap Dynamically_103	Waiting	10/11/2011 02:42:21	10/11/2011 02:42:21	User 01	
calc_216	Blocked	10/11/2011 02:42:24	10/11/2011 02:42:25	User 02	
Process_Interpreter_Integrated_108	Failed	10/11/2011 02:42:21	10/11/2011 02:42:37	User 03	
Create ValueMap Dynamically_103	Suspended	10/11/2011 02:42:21	10/11/2011 02:42:21	User 02	
calc_216	Waiting	10/11/2011 02:42:24	10/11/2011 02:42:25	User 03	

Page 1 of 2 | 25 Rows On Each Page | Displaying 1 - 22 of 22

/AAA_1/My Resource2

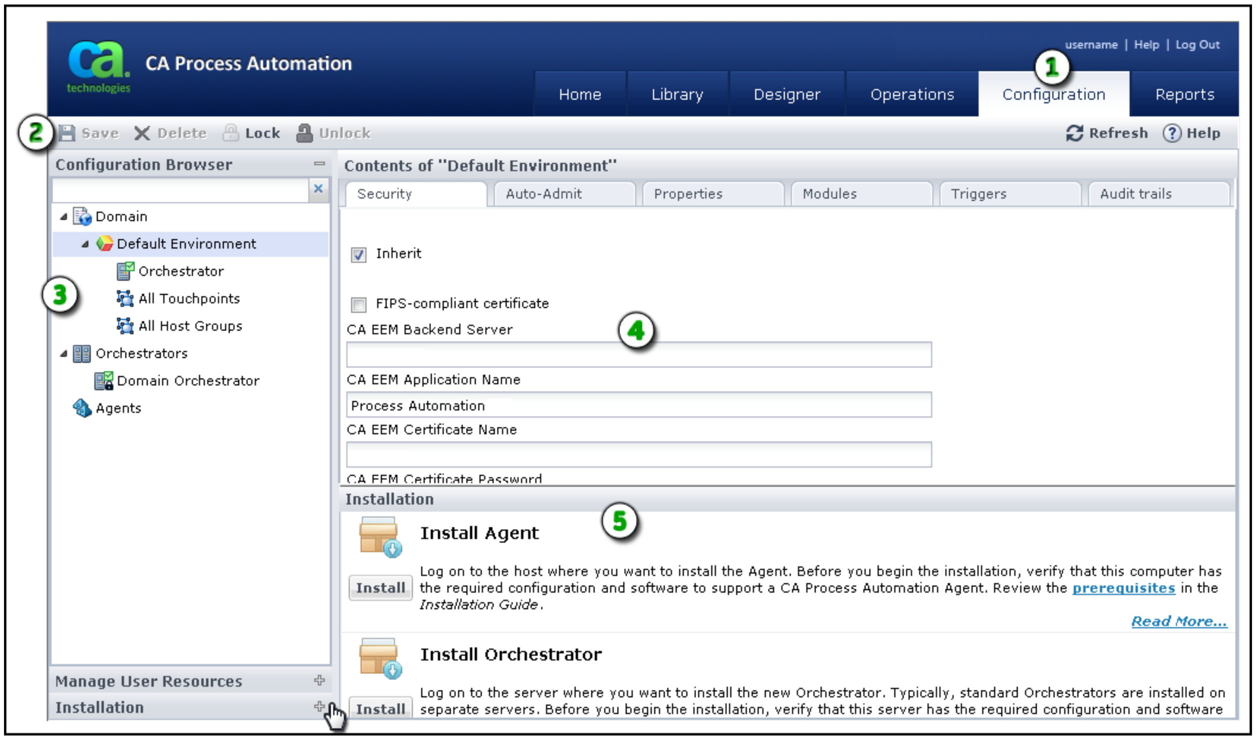
Name	State	Amount	Used	% Usage	Description
Server 02 Resource	🔒	4	3		Project Labor Resources Database
Server 01 Resource	🔒	2	1		Project Nonlabor Resources Database

Item:	Description:
①	**Operations Tab**: Click this tab to view the Operations page.
②	**Operations Toolbar**: Use these commands when working with items in the various lists of the Operations page.
③	**Operations Pane**: Make a selection from the expandable list of entries in this pane. For example, click Process Instances to view a chart and list of instances. Click a Start Request Form to start it. Enter an optional filter to find a resource object. In this example, 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**: If appropriate, CA Process Automation will display the data as a visual graph or chart. In this example, you can browse over the *Completed* instances to see the total count of 11. Click a color bar in the chart to add that status value to the filter for the resulting list below it. Click on the bar again to remove the value from the filter.
⑤	**Main Page or List**: The main area of this page displays the essential information you need. In addition to the chart, this data is often presented 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

Use this page for administrative functions including installation and configuration tasks. These functions involve the following entities:

- orchestrators
- host groups
- touchpoints
- agents
- user resources



- [✓] Inherit
- [ ] FIPS-compliant certificate

CA EEM Backend Server:

CA EEM Application Name: Process Automation

CA EEM Certificate Name:

CA EEM Certificate Password

Install Agent

Log on to the host where you want to install the Agent. Before you begin the installation, verify that this computer has the required configuration and software to support a CA Process Automation Agent. Review the prerequisites in the *Installation Guide*.

Install Orchestrator

Log on to the server where you want to install the new Orchestrator. Typically, standard Orchestrators are installed on separate servers. Before you begin the installation, verify that this server has the required configuration and software

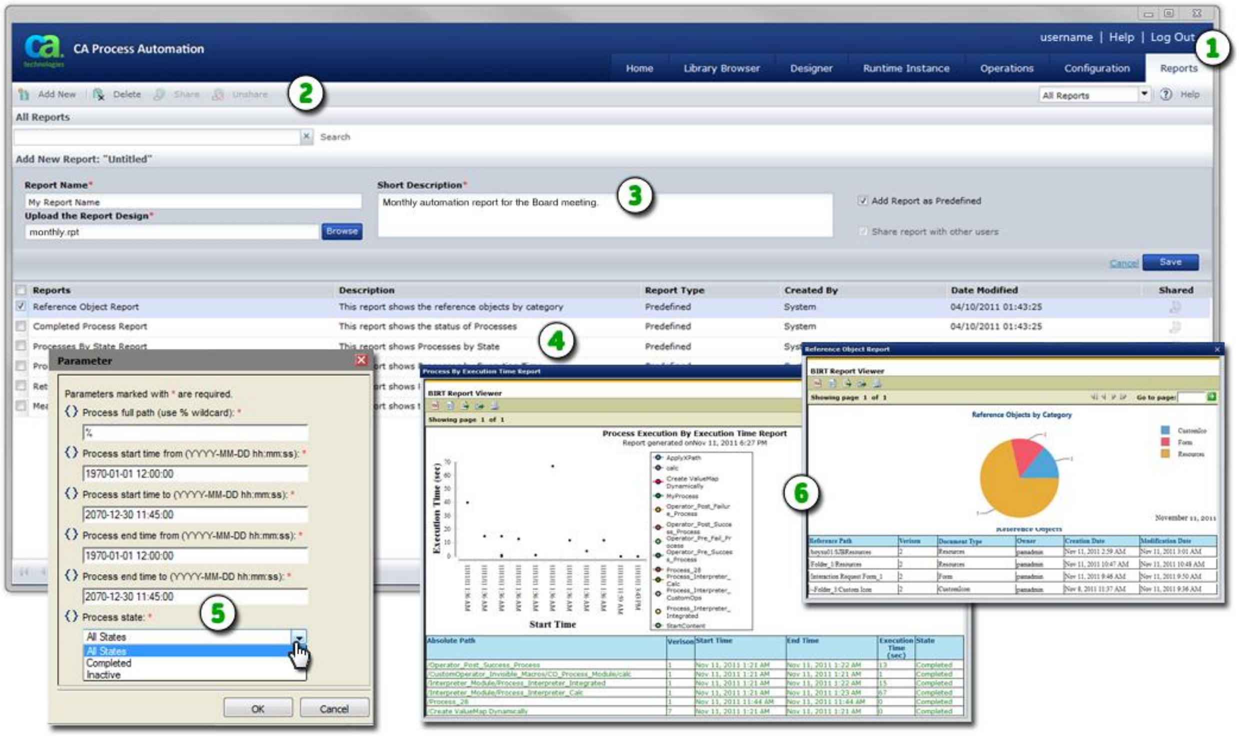
Item:	Description:
1	**Configuration Tab**: Administrators click this tab to access the administrative and configuration settings for CA Process Automation.
2	**Configuration Toolbar**: Use these tool buttons to help you install or configure the application.
3	**Configuration Pane**: This area helps you navigate the options available for this main page of the application. For example, you can choose an installation scenario, manage versions, or add and configure a touchpoint.

Item:	Description:
4	**Main Page**: The main area of this page displays the essential information you need. This data is often presented in a table, list, form, design canvas, or chart. This area may also be further divided using palettes, tabs, panes, or other visual controls. In this example, the user has selected Default Environment in the Configuration Browser pane. The domain properties for the orchestrator appear in the main page.
5	**Installation Scenarios**: When you click Installation in the Configuration Browser pane, the main page displays available installation scenarios.

Reports

Use this page to work with reports. Tasks include:

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



All Reports

Add New Report: "Untitled"

Report Name*: My Report Name

Upload the Report Design*: monthly.rpt

Short Description*: Monthly automation report for the Board meeting.

[✓] Add Report as Predefined

Share report with other users

Reports	Description	Report Type	Created By	Date Modified	Shared
✓ Reference Object Report	This report shows the reference objects by category	Predefined	System	04/10/2011 01:43:25	
Completed Process Report	This report shows the status of Processes	Predefined	System	04/10/2011 01:43:25	
Processes By State Report	This report shows Processes by State	Predefined	Sys		

Parameter

Parameters marked with * are required.

Process full path (use % wildcard): * %

Process start time from (YYYY-MM-DD hh:mm:ss): * 1970-01-01 12:00:00

Process start time to (YYYY-MM-DD hh:mm:ss): * 2070-12-30 11:45:00

Process end time from (YYYY-MM-DD hh:mm:ss): * 1970-01-01 12:00:00

Process end time to (YYYY-MM-DD hh:mm:ss): * 2070-12-30 11:45:00

Process state: * All States (All States / Completed / Inactive)

OK Cancel

Process By Execution Time Report — BIRT Report Viewer — Showing page 1 of 1

Process Execution By Execution Time Report

Report generated onNov 11, 2011 6:27 PM

Reference Object Report — BIRT Report Viewer — Showing page 1 of 1

Reference Objects by Category

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 the actual report output.
3	**New Report Area**: If you click Add New in the toolbar, this area appears so you can save details about the report.
4	**Reports List**: A list of the available reports.

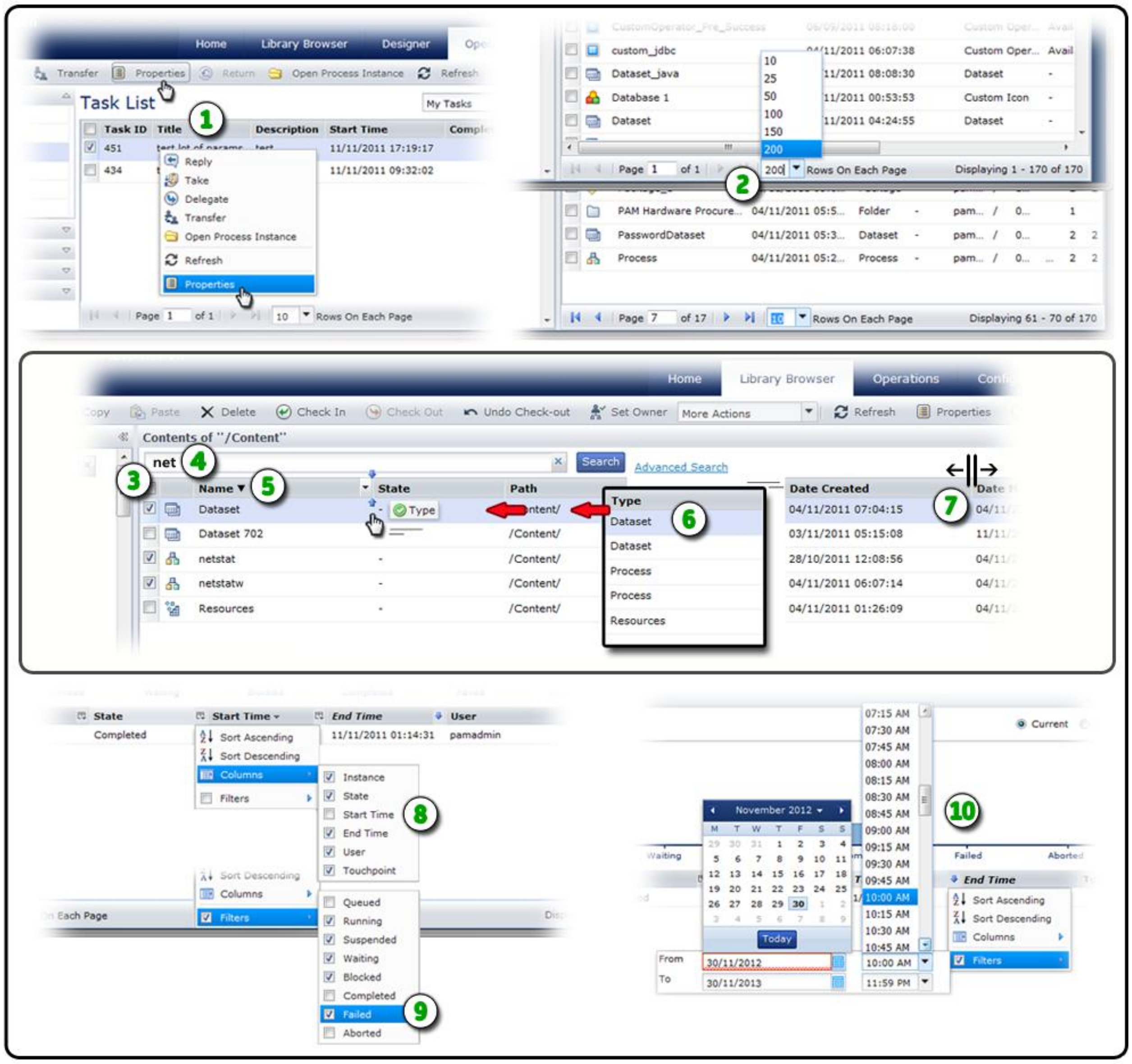
Item:	Description:
⑤	**Report Parameters Dialog**: When you click a report in the list, this dialog appears. Use it to specify the range of data you want to include in this particular instance of the report. In this example, the user has selected All States from the { } Process State field.
⑥	**Report Output**: An on-screen preview of the output appears. 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 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.

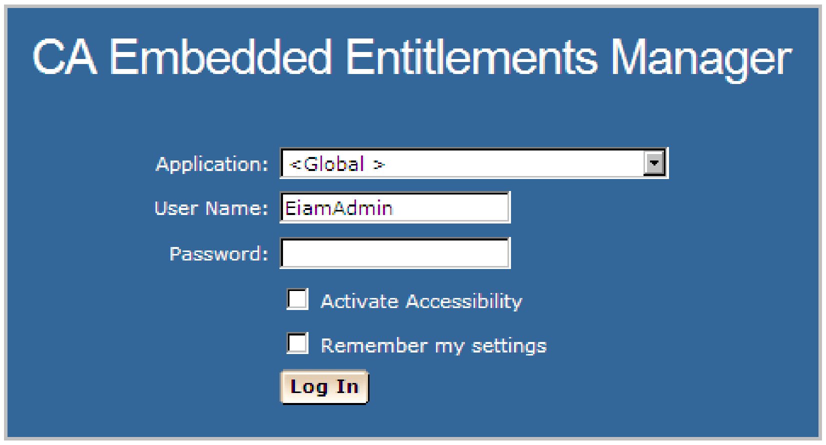
Change Your Own Password in CA EEM

CA Process Automation users can change their own passwords in CA EEM.

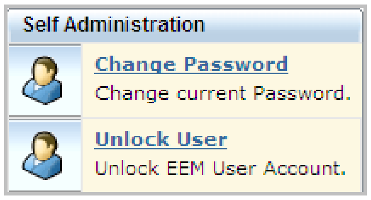
Follow these steps:

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

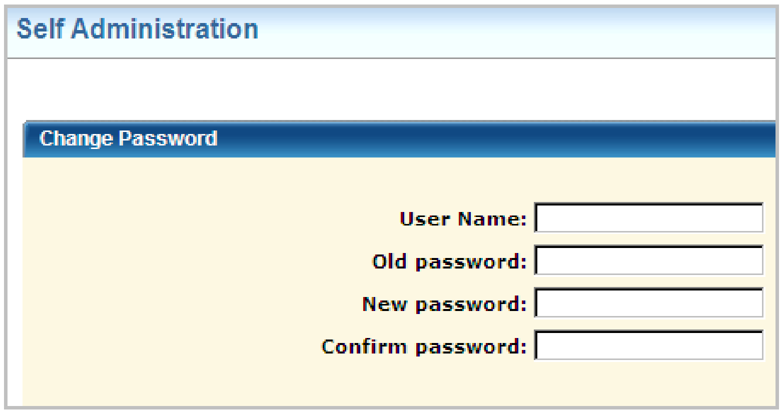
   The CA Embedded Entitlements Manager (CA EEM) Log In dialog appears.



2. For Application, select <Global>.
3. Delete EiamAdmin if this default User Name appears.
4. Enter your CA Process Automation user name and password, and then click Log In.
5. Click Change Password.



6. Enter your CA Process Automation user name and old password. Then enter your new password in both the New password and Confirm password fields and click OK.



User Name:	
Old password:	
New password:	
Confirm password:	

7. Browse to CA Process Automation and log in with your new credentials.

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

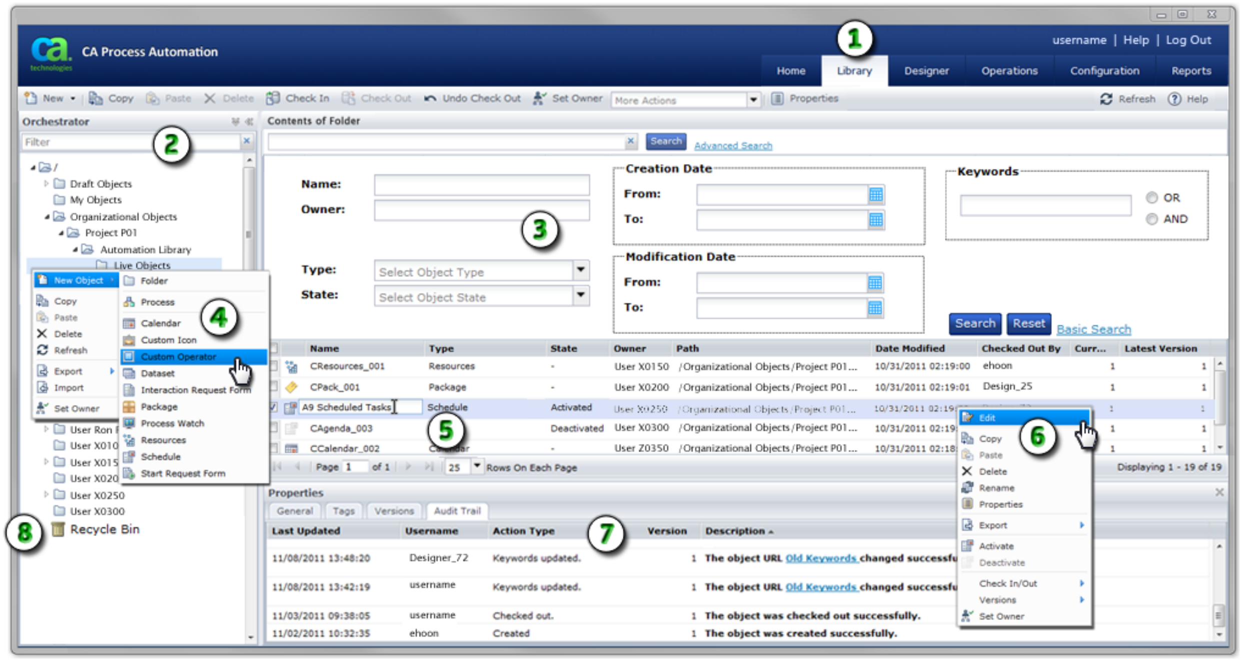
When you set Microsoft Internet Explorer 8 or 9 to Compatibility Mode (CM), CA Process Automation may exhibit poor performance or behavior. For example, the drop-down menus for the field columns on the Operations page may disappear if you click Start Requests (or another option in the Operations pane) 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: The Library Browser

Use the Library Browser to manage automation objects in folders.



	Name	Type	State	Owner	Path	Date Modified	Checked Out By	Curr...	Latest Version
	CResources_001	Resources	-	User X0150	/Organizational Objects/Project P01...	10/31/2011 02:19:00	ehoon	1	1
	CPack_001	Package	-	User X0200	/Organizational Objects/Project P01...	10/31/2011 02:19:01	Design_25	1	1
✓	A9 Scheduled Tasks	Schedule	Activated	User X0250	/Organizational Objects/Project P01...	10/31/2011 02:19		1	1
	CAgenda_003		Deactivated	User X0300	/Organizational Objects/Project P01...	10/31/2011 02:19		1	1
	CCalendar_002		-	User Z0350	/Organizational Objects/Project P01...	10/31/2011 02:18		1	1

Page 1 of 1 | 25 Rows On Each Page | Displaying 1 - 19 of 19

Properties: General | Tags | Versions | Audit Trail

Last Updated	Username	Action Type	Version	Description
11/08/2011 13:48:20	Designer_72	Keywords updated.	1	The object URL Old Keywords changed successfu
11/08/2011 13:42:19	username	Keywords updated.	1	The object URL Old Keywords changed successfu
11/03/2011 09:38:05	username	Checked out.	1	The object was checked out successfully.
11/02/2011 10:32:35	ehoon	Created	1	The object was created successfully.

Item:	Description:
1	**Library Tab**: Click this tab to navigate to the Library Browser. Here you can 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 an optional filter to view only matching folders. For example, *Folder_2* would find Folder_2, Folder_200, and MyFolder_2.
3	**Search Area**: Perform a basic or advanced search using your own input criteria. For example, locate one or more objects by name, owner, type, state, or keywords.
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 process, schedule, or start request form.
5	**Rename an Object**: Right-click an object and choose Rename from the shortcut menu to edit its name in place within the table.

Item:	Description:
⑥	**Edit an Object**: Use the toolbar or right-click an object to view a shortcut menu of commands.
⑦	**Properties**: This tabbed panel displays general data, tags, versions, and an audit trail for each object.
⑧	**Recycle Bin**: Use the Recycle Bin to manage objects you want to delete. The application allows you to restore them, leave them in the Recycle Bin, or permanently purge them.

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

This section contains the following topics:

Automation Object Types

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

You can browse an automation library associated with an orchestrator. You can also create, edit, and view objects in an automation library. You can create folders in a library to group related objects. Folders let you define a hierarchical structure so you and your co-workers can locate objects. This structure is similar to the directories or folders of any computer operating system such as Microsoft Windows.

You can create all of the following types of automation objects in the Library Browser:

Process

Process objects graphically depict the order and dependencies between operators and sometimes other processes. Operators are represented in a process graphically with links that show the sequence and logic behind the steps that a process performs.

Schedule

Schedule objects associate module invocations (including the launch of a process) with date and time conditions. You can use multiple schedules to group tasks by application, ownership, or other criteria.

Calendar

Calendar objects define rules that describe complex date conditions. Calendar objects graphically specify dates, time intervals, and conditional elements that determine when and how frequently operators are performed.

Custom Icon

Custom Icon objects specify graphic images that uniquely identify operators.

Custom Operator

Custom Operator objects allow you to extend the presentation and configuration of existing operators. You can customize your own operators based on existing operators and optimize them with specific parameters that are designed for reuse in many different processes.

Dataset

Dataset objects define and group variables used as parameters that are required by other processes, operators, and resources. Examples include application locations, passwords, and profile names. These variables can be configured easily so that processes and scheduling can be updated efficiently to reflect changes in an application environment.

Package

A single *Package* object bundles shortcuts to other automation library objects. This provides a way to manage related objects residing in any number of different folders in the library. A package can be used to bundle various related objects on an orchestrator to export for deployment on a production orchestrator. Packages also facilitate reuse of objects in different environments. Use a package when your solution spans multiple objects and folders. Apply permissions to each folder. For example, put custom operators in a *Custom Operators* folder, put network items in a *Network* folder, and put security items in a *Security* folder. A package or solution can include content from each folder.

Process Watch

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

Resource

Use *Resource* objects to synchronize independent processes that rely on common elements of the infrastructure. Resource objects are models representing elements of your system architecture. Use resources to quantify and control access to particular IT entities. Include multiple resources representing related entities in a single Resource object.

Start Request Form

Start Request Form objects define shortcuts to allow a production user to invoke processes manually. Custom dialogs prompt users for the values of parameters required to start their associated processes.

Interaction Request Form

Interaction Request Form objects allow you to prompt users to provide responses in data fields and other user interface controls. Users enter information that is required to continue a process. For example, use a form to prompt a stakeholder to review each step in an approval process.

More information:

Designing Processes (see page 87)
Running, Testing, and Debugging Processes (see page 335)
Datasets (see page 173)
Interaction Request Forms (see page 245)
Package Objects (see page 81)
Create a New Process Watch Object (see page 339)
Start Request Forms (see page 243)

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: We recommend that you organize objects in folders to set rights, export them, and perform other tasks. Do not create objects at the root level because there is no way to manage them as a group.

Create an Object

Create new automation objects in specific folders in the library associated with an orchestrator. After creating 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 choose an object type.

   A new object is created.
4. Rename the object to identify it.

   Provide a unique name. The application alerts you when the name is already applied to another object in the folder including any deleted objects that are in the Recycle Bin.

Note: We recommend that you organize objects in folders to set rights, export them, and perform other tasks. Do not create objects at the root level because there is no way to manage them as a group.

Working with Objects

The task of editing an object is divided into the following sub-steps:

- identify the location of an object or folder in the Library Browser
- open the object
- check out the object to a working version you can edit
- edit the object
- save the changes
- check in the object (in this step you also decide to apply the changes to the same or a new version)

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

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

Note: This section includes only basic information on opening and editing objects and managing versions. Other chapters in this guide describe how to edit the different types of objects in greater detail.

Edit an Object

After you create an object, you can edit its properties or modify its configuration at any time. For example, you can edit the tags for an object, add a resource to a resource object, or remove an object from a process watch.

CA Process Automation stores multiple versions of an object. When you check out and edit an object, by default, the application uses the version that is designated as the *current* version. After editing the object, you must check it back into the application. When you check in the object, you can apply the changes to either the same version or create a new version of the object. Whether you choose a new version or the same one, by default, the latest changes are applied and that version becomes the new *current* version.

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 it is not already checked out, click Check Out.
5. Edit the object.
6. Click Save or Save and Close.

   The changes you made to the object are saved. Since the object is still checked out, you can continue to edit it.
7. To check the object back in, click Check In.
8. On the Check-In dialog:
   a. In the Check-in As field, choose New Version or Same Version.
   b. Enter any optional comments about the changes made in this version.
   c. (Optional) Check the Baseline check box to use this version as a baseline.
   d. (Optional) For new versions, uncheck the Current check box only if you want a new version but do not want it to be marked as current. The previously current version will remain current.
   e. Click Check-In.

View General Properties for a Library Object

You can view the general properties of any library object including processes. All of the following properties are read-only except for Description.

To view or change general properties

1. Click the Library tab.
2. Select a folder in the left pane and a single object in the list.
3. In the toolbar, click Properties.
4. View the following fields on the General tab:

   Name

   The name of the object.

   Current Version

   The version number set as current for the object. An object can have multiple versions but can only have one current version.

   Description

   The description of the current object. You can edit this field whether the object is checked out or not. Click Save to store your changes.

   Type

   The type of automation object.

   Owner

   The ID of the CA Process Automation user who created the object.

   Checked Out By

   The ID of the CA Process Automation user who has checked out the object.

   Date Created

   The date that the folder or object was created.

   Date Modified

   The date that the folder or object was last modified.

   Path

   The path for the object in the CA Process Automation library.

   Orchestrator

   The name of the orchestrator associated with the library and who manages the database object being configured.

Specify Tags or Keywords for Objects

Use the Tags tab to assign keywords that organize objects according to any meaningful naming system recognized by your organization. You and other users can then perform an advanced search on objects or folders using the tags or keywords.

Follow these steps:

1. Click the Library tab.
2. In the Library Browser folders pane, select a folder.
3. Select an object or folder.
4. In the toolbar click Properties.

   The Properties pane appears.
5. In the Properties pane, click the Tags tab.
6. On the Tags tab:
   a. Click Edit.
   b. Enter one or more comma-separated text values. For example,

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

   c. Click Save.

Change Ownership for Automation Objects

The owner has full control of an automation object or folder. You must be logged in to CA Process Automation as the owner or the Environment Content Administrator to change ownership of an automation object. The user who creates an automation object is, by default, its owner. If you set ownership of a folder to a user who is not an administrator, that user is the only non-administrator who can access that folder.

Ownership of a process is used when Runtime Security is enabled. If you enable Runtime Security, then only the owner you set for a given process can start that process.

Follow these steps:

1. Click the Library tab.
2. Select one or more objects including folders.
3. In the toolbar, click Set Owner.

   The Set Owner dialog opens.
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

You can define how long a completed instance of a Process or Schedule object is archived on the server.

Follow these steps:

1. Click the Library tab.
2. Select a folder and then select a process or schedule object.
3. In the toolbar, click Properties.
4. In the Properties pane, click the Archival Policy tab.
5. Enter the following options:

   Minimum Days of History

   Defines the minimum number of days to retain completed and failed instances of Processes. The age of an instance measures in hours. If an instance ends at 10:00 PM and this option is set to one day, the instance remains in the Library at least until 10:00 PM the following day.

   Minimum number of Failed Instances

   Defines the minimum number of failed instances of the Process to retain in the history. Failed instances of the Process in excess of this number are deleted in order of age (oldest first) only if they have been in the history longer than the specified minimum number of days.

   Minimum number of Completed Instances

   Defines the minimum number of completed instances of the Process to retain in the history. Completed instances of the Process in excess of this number are deleted in order of age (oldest first) only if they have been in the history longer than the specified minimum number of days.

   Inherit Archival Policy From Orchestrator

   Select this check box to inherit the archival Policy property settings for the CA Process Automation server. This check box is initially selected as the default setting for an object.

6. Click Save.

Specify ROI Properties

You can specify the return on investment (ROI) properties for a Process object. The ROI values you enter appear in the dataset of process instances.

Follow these steps:

1. Click the Library tab.
2. Select a folder and then select a process.
3. In the toolbar, click Properties.
4. In the Properties pane, click the ROI tab.
5. On the ROI tab, enter the following:

   Enable ROI

   Enables the ROI fields.

   Manual Labor Time (hh:mm)

   Specifies the number of man hours and minutes it typically takes to complete this process.

   Note: Man hours is the time it takes a human being to trigger each step in the process. Man hours does not include the time each step takes to run.

   Manual Process Elapsed Time (hh:mm)

   Specifies the time taken for this process to execute manually from start to finish.

   Criticality

   Specifies the criticality for this process.

   Valid Values: High, Medium, or Low.

   Note: The ROI report displays the criticality of the process.

   App/System Group Name

   Defines the process group name.

   Note: You can group processes by group name in the ROI report. Click the Reports tab to run the Return on Investment Report for an application or system group.

6. Click Save.

Specify Runtime Security Properties

You can specify runtime security properties for Process and Schedule objects.

Follow these steps::

1. Click the Library tab.
2. Select a folder and then select a process or schedule.
3. In the toolbar, click Properties.
4. In the Properties pane, click the Runtime Security tab.
5. On the Runtime Security tab, enter the following:

   Runtime Security

   Runtime security ensures that a process cannot be started by someone who does not have the proper permissions. Select one of the following:

   Inherit from Orchestrator

   If selected, security settings are inherited from the Orchestrator.

   Enable

   Enabling runtime security lets CA Process Automation check permissions for a user before accessing automation objects from the database.

   Disable

   Disabling runtime security ensures backward compatibility to earlier versions of CA Process Automation. Processes continue to run as they have prior to this release.

   Run as Owner

   If runtime security is Enabled then Run as Owner applies. Any process and its child process executes as if the owner of the process is running it. Only Environment Content Administrators or the owner of the process can select this option.

   If Run as Owner is selected, then the current identity is the owner of the object and an access control list is enforced for the owner of the process. CA Process Automation loads private copies of the objects created by the owner (including the process itself).

   This check box is not selected by default. If Run as Owner is unchecked, then at runtime, CA Process Automation checks permissions for the user who has executed the process.

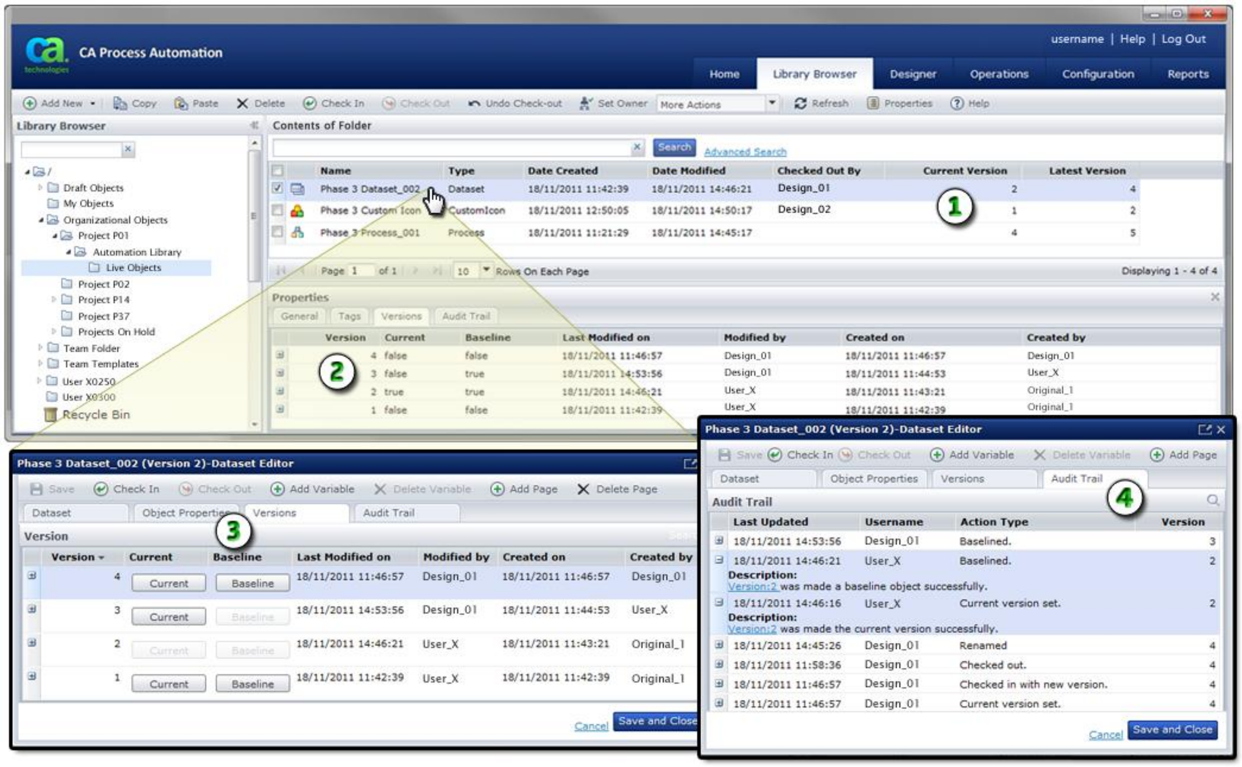
   Enable Operator Recovery

Applies to Process objects only. Specifies whether to automate recovery. Recovery applies to certain operators that fail with a SYSTEM_ERROR. Operators subject to recovery must be part of processes that are in a *Blocked*, *Running*, or *Waiting* state. If checked, recovery starts running the affected processes. The operators that target the proxy touchpoint run. If not checked, operator recovery is disabled.

Versions

CA Process Automation uses a version control system that tracks the changes you make to the objects in your library. An important point to remember is that the version tracking system always maintains what is known as a single *current* version for each object. When you open an object, the application automatically uses the one you have designated as the current version. Other versions of an object are retained for backup or archival purposes. Any archived version of an object can also be opened for viewing or editing or converted to the current version. The object owner or a user with sufficient permission can change which version is the current version.

You can check out and edit any version of an object and then check in changes either to that same version or to a new version of the object. To prevent users from changing a particular 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 can only check it in as a new version. Baseline versions serve as a benchmark or template for further development.



Name	Type	Date Created	Date Modified	Checked Out By	Current Version	Latest Version
Phase 3 Dataset_002	Dataset	18/11/2011 11:42:39	18/11/2011 14:46:21	Design_01	2	4
Phase 3 Custom Icon	CustomIcon	18/11/2011 12:50:05	18/11/2011 14:50:17	Design_02	1	2
Phase 3 Process_001	Process	18/11/2011 11:21:29	18/11/2011 14:45:17		4	5

Version	Current	Baseline	Last Modified on	Modified by	Created on	Created by
4	false	false	18/11/2011 11:46:57	Design_01	18/11/2011 11:46:57	Design_01
3	false	true	18/11/2011 14:53:56	Design_01	18/11/2011 11:44:53	User_X
2	true	true	18/11/2011 14:46:21	User_X	18/11/2011 11:43:21	Original_1
1	false	false	18/11/2011 11:42:39	User_X	18/11/2011 11:42:39	Original_1

Phase 3 Dataset_002 (Version 2)-Dataset Editor

Save | Check In | Check Out | Add Variable | Delete Variable | Add Page | Delete Page

Dataset | Object Properties | Versions | Audit Trail

Version

Version	Current	Baseline	Last Modified on	Modified by	Created on	Created by
4	Current	Baseline	18/11/2011 11:46:57	Design_01	18/11/2011 11:46:57	Design_01
3	Current	Baseline	18/11/2011 14:53:56	Design_01	18/11/2011 11:44:53	User_X
2	Current	Baseline	18/11/2011 14:46:21	User_X	18/11/2011 11:43:21	Original_1
1	Current	Baseline	18/11/2011 11:42:39	User_X	18/11/2011 11:42:39	Original_1

Cancel | Save and Close

Phase 3 Dataset_002 (Version 2)-Dataset Editor

Save | Check In | Check Out | Add Variable | Delete Variable | Add Page

Dataset | Object Properties | Versions | Audit Trail

Audit Trail

Last Updated	Username	Action Type	Version
18/11/2011 14:53:56	Design_01	Baselined.	3
18/11/2011 14:46:21	User_X	Baselined.	2
Description: Version:2 was made a baseline object successfully.			
18/11/2011 14:46:16	User_X	Current version set.	2
Description: Version:2 was made the current version successfully.			
18/11/2011 14:45:26	Design_01	Renamed	4
18/11/2011 11:58:36	Design_01	Checked out.	4
18/11/2011 11:46:57	Design_01	Checked in with new version.	4
18/11/2011 11:46:57	Design_01	Current version set.	4

Cancel | Save and Close

Item:	Description:
1	**Object Versions**: For every object, the library browser displays the user who has checked out the object, as well as the current and latest versions. In this example, the current version of the selected dataset is 2. This is the version checked out by user *Design_01*. Note that the latest version of the object is version 4.

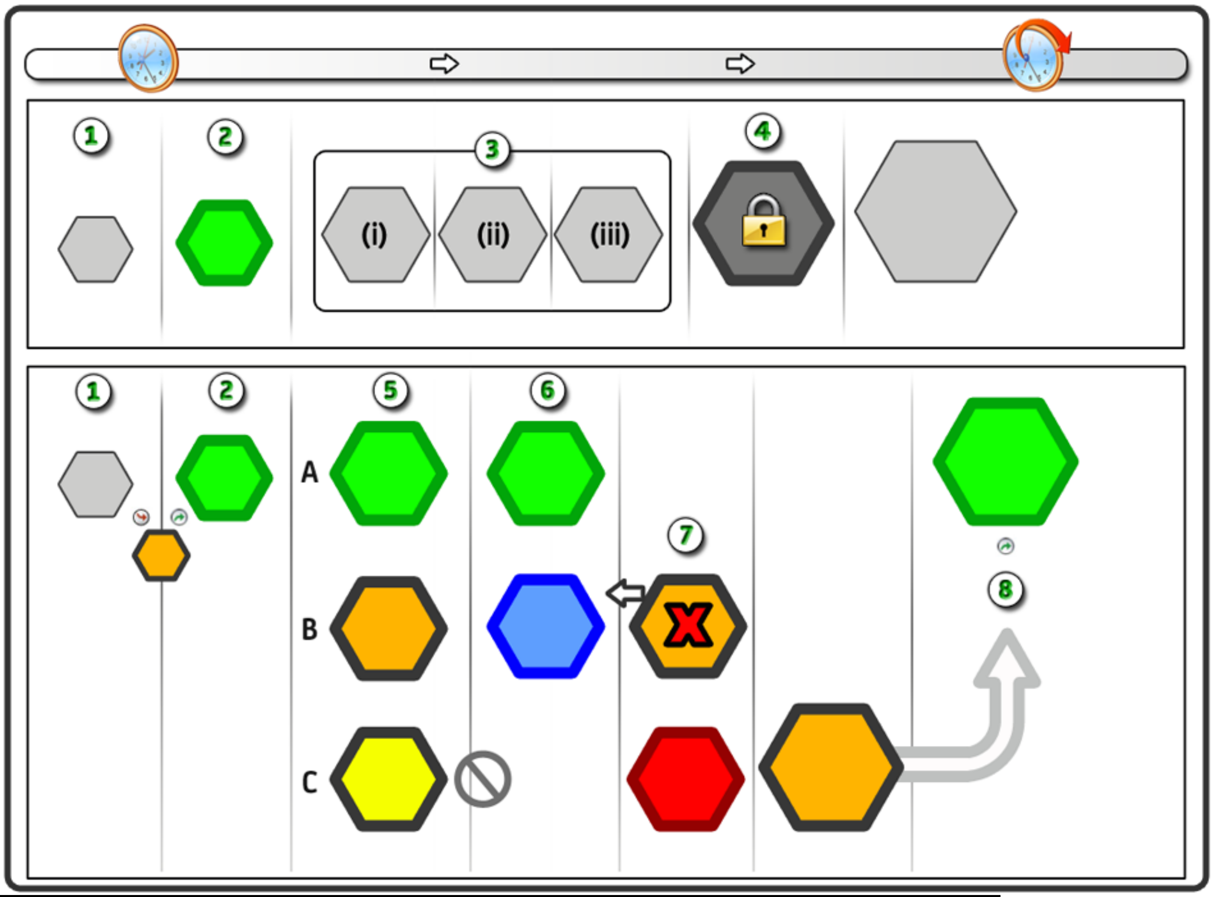
Item:	Description:
②	**Properties Panel**: The Versions tab of the object's Properties panel lists the original and all subsequent versions for that object. You can see which one is current, which one or more are baselines, as well as creation and modification information.
③	**Versions Tab in Object Editor**: When you open or edit an object, the Versions tab allows you to select a single current and one or more baseline versions. The other information is exactly the same as the information displayed in the Properties panel.
④	**Audit Trail Tab in Object Editor**: This tab provides a history of all changes made 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.`	⬢ `Current version.`
⬢ `Working version.`	⬢ `Discard working version.`
⬢ `Open a previous version.`	⬢ `Baseline version.`
⬢ `Version checked out by user 1.`	⬢ `Latest version checked out by user 2.`



Item:	Description:
1	**Typical Versioning Sequence**: A new automation object begins at ⬡ version 1. The object is checked out as ⬢ working version 1, with the option of applying pending changes to the same version 1 or new version 2.
2	**Current Version**: In this typical sequence, ⬢ working version 1 is checked in as new version 2. 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 1 is now considered just a backup or archive.

Item:	Description:
③	**Changes Applied to the Same Version**: In this example, version 3 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(i), 3(ii), and 3(iii), 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(i) and 3(ii). 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.

View Version Information

The Versions tab displays information about checked-in versions of an object.

Follow these steps:

1. Click the Library tab.
2. Select an object in the main window and click Properties.

   The Properties pane appears in the bottom portion of the window.
3. Click the Versions tab.
4. Expand a version to view all of its fields and optional comments.

   Version

   The number assigned to this iteration of an object in sequence with other iterations.

   Current

   Indicates which version is designated as the current one. Only the current version will display true in this field.

   Baseline

   Indicates which versions are designated as baselines for the object. The field displays true for baselines or false for non-baselined versions. A baseline version is locked, but can be changed and checked in as a new version.

   Last Modified on

   The date and time when the version was last checked in.

   Modified by

   The CA Process Automation user who last changed the version.

   Created on

   The date and time when the version was created.

   Created by

   The CA Process Automation user who last checked in the version.

   Comments

   Displays comments that can be entered when checking in a version of an object.

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 might 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.

1. Check Out an Object (see page 74).
2. Make changes to 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.

Check in Objects

When your changes are complete, check the object back into the system. This saves your changes to your choice of the same version or a new version.

After you complete the procedure, the Designer remains open and you can continue viewing the object in read-only mode. To make additional changes you must check out the object again.

Follow these steps:

1. Check Out an Object (see page 74).
2. (Optional) Save Changes for a Checked Out Object (see page 74).
3. On the object editor toolbar or Library Browser toolbar, click Check In.

   The Check-In dialog appears.
4. Decide how you want to check in the changes you made by configuring the following settings:

   Current

   Select this check box to make the checked-in version the *current* version of the object. For new versions, clear this check box to keep the previous current version. The new version reflects the latest changes but will not be used by the application.

   Baseline

   Select this check box to designate a version of the object that cannot be changed. Baseline versions can only serve as the basis for new versions.

   Comments

   Enter any descriptive comments to save with this version of the object.

   Check-in as

   Specify whether to create a new version of the object or apply the changes to the same version.
5. Click Check-In.

   Note: CA Process Automation displays a message if you attempt to check in a baseline version without creating a new version.

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.

Discard Changes to the Working Version of an Object

After checking out an object, making changes to it, and saving it, you may decide you do not want to continue working with that particular working version. Even if you have saved the working version numerous times, you can still discard the saved working version.

Follow these steps:

1. Click the Library tab.
2. Right-click the object in the main window and choose Versions and then click Discard Working Version.

   The saved changes to the working version of an object are ignored and the object reverts back to the originally checked out version.

   Note: Any unsaved changes continue to appear in the editor or process designer.

Copy an Object

You can copy objects or folders. You can use the current version of an object as the basis for similarly configured objects on the same orchestrator. Copy the current version of an object and then paste it to the same or a different folder where it can be renamed and edited as a new object. For example, you can configure a Custom Operator object and then add copies within a library.

Follow these steps:

1. Click the Library tab.
2. Identify the objects you want to copy in the main window. Ensure 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 the destination folder in the same automation library and click Paste.

   The copied objects are added with the same names as the original objects. If this location is the same as the source folder, then an ordinal number is appended to each object name (for example, *Process_1*).
5. (Optional) Rename the new objects.

Copy Objects Using Export and Import

You can copy objects or folders from one automation library on an orchestrator to another by exporting them to an export file. Exporting a folder also exports its contents.Then you can import the objects on the destination orchestrator. Use export/import to move objects or folders between libraries or to deploy objects from a development library to a production library. You can also use an export file to copy folders to other locations within the same automation library.

Part 1: Export

1. Click the Library tab.
2. In the Library Browser folder pane, select the parent folder.
3. Right-click one of the following:
    a. the parent folder
    b. a child folder
    c. an object
4. In the resulting shortcut menu, choose Export, and then Absolute Paths or Relative Paths.
    - Click Absolute Paths to export the full paths for the exported objects.
    - Click Relative Paths to export the relative paths within the folder containing the selected object.

    Your browser prompts you about the new XML export file.
5. Save the file to a local or network directory.

Part 2: Import

1. In the Library Browser folder pane, select the destination for the imported items. If necessary, change the orchestrator or library.
2. Right-click the parent folder and choose Import.
3. In the Import dialog:
    a. Enter the path to the XML file you exported or click Browse to locate it.
    b. Specify how you want to handle objects with conflicting names.
    c. Click Submit.

    The import recreates the exported folder structure in the selected location and imports the new objects.

When exporting and importing files, be aware of the following points:

- The export file preserves complete information, including hierarchies and paths, for each exported object. When you export objects, you can save either the full absolute folder hierarchy from the root folder, or only the relative paths starting in the selected folder.
- When the object information is imported back into an automation library, CA Process Automation uses the path information stored in the file to reconstruct the exported folder structure.
- When you export the root folder, the entire automation library is exported. Size constraints may apply.
- You can open an export file in any XML text editor application including Windows Notepad. A *<head>* element contains the CA Process Automation version information for the development environment.
- At the deployment phase of the production cycle, an export file is used to deploy the objects in an automation library to production computers.

Important! Do not change any information in an export file. External changes can corrupt the file. Any changes must be made in CA Process Automation.

Copy Objects in Packages Using Export and Import

Use a *Package* object to export select versions of multiple objects simultaneously.

As you work with objects in a package, you can select the version to be exported with the package. When you select an object, the version information shows in the browser pane. You can select a different version to export next to Version on the browser pane. Choose to include either the “Current” version or a particular version number in the list. The default setting when you add an object to a package is to export the “Current Version”.

Part 1: Export

1. Click the Library tab.
2. Right-click a Package object and choose Export, and then click either Absolute Paths or Relative Paths.

   These commands apply only to the Package object itself; the full path is always exported for each object in a package.

   - Click Absolute Paths to export the full path for the exported package.
   - Click Relative Paths to export the relative paths within the folder containing the selected package.

   Your browser prompts you about the new XML export file.
3. Save the file to a local or network directory.

Part 2: Import

1. In the Library Browser folder pane, select the destination for the imported items. If necessary, change the orchestrator or library.
2. Right-click the parent folder and choose Import.
3. In the Import dialog:
   a. Enter the path to the package XML file you exported or click Browse to locate it.

   b. Specify how you want to handle objects with conflicting names.

   c. Click Submit.

   The application imports the new objects specified by the package. The package object itself is imported in the currently selected folder. The objects in the package are imported to the same locations where they were originally stored when exported.

Package Objects

Complex processes can use various objects including operators, icons, resources, datasets, and even other processes. They can be distributed with unrelated objects among several different folders in an automation library. This complexity presents the following potential problems:

- It is difficult to work with *only* those objects required for a deployment or subsequent update.
- A developer working with disparate objects in a solution can have trouble tracking or finding individual objects.
- An administrator exporting updated objects for deployment to a production environment can easily forget a required object.

To help you work with multiple objects, use *Package* objects. Package objects allow you to define a collection of shortcuts to objects that can be worked on and exported together. You can create a package in the Library Browser just as you would create any other object type. After you create a Package object, double-click it to open the current version in the Package Editor dialog.

In the Package Editor, check one or more objects and folders and click the Add button on the toolbar to add objects to the package. The absolute path for each folder appears. For objects, you can specify an absolute or relative path.

If you add a folder, the Package Editor includes every object in the folder. To remove an object from a package, click the object and then click Delete.

Delete or Restore an Object or Folder

When you no longer need an object, empty folder, or folder containing objects, you can delete it. The application moves deleted items to the Recycle Bin as a convenience and loss prevention measure. After you delete an object and it moves into the Recycle Bin, you have three options for handling it:

a. You can *restore* it.

b. You can leave it in the Recycle Bin.

c. You can permanently delete or *purge* it.

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, Type, or use the search features to locate the items you want to find.
6. To restore objects or folders, select them and click Restore Selected.
7. To permanently delete objects or folders, select them and click Purge Selected.

   Important! If you purge the objects or folders they are permanently destroyed with no way to recover them.

Search the Library Browser

You can 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.

   Your search is limited to objects and folders within the selected folder.
3. In the Search field, enter the name of an operator or folder.
4. (Optional) Click Advanced Search to define a search based on any of the following properties:
   - Name
   - Owner
   - Type
   - Check-In State
   - Creation Date
   - Modification Date
   - Keywords
5. Click Search.

   The search results appear.
6. You can work with objects and folders here just as you would in any other library folder. Double-click an item to open it for editing. Right-click a single item to view a shortcut menu of available commands. You can also select more than one item and then right-click any selected item to apply an action to multiple objects at once.
7. To begin a new search, do one of the following:

   a. Click Reset.

   b. Click Basic Search or Advanced Search again.

   c. Clear or modify the criteria specified in the search fields.

   d. Click the x in the Basic Search box.

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.

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 344).

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.

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

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:

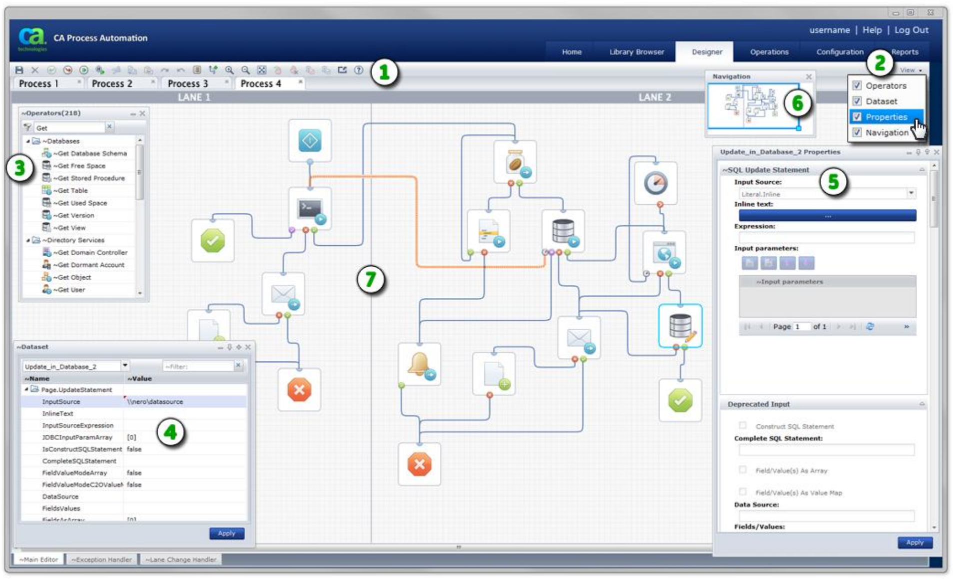
- Design and edit processes.
- Run, monitor, and control the execution of processes in production or test environments.
- Modify a running instance of a process to recover from an incident.

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: We recommend that you organize objects in folders to set rights, export them, and perform other tasks. Do not create objects at the root level because there is no way to manage them 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

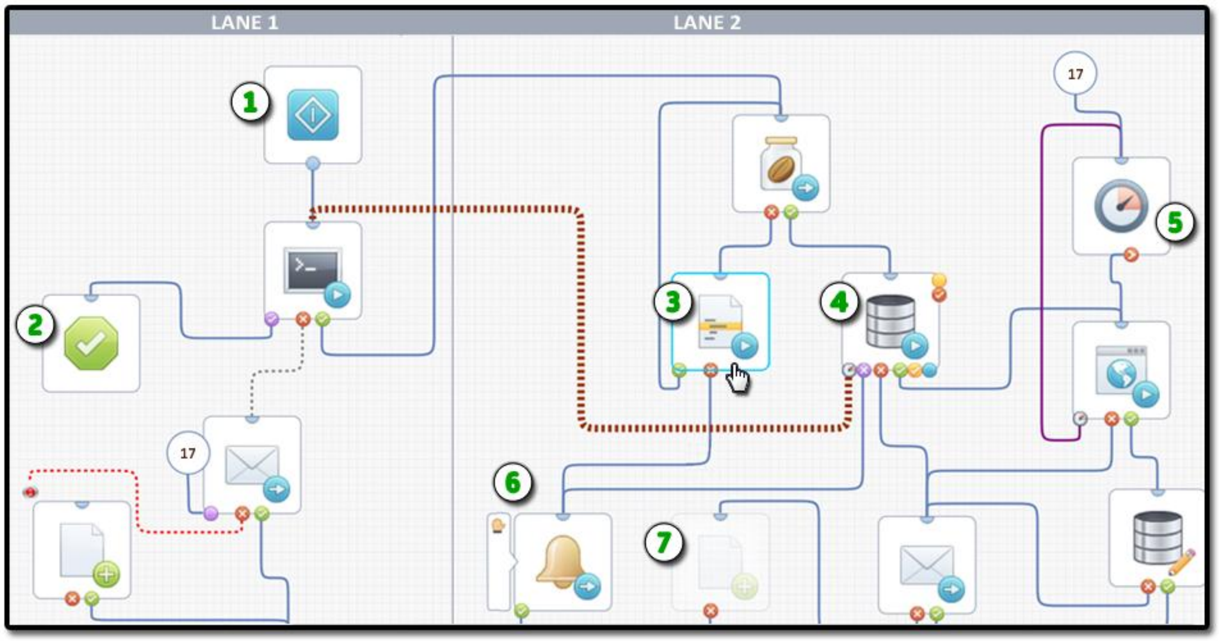
Keep the following tips in mind when working with processes:

- 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 60).

- 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.



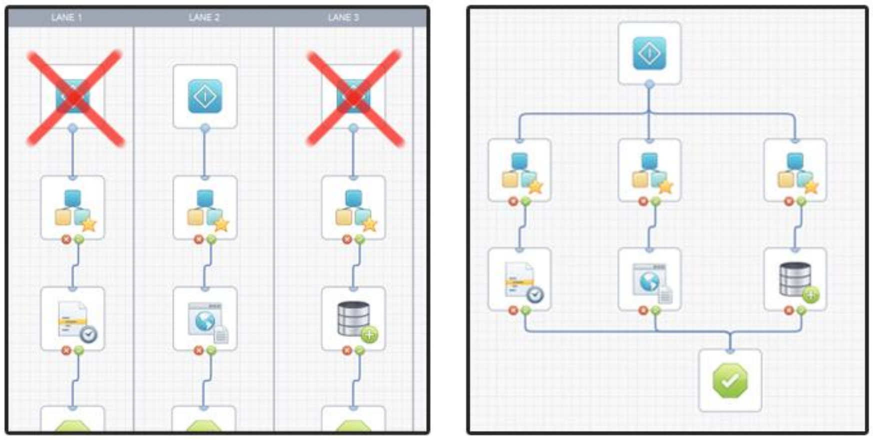
Item:	Description:
	Start Operator: All processes include at least one *Start* operator with no entry port and a single exit port. It is possible, although rare, to have more than one Start operator in a process.
2	**Stop Success Operator**: All processes include at least one *Stop* operator. Stop operators 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.

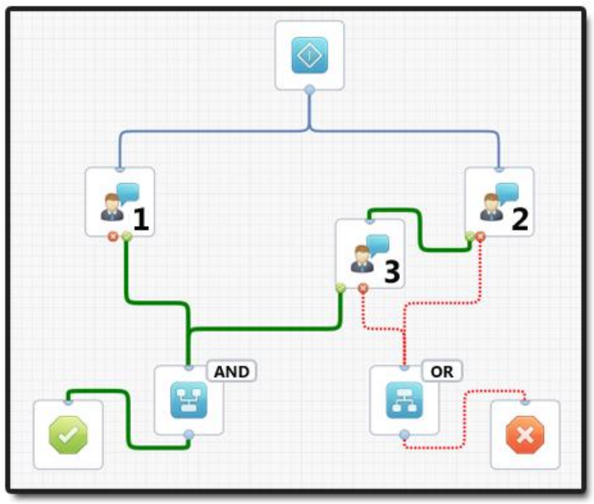
    The Process Designer tab appears. If not already checked out, click Check Out.
2. Drag an operator from the Operators palette onto the process layout anywhere below or to the right of the *Start* operator. Operators are arranged in folders for each *module* or operator category supported by CA Process 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*, and *Derivation*) are used to synchronize, unite, and split processing based on conditions. In this release, the *And* and *Or* operators are available in the Standard folder in the Operators palette. The Derivation operator is no longer available; when imported or migrated to this release from a previous release, it is converted to an *Or* operator.

Process Example: Conditional Logic

The following process example shows eight operators and nine links (2 blue, 4 green, and 3 red).



- The *Start* operator at the top initiates simultaneous processing of the first two linked operators. Let's call them *Task 1* and *Task 2*.
- *Task 3* is initiated only when *Task 2* is successfully completed.
- The *And* operator on the exit links for *Task 1* and *Task 3* is activated only after both *Task 1* and *Task 3* are successful. This path ultimately leads to the green *Stop Success* operator.
- The *Or* operator on the exit links for *Task 2* and *Task 3* is activated after an abnormal exit from either *Task 2* or *Task 3*. This path ultimately leads to the red *Stop Failure* operator.
- The Process Control operators running on an orchestrator run a process. You can run individual tasks in a process on any agent touchpoint 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

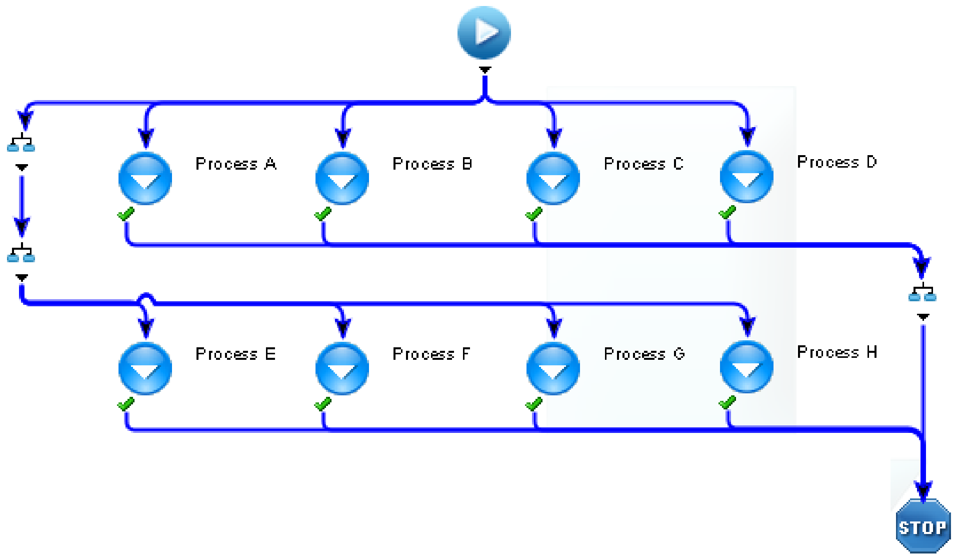
The *Or* operator activates exit links when any one of its entry links are activated. The Or operator can also be implemented with a single entry link to enable two or more output links for separate parallel branches.

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 Derivation Operator

Any *Derivation* operators imported or migrated to the current release will be converted to *Or* operators. The Derivation operator was used in previous releases to group links to make a process more readable. Logically, the Derivation operator is the same as the Or operator. It was smaller and took less space in a complicated process. The Derivation operator was most often used to neatly group links originating from or exiting to multiple operators.

The following example from a previous release shows a process using several Derivation operators to keep links from cutting across operators. The logic represents an *Or* condition with many different input links. The Stop operator is activated after any one of the eight processes exits successfully.



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

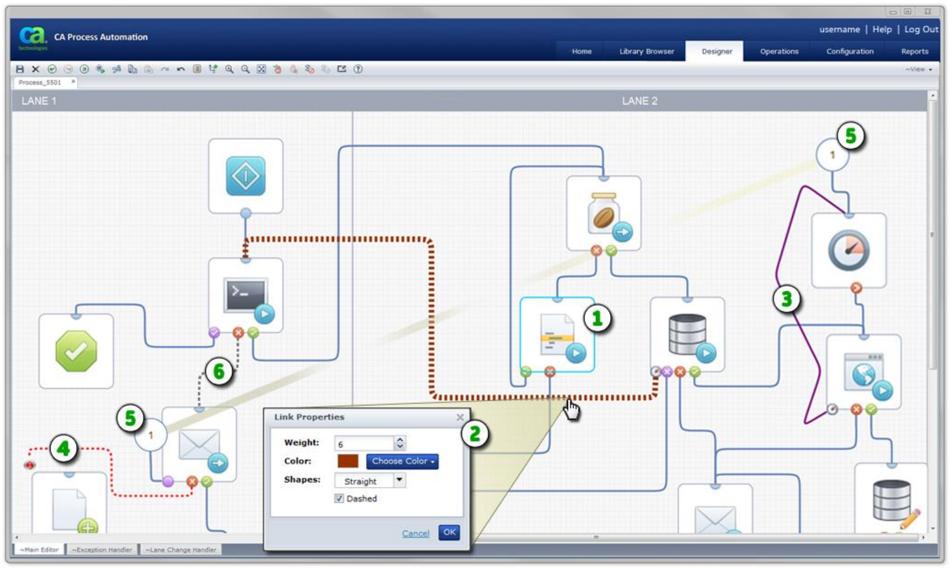
Links between operators define dependencies. The links act on the results produced by each operator. Links define the order and logic of a process as it flows.

Different kinds of actions have different predefined results or outcomes (such as successful, completed, aborted, failed, or timed out). The application calculates these outcomes to determine what exit conditions, ports, and links to activate next, in a logical sequence. For example, on some operators, you can add a custom port and define it to activate when an expression returns a *True* value.

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

When a process runs, its operators are processed only once. When a link leads to an operator that has already been activated, the destination operator is not processed again and the branch extended by the link ends.

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



Weight: 6
Color: 🟥 Choose Color
Shapes: Straight
[✓] Dashed

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**: Double-click a link, or right-click and choose Link Properties, to adjust its appearance. Select the thickness, color, shape, and dashed appearance of each link.

Item:	Description:
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 needed.
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 branch of the process 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 and choose Join Links.
⑥	**Disabled Link**: This link appears in a dashed gray color to indicate it is temporarily disabled. You can right-click to enable it again.

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.

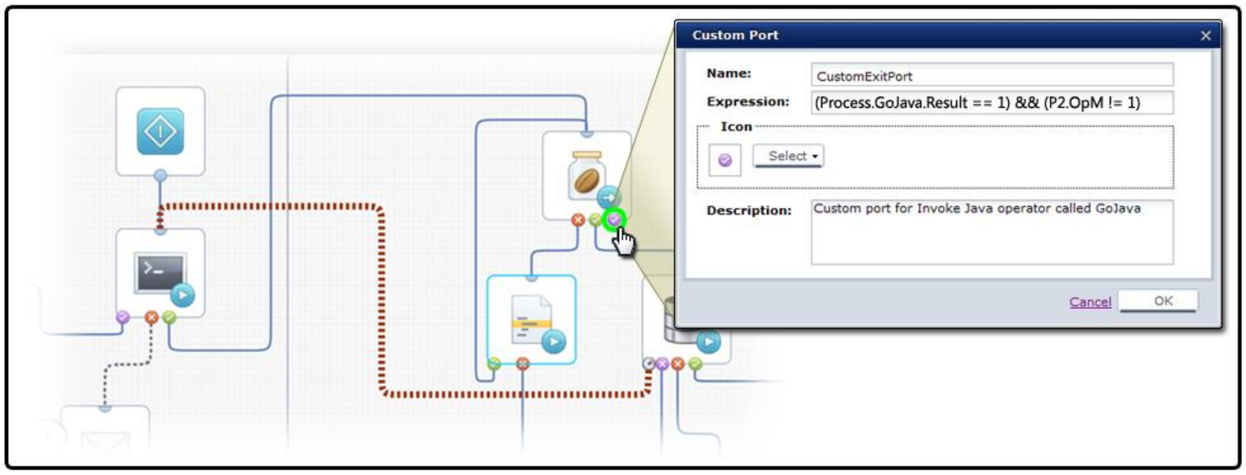
5. Repeat these steps, adding links and, if necessary, ports between operators to define the process flow.
6. In the toolbar, click Save.

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.



Name: CustomExitPort

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

Description: Custom port for Invoke Java operator called GoJava

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 359)

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).

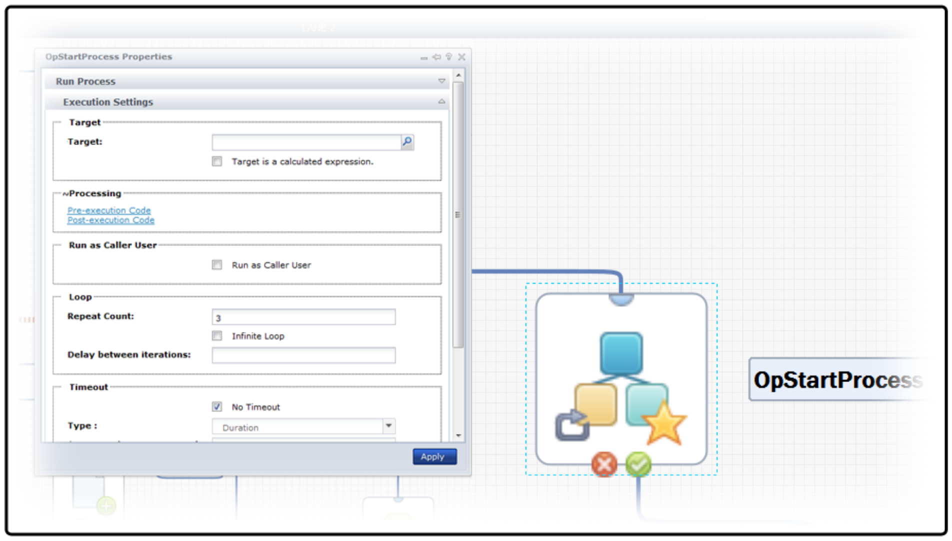
Loop an Operator in a Process

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:



OpStartProcess Properties

Run Process

Execution Settings

Target

Target:

☐ Target is a calculated expression.

Processing

Pre-execution Code
Post-execution Code

Run as Caller User

☐ Run as Caller User

Loop

Repeat Count: 3

☐ Infinite Loop

Delay between iterations:

Timeout

☑ No Timeout

Type : Duration

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.

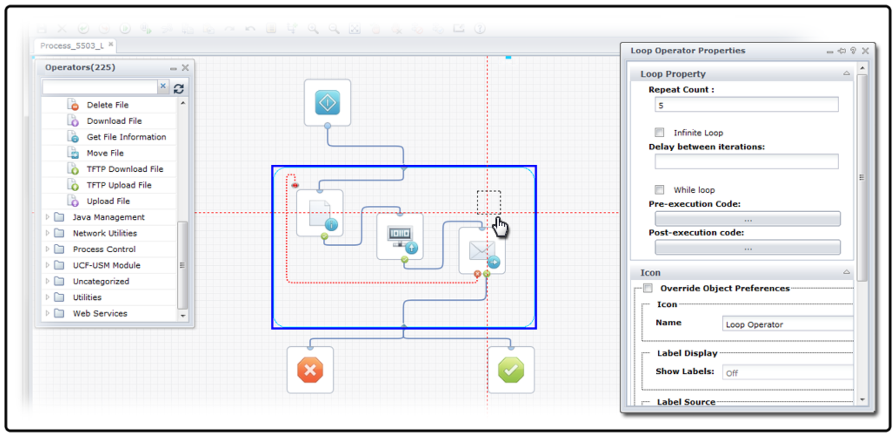
Loop a Series of Operators

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.



Loop Operator Properties

Loop Property

Repeat Count : 5

Infinite Loop

Delay between iterations:

While loop

Pre-execution Code: ...

Post-execution code: ...

Icon

Override Object Preferences

Icon

Name: Loop Operator

Label Display

Show Labels: Off

Label Source

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.

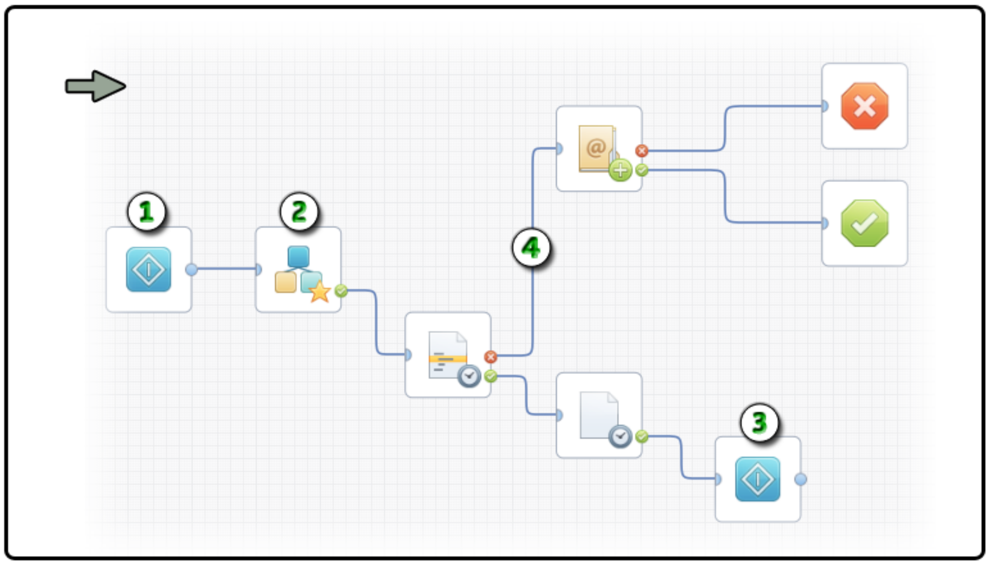
Daemons and Other Looping Processes

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.

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.

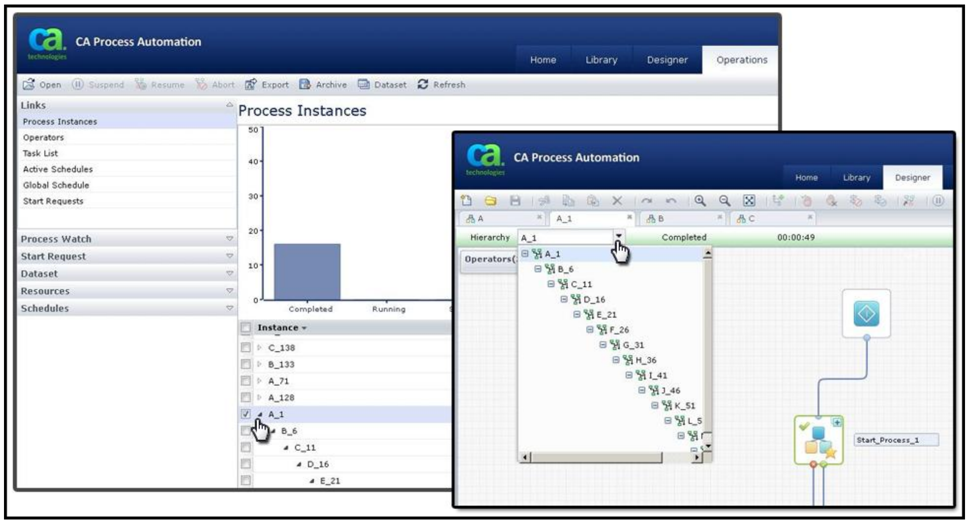
   Note: The Inherit Lane Change Handler from parent process check box is enabled only for inline child processes. CA Process Automation evaluates the child process when the parent process starts. The inline child process is loaded and expanded in the parent process.

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 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 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*)

A lane change rule looks like any sequence of operators on the Main Editor tab, except that each rule starts with a Lane Change operator instead of the Start operator. Properties on the Lane Change operator define the transition between source and destination lanes that invoke the sequence of operators linked to the Lane Change operator.

To create a lane handling rule

1. At the bottom of the Process Designer, select the Lane Change Handler tab.
2. From the Operators palette Standard group, drag a Lane Change operator onto the Lane Change Handler panel.
3. Double-click the Lane Change icon, and in the Properties pane, click the Lane Change 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 additional Operators onto 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 use comments, your own operator names, and object labels to document a process. Appropriate comments, naming, and labels help other designers understand what your process does and how it is constructed. Use comments for documenting a process, and for documenting chains, processes, or regions in a process. Comments remain stationary in a process. Do not use comments to label individual 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 an operator to label it.

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. This field is defined by the operator Text property. The Operator Information option on the View menu hides or displays this text field next to operators in a process.

CA Process Automation generates a default value for the Text property by combining the operator name and operator parameter settings.

To change the Text property of an operator

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

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

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

Embedded 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.
- Embedded content facilitated by shortcut links to specific views saves time accessing important and relevant information.

Embedded Content Links

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

In place of [Server URL] use the URL for your CA Process Automation deployment. For example:

```
http://<server>:CA Portal/itpam/Web.jsp
```

In place of <path> specify the absolute path to the automation object and its name in the Library Browser. For example:

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

Note: These links are case sensitive.

Automation Objects:

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)

Use this format to access the Process Watch as seen on the Operations tab:

`[Server URL]?page=processwatch`

Task Lists

Use this format to access *all* tasks as seen on the Operations tab:

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

Use this format to access *group* tasks as seen on the Operations tab:

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

Use this format to access *my* tasks as seen on the Operations tab or Home tab:

`[Server URL]?page=tasklist`

Process Instances

Use this format to access a runtime instance of a process in its own window:

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

Forms

Use this format to access the Start Request Form or Interaction Request Form used to reply to a task:

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

Use this format to access a list of Start Request Form instances as seen on the Operations tab:

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

Object Versions

Append the URL with the *versionid* parameter to access a specific checked in version of any library object except a process:

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

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

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

Process Versions

Append the URL with the *version* parameter to access a specific checked in version of a process:

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

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

```
[Server URL]?page=processeditor&refPath=/TestFolder/MyProcess&version=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 two steps:

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

To add a security-related variable to the context of a running process at runtime

1. Do one of the following:

   - Add a security-related variable to the context of a running process instance at runtime 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";
```

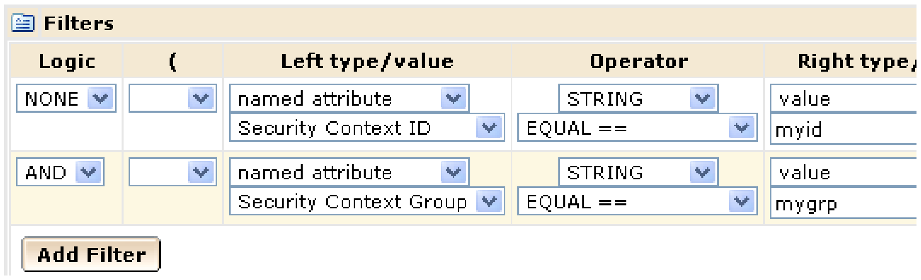
OR

- 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>
```

To create a policy for multi-tenancy

1. Add a new policy in EEM using Object as the Resource Class Name.
2. Specify the Identities for which this policy is valid.
3. Specify Resources so that it matches the path of the process for which the policy is created.
4. Specify the required policy actions.
5. Add filters and specify values for the named attributes Security Context ID and Security Context Group.



Filters

Logic	(	Left type/value	Operator	Right type
NONE		named attribute / Security Context ID	STRING / EQUAL ==	value / myid
AND		named attribute / Security Context Group	STRING / EQUAL ==	value / mygrp

This policy applies to all process instances when the values of the named attributes match those in the process instance at runtime.

Inherit Security in Sub-Processes

Sub-processes 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.

Chapter 5: Operators and Icons

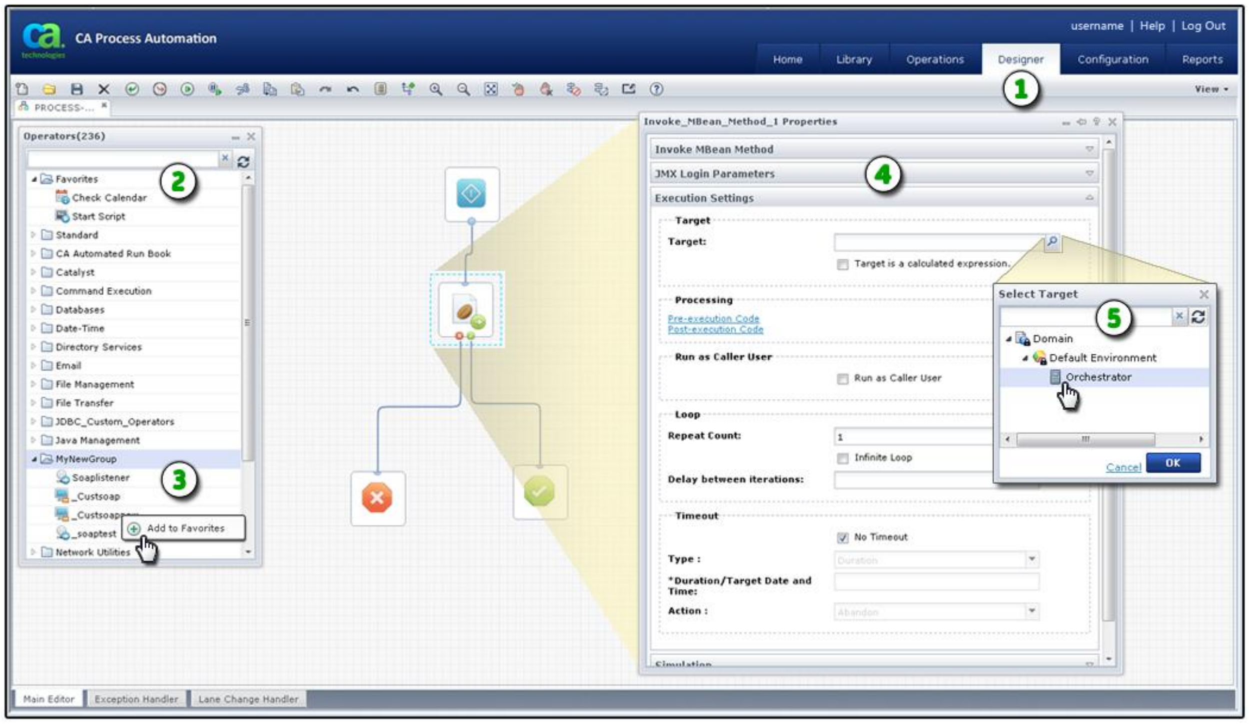
CA Process Automation carries out the instructions in the operators that you add to a process or to a schedule. This section describes how to configure operators after you have added them to a process or schedule object. This section also includes information on custom operators and connectors.

This section contains the following topics:

Operators

Process Operators (see page 93) were introduced in a previous section of this guide. To learn more about each operator, see the *Content Designer Reference Guide*.

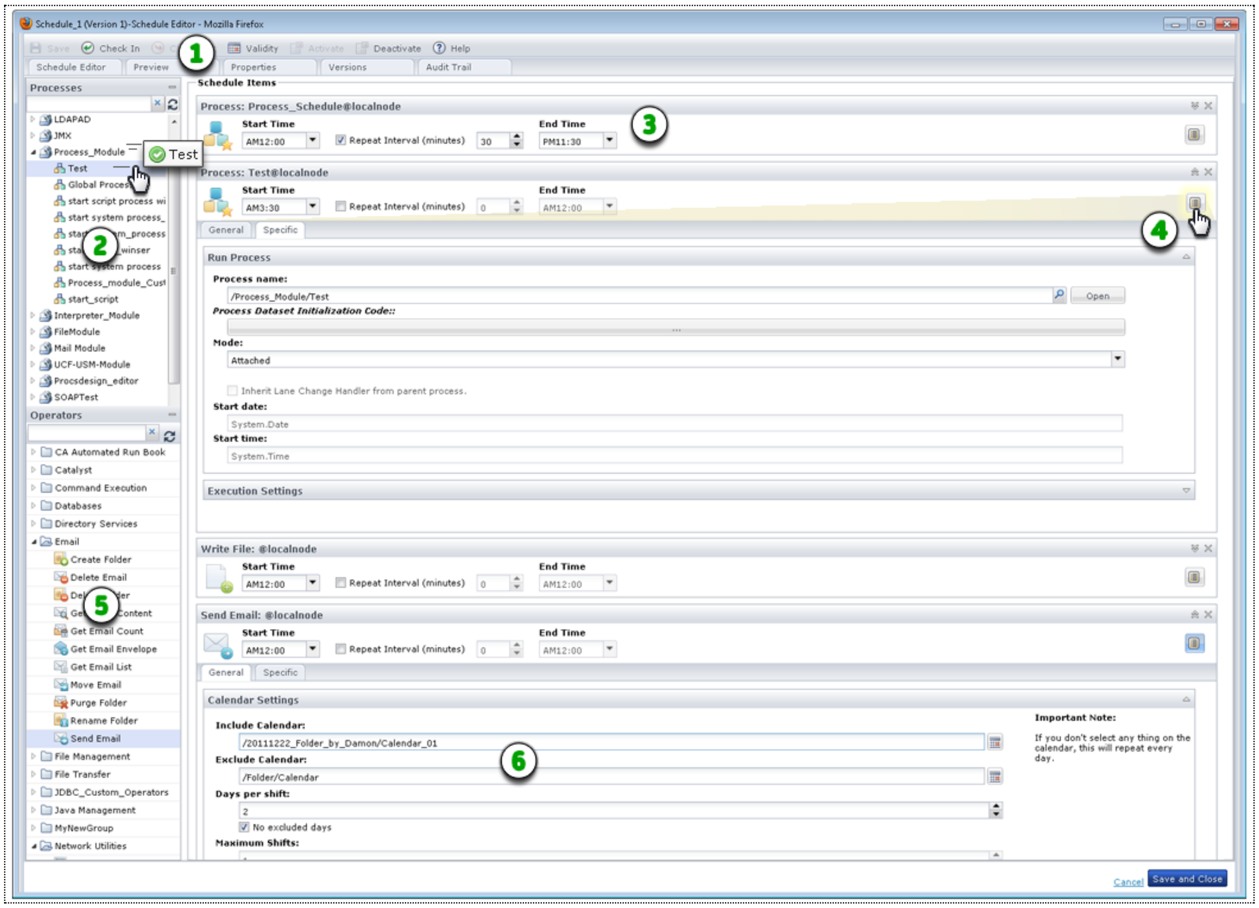
This section provides general guidelines for configuring properties and working with different types of operators including custom operators.



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 check 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 additional 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 additional windows.

You also work with operators when configuring schedules.



Schedule Items

Process: Process_Schedule@localnode
Start Time: AM12:00 — [✓] Repeat Interval (minutes): 30 — End Time: PM11:30

Process: Test@localnode
Start Time: AM3:30 — Repeat Interval (minutes): 0 — End Time: AM12:00

General | Specific

Run Process
Process name: /Process_Module/Test (Open)
Process Dataset Initialization Code::
Mode: Attached
Inherit Lane Change Handler from parent process.
Start date: System.Date
Start time: System.Time

Execution Settings

Write File: @localnode
Start Time: AM12:00 — Repeat Interval (minutes): 0 — End Time: AM12:00

Send Email: @localnode
Start Time: AM12:00 — Repeat Interval (minutes): 0 — End Time: AM12:00

General | Specific

Calendar Settings
Include Calendar: /20111222_Folder_by_Damon/Calendar_01
Exclude Calendar: /Folder/Calendar
Days per shift: 2
[✓] No excluded days
Maximum Shifts:

Important Note:
If you don't select any thing on the calendar, this will repeat every day.

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, check in, or save the schedule.
2	**Processes Pane**: Select the processes that you want to include and drag them into the Schedule Items page.
3	**Scheduled Process in Queue**:Set the duration and frequency for running the process within a single day.
4	**Process Properties**: Click the Properties button to view process properties on the General and Specific tabs.

Item:	Description:
⑤	**Operators Pane**: Select the operators that you want to include and drag them into the Schedule Items page.
⑥	**Operator Properties**: Click the Properties button 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. Expand the panels to configure the fields.
7. Click Save.

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

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 142).
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 even 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 have not already configured the Java module, complete this task first. See Configure the Java Module (see page 140).
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:
   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 cannot require authentication and must not be 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 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 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, if MyAccount.java belongs to package com.ca.tech; and MyAccount.class is in:
      ```
      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).

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 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.

5. Specify the code that you want to run.

6. Specify the input parameters to pass to the Java code.

7. Specify the output variable names. These are the names of the variables created in the operator Java code which must be saved in the operator dataset at the end of execution of the code.

   Note: Any Java object that is not a boolean, date, integer, number, string, character, or array of any of these types, is serialized and saved as a JavaObject.

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 any exceptions or errors 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 your own *Custom Operator* objects that users can add to their processes or schedules just 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 extend new functionality that can be shared across your IT organization.

As a simple example, you could create a custom operator that retrieves a specific piece of information from a network router using the Get SNMP Variable operator as its base. 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. Once published, any user can easily access this specific information from a network router by placing your custom operator in their process or schedule and simply configuring the IP address. They do not have to know the SNMP variable OID or the community string. Those values have already been configured and are likely hidden from the user.

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

- Mission-critical applications; organizations often develop and deploy these applications internally.
- Web services (SOAP)
- Command line and scripting applications.
- JMX
- SNMP

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 added advantage of easily adapting to changes in your IT environment. For example, if changes in your IT environment require you to change the way you interface with an application, then you can change only the original Custom Operator object. All processes and schedules using that custom operator would automatically apply the changes in the latest designated version. Thus, any 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
- 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 158)

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 Remove.

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 158)

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.

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.

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.

Custom Operator Specific Pre and Post Execution Code

CA Process Automation allows you to 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. Users of a custom operator cannot override any pre-execution or post-execution code already defined by designers or developers with more permissions.

The order of execution of the code is as follows:

1. User-defined pre-execution code.
2. Custom operator specific pre-execution code.
3. The actual operation.
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.

Order of Execution for Custom Operator-specific Pre-Execution Code

The order of execution for the Custom Operator specific Pre-execution code is as follows:

1. Execution of the user defined Pre-execution code i.e. Pre-execution code defined in the process.
2. Evaluation of the User Parameters that includes expression evaluation and macro expansion.
3. Execution of the Custom defined Pre-execution code. The User Parameters are exposed and you can access the parameters using the Operator keyword.

   Note: You do not have the permissions to change the User Parameter values.
4. Evaluation of the Standard Parameters including expression evaluation and macro expansion.
5. Evaluation of the Base operator.

Custom Operator: Dataset Tab

Use the Dataset tab to define and group operator dataset variables that contain information that is returned by the custom operator. For example, if your custom operator retrieves some fields from a ticket in an in-house ticketing system, you can create a page to group all the retrieved fields and place the parameters in the page corresponding to the returned fields.

Doing this as part of your Custom Operator provides two benefits:

- Users of your Custom Operator can easily see the definition of the output of the Operator without having to run the Operator first to have the parameters created at runtime.
- You can group related parameters in a page with a descriptive name, which is not possible at runtime.

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.

More information:

Datasets (see page 173)

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

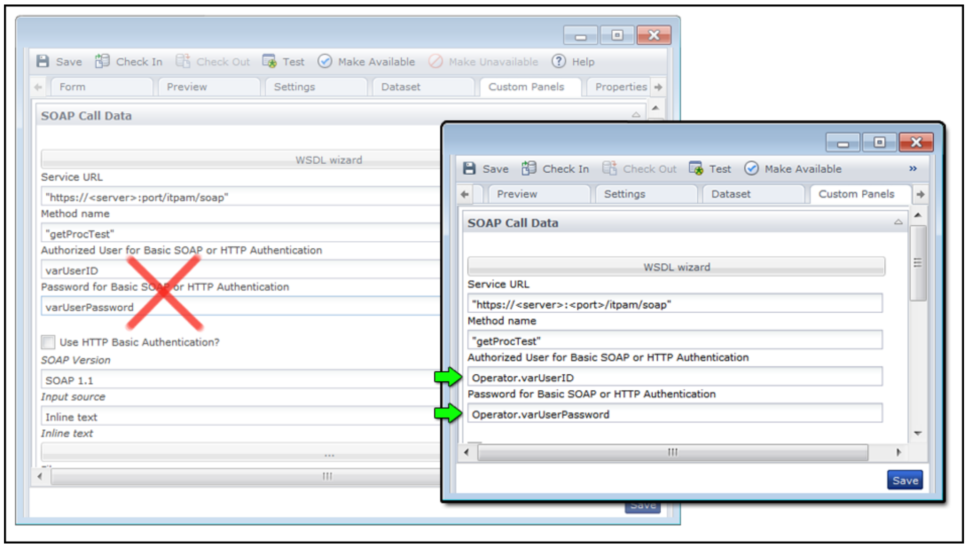
Use this large group of fields to manage security details. Expand each group box by clicking its title to view all the fields.

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
```



Service URL
"https://<server>:port/itpam/soap"
Method name
"getProcTest"
Authorized User for Basic SOAP or HTTP Authentication
varUserID
Password for Basic SOAP or HTTP Authentication
varUserPassword
Use HTTP Basic Authentication?
SOAP Version
SOAP 1.1
Input source
Inline text
Inline text

Service URL
"https://<server>:<port>/itpam/soap"
Method name
"getProcTest"
Authorized User for Basic SOAP or HTTP Authentication
Operator.varUserID
Password for Basic SOAP or HTTP Authentication
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.

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.

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 that enable additional 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 also provide bridge services from other products to CA products and solutions that embed CA Catalyst. In addition to the connectors that are provided on the CA Process Automation installation media, many connectors are available for download. Licensing restrictions may apply. A list of the most popular connectors follows:

- CA Workload Automation AE
- CA CMDB
- CA Configuration Automation
- CA eHealth
- CA Client Automation
- CA NSM
- CA Service Desk Manager
- CA Spectrum IM
- CA Workload Automation AE
- Amazon Web Services (AWS)
- AS400
- BMC Remedy
- Hyper-V
- IBM z/OS
- Microsoft Access
- Oracle
- Sybase
- VMware
- vSphere
- UCF\Catalyst

Use application-specific connectors to perform tasks, such as gathering data or applying actions, on target systems and target applications. Connectors provide operators that are executed either on the orchestrator, on agents residing on the application server or 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:

a. 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 might have more than two options, but it is always a small and well-defined set of possible values that are returned.

b. Data Collection

   Collects more complex datasets from the target application. The result is usually stored in a local dataset and can be further analyzed by other operators. If required, the data can also be made global so that it can be used by other Processes.

c. Active Management

   Performs actions on the target system. This includes all operators that, in any way, change the behavior of the target system. A few examples of this would be: sending an event, reconfiguring the application, starting or stopping a related service.

In some cases, a single operator may perform more than one of these functions. For example, a connector might perform an action and return a result set based on that update.

CA Process Automation includes a large number of connectors. It is also easy to create additional connectors and their related operators by calling standard, application-specific executables located on the agent or by using standard scripting languages for more complex functions.

To minimize the application footprint and the complexity of the user interface, only a set of generic and commonly used connectors are installed by default. Other connectors should be installed only when necessary.

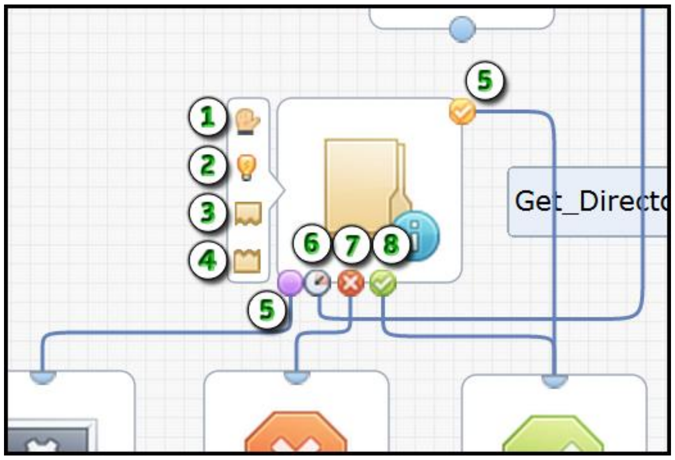
Operator Icons

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, called a *Custom 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 appear in the corners of all icons to represent different execution states in a process such as *idle*, *running*, *completed*, and *failed*.

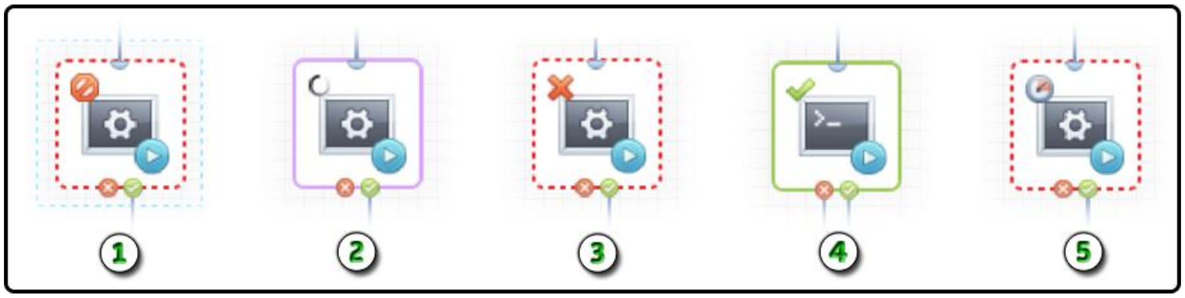
Operator Status Icons

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. Custom icons assist you in identifying the specific function that an operator performs. You can create custom icons in any automation library folder, then apply them to any operator. Each icon displays your choice of base image with a modifier image overlaid in the lower-right corner of the base. 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.

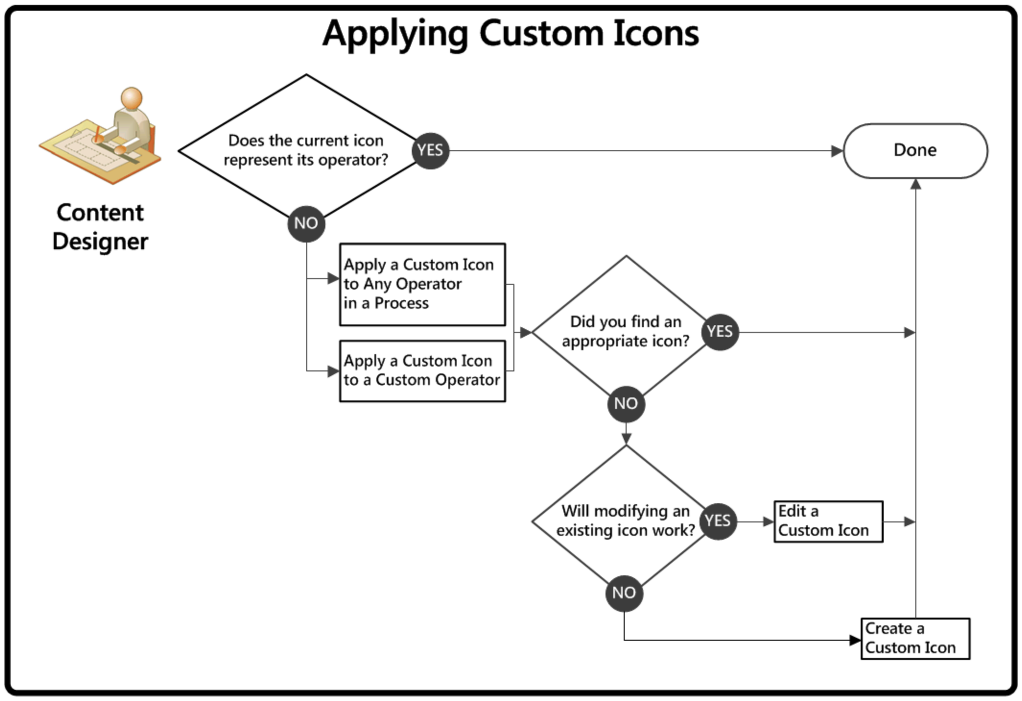
Simplicity

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:


Applying Custom Icons
Content Designer
Does the current icon represent its operator?
YES
Done
NO
Apply a Custom Icon to Any Operator in a Process
Apply a Custom Icon to a Custom Operator
Did you find an appropriate icon?
YES
NO
Will modifying an existing icon work?
YES
Edit a Custom Icon
NO
Create a Custom Icon

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 169): 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 170): 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 168): 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 168): 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.

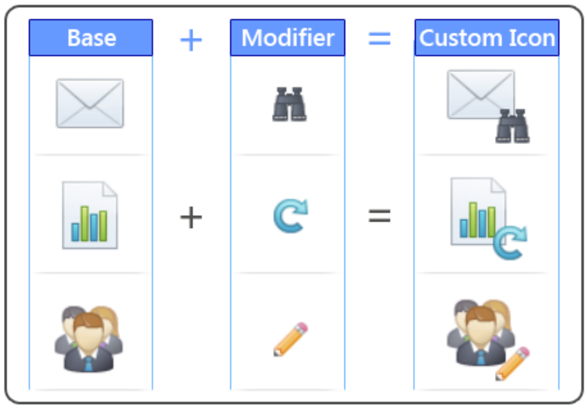
Diagnose (or Monitor) Performance

Add User Account

Debug Script or Code

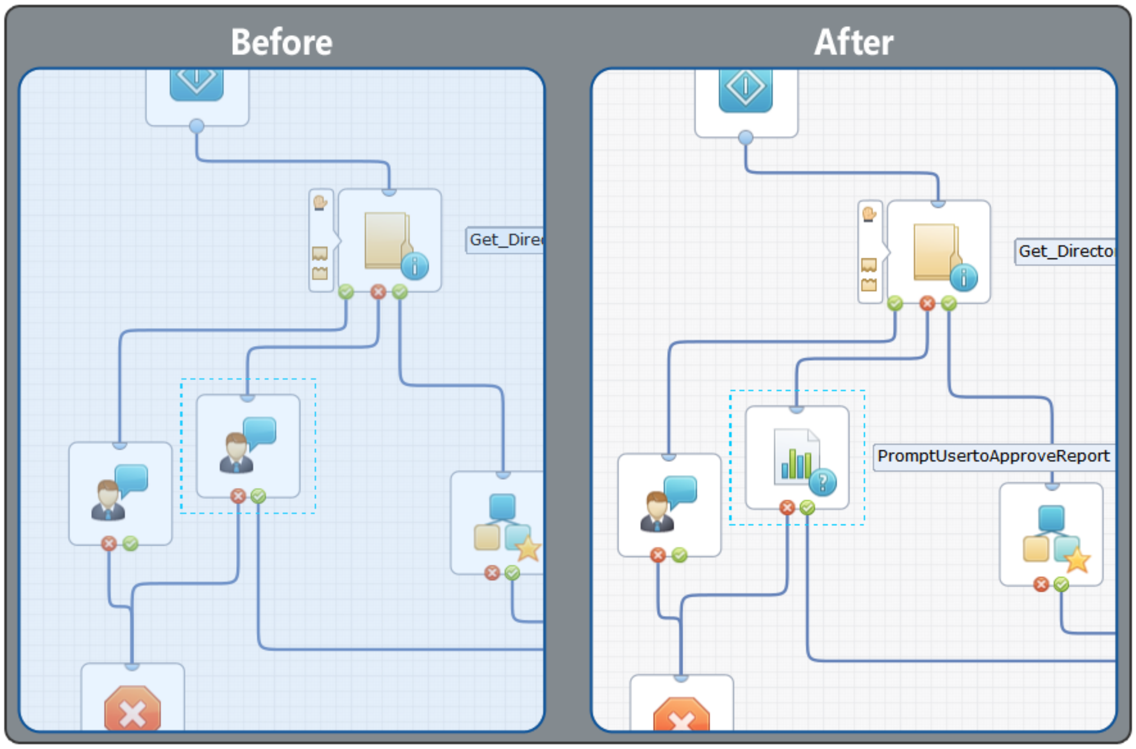
Example: Create or Edit a Custom Icon

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

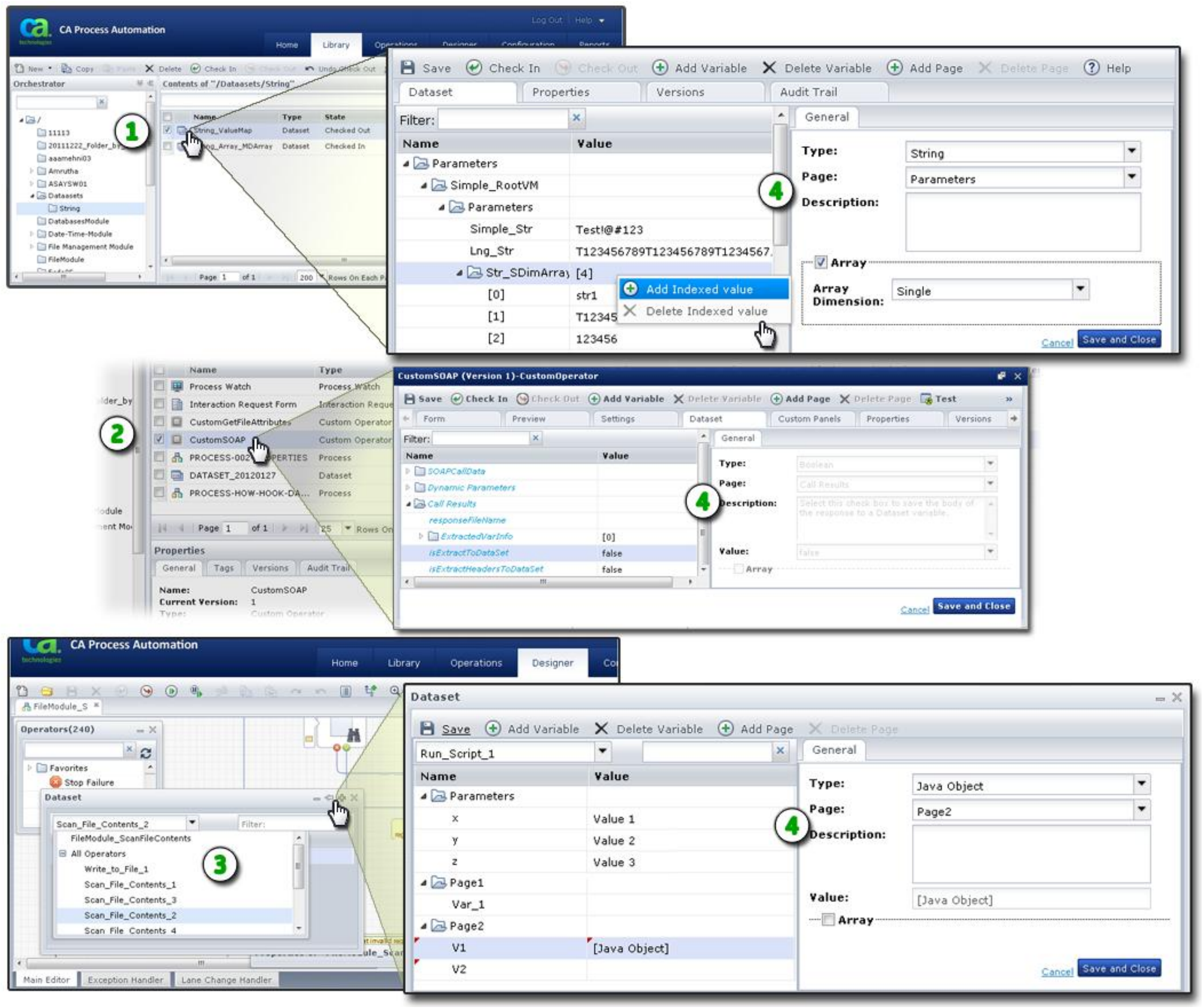
Datasets let you define groups of variables to store and organize data. Datasets provide a way to share data across multiple process instances.

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.



Name	Value
Parameters	
Simple_RootVM	
Parameters	
Simple_Str	Test!@#123
Lng_Str	T123456789T123456789T1234567.
Str_SDimArray	[4]
[0]	str1
[1]	T12345
[2]	123456

Type: String
Page: Parameters
Description:
Array
Array Dimension: Single

CustomSOAP (Version 1)-CustomOperator

Name	Value
SOAPCallData	
Dynamic Parameters	
Call Results	
responseFileName	
ExtractedVarInfo	[0]
isExtractToDataSet	false
isExtractHeadersToDataSet	false

Type: Boolean
Page: Call Results
Description: Select this check box to save the body of the response to a Dataset variable.
Value: false
Array

Dataset

Run_Script_1

Name	Value
Parameters	
x	Value 1
y	Value 2
z	Value 3
Page1	
Var_1	
Page2	
V1	[Java Object]
V2	

Type: Java Object
Page: Page2
Description:
Value: [Java Object]
Array

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 227).

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.
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 230).
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 231).

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 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 new page, click Add Page.
6. To create a new variable:
   a. Select a page.
   b. Click Add Variable.
7. Click a page or variable to rename it. Names must start with a letter and 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. Select a different page to move the variable to that page. For custom operators, the page 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 entered here when you move the mouse pointer over the field name or value.

   Value

   Specifies the default value if the field is blank or cleared. For fields of type integer, long, and double, the default value is zero. For fields of type integer, double, long, or string, 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.

You can also enter constraints for a field of these types:

- For fields of type *Boolean*, you can select True or False.
- For fields of type *Object Reference*, click the browse button (...) to select a dataset object.
- For fields of type *Date*, click the browse button (...) to select a date.

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 titlebar 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.

Edit Validation Settings for a Dataset Field

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
Dataset	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.
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.
String	Specifies the regular expression mask for text. **Note:** For more information about specifying string patterns, see Using Masks to Specify Patterns in Strings (see page 367). Minimum length is the minimum allowed number of characters for string values. Maximum length is the maximum allowed number of characters for string values.

Data Type	Validations
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 a Dataset Expression

When working with datasets, you often need 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:
    a. Double-click a dataset.
    b. Right-click a dataset and choose Edit.
    c. Right-click a dataset and choose Manage Dataset.
3. In the resulting window, click an array or a specific variable to view it.
4. Right-click it and choose Dataset Expression.

    The Dataset Expression window appears.
5. View or copy the expression.

Read Operating System Values into Dataset Variables

CA Process Automation can read values generated by a shell process into dataset variables. Before a UNIX Script or Windows Script Operator runs its associated script, it creates a folder to accept values generated by the script. The C2OSVD environment variable specifies the full path to the folder (for example, C:\TMP_VS_559) 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 passed back to CA Process Automation using the C2OSVD directory populates variables in the script Operator Dataset.

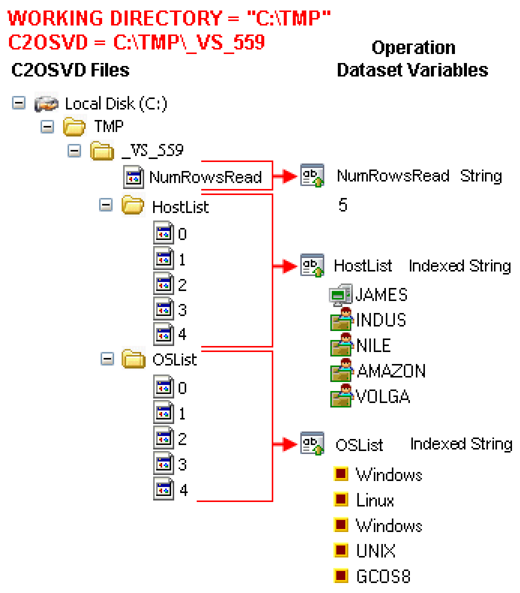
A script must save data to text files in the C2OSVD directory. After a script Operator completes its script (but before it performs post-execution actions), it determines whether there are any files in the location pointed to by the C2OSVD environment variable. Operator Dataset variables are then created according to the following rules:

- The Operator saves the contents of a file in the C2OSVD location to a string variable in the Operator Dataset with 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 (0, 1, 2, ..., n) located in the folder to the 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.
- A script should create files in the C2OSVD location with the same names as the variables you want to appear in the Operator Dataset. The script should create text files, but the files should not have any file extension (such as .txt), or the extension is included in the variable name.

The following illustration shows how the file-to-variable conversion works 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 following illustration shows two folders %C2OSVD%/HostList and %C2OSVD%\OSList containing five files named 0, 1, 2, 3, and 4. The script writes a computer name to each of the numbered files in the HostList folder and an operating system name to each of the numbered files in the OSList folder. CA Process Automation creates two zero-based indexed variables after running the script, HostList and OSList, and assigns the contents of the numbered files in the HostList and OSList folders to the corresponding elements of the indexed variables.

In the illustrated example, the Windows script uses the C2OSVD variable to create the file %C2OSVD%\NumRowsRead. CA Process Automation creates a corresponding variable after running the script, NumFilesRead, in the script Operator Dataset and assigns the contents of the NumFilesRead file to the variable.



When a script operator finishes, it deletes the C2OSVD folder and its contents. The post-execution code on the script operator can access the operator dataset variables. Typically, the post-execution code copies the operator dataset variable values to local variables in the process dataset or to operator dataset variables in subsequent operators in the process. The three example scripts included in this chapter show how the example illustrated previously is implemented using UNIX script, VBScript, or PerlScript.

Sample Scripts for Reading Operating System Values into Dataset Variables

A process can read operating system variables into operator dataset variables using UNIX script, VBScript, or PerlScript. Each script reads in lines from a text file specifying a host name and the operating system running on the host with a single space separating the two strings, as follows:

```
JAMES Windows
INDUS Linux
NILE Windows
AMAZON UNIX
YANGTZE Solaris
GILA UNIX
```

The three scripts all save the host names to an indexed field named HostList and the operating system names to an indexed field named OSList.

The process incorporates these steps:

1. The UNIX or Windows module passes the name of a text file specified by a parameter defined in the Parameters field on the UNIX Process or Windows 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.

   The folder specified by the C2OSVD environment variable is used by only one instance of a script Operator and not repeated or overwritten by any subsequent instance of that or any other script Operator .

3. The UNIX or Windows module creates the folder pointed to by the C2OSVD environment variable.

4. The script creates folders named HostList and OSList in the folder pointed to by the C2OSVD environment variable.

5. The script reads each line of the text file and writes the host names to sequentially numbered files in the %C2OSVD%\HostList folder and 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 finishes by writing the number of lines read to the files to a file named %C2OSVD%/NumRowsRead.

   This file generates an operator dataset variable 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 WinGetInfo Operator creates two indexed fields, HostList and OSList in the WinGetInfo operator dataset. It reads the source file line-by-line and assigns host names (strHost) to indexed values in sequential HostList fields and operating systems (strOS) 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 oArgs.Item variable is populated by the entries of the Parameters input area of the calling Run Script operator, with the first entry populating oArgs.Item(0). The number of rows read are assigned to the operator dataset variable named *NumRowsRead*. The Wscript.sleep 30000 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.

```
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. 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. ARGV is populated by the entries of the Parameters input area of the calling Run Script operator. 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.

```
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. To view or edit variables in the process dataset, click the Dataset tab at the bottom of the Process Designer window.
- 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. To view or edit operator dataset variables, click the Operator Dataset button on the Service Parameters properties sheet in the Properties pane. You can also click the Dataset tab and select the operator name from the drop-down menu in the palette.

More information:

Datasets (see page 173)
Calculated Parameters (see page 206)

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 213)

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"
```

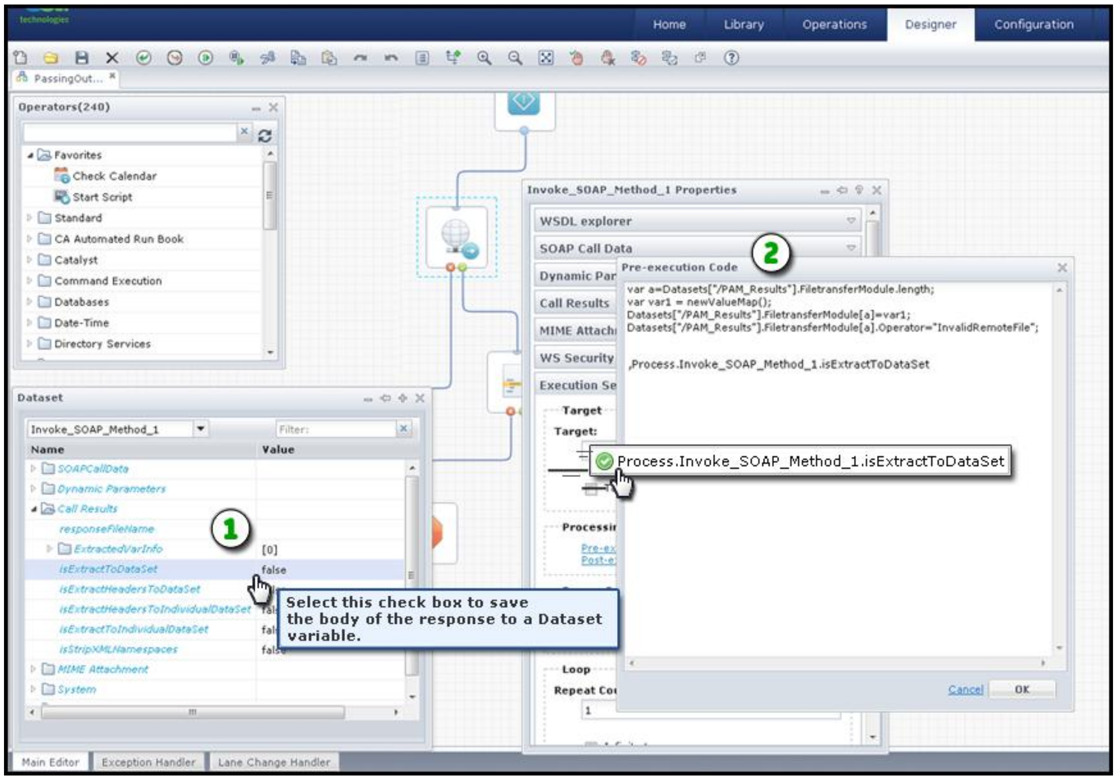
When you specify a path to a folder or object in a library, the root folder is represented by an initial slash mark, followed by the slash delimited folder hierarchy. For example:

```
"/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
```



Select this check box to save the body of the response to a Dataset variable.

```
var a=Datasets["/PAM_Results"].FiletransferModule.length;
var var1 = newValueMap();
Datasets["/PAM_Results"].FiletransferModule[a]=var1;
Datasets["/PAM_Results"].FiletransferModule[a].Operator="InvalidRemoteFile";


,Process.Invoke_SOAP_Method_1.isExtractToDataSet
```

Process.Invoke_SOAP_Method_1.isExtractToDataSet

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	**Dataset Variable in Pre-Execution Code**: You can drag and drop variables from datasets to property pages. Instead of keying in expressions that reference variables from datasets, you can drag the variable to generate the expression. This saves time and reduces potential errors.

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.

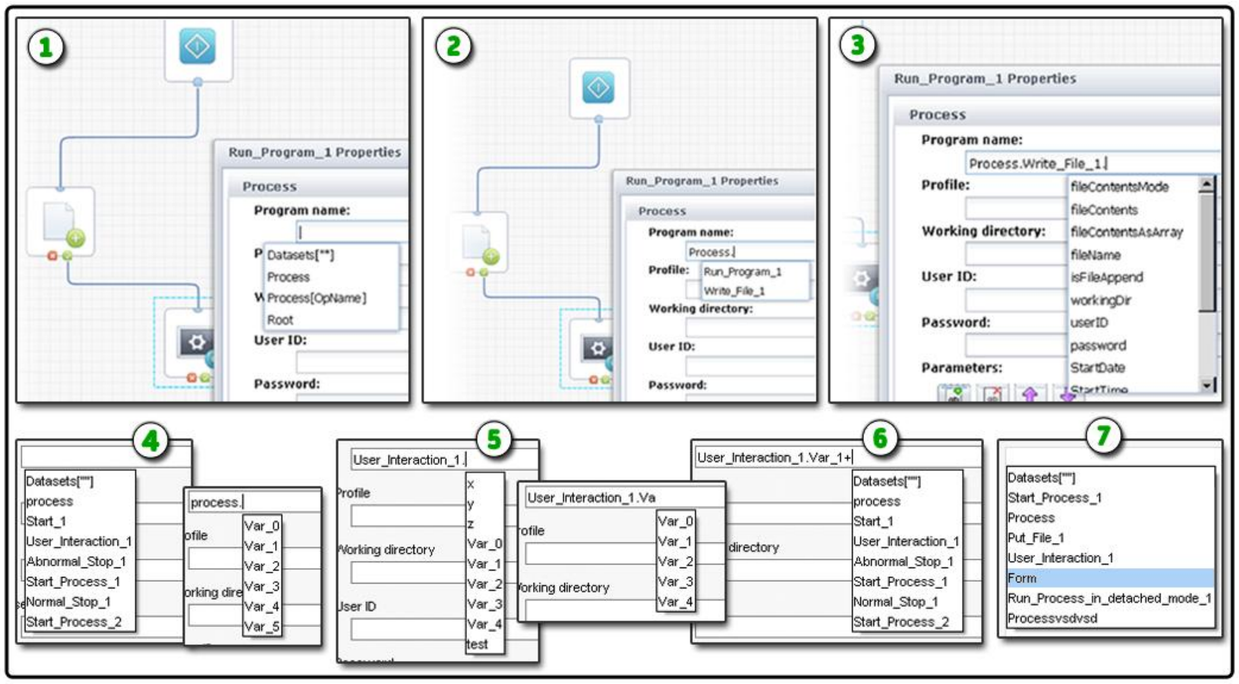
Dataset Variable Name Assistance

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. Determine the most efficient way to reference the target:

- If the target is an orchestrator, enter its touchpoint.

  Note: Do not specify the IP address of a computer hosting a clustered orchestrator. This is not a valid way to specify that an operator should run on that 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 where the touchpoint is mapped to multiple agents, enters its agent ID.
- If the target has no agent but has a proxy touchpoint, enter its proxy touchpoint.
- If the target meets none of the previous criteria, but is a remote host referenced by a host group, enter its IP address or FQDN.
- If you specify no target, the operator runs on the orchestrator where the process or schedule is being evaluated.

Important! Observe these guidelines to help ensure the most efficient processing. Enter an IP address or FQDN for a target only if the target has no associated orchestrator, touchpoint, or proxy touchpoint.

More information:

How Targets for an Operator Can Be Specified (see page 372)
Processing a Target Specified as an IP Address or FQDN (see page 373)

Operator Dataset Variables

The operator dataset contains variables associated with an operator. You can view, create, edit, or delete variables and their associated values during design time. Those variables are available to the operator 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 of these variables are standard and define information such as start time, stop time, and result, while other variables are information specific to each operator.

You can use dot syntax or bracketed 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. An element in an indexed field is returned by the following syntax where *n* is the element number:

```
field_name[n]
```

The pre-execution and post-execution code for an operator can access the name of the current operator using the *OpName* keyword. Use the following syntax to specify an operator dataset variable in the pre- or post-execution code of that same operator:

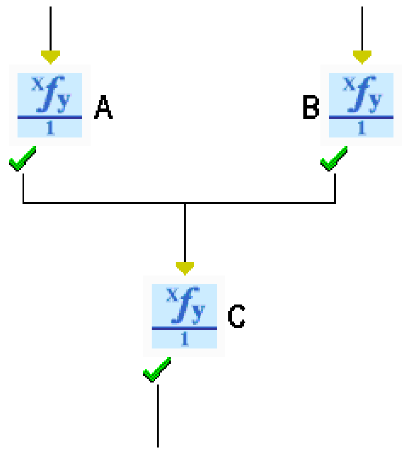
```
Process[OpName].field_name
```

For example, the following statements in the post-execution code assign the operator name and the value of its *Result* variable to the process dataset variables *iName* and *iResult* and create an operator dataset variable named *World*:

```
Process.iName = OpName
Process.iResult = Process[OpName].Result
Process[OpName].World = "Hello world!";
```

If you know the name of an operator at design time, you can use the literal name of the operator when referring to its dataset variables in an expression. Occasionally, however, you do not know the operator name at design time.

This can occur when you are editing pre- or post-execution actions for a Custom Operator object. This also occurs when you must access operator dataset variables in one of several operators, but you do not know which operator has executed until runtime. You can then use pre-execution or post-execution code in each operator to save its name to a variable, as we did with the *iName* variable in the previous example. It could be a process dataset variable or it could be an operator dataset variable belonging to the operator that requires the name. For example, in the following illustration, the process executes either Operator A or Operator B before reaching Operator C. Operator C can then use the operation name saved to a process variable by Operator A or Operator B 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 175)
[Calculated Parameters](#) (see page 206)

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 230)

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.
- The process is terminated.
- 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

- today
- formatDate
- parseDate
- rolldate
- adjdate
- now
- rolltime
- convertXml
- convertXmlURL
- applyXPath
- applyXPathURL
- convertValueToXml
- getEnvVar
- checkCalDate
- nextOpenDate
- hasField
- deleteValueMapField
- setElementValueByKey
- setFieldValueByKey
- setValByKey
- include
- load
- resolveNode
- isTouchpointUp
- absolutePath
- absPath
- absolutePath
- createRscDoc
- adjustResourceVals
- lockResource
- resetResource
- setResource

- getResourceTotal
- getResourceAvail
- newValueMap
- getValueMapFields
- deleteValueMapField
- deleteObject
- getSoapAttachments
- deleteSoapAttachments
- formatString
- logEvent
- existsDataset
- existsFolder
- existsAgenda
- existsCalendar
- existsProcess
- existsCustomOperator
- existsCustomIcon
- existsResource
- existsInteractionRequestForm
- existsProcessWatch
- existsSystem

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

Strings can be typed as literal values in an expression. Functions and JavaScript Operators in an expression also return them. To distinguish between identifiers (such as variable names) and literal strings, enclose the literal strings between string delimiters. You can select to 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, you would type it as follows:

```
"/usr/smart/program"
```

In any text box that an asterisk (*) marks, enter either an expression or a literal string between a pair of quotation marks. The asterisk indicates that CA Process Automation evaluates the contents of the text box as an expression. Do not delimit expressions between quotation marks.

Any text box that expects literal values is not labeled with an asterisk (*). In this case, do not delimit strings between quotation marks.

JavaScript uses the backslash character (\) as the escape character within 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'.

Any character following the escape character is interpreted literally rather than parsing by the CA Process Automation language interpreter. For example, to include a double quotation mark character in a string, precede it with the escape character, \”, so that the parser does not interpret it as the string delimiter. To include the backslash in a string, precede it with the escape character, \\.

If you want your string to include the same character as you are using as your delimiter (either single or double quotes), escape the character when not using it as a delimiter. For example, the following string that is delimited with single quotation marks 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)

Sequence	Character Represented
\b	Backspace (\u0008)
\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

Strings, integer, Boolean, and logical expressions can be built from a combination of entities (integers, strings, functions, and Datasets) using JavaScript Operators. JavaScript Operators are characterized by the number of operands that they expect. Most JavaScript operators are binary operators that combine two expressions into a single, more complex, expression.

JavaScript also supports several unary operators that convert a single expression into a single more complex expression. This section covers several sets of operators that are most commonly used in CA Process Automation expressions.

Array and Object Access Operators

JavaScript uses the dot (.) operator as an array element and object access operator. You can access elements of an array using either square brackets ([]) and elements of an object using a dot (.). JavaScript treats both [] and . as operators.

The . operator uses the following format:

object.identifier

- The identifier operand should not be a string or variable containing a string; it should be the literal name of the property, method, or variable name (in a Dataset), without quotation marks of any kind.
- The [] operator allows access to array elements and object properties. It also allows access to object properties without restrictions placed on the identifier operand by the . operator.

The [] operator uses the following formats:

- array[expression] // where expression evaluates to an array index

  In this case the first operand refers to an array, and the second operand (between brackets) is an expression that evaluates to an integer value for an array index.
- object[expression] // where expression evaluates to a property name

  In this case the first operand references an object, and the second operand (between brackets) is an expression that evaluates to a string that names a property of the object. Note that unlike the dot operator the second operand is a string instead of an identifier.

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

The JavaScript has the following operators for combining 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 there is more than one arithmetic operator in an expression, multiplication, and division are calculated first, followed by 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 has the following logical (or Boolean) operators for combining the outcomes of Boolean functions or Operators.

Operator	Description
&&	Logical AND
||	Logical OR
!	Logical NOT

The logical operators return True or False. The logical operators 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	Accesses a Named Dataset in a CA Process Automation Library using the following format:
		`Datasets[dataset_path].field_name`
		where *dataset_path* is a CA Process Automation expression that evaluates to the full path for a Named Dataset in the current Library.
		For example, the following path would reference a Dataset named CxLinuxDev located in the Data folder which, in turn, is located in 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]`
		where *field_name* is the Dataset variable and *expression* is 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 process) including its process dataset. The original process object determines the initial values for the dataset. Changes to a dataset in any instance of the process do not affect the original copy.

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 starts another Process using either the Run Process or the Run Detached Process Operator that the Workflow Module supports. The Process Dataset initialization code option of the Start Process operator specifies these assignments.
		
		For example, when Process A calling Process B needs to initialize fields in the Process B Dataset, Process A specifies Process B as part of the Process Dataset initialization code. In this context, Caller refers to the Dataset of the parent Process A, and Process refers to the Dataset of the child Process B.
		
		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 code 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.
		Consider the following 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).

Dataset	Dataset Context	Description

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 230).
		■ For more information about specifying Operator variables in Operator properties settings, see Dataset Variables in Parameters (see page 194).

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. For descriptions of the fields in the system Dataset, see Specify System Dataset Variables (see page 231). **Note:** For more information about specifying system variables and for a list of system variables, see Specify System Dataset Variables (see page 231).

Dataset	Dataset Context	Description
Root	Parent Dataset	Enables access to the Process Dataset of a parent instance to an Inline child Process. You can access the Process dataset of the root instance using the keyword *Root*. Examples: Root Process. ■ Process A starts an Inline process Process B. Process B starts another Inline process Process C. Process C has access to the dataset of Process A using the keyword Root. ■ Process A starts Process B in a non-Inline mode and Process B starts Process C in Inline mode. Process C has access to the dataset of Process B only and not Process A using the keyword Root.

Access Dataset Fields in Expressions

Dataset field values are specified in an expression using either (or both) the bracket ([expression]) or dot (.field) operators to specify a Dataset or a field in a Dataset.

Syntax for Specifying the Value of a Field

A value in a single non-indexed field is accessed in an expression using the dot syntax:

```
dataset_reference.field_name
```

The dataset_reference parameter corresponds to one of the keywords described previously. The field_name parameter is 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 the bracket syntax:

```
dataset_reference[field_name_expression]
```

The parameter field_name_expression is an expression that returns the name of a field in the referenced dataset. The following example returns the value for field Y in a Process Dataset:

```
Process["Y"]
```

In general, always use bracketed notation to use an expression to specify a Dataset or field name in a Dataset.

Specify the Value of an Element in an Indexed Field

A value in an indexed field (array) is accessed using the following syntax:

```
dataset_reference.indexed_field_name[index]
```

The indexed_field_name parameter specifies the field name in the Dataset. Index is an integer that addresses an indexed element in the array.

If you use a bracket notation to specify the indexed_field_name string, then 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 rather than true arrays. To access an element in an array of arrays, simply use the [] operator twice. For example, for the two-dimensional indexed integer-field called integers, every list element integers[x] is itself an indexed list of integers. To access a particular integer within 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

For accessing the length of an indexed field, CA Process Automation supports the JavaScript length property for arrays and its own size property. The length property is read-only while the size property allows you to change the number of elements in an indexed field. The JavaScript length property returns the number of elements in an indexed field using either dot or bracket notation:

```
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, both the length and size properties always return one more than the index for the last element in a field. So when either 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 any values stored in the deleted elements. The following code uses the size property to increment the length of an array X by one element, and 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 called matrix[a][b]:

```
size1 = matrix.length
```

returns the size of the first dimension of matrix, with a containing elements 0...size1 - 1.

```
size2 = matrix[2].length
```

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.

The following example illustrates how to address 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 called matrix. The code assigns the value for each element to a one-dimensional indexed field in the Process Dataset variable called 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 JavaScript array methods as listed in the following table.

Method	Description
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, then the pop function for javascript arrays should not be used in the evaluation of Operator parameters.
push()	Pushes an item onto the end of an array.
reverse()	Reverses the order of elements in an array.
shift()	Shifts an element off 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 properties, refer to a JavaScript reference guide.

Specify Named Dataset Variables

Fields in Dataset objects (called Named Datasets) are identified in expressions using either dot (.string) or bracketed ([expression]) notation:

```
Datasets[path_expression][variable_name_expression]
```

or

```
Datasets[path_expression].variable_name
```

The arguments represent the following:

- path_expression is 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 ending with the object name.
- variable_name_expression argument represents an expression that returns the name of a field in the Dataset object.
- variable_name is the actual name of a field in the Dataset object.

Either of the following would work for referencing field Y in the Named Dataset Coordinates located in the /MathValues folder:

```
Datasets["/MathValues/Coordinates"].Y
```

or

```
Datasets["/MathValues/Coordinates"]["Y"]
```

More information:

Relative Paths for Datasets (see page 195)

Specify Process Dataset Variables

The Process Dataset contains variables defined by the developer or defined automatically by CA Process Automation when an instance of a Process is started. The Process keyword is used to access variables in the Process Dataset. You can specify a process variable in an expression using either the dot (.) or bracketed ([expression]) notation:

```
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”]
```

or

```
Process.StartDate
```

The following table lists process system variables defined automatically by CA Process Automation when it runs a process.

Process Variable	Description
CallerUser	The user ID that started this instance of a process. When a parent process uses a Start Process operator to start a child process, the value of *CallerUser* is passed forward from the parent process. When a Start Process operator in a scheduled task starts another process, this variable is blank.
DisplayName	The name of the process object as seen in the library.
effectiveUser	The current owner of the process object.
EndDate	Returns the date when this instance of the process ended in the format: MM/DD/YYYY
EndTime	Returns the time when this instance of the process ended in the following format: HH:MM:SS
InstanceName	The name of the original process object ending in a unique runtime object identifier used to identify 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.

Process Variable	Description
ObjectID	Object identifier for internal use only.
ParentProcessROID	The unique runtime object identifier for the parent process, if applicable, that started this process.
rootUUID	For internal use only.
RuntimeROID	A unique object identifier appended to the process DisplayName following an underscore to identify each instance of a process. 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 following format: HH:MM:SS
TouchpointName	The name of the orchestrator managing the running process.

Specify Operator Dataset Variables

Operator Datasets contain variables defined by the developer at the time of design or by CA Process Automation at runtime for a specific Operator. Design-time variables are available immediately after a Process is started. Runtime variables are added when an Operator is executed.

Operator Datasets are labeled by the Operator name in the local Dataset at runtime, so you can specify an Operator variable in an expression using the dot (.) or bracketed ([expression]) notation:

```
Operator.field_name
```

or

```
Process["Operator_name"].field_name
```

or

```
Process["Operator_name"]["field_name"]
```

The preexecution and postexecution code for an Operator can access the name of the current Operator using the OpName keyword. You can use this keyword to access or create an Operator Dataset. Use the following syntax to specify an Operator Dataset variable in the preexecution or postexecution code:

```
Process[OpName].field_name
```

The following list shows common Operator Dataset System variables defined automatically by CA Process Automation. Additional variables may be defined for specific Operators.

Operator Dataset Variable	Runtime Scope	Description
AgentName	during, after	Name of the machine associated with the Touchpoint that executes the Operator.
AgentID	during, after	For internal use only.
EndDate	after	The date when the execution of the operator ended in the format: MM/DD/YYYY
EndTime	after	The time when the operator stopped running in the format: HH:MM:SS
Reason	after	String describing the result.
ResponseCode	after	String describing the result.

Operator Dataset Variable	Runtime Scope	Description
StartDate	during, after	The date when execution of the Operator started in the format: MM/DD/YYYY
StartTime	during, after	The time when execution of the Operator started 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	Name of the Touchpoint that executes the Operator.
UUID	during, after	For internal use only.

Specify System Dataset Variables

The System Dataset for a Process contains a number of variables that return system information. The System keyword is used to access the System Dataset. You can specify a system variable in an expression using either the dot (.) or bracketed ([expression]) notation:

```
System.variable_name
```

or

```
System[expression]
```

The following format specifies a system variable name with a literal string using the bracket notation:

System["variable_name"]

The following table lists the fields in the System Dataset.

System Variable	Description
DATE	The current date in the format MM/DD/YY
DAY	The day of the month
FIRSTDAYMONTH	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

System Variable	Description
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 next month in the format MM/DD/YY
MONTH	The current month represented as a 0-based number (for example, returns 0 for January).
TIME	The current time of day in minutes (for example, returns 600 for 10 AM)
TIMES	The current time of day in the format HHMM
TOMORROW	The date of the day following 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 preceding 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.

Statement	Syntax	Description
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."
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.

Statement	Syntax	Description
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:

Include Common Resources in CA Process Automation Scripts (see page 241)
The if Statement (see page 236)
The for/in Loop Statement (see page 239)
The switch Statement (see page 237)
Variable Declaration (see page 234)
The while loop Statement (see page 238)

Variable Declaration

The JavaScript var statement creates a variable in the Process Dataset. 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 the variables:

```
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 at the same time:

```
var i = 0, j = 0, k = 0
```

Variable Assignment

Use the = assignment operator to assign values to Dataset variables. The variable assignment uses the following syntax:

```
[dataset_reference.]variable_name = expression;
```

The expression is any combination of functions, variables, values, and operators that returns a string or integer value. Here are a few examples:

```
Process.S = "ABCDEF" + '_' + "123"
Datasets[“ThisDataset”].x = 18 * I
x = 18 * I
```

If the Dataset reference is omitted, the Process Dataset is referenced automatically. If the variable does not exist in the Process Dataset, then a temporary variable is created. If you want 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 variable in an object. The for/in loop has the following syntax:

```
for (variable in object)
        statement
```

The variable should be either the name of a variable, a var statement declaring a variable, an element of an array, or a property of an object. In other words, it should be suitable as the left side of an assignment expression. The object should be the name of an object or an expression that evaluates to an object.

For example, you can loop through elements of an indexed field in a Dataset by defining an index variable and specifying 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

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.

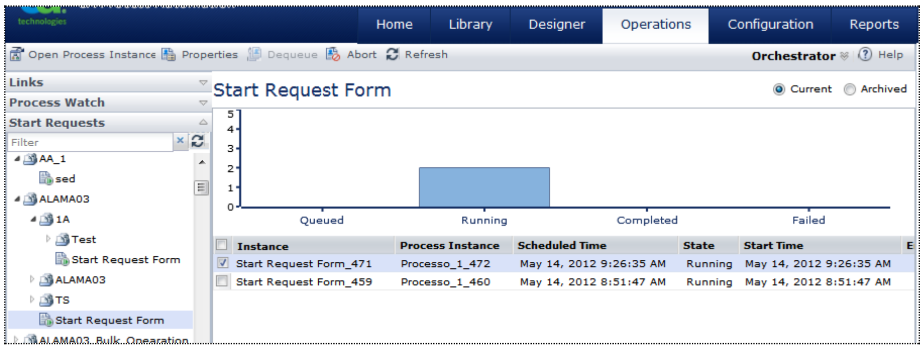
Monitor Start Request Form Instances and Process Instances

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. Exand 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. Exand 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.



	Instance	Process Instance	Scheduled Time	State	Start Time	E
✓	Start Request Form_471	Processo_1_472	May 14, 2012 9:26:35 AM	Running	May 14, 2012 9:26:35 AM	
	Start Request Form_459	Processo_1_460	May 14, 2012 8:51:47 AM	Running	May 14, 2012 8:51:47 AM	

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.

Create and Edit a Start Request Form Object

Use the following procedure to create a form. You can also double-click an existing form, check it out, and edit it.

Follow these steps:

1. Click the Library tab.
2. Select a folder.
3. In the toolbar, click New and then select one of the following:
    a. Start Request Form
    b. Interaction Request Form
    c. Custom Operator
4. Double-click the new object.

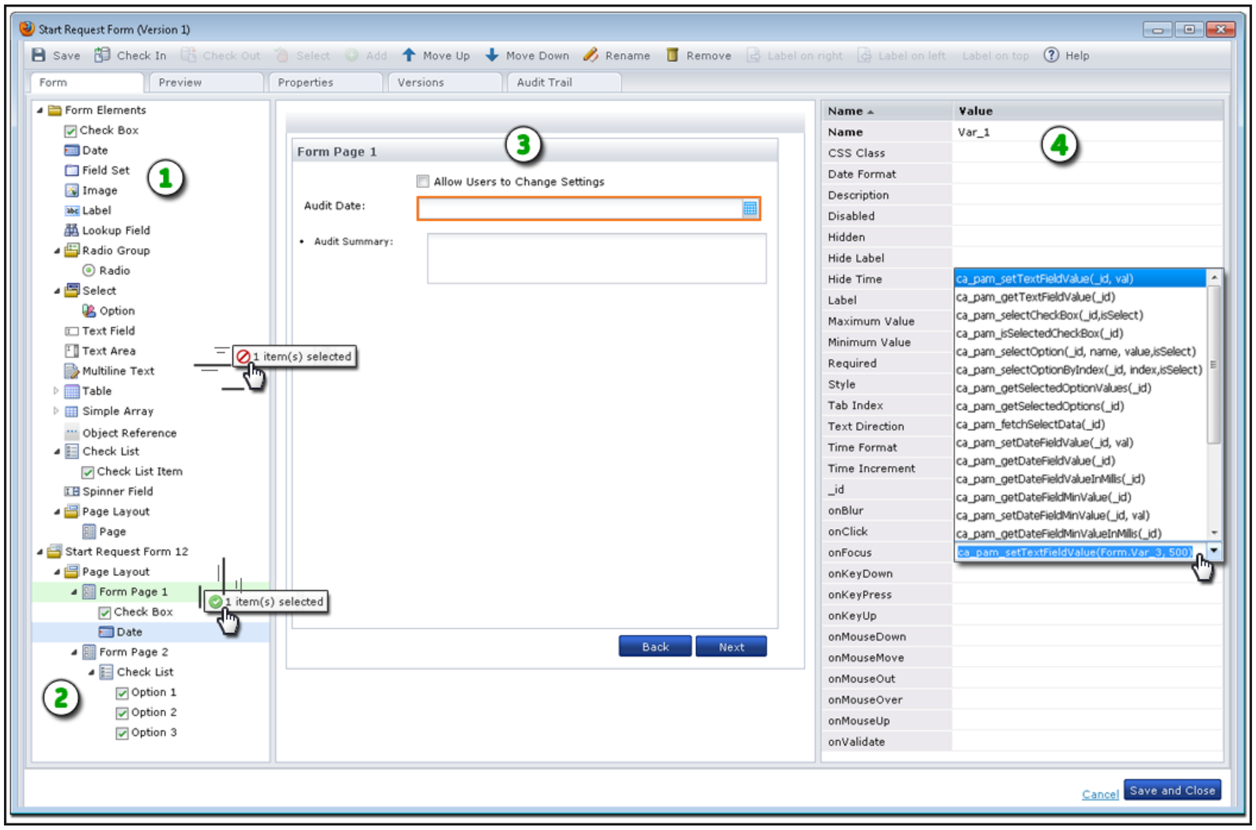
    The object opens in a dialog that shows the Form tab.
5. In the Form Elements pane at left:
    a. Expand Form Elements to view the available list of controls that you can add to the pages of the form.
    b. Below Form Elements, expand the name of your form or custom operator to view its pages and existing form structure.
    c. Drag a form element down the list and drop it on an appropriate parent. For example:
        - Drag a text field onto a page.
        - Drag a check list item onto a check list.
        - Drag a new page onto the page layout for your form to add a new page to your form.

        Note: You cannot drag and drop directly to the form page layout in the center of the page. You must drag elements to the form structure at the bottom of the left pane.
    d. Click one of the form elements that you added to your structure.

        A border appears around the form element in the center of the page. The properties for the form element appear in the right pane.
6. In the Properties pane at right:
    a. Select a property or event.
    b. Specify a value. Most properties accept input strings or pop-up menus for true or false. For events, select from a list of predefined functions.
7. Click Save.

The Form Designer

When you open a form from the Library Browser, the Form Designer dialog appears. It features a standard toolbar and series of tabs. Use the Form tab to design a form and use the Preview tab to check its appearance.



① Form Elements

The top part of this pane displays all the possible types of controls that are available.

② Form Structure

The bottom part of this pane shows the structure of your form. Drag and drop form elements to the pages of your form here.

③ Form Pages

The layout for the pages of your form appears here. Click a control to edit its properties. For forms with more than one page, click the Back and Next buttons to view other pages. Later, when your users need to view the form pages, they also click the Next and Back buttons.

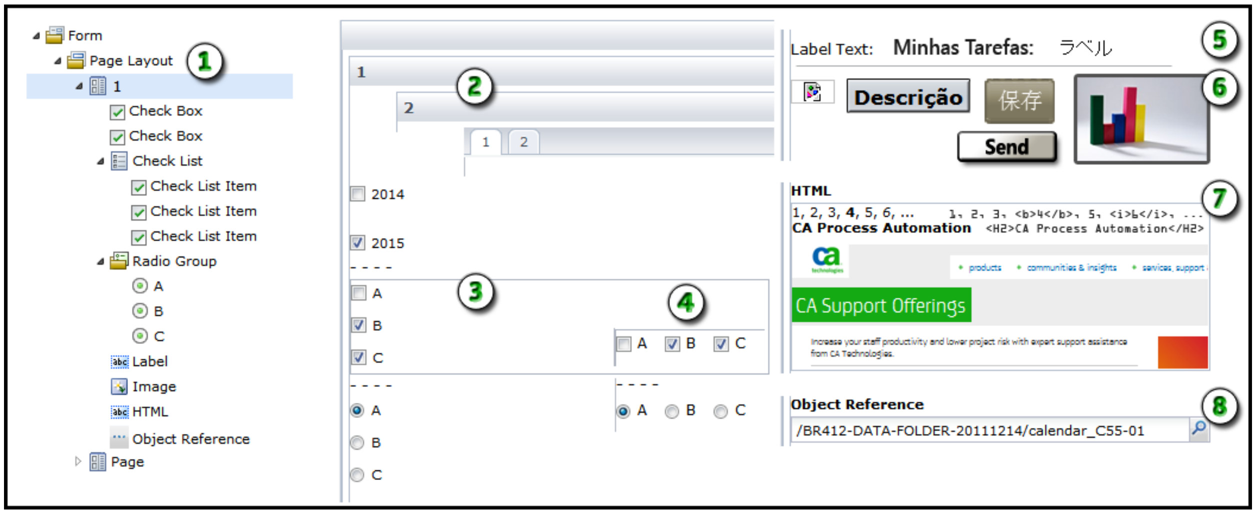
④ Property Pane

Use this pane to view or edit the variables in the form elements. For example, set the Required property to true, change the Label used to identify a field, or specify a function for an event. In this example, you want the onFocus event for a date field to set the value of a text field named Form.Var_3 to 500. Specify the following arguments in the function:

```
ca_pam_setTextFieldValue(Form.Var_3, 500)
```

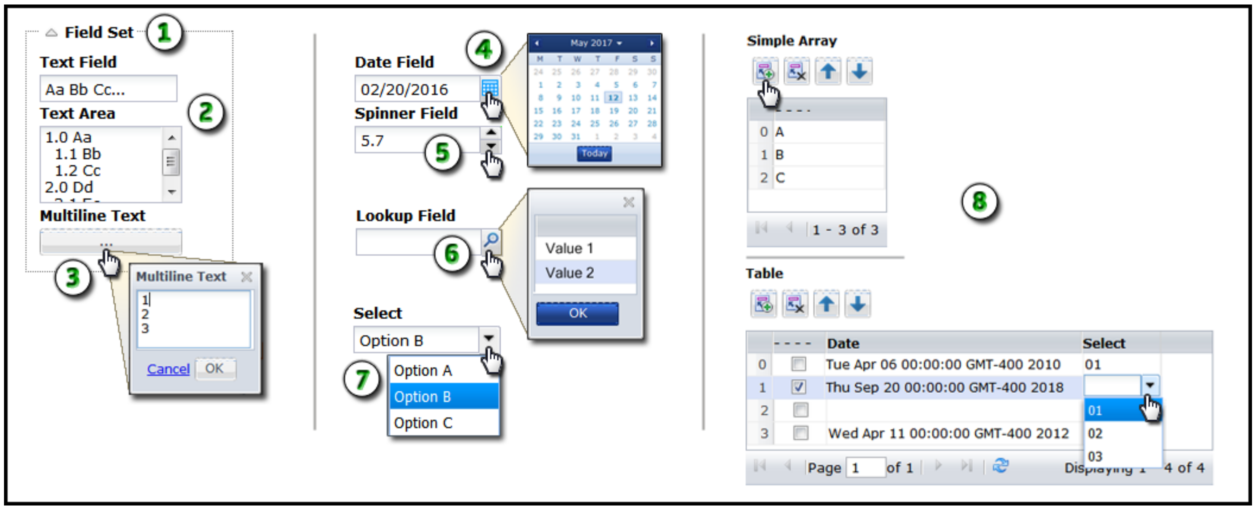
Form Elements

This topic presents basic examples of each type of form element.



Item:	Description:
①	**Form Structure**: While you design the form, the arrangement of pages and form elements appears at the bottom of the Form tab.
②	**Page Layout**: You can set the layout to display pages as cards or tabs. Cards appear in sequence when the user chooses Back or Next. Tabbed pages allow the user to select any tab to view the page.
③	**Check Boxes and Radio Buttons**: Use a group of check list items when a user can select multiple related items. Use a radio group of items when a user must select only one related item. Use individual check boxes to control settings for unrelated items.
④	**Orientation**: Check lists and radio groups can appear in vertical or horizontal orientations.
⑤	**Labels**: Use labels to identify specific fields or regions of the form.

Item:	Description:
⑥	**Images**: Use an image element to specify a graphic. You can use images to display logos, icons, status indicators, or as buttons that users can click. When the form cannot locate an image, the broken link icon appears.
⑦	**HTML**: Use this form element to specify HTML code to render for the form user.
⑧	**Object Reference**: Use this form element to provide form users with 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 form user can specify a touchpoint on an orchestrator and then run a process on the selected touchpoint. As a form designer, 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 avoid clutter by focusing on specific parts of a form.
②	**Text Field and Text Area**: Use text fields for basic data entry including names, addresses, email accounts, phone numbers, and other details. Use Text Area form elements when you want to allow users to enter multiple lines of text. They can view a fixed amount of text on the form (set the height property) and scroll the remainder of the field.
③	**Multi-line Text**: Use this form element for large amounts of text that appear in their own resizeable scrolling window. On the form, this text element only occupies a single line of space. On the form, it appears as a button with ellipsis (...) to indicate you can click it to browse the full window.
④	**Date Field**: Use this form element to store a date. Users can enter a date or select one from the integrated calendar control.
⑤	**Spinner**: Use this type of field to allow the user to nudge or adjust a value up or down in predefined increments.

Item:	Description:
6	**Lookup**: Use a lookup to present a one-column table of values in a popup window. You can use a dataset or external datasource to provide the values. A user can click a value and then click OK to populate the lookup field and store the value.
7	**Select**: Use this form element to present a drop-down list of options to the user.
8	**Simple Array and Table**: Use a simple array to store a single type of data in a table. Use a table when you need to store multiple columns of data. These types of form elements include options for adding, deleting, and moving rows.

You can insert the following elements in a table:

- check box
- date
- lookup
- select field
- text field
- object reference
- spinner

Form Element Properties

Allow Adding of Rows

A Boolean (true or false) value for tables and simple arrays. When true, a tool button allows form users to create rows in the table. When false, users cannot add new 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 tool button allows form users to remove entries 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 tool buttons allow form users to 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 this type. When false or empty, this type of automation object does not appear. The object properties of object reference fields work together like a filter to include only certain types of automation objects when the user views the available choices.

CheckBox Label

The text string or name applied to a single check box or check box item in a group.

Created By

The name of the user or user account responsible for the creation of 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 this type. When false or empty, this type of automation object does not appear. The object properties of object reference fields work together like a filter to include only certain types of automation objects when the user views the available choices.

Custom Operator

A Boolean (true or false) value for object reference fields. When true, the Object Browser lookup dialog includes automation objects of this type. When false or empty, this type of automation object does not appear. The object properties of object reference fields work together like a filter to include only certain types of automation objects when the user views the available choices.

Dataset

A Boolean (true or false) value for object reference fields. When true, the Object Browser lookup dialog includes automation objects of this type. When false or empty, this type of automation object does not appear. The object properties of object reference fields work together like a filter to include only certain types of automation objects when the user views the available choices.

Dataset Expression

A reference to a specific array variable in a dataset object in the library. For example:

```
Datasets["/MyFolder/MyForm/MyDatasetObject"].MyArray
```

Date Format

The preferred format for dates. For example, enter 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 a to display the date and time value 05/01/2014 10:17:43 AM.

Description

Additional information about the form element beyond its name or short label. A tooltip for a field displays the label; however, if you provide a description, the tooltip displays that instead to assist the user in completing the form.

Disabled

An optional mode or setting for a form element characterized by the following behavior:

- The form element cannot receive focus.
- The form element appears grayed-out or unavailable.
- You cannot change the value of a disabled form element; however, you may be able to copy its value.

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 that you want the form element to occupy. Specify a value for this property in pixels from the top of the form element extending down.

Hidden

A Boolean (true or false) value for a field. When true, the field is not visible to users when it loads. When false or empty, the field is visible.

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 appears.

_id

A unique read-only identifier for a specific instance of a form element. The _id joins the name of the form, the name of any parent object such as a table, and the name of the form element with a dot separator. For example, Form1 has two radio button groups with the following _id properties:

```
Form1.rgName1
Form1.rgName2
```

An actual radio button would have the following _id:

```
Form1.rgName2.RadioOptionA
```

A form element in a table named Table_2 might have the following _id:

```
Form_1.Table_2.Var_3
```

The _id for a form is equal to its Name property.

Note: The _id is used in any 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 in value would result in 6.57 and one click down would yield 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 this type. When false or empty, this type of automation object does not appear. The object properties of object reference fields work together like a filter to include only certain types of automation objects when the user views the available choices.

Label

The short descriptive name for a form element that is displayed to the user.

Label Align

A read-only property that indicates how field labels appear relative to the form elements that they describe. When you click a form in the form designer, you can choose buttons in the toolbar to align labels to the left, right, or on top of their associated form elements.

Label Width

The size of a line of label text in pixels. Long labels wrap to the next line.

Layout

A design-time only property that determines how multiple pages on a form appear on the Form and Preview tabs. Select card to view a single page at a time with Back and Next buttons to navigate between pages in sequence. This setting is the default. It is how the pages of every form appear to form users. Select tab to view the names of available pages in their own tab. As a convenience, designers can click a tab to navigate to the selected page in any order.

Maximum Length

The greatest number of characters a user can enter in the field. For example, if your company requires an 8-digit number in a given field, you can constrain input by setting the Maximum Length and Minimum Length properties to 8.

Maximum Rows

The highest number of entries allowed in a table.

Minimum Length

The least amount of characters a user can enter in the field. For example, if your company requires an 8-digit number in a given field, you can constrain input by setting the Maximum Length and Minimum Length properties to 8.

Minimum Rows

The lowest number of entries allowed in a table.

Modified By

The name of the user or user account responsible for the modification of the form.

Modified On

The date when this version of the form object was most recently changed.

Name

A unique string that identifies a form element. The system assigns an initial name such as Var_3; however, you can modify this value. Changing the name also changes the _id.

Form elements can have two separate names. A Name property and an internal name used to identify the form element in the Form Designer.

- Change the value of the Name property to set the value of the _id variable used to identify the form element.

- Click the Rename button to change the internal name for an element that appears in the hierarchical form structure layout at design time. This internal name is the default value given to the Label of a form element. This label does appear to form users at runtime.

Number Format

Indicates the format for numeric input in the field. For example, enter $#.## to display $3.14.

Orientation

Specifies whether radio groups and check lists appear in a horizontal or vertical arrangement on the form. Vertical orientation is preferred and is the default selection.

Package

A Boolean (true or false) value for object reference fields. When true, the Object Browser lookup dialog includes automation objects of this type. When false or empty, this type of automation object does not appear. The object properties of object reference fields work together like a filter to include only certain types of automation objects when the user views the available choices.

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 appears using bullet characters to hide input from other users. When false or empty, data appears exactly as keyed.

Pattern

An input constraint or validation requirement you can place on the values a user enters for a text field or text area. For example, enter [a-z] to require only lowercase alphabetic characters in a field.

Pattern Message

The on-screen alert or hint that you want to appear when a user's entry does not meet the pattern that you specified for the text field or text area.

Process

A Boolean (true or false) value for object reference fields. When true, the Object Browser lookup dialog includes automation objects of this type. When false or empty, this type of automation object does not appear. The object properties of object reference fields work together like a filter to include only certain types of automation objects when the user views the available choices.

Process Watch

A Boolean (true or false) value for object reference fields. When true, the Object Browser lookup dialog includes automation objects of this type. When false or empty, this type of automation object does not appear. The object properties of object reference fields work together like a filter to include only certain types of automation objects when the user views the available choices.

Render as HTML

A Boolean (true or false) value for multi-line text fields. When true, the form interprets the content of the multi-line text field as HTML code and displays it much like a web browser does. For example, text that you tag with <H2> would appear as a second-level heading and text that you tag with <b> would appear bold.

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 this type. When false or empty, this type of automation object does not appear. The object properties of object reference fields work together like a filter to include only certain types of automation objects when the user views the available choices.

Schedule

A Boolean (true or false) value for object reference fields. When true, the Object Browser lookup dialog includes automation objects of this type. When false or empty, this type of automation object does not appear. The object properties of object reference fields work together like a filter to include only certain types of automation objects when the user views the available choices.

Start Request Form

A Boolean (true or false) value for object reference fields. When true, the Object Browser lookup dialog includes automation objects of this type. When false or empty, this type of automation object does not appear. The object properties of object reference fields work together like a filter to include only certain types of automation objects when the user views the available choices.

Style

One or more statements that determine how text attributes in a field appear to users. Format style attribute assignments in mixed case. Examples include:

```
color:blue;
textAlign:center;
textDecoration:underline;
textTransform:uppercase
textIndent:30px;
fontStyle:italic;
fontFamily:"Courier";
fontSize:14px;
```

Tab Index

A sequential number that you can assign to control the tab order of the form at both design time and runtime. Because the application does not enforce uniqueness, you can use multiples of 5 or 10 as you set the tab order. When a new field is inserted in the layout between, for example, 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 is top to bottom in the layout.

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. Choose ltr or leave empty (the default) to display text from left to right. This is the default text direction for ISO-8859 Latin I codesets. Choose 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 any automation object types that are set to true and allows the user to browse touchpoints by domain, environment, and orchestrator. When false or empty, objects from only the default touchpoint appear; 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 any automation object types that are set to true and allows the user to browse touchpoint groups by domain, environment, and orchestrator. When false or empty, objects from only the default touchpoint group appear; the user cannot select another touchpoint group.

URL

For Image form elements, the URL path to an image file. For example:

```
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

A form element's stored data. 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 this value property to store a string or number for each option. For example, the user might see the options listed as Easy, Moderate, and Difficult. The values for each option might be set to 0.8, 1.0, and 1.2; and these values would be used in additional calculations.
- An HTML form element displays a value property. Use this property to specify the actual text including any code that you want to store in the field. For example, enter the following 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 that you want the form element to occupy. Specify a value for this property in pixels from the left edge of the form element extending to the right.

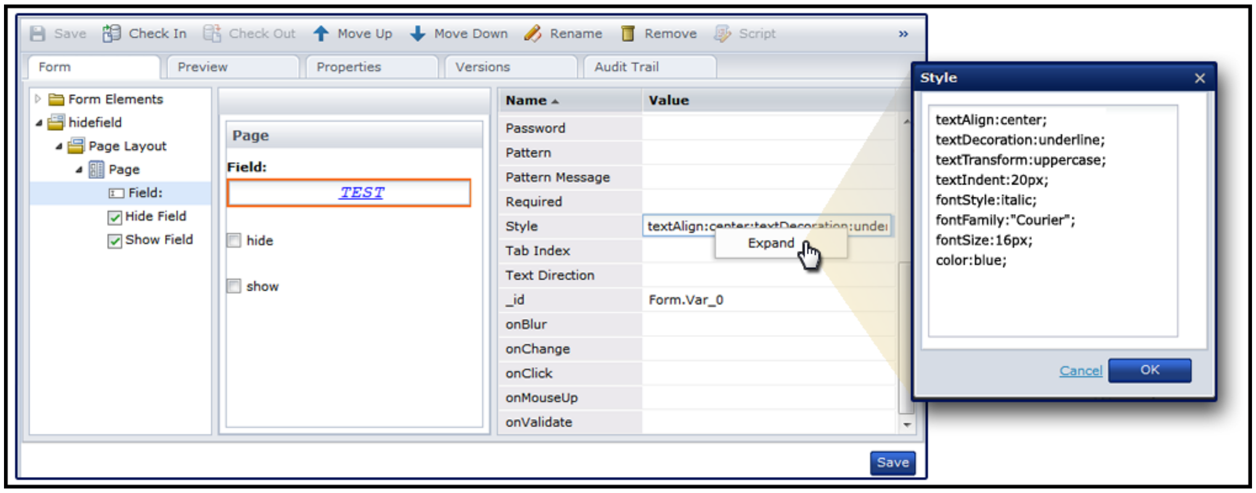
View Form Properties in a Separate Window

When form properties do not fit in the single line of the Form Designer Property pane, you can view form properties 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 inside the property or event value and choose Expand.

      The expanded properties appears in a separate window.



Name	Value
Style	textAlign:center;textDecoration:unde
_id	Form.Var_0

```
textAlign:center;
textDecoration:underline;
textTransform:uppercase;
textIndent:20px;
fontStyle:italic;
fontFamily:"Courier";
fontSize:16px;
color:blue;
```

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

The event that occurs when a form element loses focus. For example, a form contains a field called *User Name*. A user first 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

The event that occurs when a form element loses focus and determines that its new value is not the same as its old value. For example, a form contains a field called *Quantity* with a value of 10. After changing the value to 15, the user remains in the field. The onChange event does not occur until 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

The event that occurs when a user clicks a form element. A valid click by the user must include both the onMouseDown and onMouseUp events on the same object. This requirement helps prevent calling functions or other code accidentally since the user's mouse must remain on the clickable object.

onFocus

The event that occurs when a form element receives focus. A user establishes focus in a field by tabbing to it or clicking in it. You can also write scripts or code to establish focus in a field.

onKeyDown

The event that occurs when a user first presses down on a key. For example, a user clicks or tabs into a Name field. A script associated with the onKeyDown event for the Name field alerts users when they attempt to type number keys.

onKeyPress

The event that occurs after a user presses a key down and continues to hold that key down. For example, a user clicks or tabs into a Select field called Name. A script associated with the onKeyPress event for the Name field cycles through names that match the pressed alphabetic key.

onKeyUp

The event that occurs when a user releases a key. For example, a user clicks or tabs into a Spinner field called Temperature. A script associated with the onKeyUp event for the field adjusts its value up or down incrementally each time the user presses and releases a specific key.

onLoad

The event that occurs when the form first appears for the user to complete.

onLookup

The event that occurs when a user clicks the Browse button in a Lookup field to view the available values that are calculated (or "looked up") based on other field values.

onMouseDown

The event that occurs when a user presses the left mouse button down on a form element.

onMouseMove

The event that occurs when a user moves the mouse cursor inside the boundary of a form element.

onMouseOut

The event that occurs when a user moves the mouse cursor outside the boundary of a form element.

onMouseOver

The event that occurs when a user positions the mouse over a form element and stops moving the mouse.

onMouseUp

The event that occurs when a user releases the left mouse button on a form element after it was down.

onMouseWheel

The event that occurs when a user equipped with an optional mouse wheel rolls the wheel forward or back to scroll a form element.

onSubmit

The event that 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.
- A change in the value of a field.
- The user reaches the end of the form in the tab order.

onValidate

The event that occurs when the value of a field is verified against business rules before it is stored in the data buffer or written to the database. For example, the user enters 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 might alert the user if the serial number does not meet validation rules so they can adjust their entry.

Form Element Functions

A user completing a form causes, often unknowingly, any number of user interface *events* to occur. Events, such as changing a field value (onChange) or positioning the mouse pointer over a table (onMouseOver) can then initiate JavaScript functions. As a form designer, use functions to achieve the following sample goals:

- Change the appearance of fields on a form.
- Get data from a field or table.
- Set the data in a field or table.
- Help users navigate through potentially complex choices, based on logic. For example, update a list of cities based on the user's selection of *Country*.
- Help users minimize data-entry errors.

The remainder of this reference topic lists the predefined JavaScript functions that you can use in fields on start request and interaction request forms.

User Interface

ca_pam_disableField(_id, isDisable)

Use this function to enable or disable a field. When a field is enabled, users can enter the field and can provide input. Disabled fields are still visible but appear unavailable and do not accept user focus or input.

Parameters:

- **_id**: The unique identifier for the field that you want to enable or disable.
- **isDisable**: A Boolean (true or false) value that is used to indicate whether the field is disabled (true) or enabled (false).

Example:

```
ca_pam_disableField('Form.ckListField27', true)
```

- This example disables the specified check list field including all check list items inside it.

ca_pam_hideField(_id, isHide)

Use this function to show or hide a field. When a field is visible and enabled users can enter the field and can provide input. Hidden fields are not visible to the user. The user cannot enter a hidden field. The tab order of the form skips hidden fields.

Parameters:

- **_id**: The unique identifier for the field that you want to show or hide.
- **isHide**: A Boolean (true or false) value used to indicate if the field is hidden (true) or shown (false).

Example:

```
ca_pam_hideField('Form.Var_0', true)
```

- This example hides the specified field. The hidden field is effectively removed from the form although it can still hold a value.

Selected Values

ca_pam_getSelectedOptions(_id)

Use this function to retrieve the index value of the currently selected option in a Select field.

Parameters:

- **_id**: The unique identifier for the field with options that you want to retrieve.

Example:

```
ca_pam_getSelectedOptions('Form.VarFillerType')
```

- This example retrieves the index value of the user's current choice of *Filler Type*.
- *Filler Type* is a Select field with options for Rocks [0], Sand [1], and Water [2].
- When the user selects *Sand* from the Filler Type drop-down list, this function returns the index value *1*.

ca_pam_getSelectedOptionValues(_id)

Use this function to retrieve the option value of the currently selected option in a Select field.

Parameters:

- **_id**:The unique identifier for the field with option values that you want to retrieve.

Example:

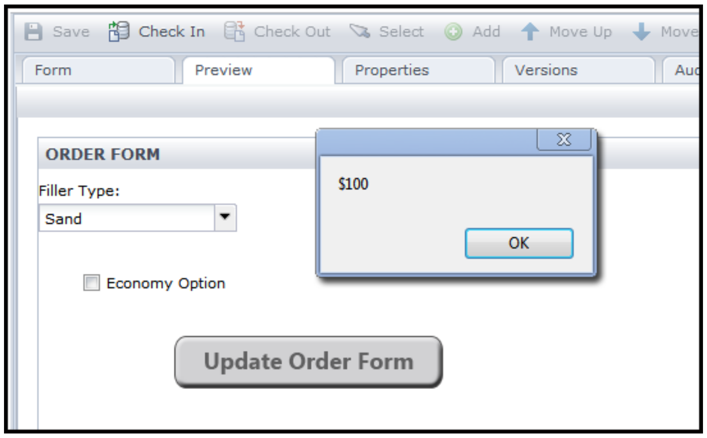
```
alert(ca_pam_getSelectedOptionValues('Form.VarFillerType'))
```

- This example retrieves the option value number or text string for the user's current choice of *Filler Type*.
- The alert command displays the option value result to the user in a browser pop-up dialog or message box.
- *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.
- *Update Order Form* is a button image with this function in its onMouseUp event.
- When the form user clicks the *Update Order Form* button, this function retrieves the value for the option they chose in the Filler Type drop-down list.
- When the form user chooses *Sand* in the *Filler Type* field, and clicks the update button, the pop-up alert box displays the value $100.



Filler Type: Sand

$100

ca_pam_getTextFieldValue(_id)

Use this function to retrieve a numeric value or text string from a Text field.

Parameters:

- **_id**: The unique identifier for a Text field.

Examples:

```
ca_pam_getTextFieldValue('Form.top_vendors')
```

- This example retrieves the value of the top_vendors Text field.

```
ca_pam_setTextFieldValue('Form.dynamic_field',
ca_pam_getTextFieldValue('Form.top_vendors'))
```

- This example retrieves the value of the *top_vendors* field and sets the *dynamic_field* Text field to that retrieved value.

ca_pam_isSelectedCheckBox(_id)

Use this function to determine whether a check box field is selected or checked (true), or clear (false).

Parameters:

- **_id**: The unique identifier for the Check Box field that you want to evaluate.

Example:

```
ca_pam_isSelectedCheckBox('Form.ckbxInsBuy')
```

- This example returns the value *true* when the Purchase Insurance check box is selected. It returns *false* when the check box is clear.
- The check box named ckbxInsBuy has a label the user sees as Purchase Insurance.

ca_pam_isSelectRadio (_id, radio_id)

Use this function to determine if a specific radio button option is selected.

Parameters:

- **_id**: The unique identifier for a Radio Group field.
- **radio_id**: The unique identifier for a specific radio button option in a radio group.

Example:

```
ca_pam_isSelectRadio('Form.Transportation',
'Form.Transportation.Option_01_Air')
```

- This example returns a value of true when the *Transportation* radio group is set to *Air*. The function returns *false* when the user selects *Rail*.
- *Transportation* is a radio group with two radio button options: Option_01_Air and Option_02_Rail.

ca_pam_selectCheckBox(_id, isSelect)

Use this function to select or clear a check box.

Parameters:

- **_id**: The unique identifier for the Check Box field that you want to select (check) or clear (uncheck).
- **isSelect**: A Boolean (true or false) value that is used to indicate whether the field is checked (true) or cleared (false).

Example:

```
ca_pam_selectCheckBox('Form.ckbxInsBuy',true)
```

- This example marks the *ckbxInsBuy* check box field on the form.
- The check box named *ckbxInsBuy* has a label the user sees as *Purchase Insurance*.
- This function might be used if the user selects some other options on the form that require the purchase of insurance.

ca_pam_selectOption(_id, name, value, isSelect)

Use this function to set a Select field to one of its specific options. The function can also be used to prevent the selection of a specific option.

Parameters:

- **_id**: The unique identifier for the Select field that you want to set to a specific value.
- **name**: The name of a specific Option listed under the Select field.
- **value**: The specific numeric or text string already assigned in the Value property on the Form tab of the Form Designer dialog.
- **isSelect**: A Boolean (true or false) value that is used to indicate whether the Select field option is applied (true) or cleared (false).

Example:

```
ca_pam_selectOption('Form.City', 'Regional Hub', 'Sydney', true);
```

- This example sets the City field to the option named *Regional Hub*.
- If more than one option is named *Regional Hub*, 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 direct business to the Regional Hub using a set of standard forms. However, for 2014, the regional hub city *value* is Sydney and for 2015 the city *value* is Seoul.

ca_pam_selectOptionByIndex(_id, index, isSelect)

Use this function to select or clear a specific option in a field identified by the index value for the option.

Parameters:

- **_id**: The unique identifier for the field with options that you want to select or clear.
- **index**: The fixed identifier for each option in a field.
- **isSelect**: A Boolean (true or false) value that is used to indicate whether the Select field option is applied (true) or cleared (false).

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, this function sets the *Filler Type* field to the option at index position 2, *Water*.

```
ca_pam_selectOptionByIndex('Form.VarFillerType', 2,false)
```

- When *Filler Type* is already set to *Water*, this function clears the Filler Type field.

ca_pam_selectRadio(_id, radio_id)

Use this function to select a specific radio button option in a radio group.

Parameters:

- **_id**: The unique identifier for a Radio Group field.
- **radio_id**: The unique identifier for a specific radio button option in a radio group.

Example:

```
ca_pam_selectRadio('Form.Transportation',
'Form.Transportation.Option_01_Air')
```

- This example selects the *Air* radio button option in the *Transportation* radio group.
- *Transportation* is a radio group with two radio button options: Option_01_Air and Option_02_Rail.

ca_pam_setTextFieldValue(_id, val)

Use this function to insert a numeric value or text string into a Text field. This function replaces any existing value in the target field with the specified value.

Parameters:

- **_id**: The unique identifier for a field that you want to set to a specific value.
- **val**: A numeric or text value.

Examples:

```
ca_pam_setTextFieldValue('Form.top_vendors', 'I vote for vendor 3 because: ')
```

- This example sets the top_vendors Text field to the following string: *I vote for vendor 3 because:*
- To continue this example, after users enter their reasons, use ca_pam_getTextFieldValue('Form.top_vendors') to submit their full statements.

```
ca_pam_setTextFieldValue('Form.dynamic_field',
ca_pam_getTextFieldValue('Form.top_vendors'))
```

- This example retrieves the value of the *top_vendors* field and sets the *dynamic_field* to that retrieved value.

Select Fields and Options

Use the following six functions for working with the possible options that appear in a Select field.

ca_pam_addValuesInSelectStore(_id, values)

Use this function to replace the list of options available in a Select field with a simple set of values that you define in code. The options that appear at design time on the form are not used.

Parameters:

- **_id**: The unique identifier for a Select field.
- **values**: An array or reference to an array of the series of values in the two fields necessary to represent the list of options for a Select field. The two fields are Name and Value. In the example below, the two fields are cityOptionNames and cityOptionValues.

Examples:

```
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 ))
```

- This example dynamically updates the available list of options in the City field based on the user's selection of *West* or *North* in a separate Region field.

ca_pam_clearSelectStore(_id)

Use this function to clear any current value in a Select field.

Parameters:

- **_id**: The unique identifier for a Select field that you want to clear.

Examples:

```
ca_pam_clearSelectStore(Form1.City)
```

- This example clears any existing value previously selected in the City field on Form1.

ca_pam_convertToJavaScriptObject(resultFromSQLQuery)

Use this function to convert a limited Process Automation array (C2OValueArrayDTO) into a Javascript object array.

Parameters:

- **resultFromSQLquery**: The resulting array of data from a query.

Examples:

```
var array = ca_pam_convertToJavaScriptObject(result);
```

- This example declares a variable called array and sets it to the converted JavaScript array.

ca_pam_convertToSimpleArray(objectArray,fieldName)

Use this function to create a simple array of text or numeric strings from any more complex JavaScript array of objects.

Parameters:

- **objectArray**: A reference to a complex JavaScript array of JavaScript objects.
- **fieldName**: The single field in the complex array that you want to convert to a simple array.

Examples:

```
ca_pam_convertToSimpleArray(LocationArray,BuildingCode)
```

- This example converts the complex *LocationArray* to a simple array of building codes based on the *BuildingCode* field.

ca_pam_createSelectStore(nameArray,valueArray)

Use this function to specify a list of options, or store, for a Select field. The options you specify in code replace any design-time options set for the form elements.

Parameters:

- **nameArray**: A numeric or text value.
- **valueArray**: An optional text or numeric string that you want to associate with each option. The option names appear to the user to select on the form. The option values are hidden from the user. If you do not provide or specify these values, by default, the form uses the names as values.

Examples:

```
ca_pam_addValuesInSelectStore('Form1.City',
ca_pam_createSelectStore(cityOptionNames ,cityOptionValues ))
```

- This example sets the available list of options for the City field to the specified names and associated values.

ca_pam_createSelectStoreFromSQLResult(resultFromSQLQuery,nameColumnID,valueColumnID)

Use this function to directly create a a list of options, or store, for a Select field from the result of a SQL query. valueColumnId is not mandatory if same as nameColumnID.

Parameters:

- **resultFromSQLQuery**: The resulting array of data that the SQL statement retrieves.
- **nameColumnID**: A field that stores the options used to build the drop-down list for a Select field.
- **valueColumnID**: An optional field that stores the values associated with the options in a Select field.

Examples:

```
ca_pam_addValuesInSelectStore('Form1.City',ca_pam_createSelectStoreFromSQLRes
ult(result,'txtRegion'));
```

- This example sets the options for the City field based on the results of a query to an external datasource. The user, without necessarily knowing it, initiates the query by selecting an option from the Region field.

Date Fields

Note: Except for functions that return a value in milliseconds, all date functions return the date as a string value in the format that you specify in the Date Format property.

ca_pam_getDateFieldMaxValue(_id)

Use this function to retrieve the value of the Date field property called *Maximum Value*.

Parameters:

- **_id**: The unique identifier for a Date field.

Example:

```
ca_pam_getDateFieldMaxValue('Form.DateDeparture')
```

- This example returns the maximum value allowed for the *DateDeparture* field as a date string, such as 05/20/2025.

ca_pam_getDateFieldMaxValueInMillis(_id)

Use this function to retrieve the value of the Date field property called *Maximum Value* expressed as an integer. This integer represents the number of milliseconds before or after January 1, 1970 00:00:00 UTC (known as the *Unix Epoch*).

Parameters:

- **_id**: The unique identifier for a Date field.

Example:

```
ca_pam_getDateFieldMaxValueInMillis('Form.DateDeparture')
```

- This example returns the maximum value 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. (Shown with commas as a thousands separator, this is 1,748,188,800,000 or 1.7 trillion milliseconds.)
- For dates before 1970, this function returns a negative result.

ca_pam_getDateFieldMinValue(_id)

Use this function to retrieve the value of the Date field property called *Minimum Value*.

Parameters:

- **_id**: The unique identifier for a Date field.

Example:

```
ca_pam_getDateFieldMinValue('Form.DateDeparture')
```

- This example returns the minimum value allowed for the *DateDeparture* field as a date string, such as 05/05/2025.

ca_pam_getDateFieldMinValueInMillis(_id)

Use this function to retrieve the value of the Date field property called *Minimum Value* expressed as an integer. This integer represents the number of milliseconds before or after January 1, 1970 00:00:00 UTC (known as the *Unix Epoch*).

Parameters:

- **_id**: The unique identifier for a Date field.

Example:

```
ca_pam_getDateFieldMinValueInMillis('Form.DateofBirth')
```

- This example returns the minimum value 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 about 126291600000. (Shown with commas as a thousands separator, this is 126,291,600,000 or 126 billion milliseconds.)
- For dates before 1970, this function returns a negative result.

ca_pam_getDateFieldValue(_id)

Use this function to retrieve the value of a Date field as a date string, such as 05/05/2025.

Parameters:

- **_id**: The unique identifier for a Date field.

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 *LastDay*.

ca_pam_getDateFieldValueInMillis(_id)

Use this function to retrieve the value of a Date field as an integer, such as 61238000. This integer represents the number of milliseconds before or after January 1, 1970 00:00:00 UTC (known as the *Unix Epoch*).

Parameters:

- **_id**: The unique identifier for a Date field.

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 about -132307200000. (Shown with commas as a thousands separator, this is -132,307,200,000 or -132 billion milliseconds.)
- For dates before 1970, this function returns a negative result.

ca_pam_setDateFieldMaxValue(_id, val)

Use this function to set the value of the Date field property called *Maximum Value*. Form users cannot enter a date in the Date field beyond the Maximum Value.

Parameters:

- **_id**: The unique identifier for a Date field.
- **val**: The value that you want to use as a maximum, expressed as a date string such as 05/20/2014.

Example:

```
ca_pam_setDateFieldMaxValue('Form.DateDeparture',LastDay);
```

- This example sets the Maximum Value property of the *DateDeparture* field to the date stored in the *LastDay* variable.

ca_pam_setDateFieldMinValue(_id, val)

Use this function to set the value of the Date field property called *Minimum Value*. Form users cannot enter a date in the Date field that occurs before the Minimum Value.

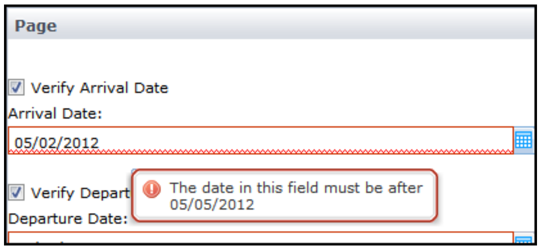
Parameters:

- **_id**: The unique identifier for a Date field.
- **val**: The value that you want to use as a minimum, expressed as a date string such as 05/05/2012.

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.
- Form users receive a message when they enter a date below the minimum value.
- Popup calendars on the form display invalid dates outside the permitted range in gray and do not allow users to click on invalid dates.



05/02/2012

The date in this field must be after 05/05/2012

ca_pam_setDateFieldValue(_id, val)

Use this function to set the value of a Date field to a date string such as 05/15/2014.

Parameters:

- **_id**: A unique identifier for a Date field.
- **val**: A date value that you specify, such as 05/05/2014.

Example:

```
ca_pam_setDateFieldValue('Form.DateArrival','05/05/2014');
```

- This example sets the *DateArrival* field to 05/05/2014.

Tables

ca_pam_clearTableData(_id, startIndex, endIndex)

Use this function to delete one or more rows of data from a table.

Parameters:

- **_id**: A unique identifier for a Table form element.
- **startIndex**: The numeric index value of the first row of table data that you want to delete.
- **endIndex**: The numeric index value of the last row of table data that you want to delete.

Example:

```
ca_pam_clearTableData('Form.employeeList',0,ca_pam_getTableRowCount('Form.emp
loyeeList')-1);
```

- This example deletes all rows in the employeeList table.
- The specified range begins with row 0 and ends with the last row.
- The last row is calculated by getting the current row count for the table and subtracting 1 because the table index begins at row 0.

ca_pam_getTableData(_id, startIndex, endIndex)

Use this function to retrieve one or more values from a Text Field form element in a Table form element.

Parameters:

- **_id**: The unique identifier for a table on a form.

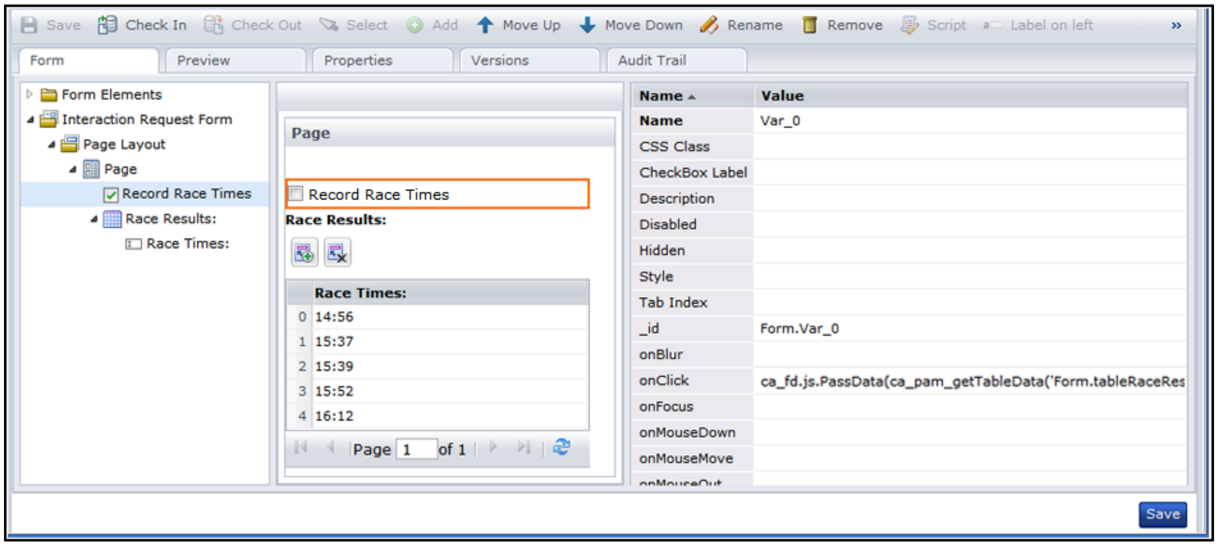
Example:

```
ca_fd.js.PassData(ca_pam_getTableData('Form.tableRaceResults',0,4))
```

- This example passes the finishing time values for the top five winners in a race from a table on a form to a custom database with the *RecordResults()* function.
- The *Record Race Times* check box onClick event initiates function calls to get the table data, pass the data, and record the results.
- The form script is:

```
{
 PassData : function(result) {
   for(i=0;i<result.length;i++)
   {
    var firstRow = result[i];
    var topFive = firstRow['Var_0'];
    RecordResults(topFive );
   }
 },
}
```

- The graphic below demonstrates how the form appears during design.



Form Elements
Interaction Request Form
Page Layout
Page
Record Race Times
Race Results:
Race Times:

Page

Record Race Times
Race Results:

Race Times:	
0	14:56
1	15:37
2	15:39
3	15:52
4	16:12

Page 1 of 1

Name ▲	Value
Name	Var_0
CSS Class	
CheckBox Label	
Description	
Disabled	
Hidden	
Style	
Tab Index	
_id	Form.Var_0
onBlur	
onClick	ca_fd.js.PassData(ca_pam_getTableData('Form.tableRaceRes
onFocus	
onMouseDown	
onMouseMove	

ca_pam_getTableRowCount(_id)

Use this function to retrieve the total number of rows of data from a table.

Parameters:

- **_id**: A unique identifier for a table on a form.

Example:

```
ca_pam_clearTableData('Form.employeeList',0,ca_pam_getTableRowCount('Form.emp
loyeeList')-1);
```

- This example specifies rows to delete beginning with row 0 and ending with a calculated result to determine the last row. Using the function to calculate the last row permits the same code to be used for all tables of all sizes.
- The last row is calculated by getting the current row count for the *employeeList* table and subtracting 1 because the table index begins at row 0. For example, when ca_pam_getTableRowCount returns 100, the first row is row 0 and the last row is row 99.

ca_pam_setTableData(_id, values)

Use this function to set the data in a table.

Parameters:

- **_id**: A unique identifier for a table on a form.
- **values**: The data that you want to display in the rows and columns of the table.

Example:

```
ca_pam_setTableData('Form.employeeList',result);
```

- This example sets the employeeList table to the values retrieved by the result variable.
- This function is part of *Example 1: A Form Script* at the end of this topic.

ca_pam_showDataInTable(result,_id,tableHeader)

Use this function to display resulting rows of data in a Lookup Field as a single column table with the specified header or title.

Parameters:

- **result:** A function call with a result set of the values that you want to show in a table.
- **_id**: The unique identifier for the Lookup Field table where you want to show the result set.
- **tableHeader:** A text string to display to the user above the Lookup Field table on the form.

Example:

```
ca_pam_showDataInTable(result,'Form.office','Office');
```

- This example shows an *Office* table listing the results of a separate query to get the names of offices. In this example, the list of offices to display varies based on the user's selection of Region.
- This function is called by the *onLookup* event for a Lookup Field called *Office*.
- This function is part of *Example 1: A Form Script* at the end of this topic.

Data Sources

ca_pam_getDataUsingDatasetExpression(datasetExpression,callBack)

Use this function to retrieve all the field values for all index entries of a specific ValueMap array variable from a dataset. The callBack variable stores the values.

Parameters:

- **datasetExpression**: 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**: A reference to the function object, including its onSuccess and onFailure methods, that you want to call after retrieving the data.

Example:

```
ca_pam_getDataUsingDatasetExpression(ca_pam_getTextFieldValue('Form.TxtFieldW
ithDatasetPath'),callBack)
```

- This example uses a Text field called *TxtFieldWithDatasetPath* to store the dataset expression.
- The getTextFieldValue function gets the value of the dataset expression.

- The getDataUsingDatasetExpression function retrieves the values from the dataset and calls the callBack function to determine what happens next.
- To continue with this example, you can apply the data to a table or Lookup field, respectively, using the following functions in the callBack.onSuccess declaration:

  ```
  ca_pam_setTableData('Form.employeeList',result);
  ```

  ```
  ca_pam_showDataInTable(result,'Form.Results','Dataset results');
  ```

ca_pam_getDataUsingSQLQuery(driverName,connectionURL,userName,password,query,callBack)

Use this function to retrieve data from an external datasource using SQL and to store that data in a variable you define named callBack.

Parameters:

- **driverName**: The class name for the installed SQL driver that you are using. For example, com.mysql.jdbc.driver.
- **connectionURL**: The URL for the database application that you want to query. For example, jdbc:mysql://myPC-xp.myCompany.com:CA Portal/<path> or https://<server>:CA Portal/.
- **userName**: A user name or login name credentials for a predefined user account with sufficient permissions to run the query.
- **password**: The corresponding password for the specified user name.
- **query**: A specific SQL query statement or a reference to one. The following line represents a sample SQL query:

  ```
  return "select * from employeedatatable where empName like
  '%"+ca_pam_getTextFieldValue('Form.empName')+"%'";
  ```

- **callBack**: A variable that you define to store the data that the query retrieves.

Example:

```
ca_pam_getDataUsingSQLQuery(ca_fd.js.driverName
(),ca_fd.js.connectionURL(),ca_fd.js.userName(),ca_fd.js.password(),ca_fd.js.
queryEmployeeName(),callBack);
```

- This example queries an external data source and stores the resulting data in the *callBack* variable.
- The SQL query function is part of *Example 1: A Form Script* at the end of this topic.

Example 1: A Form Script

```
{ populateDataInTable : function()
 { var callBack = new Object();
    callBack.onSuccess = function(result)
       { ca_pam_setTableData('Form.employeeList',result); }
    callBack.onFailure = function(caught)
       { alert(caught); }

    ca_pam_clearTableData('Form.employeeList',0,ca_pam_getTableRowCount('Form.emp
    loyeeList')-1);
      var useEnteredValue = ca_pam_getTextFieldValue('Form.empName');
      if(!(useEnteredValue == ''))
```

```
      { ca_pam_getDataUsingSQLQuery(ca_fd.js.driverName
    (),ca_fd.js.connectionURL(),ca_fd.js.userName(),ca_fd.js.password(),ca_fd.js.
    queryEmployeeName(),callBack); }
 },
    driverName : function()
     { return 'com.mysql.jdbc.Driver'; },
    connectionURL : function()
     { return 'jdbc:mysql://myPC-xp.myCompany.com:1234/example'; },
    userName: function()
     { return 'root'; },
    password: function()
     { return 'root'; },
    queryEmployeeName:function()
     { return "select * from employeedatatable where empName like
    '%"+ca_pam_getTextFieldValue('Form.empName')+"%'"; },
    queryRegion:function()
     { var array = ca_pam_getSelectedOptionValues('Form.country');
       var country = array[0];
       return "select region from regiontable where country = '"+country+"'";
     },
    populateRegion :function()
     { var callBack = new Object();
       callBack.onSuccess = function(result)
        { ca_pam_showDataInTable(result,'Form.office','Office'); }
       callBack.onFailure = function(caught)
        { alert(caught); }
       ca_pam_getDataUsingSQLQuery(ca_fd.js.driverName
    (),ca_fd.js.connectionURL() ,ca_fd.js.userName() ,ca_fd.js.password()
    ,ca_fd.js.queryRegion(),callBack);
     },
}
```

Create a Simple Form with Basic Functions

In this second example, you are developing a form for a company that operates in three primary regions. The regions are the states of California, Pennsylvania, and North Carolina. In each state, the company has a primary and secondary office location identified by the name of the city. The stakeholders responsible for implementing company policy want the form to meet the following business requirements:

- All business originating in California must be routed by default to the Anaheim office.
- All business originating in Pennsylvania must be routed by default to the Philadelphia office.
- All business originating in North Carolina must be routed by default to the Raleigh office.
- All business originating 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 2: 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 not already checked out, click Check Out.
5. In the Form Elements pane, expand the following entries:
   a. Form Elements
   b. Your form (e.g., Interaction_Request_Form_4)
   c. Page Layout
   d. Page

6. Click Page. In the toolbar, click Rename. Enter the name *Location:* and click OK.
7. In the Form Elements pane, drag a Select field down to the Location page.
8. Repeat the last step.

   Two Select fields appear under your Location page layout.
9. In the Form Elements pane:
   a. Drag a Select Option down to the first of your two Select fields.
   b. Drag two more Select Options under the same Select field.
   c. Drag six Select Options under the second Select field.
10. Click each of the following form elements and then click the Rename toolbutton 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*.

11. 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.

12. In the Form Elements pane, click the name of your form and then, in the toolbar, click Script.

    The Script dialog appears.

13. In the Script dialog, copy and paste the following code:

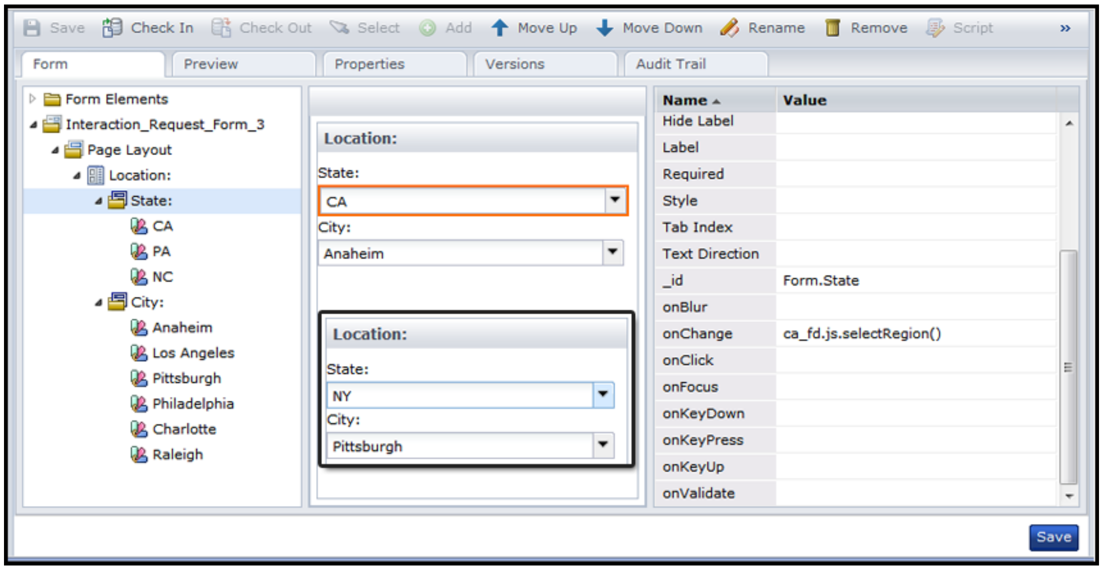
```
{
 selectRegion: function()
  {
    var selectedState = ca_pam_getSelectedOptionValues('Form.State')[0];

    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);
     }
  }
}
```

    Click Save. The Script dialog closes.

14. In the Forms Designer, click Save.

15. Click the Preview tab.

16. Confirm your results are similar to the sample form results and table values shown.



- Form Elements
- Interaction_Request_Form_3
  - Page Layout
    - Location:
      - State:
        - CA
        - PA
        - NC
      - City:
        - Anaheim
        - Los Angeles
        - Pittsburgh
        - Philadelphia
        - Charlotte
        - Raleigh

Location:
State: CA
City: Anaheim

Location:
State: NY
City: Pittsburgh

Name	Value
Hide Label	
Label	
Required	
Style	
Tab Index	
Text Direction	
_id	Form.State
onBlur	
onChange	ca_fd.js.selectRegion()
onClick	
onFocus	
onKeyDown	
onKeyPress	
onKeyUp	
onValidate	

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

Use *Resource* objects to synchronize independent processes that rely on common elements of the infrastructure. Resource objects are models representing elements of your system architecture. Use resources to quantify and control access to particular IT entities. Include multiple resources representing 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.

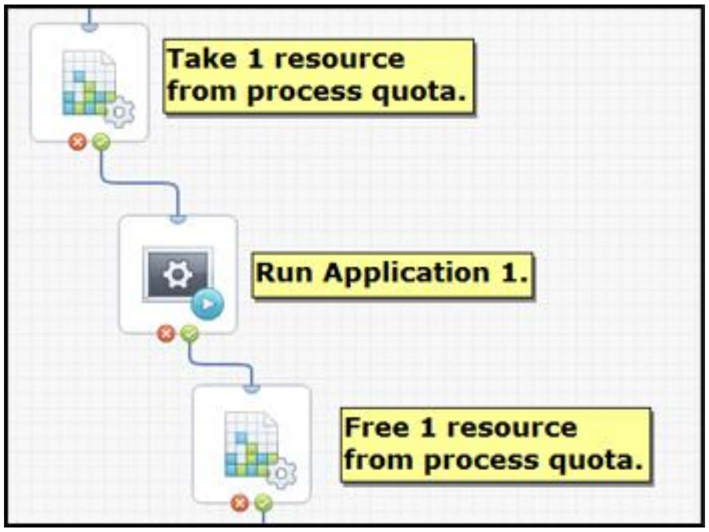
You group resources because they are related to each other in some way. Examples include shared databases, transmission links, simultaneous access to a limited number of software licenses, concurrent processes on a touchpoint, numeric quotas, and other resources. After measuring performance, you can 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

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.


Take 1 resource from process quota.
Run Application 1.
Free 1 resource from process quota.

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.

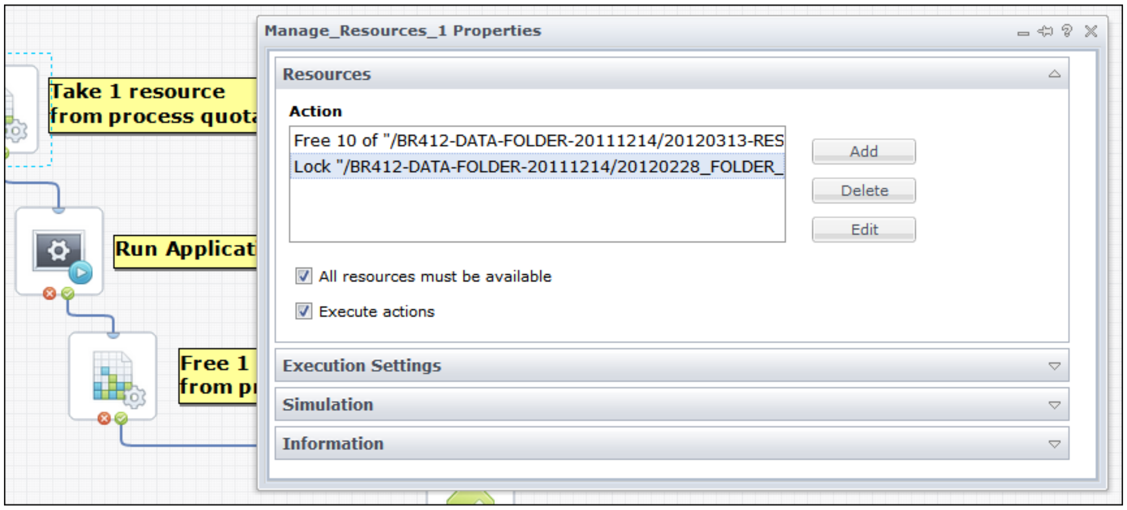
   Unlock Resource

   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.


Take 1 resource from process quota
Run Applicat
Free 1 from pr
Manage_Resources_1 Properties
Resources
Action

Free 10 of "/BR412-DATA-FOLDER-20111214/20120313-RES
Lock "/BR412-DATA-FOLDER-20111214/20120228_FOLDER_

☑ All resources must be available
☑ Execute actions

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

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:

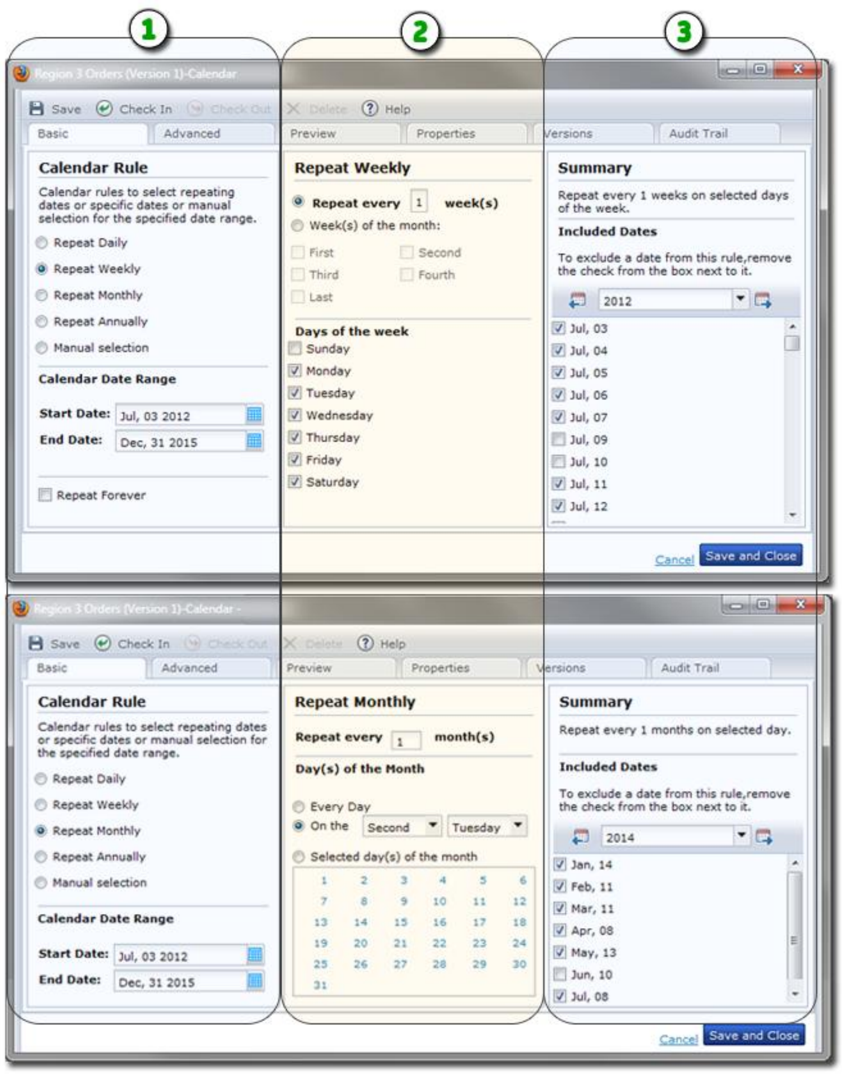
1. Click the Library tab.
2. Click a folder.
3. In the toolbar, click New, select Object, and then choose Calendar.

   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.



Calendar Rule

Calendar rules to select repeating dates or specific dates or manual selection for the specified date range.

- ○ Repeat Daily
- ● Repeat Weekly
- ○ Repeat Monthly
- ○ Repeat Annually
- ○ Manual selection

Calendar Date Range

Start Date: Jul, 03 2012

End Date: Dec, 31 2015

☐ Repeat Forever

Repeat Weekly

● **Repeat every** 1 **week(s)**

○ Week(s) of the month:

☐ First ☐ Second ☐ Third ☐ Fourth ☐ Last

Days of the week

- ☐ Sunday
- ☑ Monday
- ☑ Tuesday
- ☑ Wednesday
- ☑ Thursday
- ☑ Friday
- ☑ Saturday

Summary

Repeat every 1 weeks on selected days of the week.

Included Dates

To exclude a date from this rule,remove the check from the box next to it.

2012

- ☑ Jul, 03
- ☑ Jul, 04
- ☑ Jul, 05
- ☑ Jul, 06
- ☑ Jul, 07
- ☐ Jul, 09
- ☐ Jul, 10
- ☑ Jul, 11
- ☑ Jul, 12

Calendar Rule

Calendar rules to select repeating dates or specific dates or manual selection for the specified date range.

- ○ Repeat Daily
- ○ Repeat Weekly
- ● Repeat Monthly
- ○ Repeat Annually
- ○ Manual selection

Calendar Date Range

Start Date: Jul, 03 2012

End Date: Dec, 31 2015

Repeat Monthly

Repeat every 1 **month(s)**

Day(s) of the Month

○ Every Day

● On the Second Tuesday

○ Selected day(s) of the month

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31					

Summary

Repeat every 1 months on selected day.

Included Dates

To exclude a date from this rule,remove the check from the box next to it.

2014

- ☑ Jan, 14
- ☑ Feb, 11
- ☑ Mar, 11
- ☑ Apr, 08
- ☑ May, 13
- ☐ Jun, 10
- ☑ Jul, 08

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

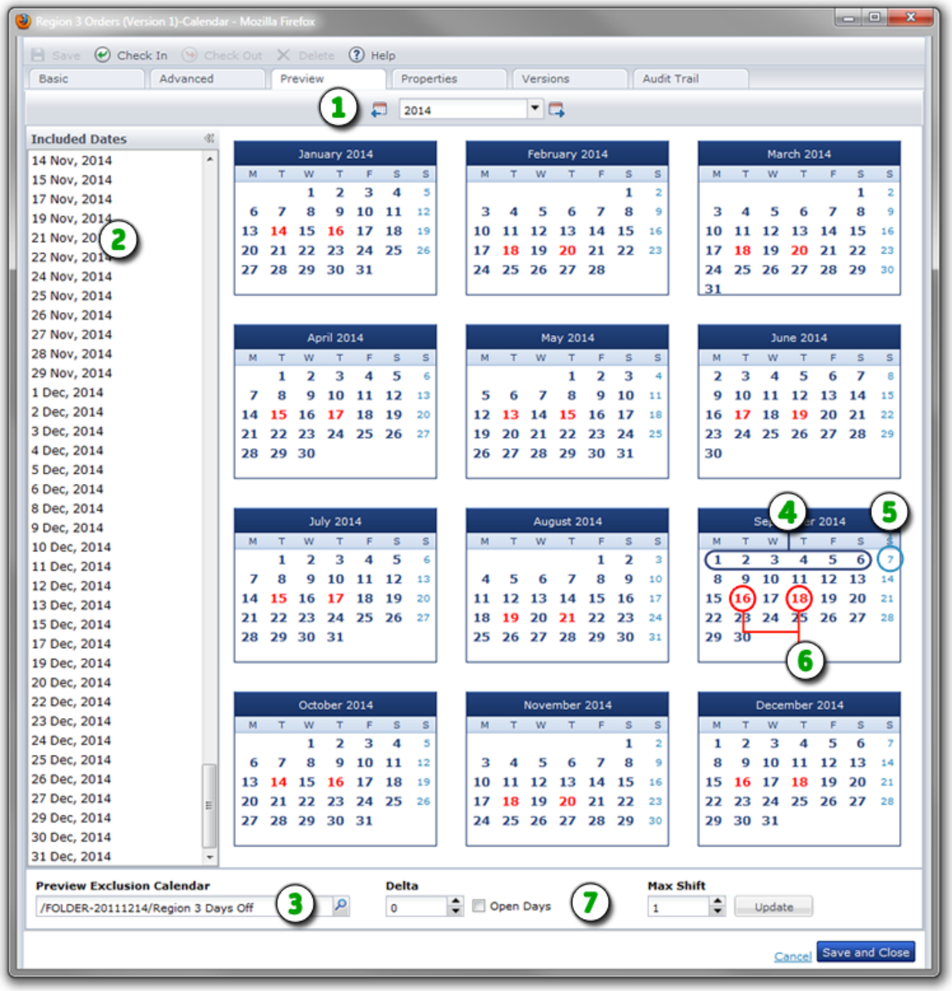
1. Open a calendar.
2. Click the Advanced tab.
3. Expand Manually Included Dates or Manually Excluded Dates.
4. In the month viewer in the Properties pane, right-click a selected date and click one of the include or exclude date commands on the shortcut menu.
   - To include specific dates for a particular 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 visually inspect the dates that you want included and excluded from a calendar.

You can also examine the set of dates in one calendar against another set of dates in a second calendar. For example, define a standard work calendar that needs to omit the holidays or vacation days that you defined in another *exclusion calendar*. While you are editing the standard work calendar, you can test how a vacation exclusion calendar would affect dates.



2014

Included Dates

14 Nov, 2014
15 Nov, 2014
17 Nov, 2014
19 Nov, 2014
21 Nov, 2014
22 Nov, 2014
24 Nov, 2014
25 Nov, 2014
26 Nov, 2014
27 Nov, 2014
28 Nov, 2014
29 Nov, 2014
1 Dec, 2014
2 Dec, 2014
3 Dec, 2014
4 Dec, 2014
5 Dec, 2014
6 Dec, 2014
8 Dec, 2014
9 Dec, 2014
10 Dec, 2014
11 Dec, 2014
12 Dec, 2014
13 Dec, 2014
15 Dec, 2014
17 Dec, 2014
19 Dec, 2014
20 Dec, 2014
22 Dec, 2014
23 Dec, 2014
24 Dec, 2014
25 Dec, 2014
26 Dec, 2014
27 Dec, 2014
29 Dec, 2014
30 Dec, 2014
31 Dec, 2014

January 2014

M	T	W	T	F	S	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

February 2014

M	T	W	T	F	S	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28		

March 2014

M	T	W	T	F	S	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

April 2014

M	T	W	T	F	S	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

May 2014

M	T	W	T	F	S	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

June 2014

M	T	W	T	F	S	S
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

July 2014

M	T	W	T	F	S	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

August 2014

M	T	W	T	F	S	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

September 2014

M	T	W	T	F	S	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

October 2014

M	T	W	T	F	S	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

November 2014

M	T	W	T	F	S	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

December 2014

M	T	W	T	F	S	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Preview Exclusion Calendar: /FOLDER-20111214/Region 3 Days Off

Delta: 0

Max Shift: 1

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 panel displays all the dates included in your calendar rule settings.

Item:	Description:
3	**Exclusion Calendar**: (Optional) Select any separate calendar to easily detect any conflicts shown in red.
4	**Included Dates**: Dates that are included in your calendar appear using dark blue bold numbers.
5	**Excluded Dates**: Dates that are manually or automatically omitted from the calendar rules appear with light blue numbers.
6	**Dates in Conflict with the Optional Exclusion Calendar**: Dates that overlap or conflict with the dates defined by an optional exclusion calendar appear in red.
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 red and omitted. Mark the Open Days check box to count only included days when shifting the schedule to avoid an excluded or omitted date. Open days are those days not specified by a condition or rule that omits or excludes dates. If the Open Days check box is not selected, a shifted date can potentially fall on another excluded or omitted day. Use the Max Shift field to define the maximum number of shifts or adjustments that are allowed 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.

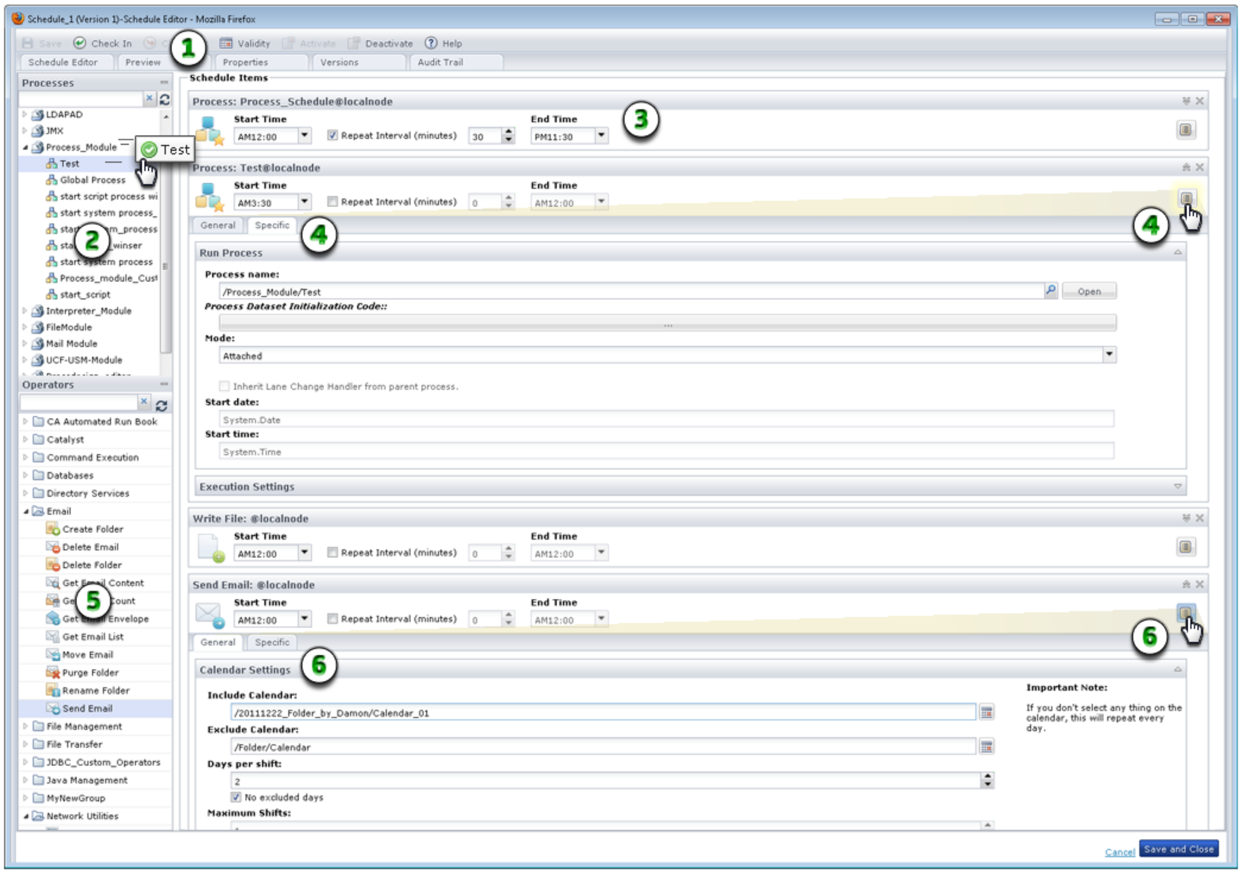
Schedules

Use *Schedule* objects to configure when process or operator tasks run. Specify valid days in the schedule or by reference to previously defined *Calendar* objects. Schedule objects allow you to group, coordinate, and schedule the execution of tasks related to organizational or architectural elements of an enterprise, such as applications, ownership, monitoring, maintenance, and functional processes.

Schedule objects specify:

- The tasks (processes and operators) you want 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, specifying explicit days, or some combination)
- The days when tasks are not permitted to start (using vacation calendars and/or 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 explicitly specified days, do not require a calendar object. To schedule dates using calendar rules, first create calendar objects and then specify the calendars in schedule objects. Calendars define rules that specify valid dates for performing tasks and closed days on which the tasks cannot be run. Use a schedule object to associate operationally related tasks with a calendar and to specify the times at which the tasks are run on the valid days defined by the calendar rules.



Process: Process_Schedule@localnode — Start Time: AM12:00; Repeat Interval (minutes): 30 (checked); End Time: PM11:30

Process: Test@localnode — Start Time: AM3:30; Repeat Interval (minutes): 0; End Time: AM12:00

Run Process
Process name: /Process_Module/Test
Process Dataset Initialization Code::
Mode: Attached
Inherit Lane Change Handler from parent process.
Start date: System.Date
Start time: System.Time
Execution Settings

Write File: @localnode — Start Time: AM12:00; Repeat Interval (minutes): 0; End Time: AM12:00

Send Email: @localnode — Start Time: AM12:00; Repeat Interval (minutes): 0; End Time: AM12:00

Calendar Settings
Include Calendar: /20111222_Folder_by_Damon/Calendar_01
Exclude Calendar: /Folder/Calendar
Days per shift: 2
No excluded days (checked)
Maximum Shifts:

Important Note:
If you don't select any thing on the calendar, this will repeat every day.

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 five tabs including Schedule Editor.
2	**Processes Pane**: Select the processes that you want to include and drag them into the Schedule Items list.
3	**Scheduled Process**:Set the duration and frequency for running the process within a single day from 12:00 a.m. to 11:45 p.m.
4	**Process Properties**: Click the Properties button to view process properties on the General and Specific tabs.

Item:	Description:
5	**Operators Pane**: Select the operators that you want to include and drag them into the Schedule Items list.
6	**Operator Properties**: Click the Properties button 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 59).

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 occurrences of tasks for a specific day. When you configure a task to repeat on a particular day, the application tracks each occurrence of the task. For example, if a task repeats every 10 minutes for half an hour, three occurrences of the task appear in the preview. Use this procedure to plan for a future date or to view the results of tasks scheduled on a specific date in the past.

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. In the Nodes drop-down list, select All Nodes or a particular orchestrator touchpoint.
   c. Select the Current or Archived options.
   d. Click Refresh.

   The application evaluates calendar rules and the validity period associated with tasks before displaying the appropriate occurrences in the list.

   - When you select a past date, the Preview tab includes tasks that were started, their state, their actual start time, and their end time.
   - When you select the current date, the Preview tab indicates whether occurrences have started and their actual 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 showing the configuration settings for the task.
5. Click the Schedule Editor tab to make any changes.
6. To set the valid date range for the entire schedule, click Validity.
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 319).

   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 per day for more than one day, the application 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, the application tracks all 3 x 365 = 1,095 occurrences of the task. Use this procedure to plan for a future date or to view the results of all tasks scheduled on a specific date in the past.

Follow these steps:

1. Click the Operations tab.
2. On the Operations page, expand the Links pane.
3. Click Global Schedules.
4. In the toolbar, select an orchestrator and environment and 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. In the Nodes drop-down list, select All Nodes or a particular orchestrator touchpoint.
    c. Select the Current or Archived option.
    d. Click Refresh.

    The application evaluates calendar rules and the validity period associated with tasks before displaying the appropriate occurrences in the list. If your task repeats, multiple occurrences on the same day appear.

    - When you select a past date, the list includes tasks that were started, their state, their actual start time, and their end time.
    - When you select the current date, the list indicates whether occurrences have started and their actual 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. Double-click an occurrence to view a read-only Properties window showing the configuration settings for the scheduled task.
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 start an occurrence over.

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

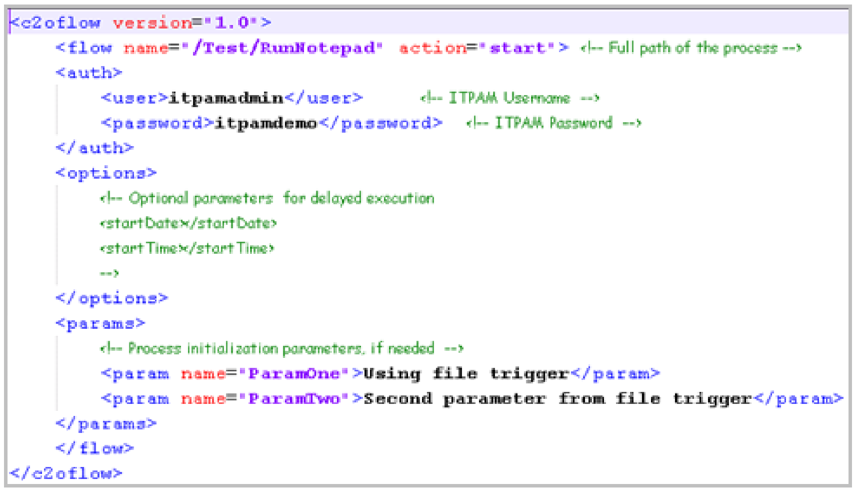
This topic provides a description of the processing sequence for triggers. It uses file and mail triggers as examples.

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.

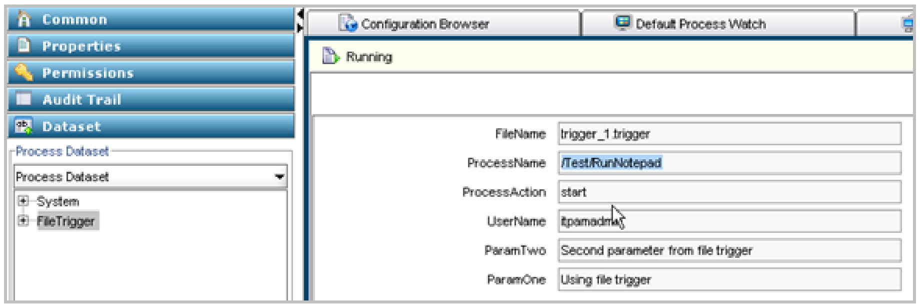
2. If a new file object or mail object is found, CA Process Automation attempts to run the process, based on the XML content.

   An illustration of valid XML content for the file trigger follows:



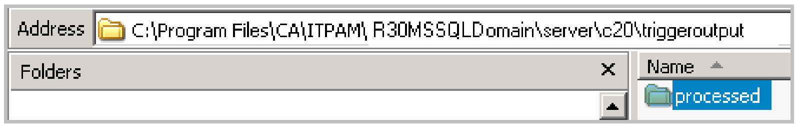
```
<c2oflow version="1.0">
    <flow name="/Test/RunNotepad" action="start"> <!-- Full path of the process -->
    <auth>
        <user>itpamadmin</user>       <!-- ITPAM Username -->
        <password>itpamdemo</password>   <!-- ITPAM Password -->
    </auth>
    <options>
        <!-- Optional parameters 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.

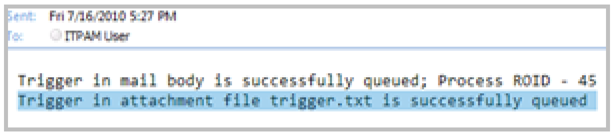


FileName	trigger_1.trigger
ProcessName	/Test/RunNotepad
ProcessAction	start
UserName	itpamadmin
ParamTwo	Second parameter from file trigger
ParamOne	Using file trigger

4. The results are posted to the processed folder in the configured path.


Address
C:\Program Files\CA\ITPAM\ R30MSSQLDomain\server\c20\triggeroutput
Folders
Name
processed

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:



```
Trigger in mail body is successfully queued; Process ROID - 45
Trigger in attachment file trigger.txt is successfully queued
```

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 *Reference Guide* 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 -->
       <auth>
              <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.
   1. Stop the Orchestrator.
   2. 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).

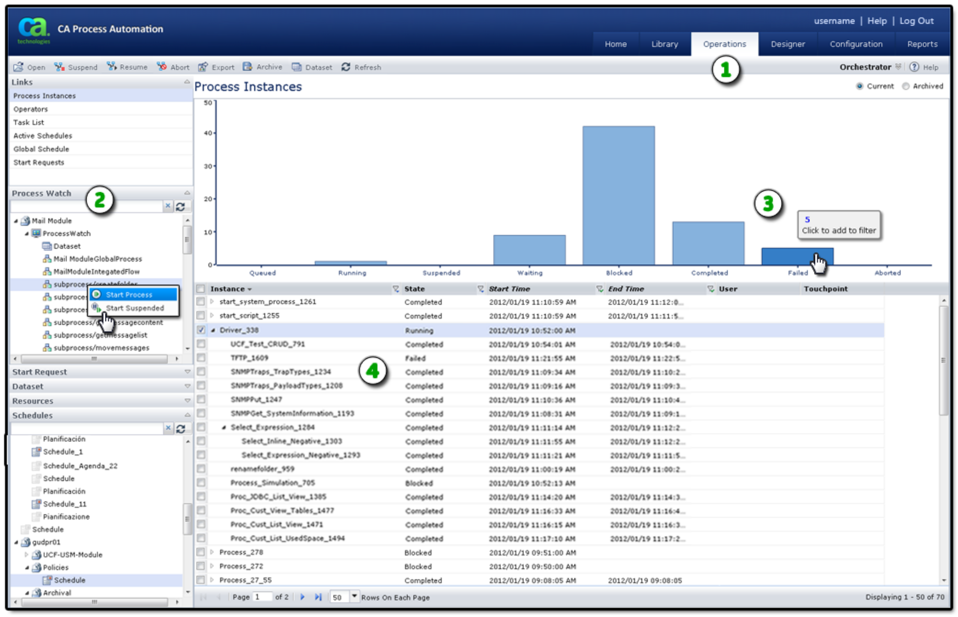
You can open, view, and work with an instance of a process while it is running or after it finishes. Changes made to an instance of a process affect only that instance and do not affect the original process object stored in an automation library.

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:

Process Watch Objects

The Operations page monitors selected tasks and system elements managed by CA Process Automation. The Operations page display all process activities such as process instances, schedules, module invocations, datasets, and user prompts for a selected automation library.



Instance	State	Start Time	End Time	User	Touchpoint
start_system_process_1261	Completed	2012/01/19 11:10:59 AM	2012/01/19 11:12:0...		
start_script_1255	Completed	2012/01/19 11:10:59 AM	2012/01/19 11:11:5...		
Driver_338	Running	2012/01/19 10:52:00 AM			
UCF_Test_CRUD_791	Completed	2012/01/19 10:54:01 AM	2012/01/19 10:54:0...		
TFTP_1609	Failed	2012/01/19 11:21:55 AM	2012/01/19 11:22:5...		
SNMPTraps_TrapTypes_1234	Completed	2012/01/19 11:09:34 AM	2012/01/19 11:10:2...		
SNMPTraps_PayloadTypes_1208	Completed	2012/01/19 11:09:16 AM	2012/01/19 11:09:3...		
SNMPPut_1247	Completed	2012/01/19 11:10:36 AM	2012/01/19 11:10:4...		
SNMPGet_SystemInformation_1193	Completed	2012/01/19 11:08:31 AM	2012/01/19 11:09:1...		
Select_Expression_1284	Completed	2012/01/19 11:11:14 AM	2012/01/19 11:12:2...		
Select_Inline_Negative_1303	Completed	2012/01/19 11:11:55 AM	2012/01/19 11:12:2...		
Select_Expression_Negative_1293	Completed	2012/01/19 11:11:21 AM	2012/01/19 11:11:5...		
renamefolder_959	Completed	2012/01/19 11:00:19 AM	2012/01/19 11:00:2...		
Process_Simulation_705	Blocked	2012/01/19 10:52:13 AM			
Proc_JDBC_List_View_1385	Completed	2012/01/19 11:14:20 AM	2012/01/19 11:14:3...		
Proc_Cust_View_Tables_1477	Completed	2012/01/19 11:16:33 AM	2012/01/19 11:16:4...		
Proc_Cust_List_View_1471	Completed	2012/01/19 11:16:15 AM	2012/01/19 11:16:3...		
Proc_Cust_List_UsedSpace_1494	Completed	2012/01/19 11:17:10 AM	2012/01/19 11:17:2...		
Process_278	Blocked	2012/01/19 09:51:00 AM			
Process_272	Blocked	2012/01/19 09:50:00 AM			
Process_27_55	Completed	2012/01/19 09:08:05 AM	2012/01/19 09:08:05		

Page 1 of 2 50 Rows On Each Page Displaying 1 - 50 of 70

Item:	Description:
①	**Operations Tab**: Click this tab to navigate to the Operations page, a high-level automation dashboard. In the toolbar you can choose an orchestrator, open a process instance, and control it.
②	**Links Pane**: Select a link to view its associated items. For example, click Process Instances to view all processes by state. Expand any of the groups to make a selection by browsing folders. Some objects include a shortcut menu of commands that you can invoke by right-clicking. In this example, the user has clicked Start Process for a subprocess in a *process watch* object.
③	**Chart Area**: Hover over a bar to view the number of items matching a given state. Click a bar to apply a filter that shows only items matching the selected state.
④	**Operations Table or List**: Depending on your selection in the left pane, this area displays the resulting data in a table or list.

Filter Objects Displayed by a Shortcut

You can apply any of the following filters to a Touchpoint Manager shortcut in a Process Watch object to define the objects displayed by the shortcut. The following table lists the available filters.

All Instances

Displays all the process instances in the current orchestrator.

Abnormally Ended Instances

Displays abnormally ended instances of Processes on the Touchpoint. Examine this folder to quickly identity the Processes with problems. Open the abnormally ended instance of a Process to troubleshoot the Process.

Active Schedules

Lists all the active schedules in the orchestrator.

Active Operators

Lists all operators that are being executed.

Active Systems

Lists all the active system objects in the orchestrator.

Ended Instances

Displays ended instances of Processes on the Touchpoint.

Normally Ended Instances

Displays all instances of Processes that completed without any issues.

Queued Instances

Displays all instances of start requests that are in a queued state and waiting for the execution to start.

Running Instances

Displays running instances of Processes on the Touchpoint. This filter does not fetch Waiting, Suspended, or Breakpoint suspended instances.

Suspended Instances

Displays instances of Processes currently suspended by a user or by the application on the Touchpoint.

User Interactions

Displays all tasks pending user action.

Waiting Instances

Displays instances of Processes with an inactive run state. These are 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 Instances

Displays instances of Processes which are blocked because there is not another path to take. For a blocked process instance to complete, a user must suspend the process and provide a path that permits the process to complete. This state not only reflects situations where user intervention is required, but also situations where a process instance cannot proceed because it met an unexpected condition.

The Default Process Watch (opened by clicking an Automation Library in the CA Process Automation Client and then clicking the Default Process Watch button on the toolbar) includes a shortcut for each of these filters. You can use the following procedure to filter any shortcut in a Process Watch object:

To apply a filter to a Process Watch shortcut

1. Expand a shortcut on the Filter palette of the Process Watch window.
2. Click the filter that you want to apply to the shortcut.

   You can add several shortcuts to the same Process Watch object and apply a different filter to each one. This allows your users to quickly view different objects in the same container.

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: We recommend that you organize objects in folders to set rights, export them, and perform other tasks. Do not create objects at the root level because there is no way to manage them 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 dialog appears.
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.

Edit Objects from within a Process Watch

You can right-click the shortcut to an object in the Filter palette of a Process Watch object, and click Open to open the object in its corresponding Designer. You can also open an instance of a Process shown in the details pane of the Process Watch window.

Opening an object by right-clicking a shortcut in the Filter palette opens the object definition on the Orchestrator. Any changes you make to the object pointed to by the shortcut are saved to the object definition in the Automation Library. All future versions of the object will then incorporate the changes, which is the same as opening the object definition inside the Orchestrator object using CA Process Automation Client.

Opening an instance of a Process in the details pane of the Process Watch window opens the instance of the object. Doing so allows you to monitor execution of the instance closely and change it. Any changes you make to an instance of a Process object affects only the execution of that instance of the object and does not change the object definition in the Automation 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 instance of a Process, you must export it to a CA Process Automation export file. To do this, right-click the instance of the Process in Process Watch, and click Export.

Note: For more information about working with instances of Processes in Process Watch, see “Monitor Processes” in the *Administration 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 allows the users to move an automation solution from any level throughout the library hierarchy. For example, moving objects from one folder to another folder or moving the automation objects to 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 relation between the objects though the root folder changes, use the Extended relative path option.

Note: CA Process Automation provides *Use Relative Path for Objects* check box in Process Watch and Package Object Browser windows. Select the *Use Relative Path for Objects* check box to add the relative path to Package or Process watch automatically.

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.

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.

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.
2	Unexpected Results	There is no exit link for a particular exit condition.
3	Abort	An operator aborts or a user aborts an operator.

Priority	Exception Type	Occurs When
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 refers to an agent that is not running.
- **Timeout**: Occurs when the operator has not finished by the specified time and the operator is configured to take the timeout path and either end or continue with result. The exception handler defines the timeout path.
- **Unidentified Response**: Occurs when there is no output connector corresponding to the response.

To create exception handling rules

1. Click the Design 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 onto the process.
5. Expand the Exception operator properties dialog.
6. Expand Information and type a name in the Name field.

   Note: We recommend that you name operators in a default exception handler with a prefix so that 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 additional operators onto 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 process execution for one or more paths in the Exception Handler.
10. In the toolbar, click the Save button.

    The new exception rule is added.

As part of exception handling, you can reset the operator and continue with the process. Additionally, you can select to ignore an exception and continue with the process. To ignore an exception, you must set the operator in simulate mode and continue with the process. You can find the Reset operator on the Common palette. You can use the Reset operator in the process pane, exception handler, and lane change handler.

To configure a Reset operator

1. Right-click Add, Reset to add a Reset operator.

   An entry is shown in the text box with a drop-down to select one of the available Operator names in the current Process. You can add multiple Operator names.

2. Click the Delete, Move Up, and Move Down buttons to manipulate an operator name.

3. Enter an expression (instead of choosing an operator name from the drop-down) which resolves to a string (for example, operator name) or a list of values (for example, a list of operator names) at runtime.

4. Select Continue with Result selected to use Successful or Unsuccessful.

   - When Continue with Result is unchecked, when an error condition is met at runtime, CA Process Automation resets the selected Operators and continues with the process flow.
   - When Continue with Result is selected, the EndCondition option is made available to select either Successful or Unsuccessful. When Successful is selected and if an error condition is met at runtime, CA Process Automation assumes that the selected Operators are successful and continues with the rest of the Process flow. When Unsuccessful is selected and an error condition is met at runtime, CA Process Automation assumes that the selected Operators failed and continues with the rest of the Process flow.

5. (Optional) To ignore an exception and continue with the process, add a Reset operator in the exception handler mode to ignore the exception by specifying the following in the operator list:
   `exceptionStart-<operator-name>.Source`

Note: When you add a new field to the Operators List of the Reset operator, the names of the process Loop operators appear in the drop-down menu of the new field. The Reset operator resets all operators inside the loop operator and resets the loop operator to the first iteration. After the reset, the Loop operator will restart from the first iteration. Because the Loop operator does not support simulation, the Reset operator will *always* reset a Loop operator. The Loop operator will reset and the following field values will be ignored:

- Continue with Result
- End condition
- Pre-execution and Post-execution code

Run Processes Interactively

You can start an instance of a process immediately or in suspended mode.

When a process starts immediately, the Workflow module creates an instance of the process, loads it to memory, and immediately starts processing operators. If a process is started in suspended mode, the Workflow module loads the instance of the process to 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.

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 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 available using Process Watch. You can open Process Watch using either the default Process Watch or with a Process Watch object.

More information:

Process Watch Objects (see page 336)

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.

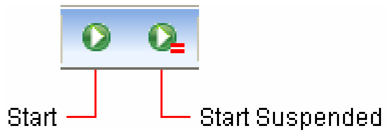
   The Monitor Process Instance prompt opens.
5. Take one of the following actions:
   - Click Yes to open the Designer tab with the debug toolbar. You can begin working with the suspended instance immediately.
   - Click No to load the process into the Operations tab. The Designer tab does not open. You 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 352)

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 Start and Start Suspended commands in the Process Designer run the current version of the process, including any changes you have made.

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 Workflow module creates the instance and starts executing the process immediately. If you click Start Suspended, the Workflow module creates the instance 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 and 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:

Process Watch Objects (see page 336)

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 **Keep Process Loaded 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 **Keep Process Loaded 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 359)

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.

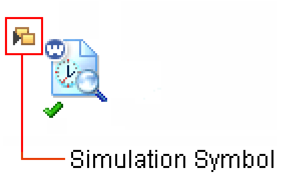
Simulate Processing of a Selected Operator

You can simulate the processing of a selected operator.

Follow these steps:

1. Open any operator in the Library Browser or an open process.
2. Configure the Simulation properties for any operator.
3. If simulation is not turned on for the process, first select the *Override simulation options in the Process* check box.
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 preconditions 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 Outcome* 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 202)

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.

To simulate processing of an entire process

1. Open the process.
2. From the File menu, click Properties or click the Properties button on the Process Designer toolbar:
3. On the Simulation tab of the Properties pane, configure the default simulation settings for all operators in the process:
   - Type
   - Delay
   - End Condition
   - Evaluate pre-conditions and post-execution code

Except for the option to override the process settings, these settings are the same as described for configuring an operator.

The simulation parameters for a process are the same for an operator, except that they are the default settings for every operator in a process. You can always change the simulation property settings for an individual operator by selecting the *Override simulation options in the Process* check box on the Simulation properties pages for an operator.

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 file contents. Masks used by CA Process Automation are JavaScript regular expressions that describe sets of strings without actually listing their contents.

Masks can be used to determine whether a particular pattern is matched and to extract parts of the matched string. Extracted substrings are placed in variables and can in turn be used to specify values for parameters of other Operators. Examples of Operators that use masks are Directory Entries Operator and extraction Macros. Masks are also used for text input fields to specify permissible strings such as date formats.

This section contains the following topics:

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 each of the following entries as an Operator target:

- IP address or FQDN.

  You could know the IP address or FQDN of a host that you want to target without knowing its Touchpoint or Proxy Touchpoint. Because of the robust processing logic for IP addresses and FQDNs, you can specify any target with its IP address or FQDN. In this case, CA Process Automation searches active Orchestrators and Agents in the Environment for an enabled Touchpoint, Proxy Touchpoint, or Host Group that supports processing on this host. If found, CA Process Automation executes the Operator on that Touchpoint, Proxy Touchpoint, or Host Group. Processing a target specified with its IP address or FQDN is more costly than processing that same target expressed with a Touchpoint or Proxy Touchpoint. Processing a target specified as an IP address or FQDN requires time consuming queries of the DNS. The advantage of this option outweighs the cost in certain circumstances.

- AgentID identifier of the target Orchestrator or Agent.

  You could need to target a specific host with an installed Agent, where the Touchpoint for that Agent is not an acceptable target because it is mapped to multiple Agents. As a target, AgentID is an alternative to Touchpoint. For example, if you need consecutive Operators to target the same host, specify the AgentID as the target for both Operators.

This section contains the following topics:

How Targets for an Operator Can Be Specified

The target of an Operator is the host on which the Operator executes. A host that can be a target of an Operator must be configured with a Touchpoint, a Proxy Touchpoint, or a Host Group. When determining the execution settings for an Operator, content designers typically specify the target host based on how that target is configured. The exception to specifying a Touchpoint is when the Process requires that the Operator executes on the same host as the previous Operator or the same host on every execution.

Specify the Target as	When
Touchpoint	The target host has an Orchestrator or Agent installed.
Touchpoint Group	The Operator is to execute on all of the hosts associated with Touchpoints in the specified Touchpoint Group.
<AgentID>	*Exception to Touchpoint:* The Operator must execute on a specific host each time it runs. Specify the same AgentID for consecutive Operators when the Touchpoint is mapped to multiple hosts, but the same host must be used as the target for both Operators.
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 **Note:** The host name can be used, but FQDN is preferred.	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.

Important! If you specify the target with a Touchpoint that has a name that is also a hostname of a computer in the Environment, unexpected results can occur. Unexpected results occur if the Agent associated with that Touchpoint is inactive when you run the Operator. In this case, CA Process Automation does run the Operator on the Touchpoint. Instead, CA Process Automation attempts to resolve the target (intended to be the Touchpoint) to the name of a host in the Environment. If found, 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 found.

Processing a Target Specified as an IP Address or FQDN

When you specify the target host of an Operator with an IP address or an FQDN, CA Process Automation queries the Domain Name Server (DNS) multiple times. 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 FQDN for that host and any additional IP addresses associated with that FQDN. Subsequent searches test against all of the host specifiers 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. Because Host Groups define remote hosts with subnet and host name patterns, Host Groups can include hosts with Orchestrators or Agents that are mapped to Touchpoints. Host Groups can also include individual hosts that are mapped to a Proxy Touchpoint. This means that a host identified with its IP address or FQDN could be processed with a Touchpoint name or Proxy Touchpoint name, if it exists.

Some searches find any active Orchestrators or Agents defined on that host that have enabled Touchpoints. Another search finds any enabled Proxy Touchpoint on an active Agent that is mapped to this host. This same search finds any enabled Host Group on an active Agent with host name patterns or IP address subnets that match the FQDN or an IP address of this host. When search criteria are met, these queries delay the start of execution of the Operator. CA Process Automation executes the Operator on the first found enabled Touchpoint, Proxy Touchpoint, or Host Group that is running on an active Orchestrator or Agent on the target host. 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 following processing resolves whether to run the Operator on a Touchpoint, Proxy Touchpoint, or Host Group, given the target IP address or FQDN:

1. Searches the DNS to find all the identifiers for the specified target. If the target is specified as an FQDN, searches for all the IP addresses for that FQDN. If the target is specified as an IP address, searches for the FQDN associated with the target IP address and any additional IP addresses for that FQDN.
   - If connection to the DNS fails, writes an error to the log file and exits.
   - If the query returns the FQDN and additional IP addresses, uses not only the specified IP address but the FQDN and the additional IP addresses in all subsequent searches.
   - If the query returns one or more IP addresses, uses not only the specified FQDN but also the associated IP addresses in all subsequent searches.

2. Searches for an active Orchestrator with an enabled Touchpoint in the current Environment that is installed on a host with an identifier retrieved in Step 1.
   - If found, uses that Orchestrator Touchpoint to execute the Operator.
   - If not found, continues processing.

     Note: Clustered Orchestrators are not detected because hosts with clustered Orchestrators do not appear in the CA Process Automation Domain.xml file.
3. Searches for an active Agent with an enabled Touchpoint in the current Environment that is installed on a host with an identifier retrieved in Step 1.
   - If a host with an identifier retrieved in Step 1 exists that has an active Agent with one enabled Touchpoint, uses that Touchpoint to execute the Operator.
   - If a host with an identifier retrieved in Step 1 exists that has an active Agent with more than one enabled Touchpoints, uses one of the enabled Touchpoints to execute the Operator.
   - If not found, continues processing.
4. Searches for an active Agent with an enabled Proxy Touchpoint or Host Group in the current Environment that is mapped to the target host as a remote host. For Host Groups, the queries use pattern matching to find whether the referenced subnet or the referenced host name pattern includes any of the host identifiers retrieved in Step 1. For Proxy Touchpoints, searches for an exact match to any of the host identifiers retrieved in Step 1.
   - If the only match is a Proxy Touchpoint, uses that Proxy Touchpoint to execute the Operator.
   - If the only match is a Host Group, uses that Host Group to execute the Operator.
   - If multiple matches are found, uses the first found Proxy Touchpoint or Host Group to execute the Operator.
   - If there are no matches to the search criteria, continues processing.

5. Determines why the Operator is not executed, then fails the Operator with a message explaining the failure.

   - If none of the identifiers retrieved in Step 1 belongs to any host in the current Environment, fails the Operator. The error message follows:

     ```
     Message could not be posted to the node.
     ```

   - If one or more identifiers retrieved in Step 1 belong to a host where all Orchestrators and Agents are inactive, the Operator fails with the following error message:

     ```
     Message could not be posted to the node.
     ```

   - If one or more identifiers retrieved in Step 1 belong to a host with all disabled Touchpoints or are referenced with only disabled Proxy Touchpoints or Host Groups, proceeds as follows. First, returns a disabled Touchpoint or Host Group in the TouchpointName field of the Operator dataset system output variables. Then, fails the Operator with the following error message:

     ```
     Node is disabled.
     ```

More information:

Target Settings (see page 199)

Processing a Target Specified as the ID of an Agent or Orchestrator

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 different agents on different hosts.

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 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 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 mapped to the target are disabled. CA Process Automation posts the following message with one of the disabled touchpoints:

  ```
  Node is disabled.
  ```

Use Case: Track Recovered Processes through Logs

Consider the following scenario, which offers an alternative to opening hundreds of Processes to identify the Processes that had been recovered through auto Operator recovery. 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 attempt to run Operators on the two Touchpoints.
3. Three hundred Processes go into BLOCKED state.
4. The Agent becomes active.
5. Operators recovery, which is initiated automatically, recovers the 300 Processes.
6. A few hours later, you notice that 1000 Processes finished executing. You open one of them to examine its logs and learn that it was automatically recovered.
7. To find out which of the 1000 Process that finished executing, finished after being recovered, do one of the following:
   - Open each of the 1000 Processes to determine which were auto recovered.
   - Examine the log file.

   Note: The following topic walks you through the preferred alternative, examining the log file.

Operators Auto Recovery Example

If *Operators Auto Recovery* is selected, the default setting, you do not have to initiate the recovery. Each orchestrator within the environment detects that the agent or orchestrator is active again and runs the recovery. Each orchestrator runs recovery on any enabled touchpoint, host group, or orchestrator that is mapped to the agent or orchestrator within 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 primary node runs the recovery.

Automatic Recovery Example

Consider the following scenario where 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 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 attempts to run the automatic recovery on the three touchpoints.

Assume that 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 was started automatically, the process logs state 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, within the process and finally 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