BUILDING SOLUTIONS

Pega Robotic Automation Studio lets you create solutions that integrate and automate desktop applications without writing code. When creating solutions, you go through many of the same logical steps and practices required by software programming. This document describes both general advice applicable to any programming project, and specific practices recommended for creating solutions with Studio.

To use this guide, you should be proficient creating Studio solutions and have completed the Pega Robotic Automation Architect Essentials course.

This document includes these topics:

Installation considerations

Keep in mind these considerations when installing Studio:

- Do not install Studio or the Studio plug-in and Runtime on the same machine. Doing so is not supported and can cause issues if you are also installing Studio Services and you plan to use MonitorAll.
- Review the Minimum Requirements topic in the appropriate installation instructions before you launch the Setup wizard. You can find these installation instructions on the Robotic Automation site:

  Robotic Automation
- Launch the Setup wizard by right-clicking on it and selecting the Run as Administrator option.
- Install from a local drive, not a mapped drive.

Naming and design considerations

Designing solutions based on user actions

Capture the sequence of actions you want to automate in a series of screen shots and make a note of all your keyboard and mouse interactions. The more detail you capture, the easier the design and testing process will be. Base your solution design on the user interactions with the applications and processes you plan to automate. Remember, each automation in your solution must begin with an event, therefore user and system actions play an important role in solution design.

Naming convention for solutions, projects, adapters, controls, and components

Establish a standardized naming convention for all parts of the solution. By doing so, you can quickly identify the target in the event you need to diagnose a problem or troubleshoot match rules. Standardized naming conventions are especially helpful when working with a group of designers who are designing separate areas of a solution.

Naming solutions and projects

Solutions and projects should be named in such a way that lets you easily identify the high-level function they perform.

Naming adapters

Adapters should be named in such a way that lets you easily identify the application they represent. We suggest naming adapters identically to their current application shortcut names. It is also recommended that the names be readable since they are visible to the user on the Runtime menu.

For example, in the CRM web solution you created in the Pega Robotic Automation Architect Essentials course, the adapter for the Windows application is named: *CRM Information*, and the adapter for the web application is named *ACME Product Search System,* making them easily identifiable in the Solution Manager. In a more complex solution, it can become difficult to locate the correct adapter without a logical naming convention.

Naming automations

Automations should be named in such a way that lets you easily identify the action they perform. We suggest that automations follow the same naming conventions as methods in their code counterparts. Essentially, if you have a verb-noun rule for method names, use the same for automations that behave like methods. For other automations, we suggest naming them based on what they are trying to accomplish.

For example, in the CRM web solution, automations relating to the CRM Information Windows adapter are:

Automation Name	Action Performed
Set Account Window	Sets focus on the New Call Account window.
Get Account Window	Gets data from the Account window.

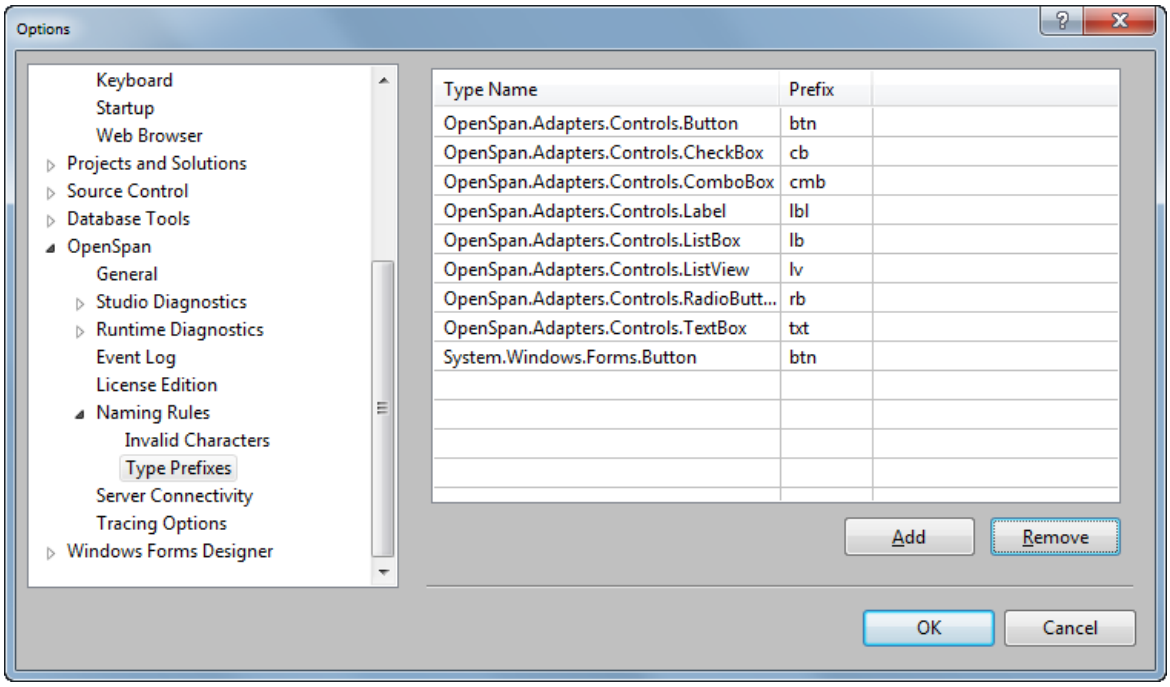
Naming controls

Establishing a standardized naming convention for controls and components is extremely important. Whether you are building an automation, debugging a solution, or troubleshooting match rules, being able to quickly identify the controls makes the task much easier.

Here are some examples of standardized naming convention prefixes for several common controls.

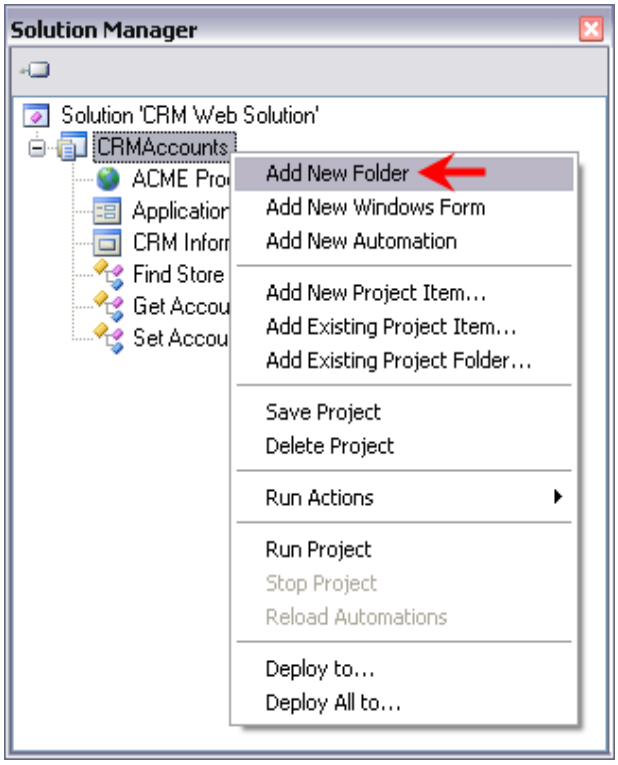
Common Control	Standard Prefix
Button	btn
Check box	cbx
Combo box	cmb
Label	lbl
List box	lbx
List view	lv
Radio button	rbtn
Text box	txt
Toolbox	tb
Windows form	frm

You can modify the default prefixes used by Studio for common controls by using the Auto-Naming Rules feature. Access this feature from the Tools > Options > OpenSpan > Naming Rules > Type Prefixes option. Here is an example of the dialog:

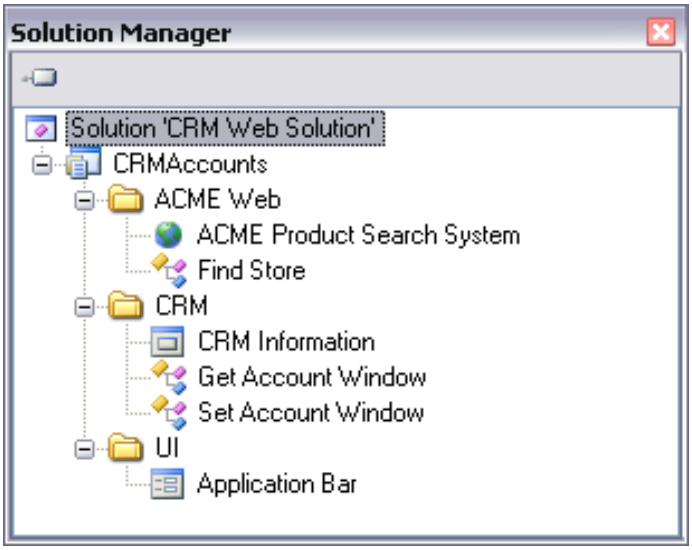


Creating a Solution Manager hierarchy based on tasks

Keep the Solution Manager hierarchy organized. Use folders to group related items.

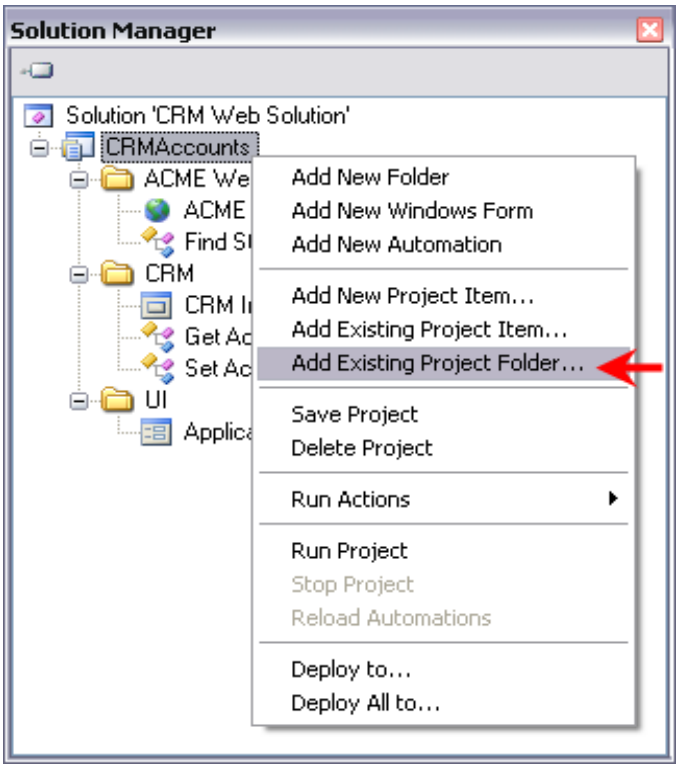


For example, using the CRM web solution from the Pega Robotic Automation Architect Essentials course, you can create folders to store common CRM, ACME, and User Interface (UI) elements. This lets you easily locate objects when working with the solution. See the following example:



In addition to grouping related functionality or automations, we suggest using the top-level folders to group items into *Systems*. For this purpose, a *System* is defined as a group of related functionality surrounding an individual adapter (or possibly multiple adapters) with the ability to share the entire contents of the system between different solutions.

For example, you can create a CRM folder that completely encompasses the functionality of CRM. When another solution needs CRM, you can simply take the entire system (that is, copy the existing folder, including all of its contents), into the solution where CRM is needed.



You can then access any of the properties, methods, or events which have been exposed. This lets you test individual pieces of functionality separate from the overall solution.

Saving backup deployment packages of your solutions

As with any development project, it is essential that you backup your solutions at regular intervals. We strongly recommend using a source control system for solutions.

Interrogating applications

Interrogating all applications before building automations

It is always a good practice to interrogate every application you need in the solution before building the automations. By doing so, you can uncover any potential application integration issues before building the automation logic.

Testing applications in the Studio environment by unit testing solutions

For complex applications, create a solution and add a Windows form and the application. Interrogate all areas of the application that you want to use in your final solution. Add controls to the Windows form to unit test interaction with the application.

When you are satisfied that the application runs properly in the Studio environment, add the adapter by adding an Existing Project Item to the final solution.

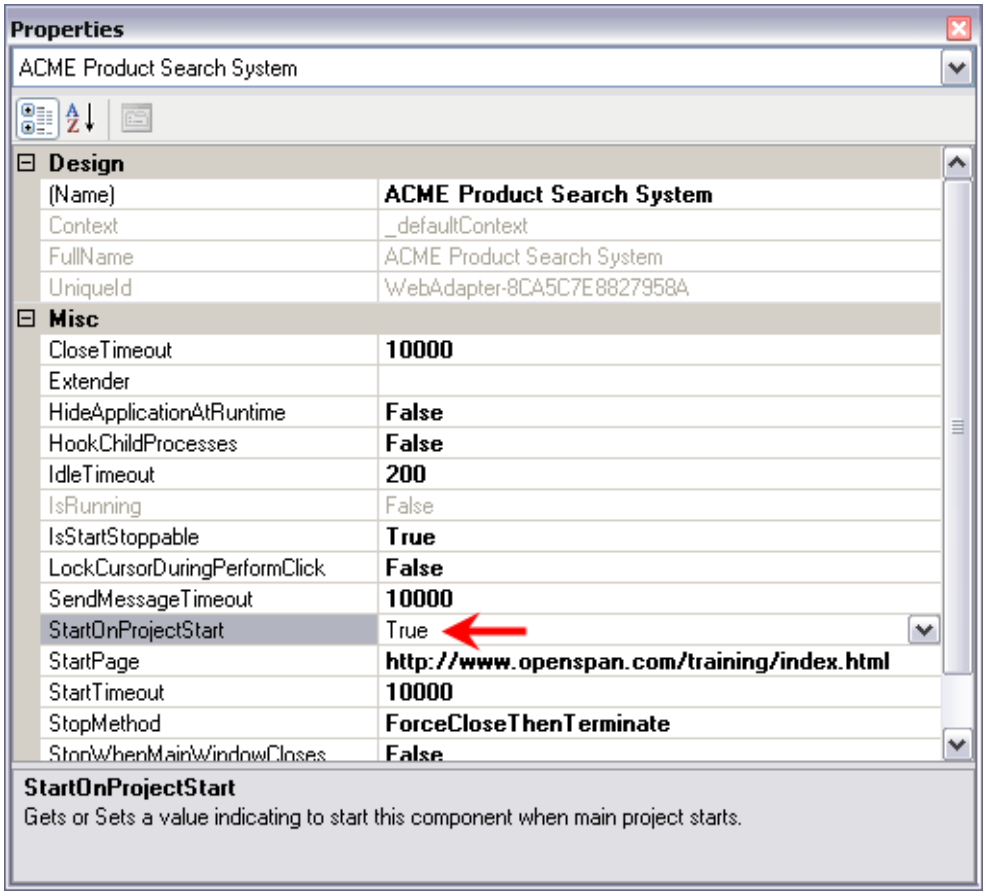
Following match rules best practices

Each application type – Windows, web, text – has special considerations regarding how match rules are applied. Refer to the Best Practice: Using Match Rules document for details on working with match rules.

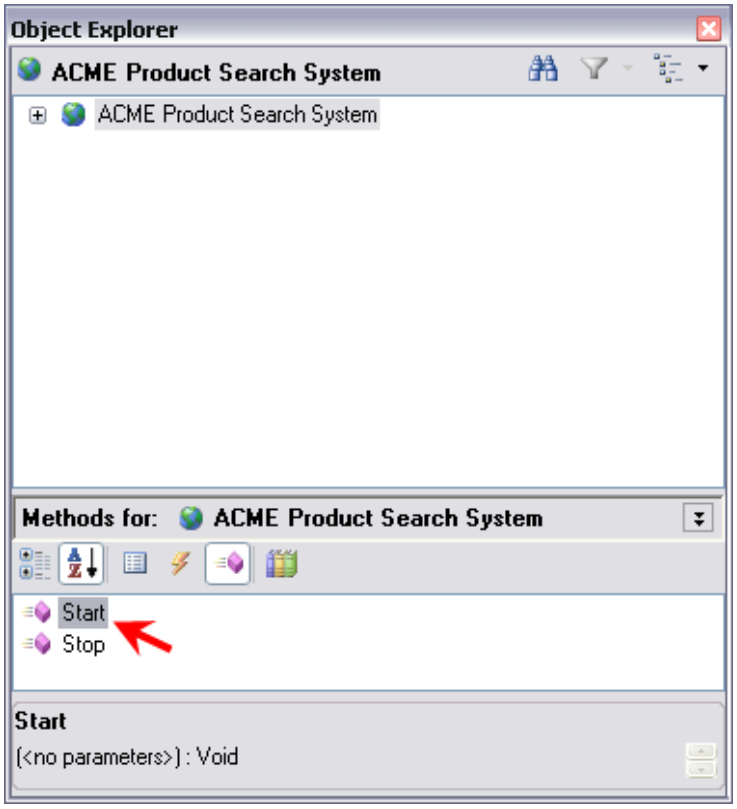
Using integration processes

Starting applications as required

When you add an adapter to a project, the StartOnProjectStart property for the adapter is set to True by default. This means that the application starts when the project starts, whether it is required at startup or not. It is a good practice to only start the applications that are required at the start of the project.



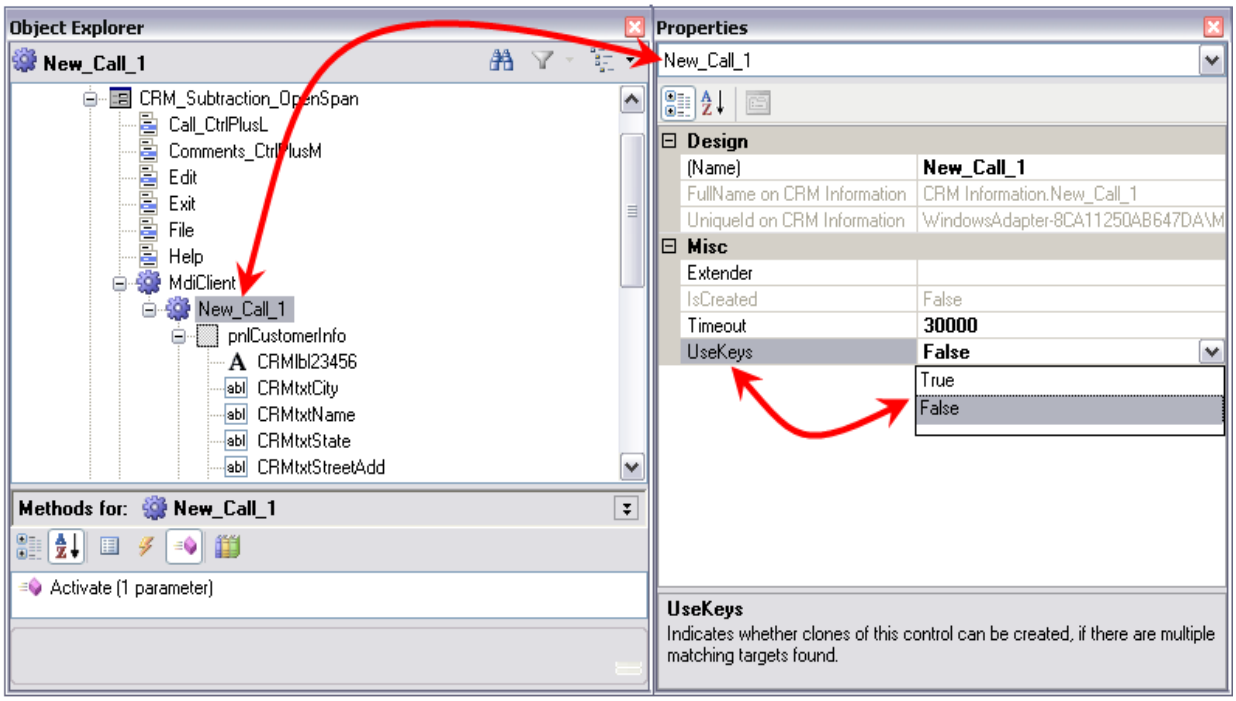
Use the Start method in an automation for the adapter to start an application as needed by the project execution. By controlling the number of applications running during the execution of your project, you improve the overall performance of the project.



For example, you can use a main automation that uses the Started event to start anything you need. By default, you can turn all the adapters off and start them from the automation as needed.

Setting the UseKeys property when required

When working with applications that contain objects that can be cloned, such as Multiple-Document Interface (MDI) child windows, you can use the UseKeys property. See the following example from the CRM application used in the Core Training where multiple instances (clones) of the New Call (New_Call_1) window are created.



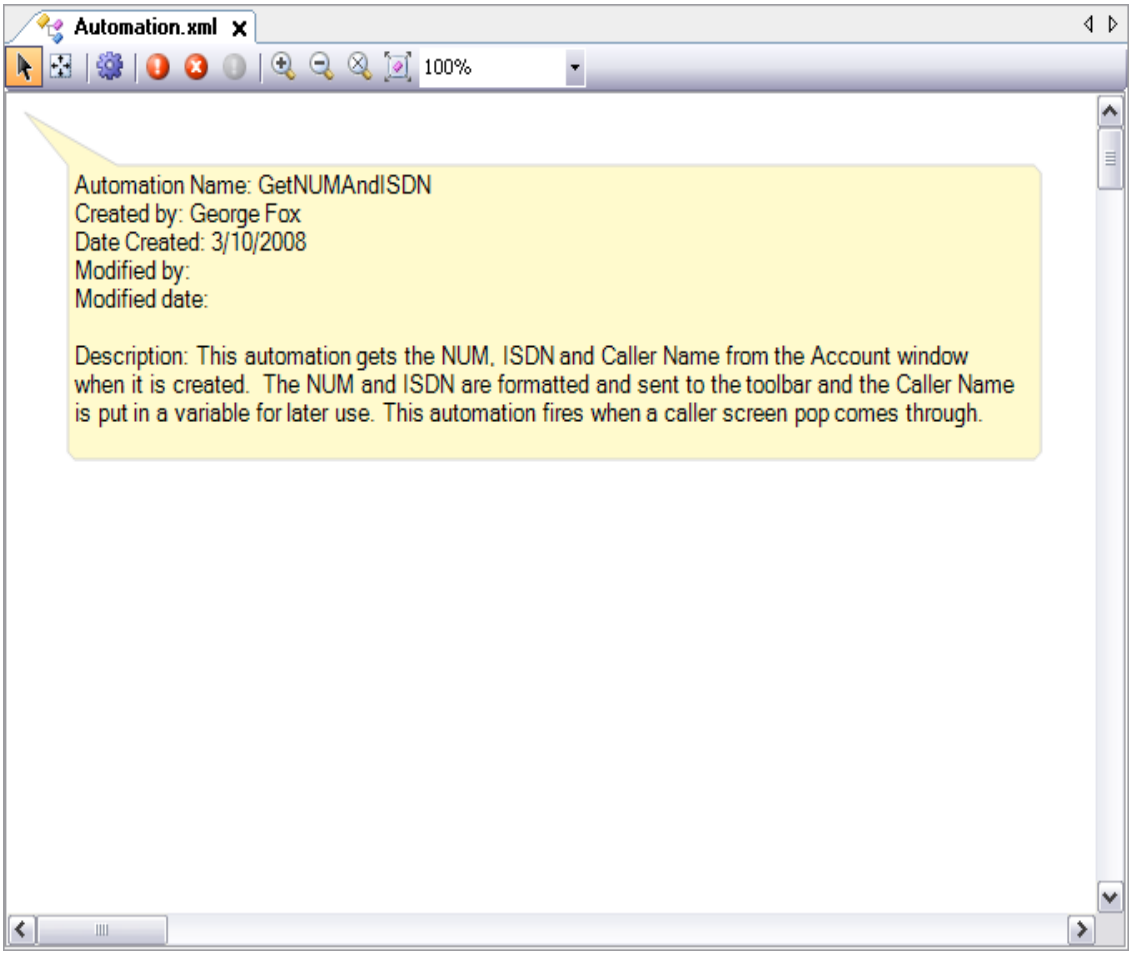
By default, this property is False, indicating that keys are not assigned to the object. However, if multiple copies of the object exist at runtime, and your solution needs to distinguish between the various instances of the object, then set the property to True.

Make sure to set the property to True for the object that is cloned. This means you need to set the value for the base object in the hierarchy that can be cloned to True. If your solution does not use objects that can be cloned, leave this value set to False.

Using automations

Adding comment text as header to all automations

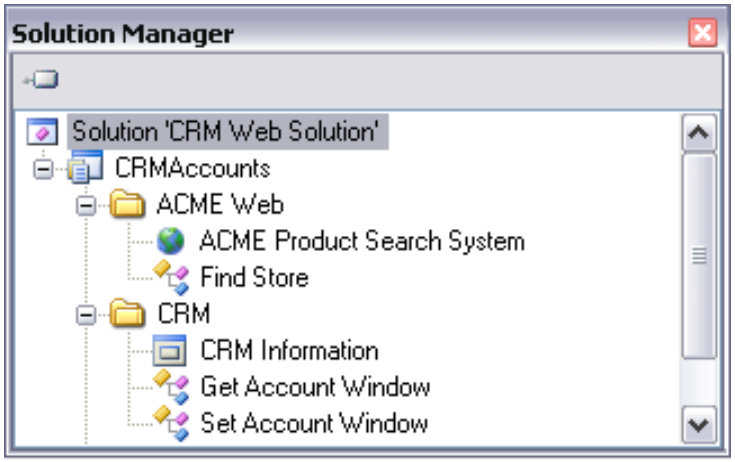
Comments within an automation can be extremely helpful when working with multiple designers. It will enable a designer to look at an automation and quickly discern the logic and flow or the process. For example, a standard Automation header comment such as the following can be used:



Keeping automations with the task or adapter

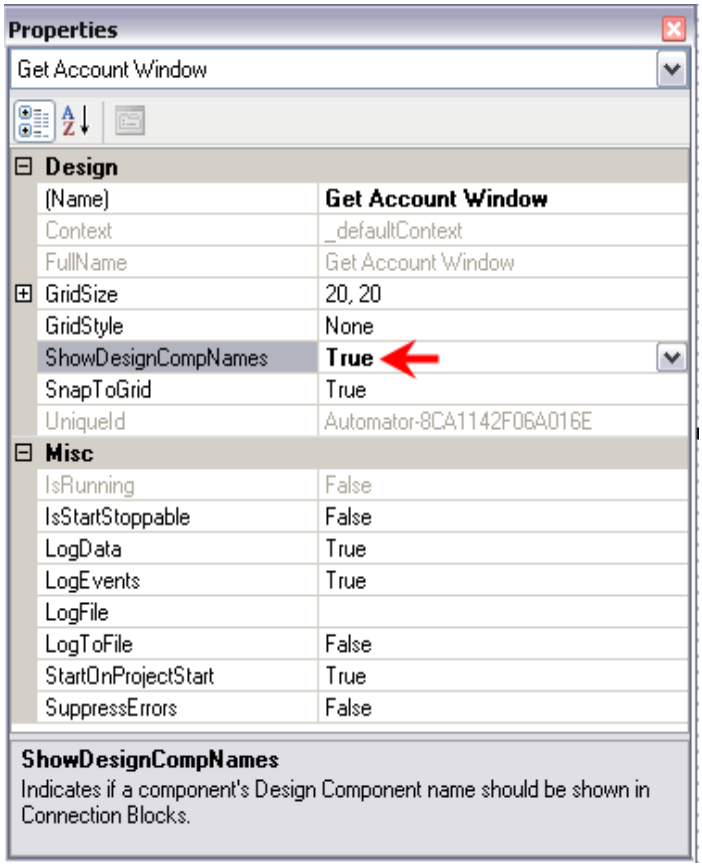
In a complex solution, you are likely have many automations. Keeping those automations with the task or adapter makes it much easier to locate when you debug or edit your solution.

In this example, you can see the automation associated with the ACME web application and the automations associated with the CRM application.

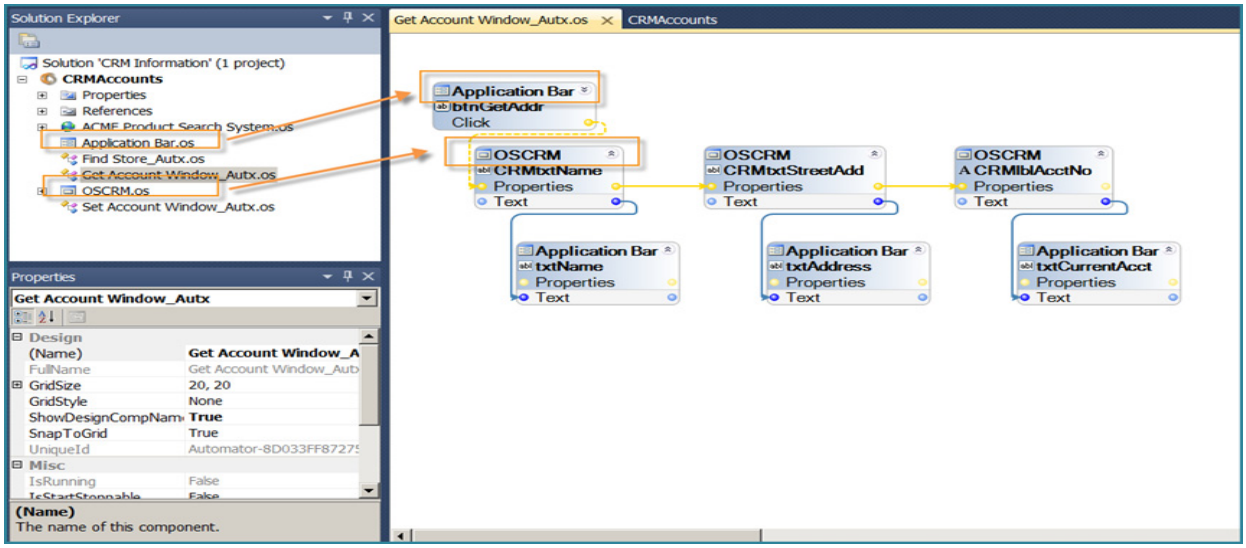


Setting the automation's ShowDesignCompNames property to True

By setting the Automation property ShowDesignCompNames to True, Studio displays the full design name of the components in the automation design blocks.



See the following example:



Using this option, viewed or printed automations clearly show the adapter or form associated with each component.

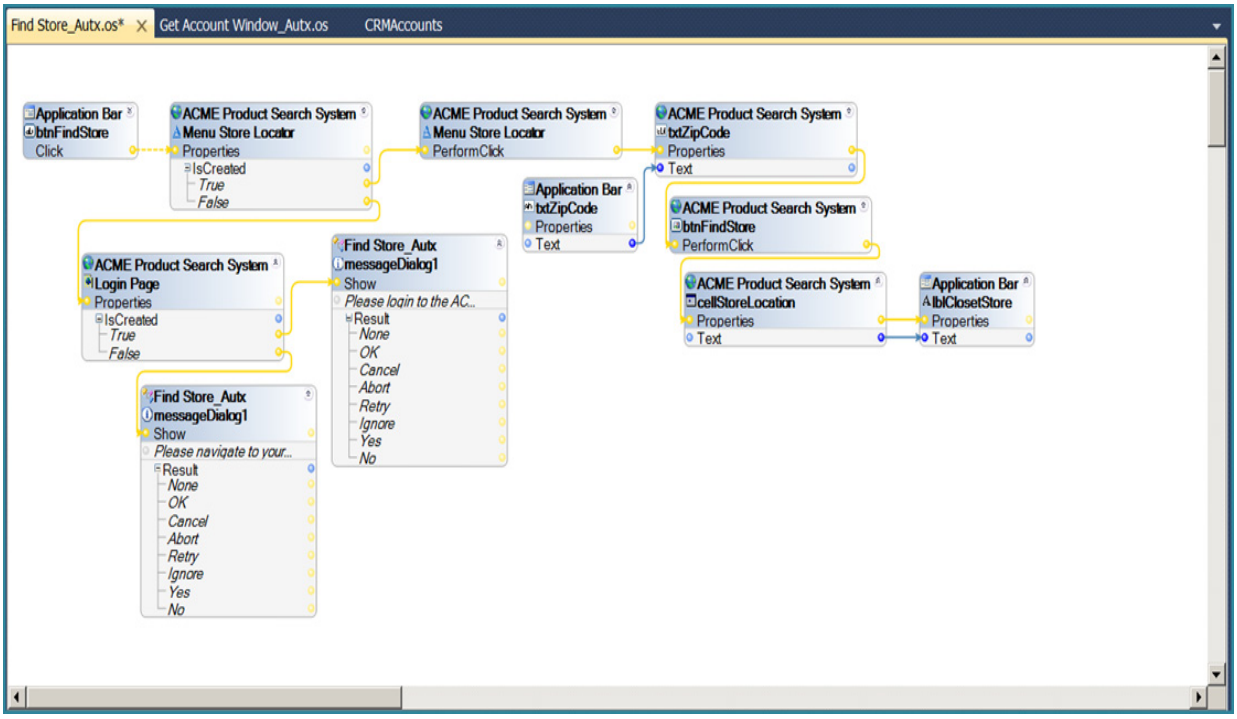
Using WaitForCreate and IsCreated

Before using the properties, methods, or events of a component in an automation, confirm that the component is created and matched. Use error trapping for unmatched targets in automations.

You do not need to check for all objects on a web page to be created before using them in the automation. Once one object is loaded, you can use the rest of the objects on the page without checking the matching status. The following are helpful checking the matching status:

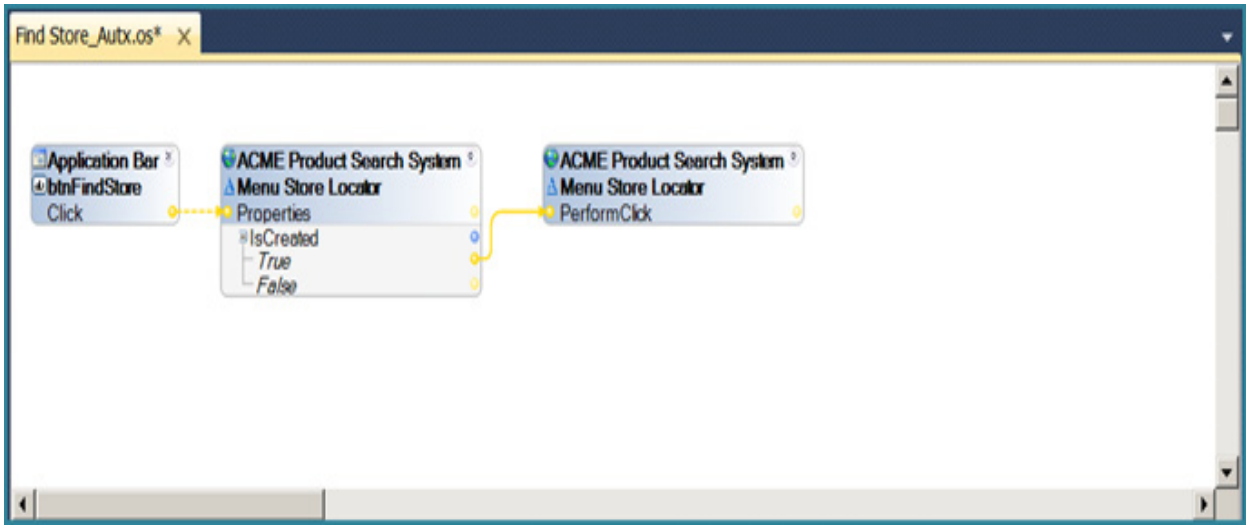
- **WaitForCreate Method** — Waits for an object to be created. Returns True if the object is matched before the WaitTimeout period elapses. Otherwise, it returns False.
- **IsCreated Property** — Returns True if the object is matched at a specific instance in time. Otherwise, it returns False.

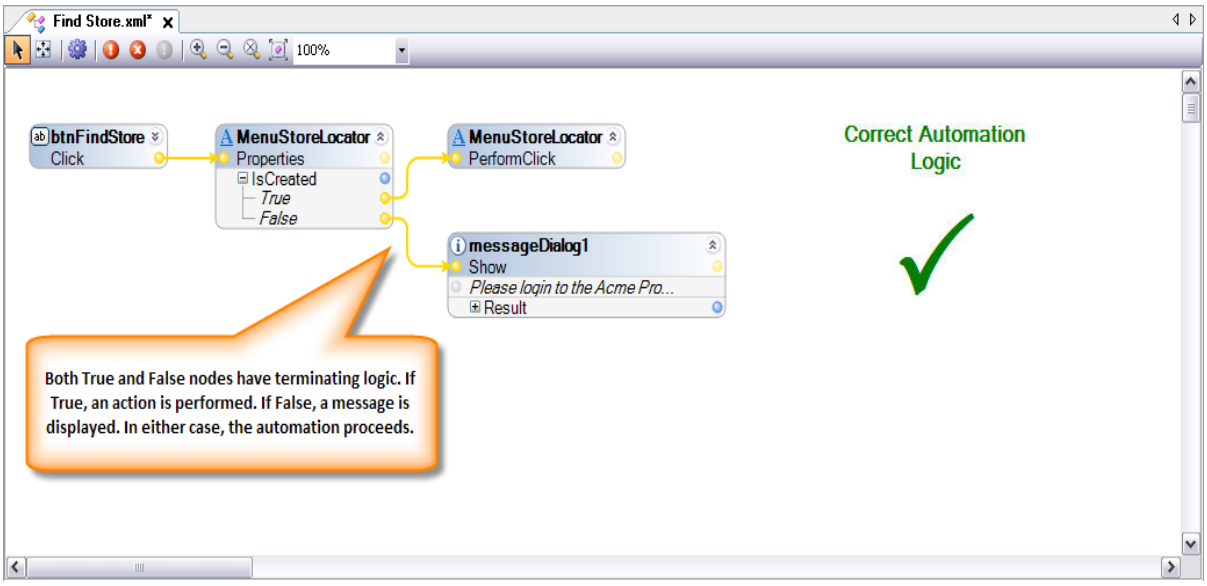
The following example of the Find Store automation from the CRM web solution in the Pega Robotic Automation Architect Essentials course shows how the IsCreated property is used to make sure certain web objects are created (matched) before being used by the automation logic.



Terminating branched event paths

Anytime the event path in your automation branches, make sure all event paths terminate logically. The illustrations below shows the wrong way and the correct way to handle a branched event path:



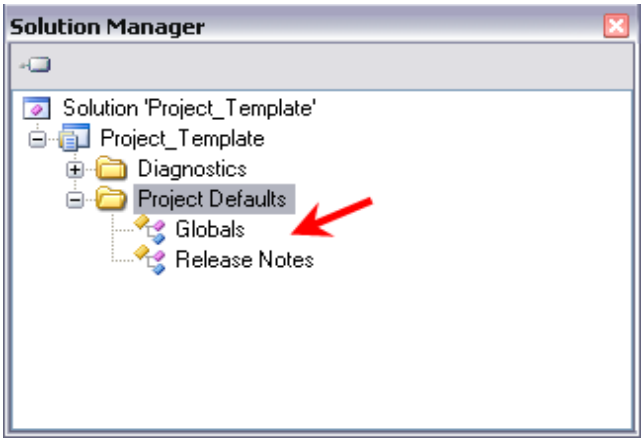


In this example, a message dialog was used to terminate the False event path. If you do not want to alert the user or take any action based on the event, try using the DiagnosticLog component to capture the event in the Studio log. See “Adding diagnostic logging” on page 21 for an example of logging automation that can be called from any automation in a solution.

Using global storage for variables and a release notes automation

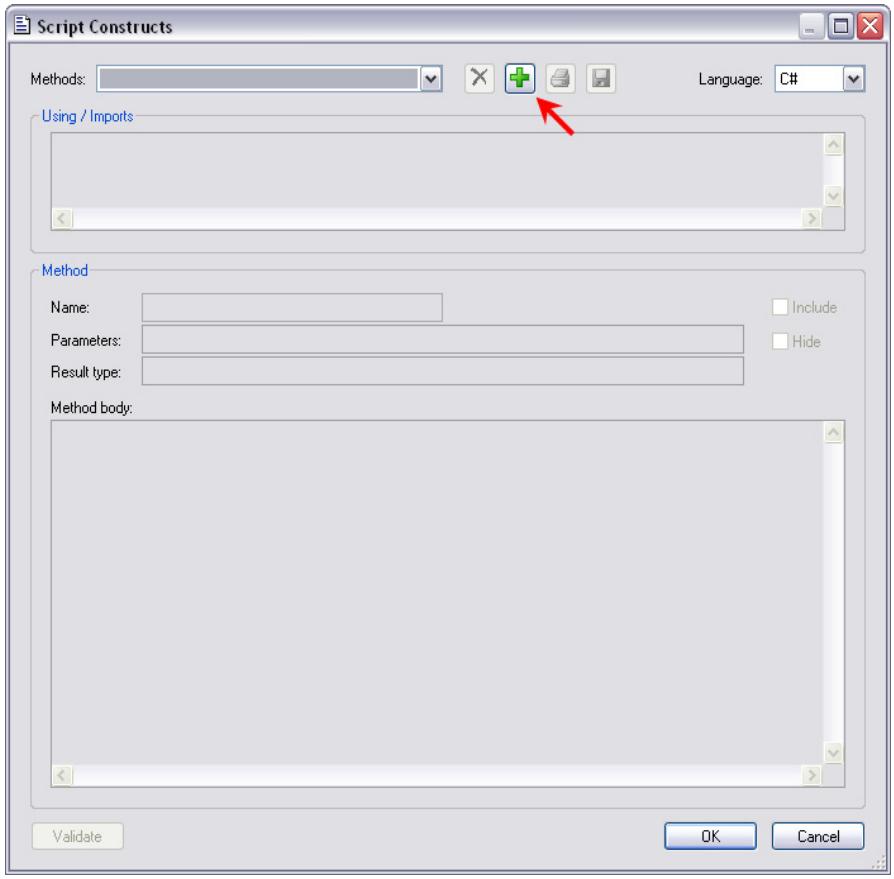
It is a good organizational practice to set up global storage for variables or components that can be called from multiple locations within the solution.

Another good practice is to include an automation for storing notes and tracking any revisions to the solution. The Project_Template solution available in “Finding additional information” on page 25 provides examples of these automations.



Using a single script with multiple methods

If you require Script components in your solution, use a single Script component and add multiple methods to the component rather than using multiple Script components containing only one method. Each Script component adds an assembly file to the deployed project so by using a single Script with multiple methods, you reduce the overhead of the project.

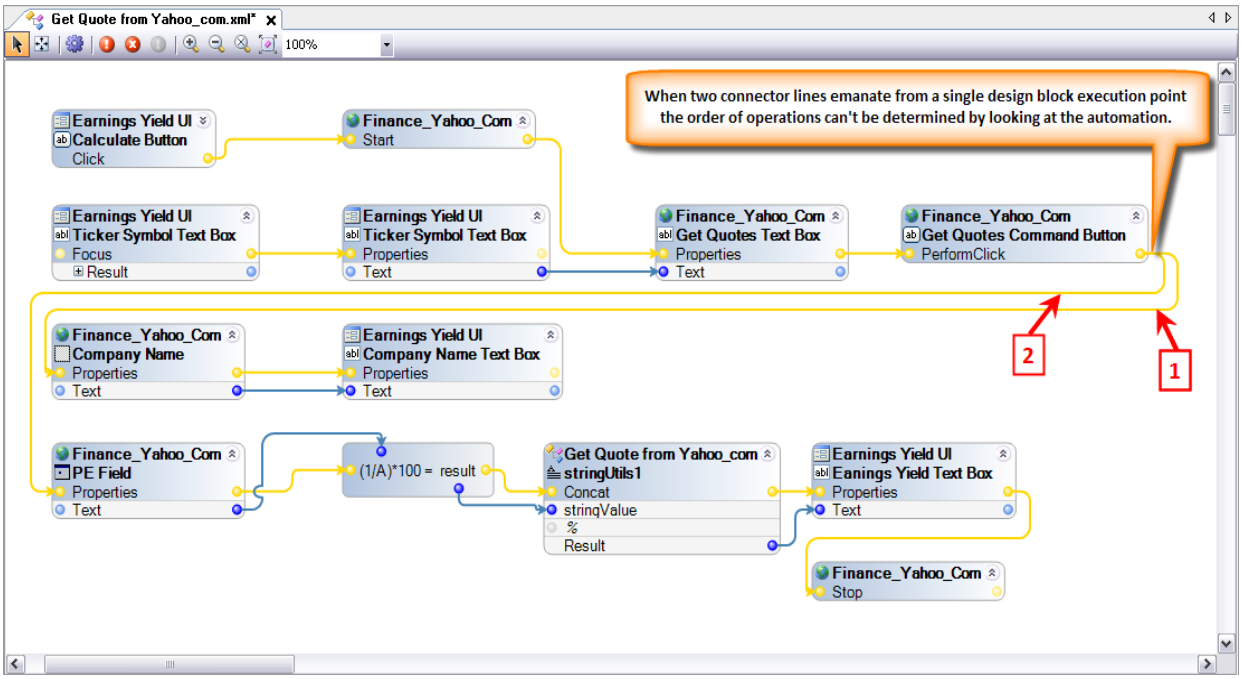


Following threading best practices

Design your automations following the guidelines set forth in the Best Practices: Understanding Threading document. This document provides details on threading in the Studio framework.

Running only one connector line from a design block execution point

It is a good practice to run just one connector line from a single design block execution point. When there is a required order of operations and multiple connector lines originating from the same design block execution point, the correct order of operations cannot be determined and the automation might fail.



In the above automation example, the connector line labeled 1 (Finance_Yahoo_Com.Company Name.Text -> Earnings Yield UI.Company Name Text Box.Text) must execute before the connector line labeled 2 (Finance_Yahoo_Com.PE Field.Text -> Numeric Expression).

Presenting your solutions

Agile Desktop was designed for Studio developers who create solutions for Runtime end users. It provides a framework for presenting the solutions you design to the end user. This framework ensures a user-friendly and attractive design and one that brands your work appropriately.

Agile Desktop includes these ready-to-go plug-ins which you can use to create a unified user experience:

- Customer 360 View — Presents the most critical client information to the user.
- Automated Notes — Agile Desktop automatically creates notes using text templates populated by pre-defined activities. You can easily add comments about an interaction and review the notes you have already entered and generated.
- Shortcuts — Lets you define and present frequently used shortcuts to your users. These shortcuts can start an executable or a solution or go to a web address. These shortcuts can start an execution, navigate to a web page, start an activity, or execute an automation.

You configure Agile Desktop using the interaction.xml file. During this configuration, you define the information which will be presented to end users and set options specific to each plug-in. For more information, see Using the Interaction.xml File.

When an end user installs Runtime, the Setup wizard installs both the standard Runtime application and Agile Desktop. The wizard also prompts you to decide whether you will use the standard Runtime application or Agile Desktop. For more information, see the applicable Runtime Installation Guide.

For more information, see the Agile Desktop Implementation Guide.

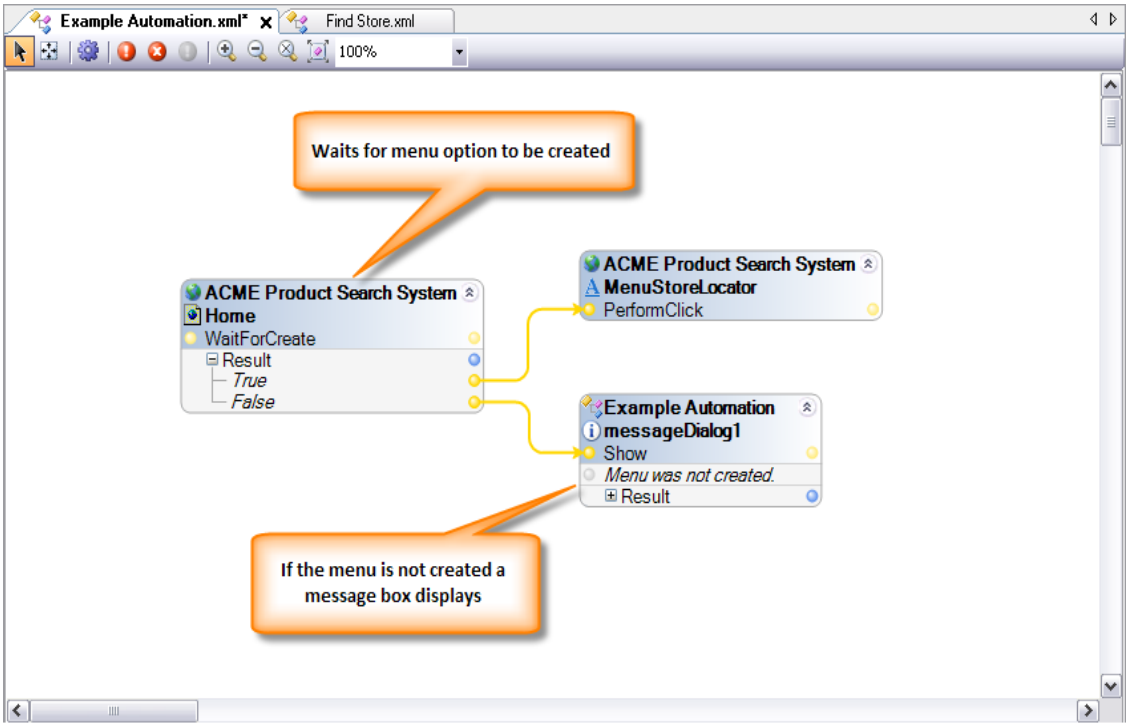
Debugging your solutions

Using message dialogs

Add MessageDialog components to automations when you are building and testing solutions to confirm event handling. Using the MessageDialog component creates and displays a message dialog from a solution.

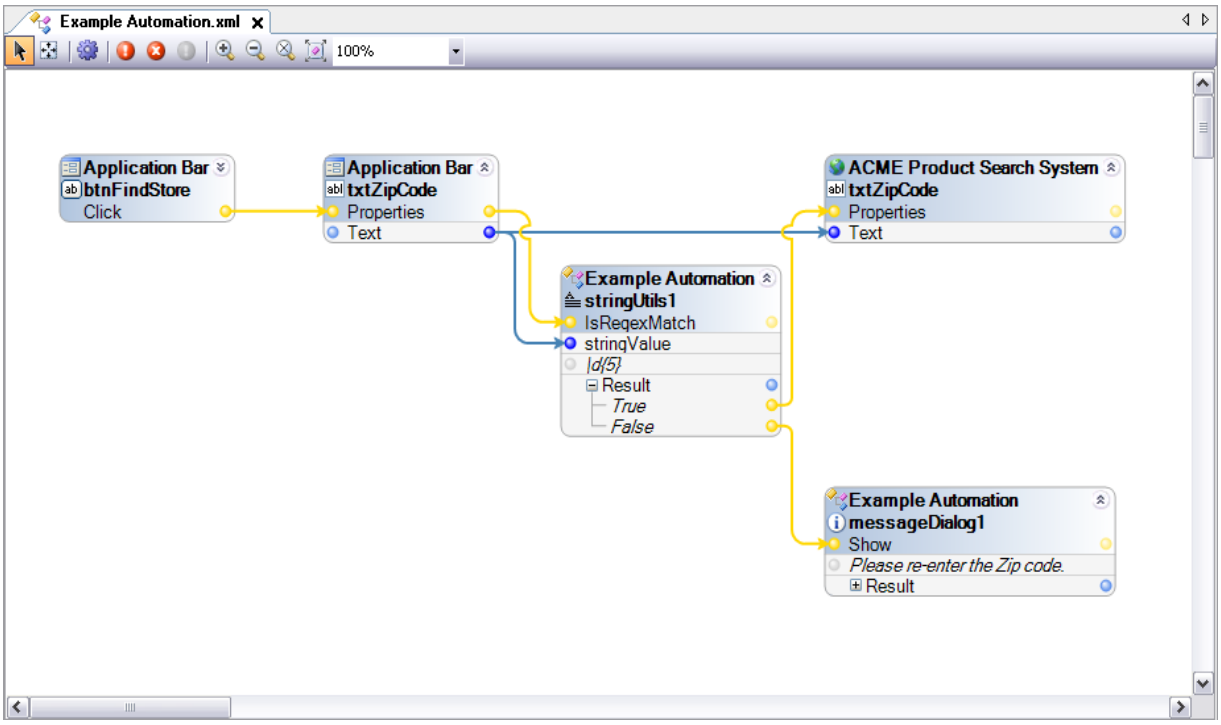
In this example, if the menu is not created, WaitForCreate will return a False result and the message dialog displays this message:

Menu was not created.



Using error trapping around data entry

Most solutions minimize the need for data entry. If, however, your solution requires data input, make sure you provide the logic to make sure the input is valid for use in your solution. For example, the following logic uses the StringUtilities.IsRegexMatch method to determine if an entry contains five (5) digits:



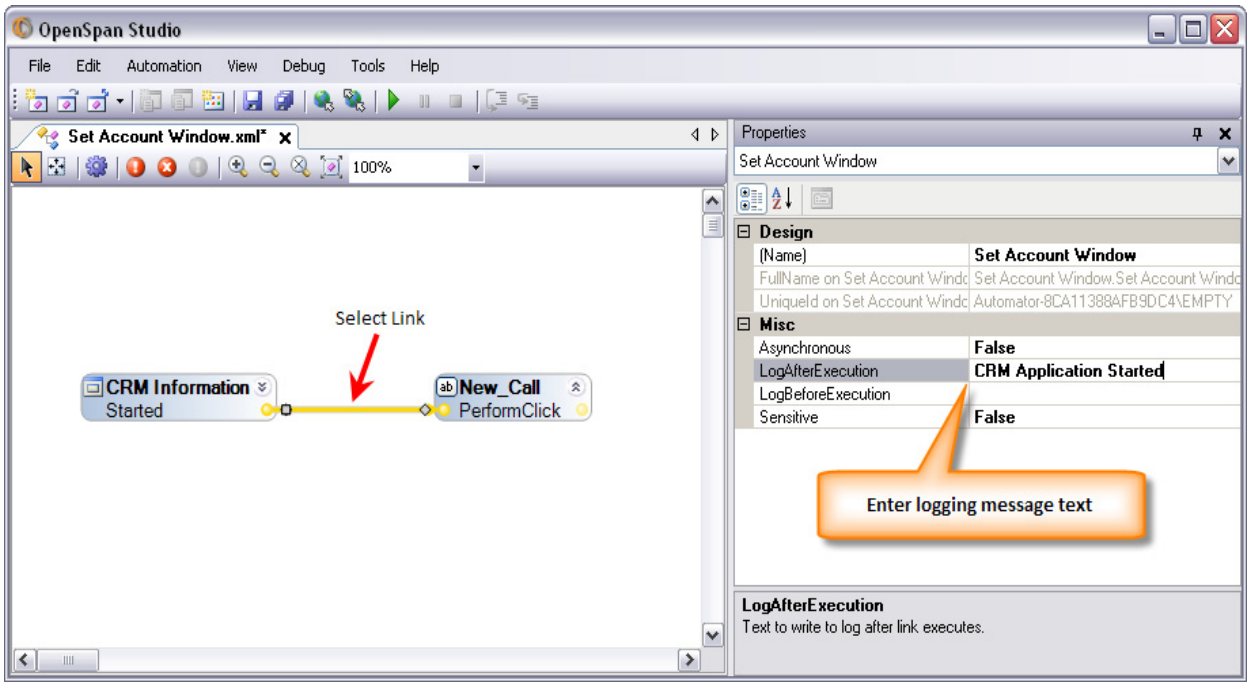
Adding diagnostic logging

The LogBefore and LogAfter execution functions enable you to set messages to be written to the diagnostics file. These messages are written before and after execution of a particular event link or transfer of data across a data link on an automation.

Add the DiagnosticLog component to automations to track solution processing. Create a logging automation that can be called by any automation in the solution.

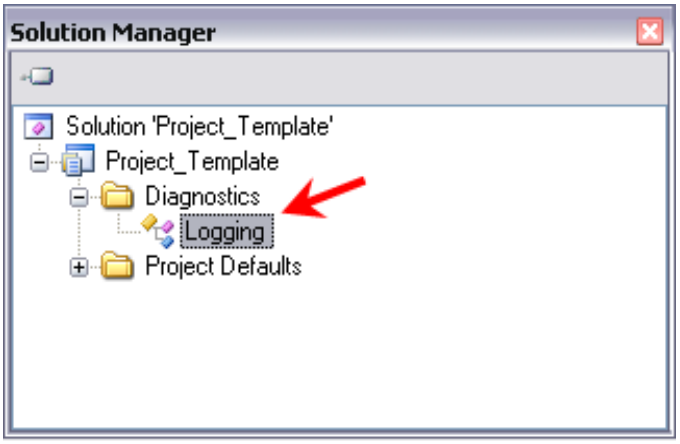
To invoke this type of diagnostics, select the link for which you want to enable messaging, the following properties display in the Properties Grid:

- Log Before Execution
- Log After Execution

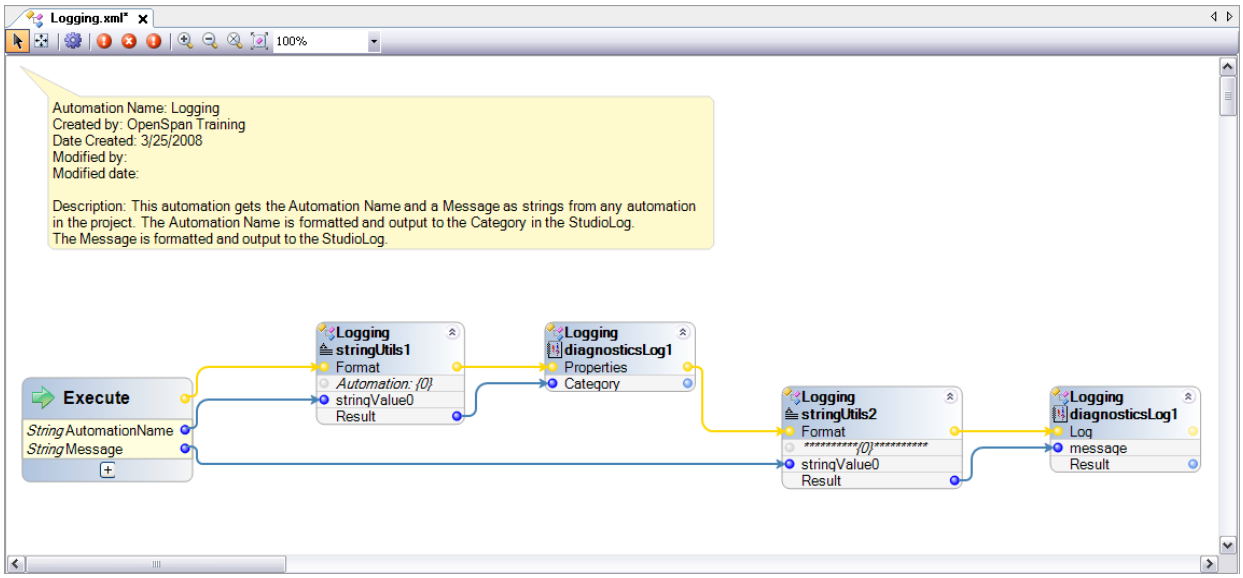


Setting up a universal diagnostic automation

Creating a single diagnostic automation that can be called from anywhere in the project is an efficient way to implement diagnostic logging in a larger, more complex solution.



Here is an example of a diagnostic logging automation that can be called from anywhere in the project and will write with a standard format to the diagnostic log file:



The Logging automation in the Project_Template solution (see“Finding additional information” on page 25) contains this logic.

Deploying your solutions

Using Studio's Deployment Portal

There are three ways to deploy the solutions (packages) you build:

- Deployment Portal
- File share
- Web site

We recommend that you use the Deployment Portal to deploy packages to your various Runtime machines. For more information on deploying solutions, refer to the Deployment Portal User Guide.

Using a common extract directory

By default, Runtime extracts the deployment package to each user's application data folder. It is recommended that a common extract directory is used. Change the Deployment Extract Directory key in the RuntimeConfig.xml file from blank to the designated location.

Here is an example:

```
<add key="DeploymentExtractDirectory" value="C:\OpenSpan_Runtime\" />
```

Finding additional information

Using Pega Robotic Automation Studio Help

To access material within Studio, choose Help > Help Contents. You can also access the Help system via this website:

help.openspan.com/80/

Using the Support Portal

The Pega Support Portal lets you search our extensive Knowledge Base of support articles, how-tos, and documentation. If you can't find the answer here, ask a question in our Product Support Community.

Viewing example solutions

You can also view example solutions to see how to build Studio solutions.