openspan

where people and technology meet®

OpenSpan Best Practices

Building Solutions

01 September 2013

This document includes these topics:

- "Naming and Design Considerations" on page 1-2
- "Interrogating Applications" on page 1-6
- "Using Integration Processes" on page 1-7
- "Using Automations" on page 1-10
- "Debugging" on page 1-17
- "Deployment" on page 1-21
- "More Resources" on page 1-22

Public

©Copyright 2012 - 2014 OpenSpan, Inc. All Rights Reserved.

No part of this publication may be reproduced or distributed in any form or by any means, electronic or otherwise, now known or hereafter developed, including, but not limited to, the Internet, without explicit prior written consent from OpenSpan, Inc. Requests for permission to reproduce or distribute to individuals not employed by OpenSpan any part of, or all of, this publication should be mailed to:

OpenSpan, Inc.
Suite 200
11175 Cicero Drive
Alpharetta, Georgia 30022

Phone (International) +1 (678) 527-5400
Phone (US and Canada) (877) 733-1136
Email: sales@openspan.com

OpenSpan® and Surface Integration® are registered trademarks of OpenSpan Inc., a Delaware Corporation.

Microsoft®, Visual Studio®, MSDN®, and Windows® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

TIBCO Spotfire ® is a registered trademark of TIBCO Software Inc. in the United States and/or other countries.

Public

Contents

1 Contents

2 Naming and Design Considerations

- 2 Designing Solutions Based on User Actions
- 2 Naming Convention for Solutions, Projects, Adapters, Controls and Components
- 2 Naming Solutions and Projects
- 2 Naming Adapters
- 2 Naming Automations
- 3 Naming Controls
- 4 Creating a Solution Manager Hierarchy Based on Tasks
- 5 Saving Backup Deployment Packages of Your Solutions

6 Interrogating Applications

- 6 Interrogating All Applications Before Building Automations
- 6 Testing Applications in the OpenSpan Environment by Unit Testing Solutions
- 6 Following Match Rules Best Practices

7 Using Integration Processes

- 7 Starting Applications as Required
- 8 Setting the UseKeys Property When Required

10 Using Automations

- 10 Adding Comment Text as Header to all Automations
- 10 Keeping Automations with the Task or Adapter
- 11 Setting the Automation ShowDesignCompNames Property to True
- 12 Using WaitForCreate and IsCreated
- 13 Terminating Branched Event Paths
- 14 Using Global Storage for Variables and a Release Notes Automation
- 15 Using a Single Script with Multiple Methods

15 Following Threading Best Practices
15 Running Only One Connector Line from a Design Block Execution Point
17 Debugging
17 Using Message Dialogs
17 Using Error Trapping Around Data Entry
18 Adding Diagnostic Logging
19 Setting up a Universal Diagnostic Automation
21 Deployment
21 Using a Single Deployment Location
21 Using a Common Extract Directory
22 More Resources
22 Using OpenSpan Studio Help

Public

BUILDING SOLUTIONS

The OpenSpan Platform 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 enterprise solutions with OpenSpan Studio.

To use this guide, you should be proficient creating OpenSpan solutions and have completed the OpenSpan Studio Basics Training course.

Note An OpenSpan solution named Project_Template is referenced throughout this document. The Project_Template.OpenSpan deployment file is available in the More Resources section of this document.

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 OpenSpan Integrator menu.

For example, in the CRM Web Solution you created in the OpenSpan Studio Basics Training 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.

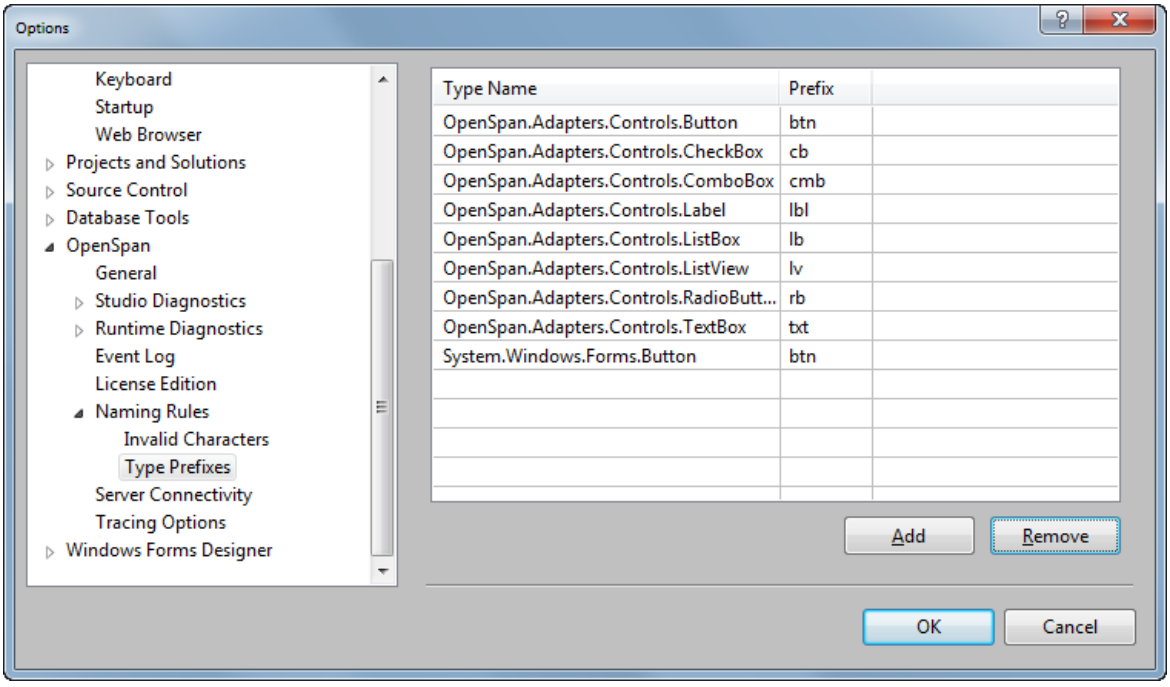
Naming Controls

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.

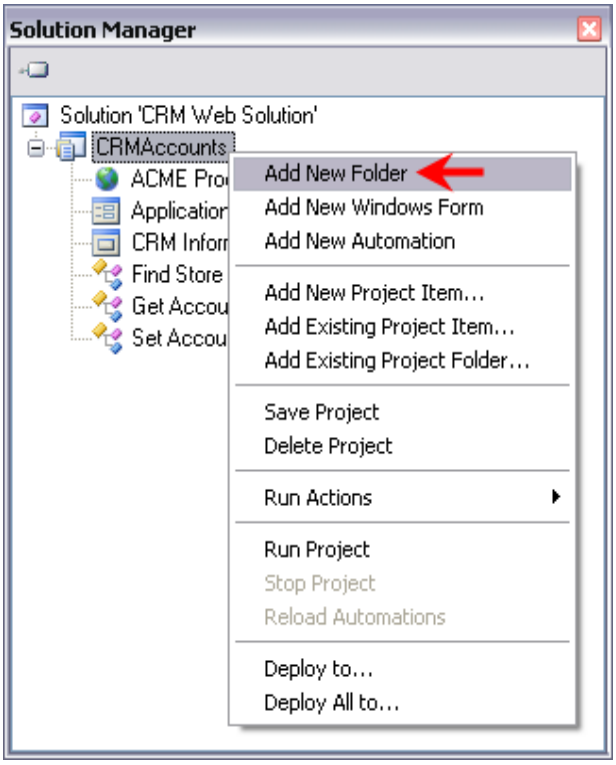
Common Control	Standard Prefix
Button	btn
Check Box	cbx
Combo Box	cmb
Label	lbl
List Box	lbx
List View	lv
Radio Button	rb
Text Box	txt

You can modify the default prefixes used by OpenSpan 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:

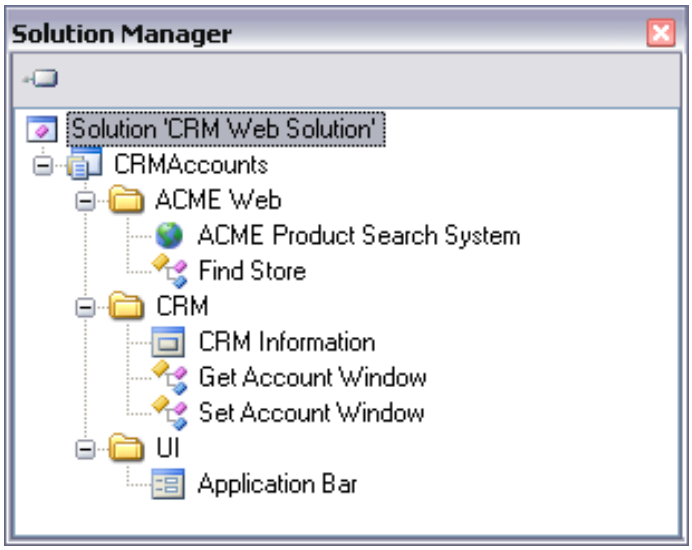


Creating a Solution Manager Hierarchy Based on Tasks

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

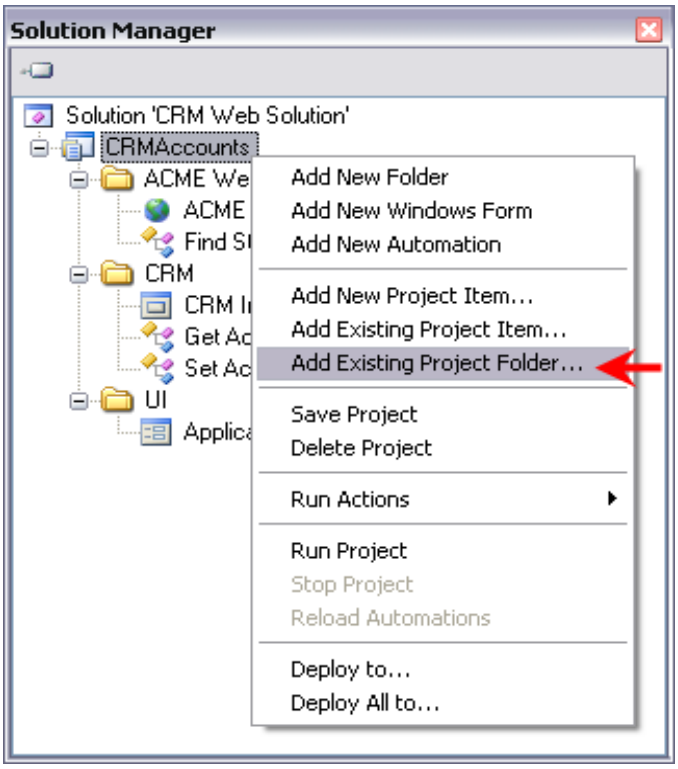


For example, using the CRM Web solution from the OpenSpan Studio Basics Training course, you can create folders to store common CRM, ACME, and User Interface (UI) elements. This lets you easily locate objects when working with the solution. See the following example:



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 OpenSpan Environment by Unit Testing Solutions

For complex applications, create a solution and add a WinForm and the application. Interrogate all areas of the application that you want to use in your final solution. Add controls to the WinForm to unit test interaction with the application. When you are satisfied that the application runs properly in the OpenSpan environment, add the adapter by adding an Existing Project Item to the final solution.

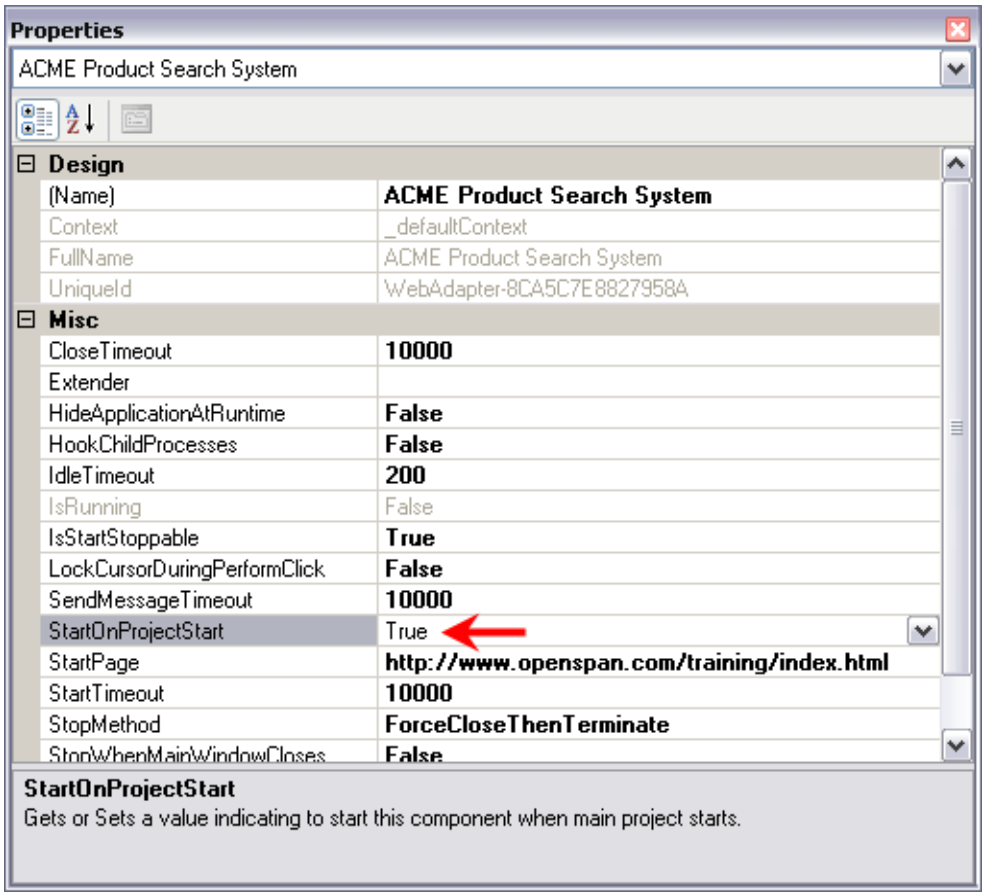
Following Match Rules Best Practices

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

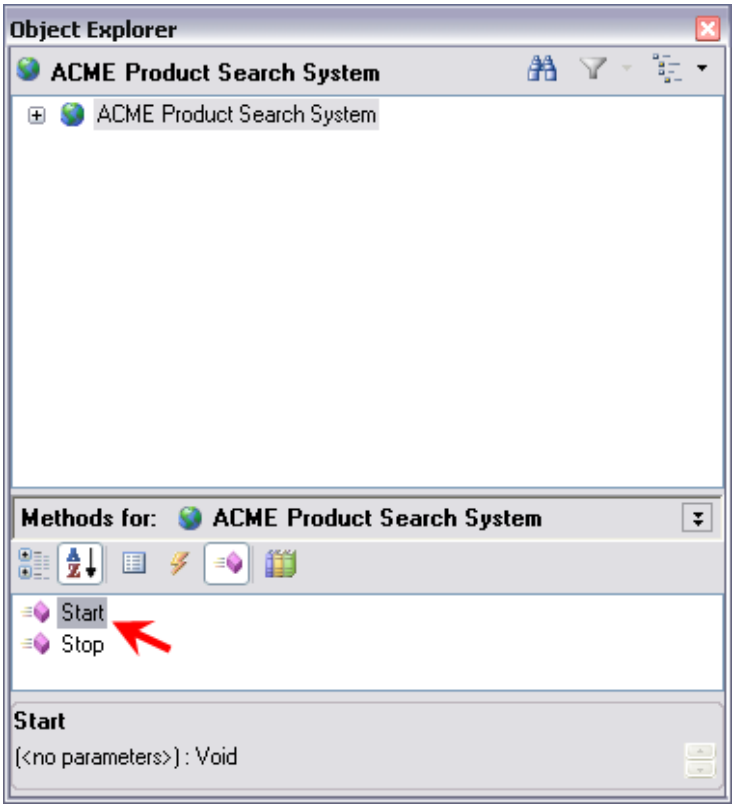
Using Integration Processes

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.



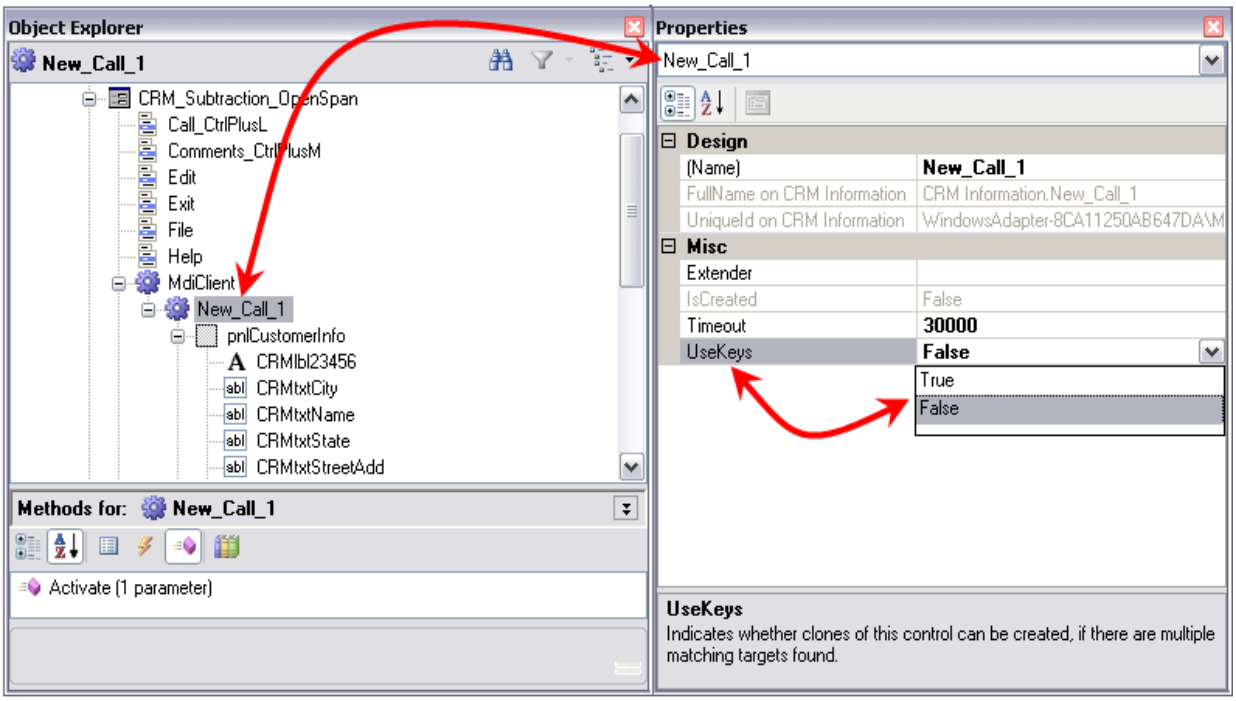
Use the Start method in an automation for the adapter to start an application as needed by the project execution. By controlling the number of applications running during the execution of your project, you improve the overall performance of the project.



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 OpenSpan Studio Basics Training where multiple instances (clones) of the New Call (New_Call_1) window are created.



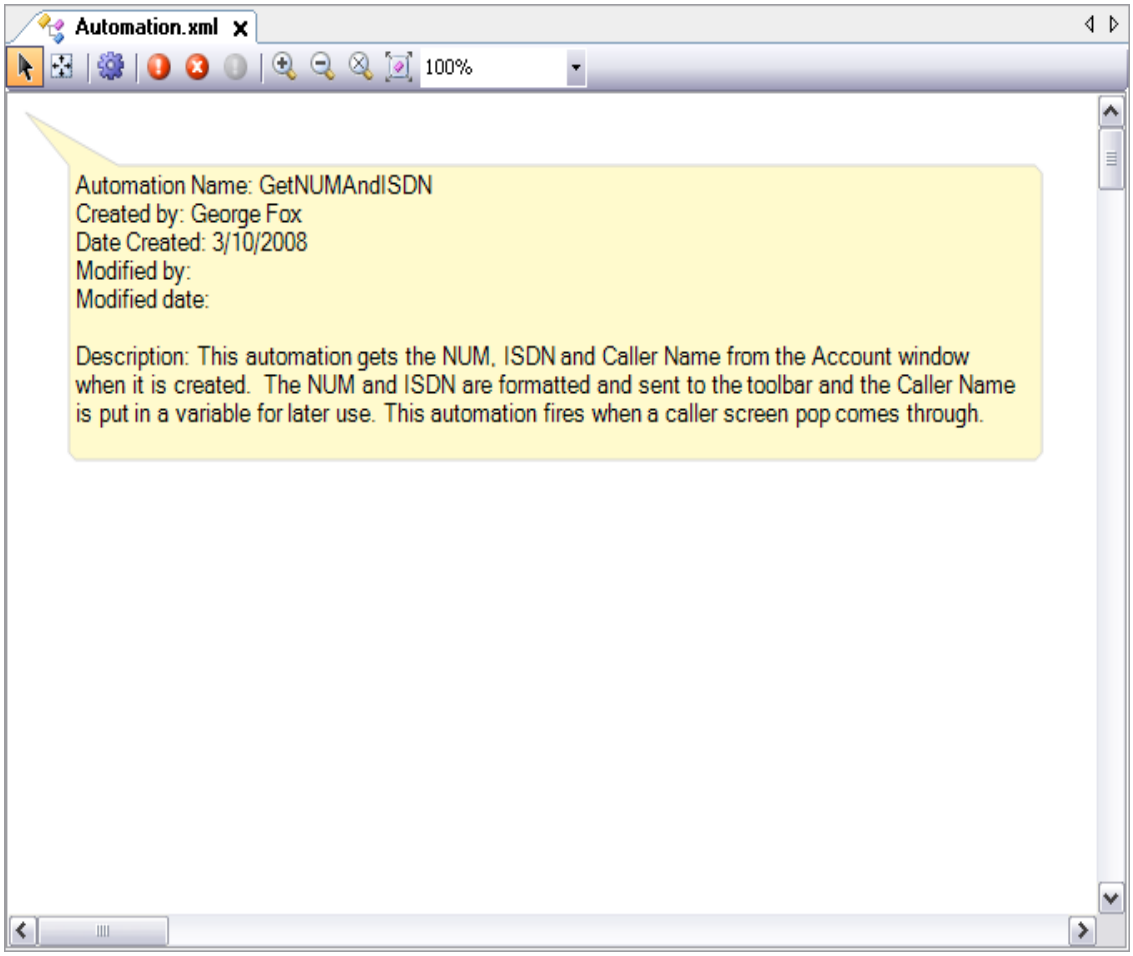
By default, this property is False, indicating that keys are not assigned to the object. However, if multiple copies of the object exist at runtime, and your solution needs to distinguish between the various instances of the object, then set the property to True.

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

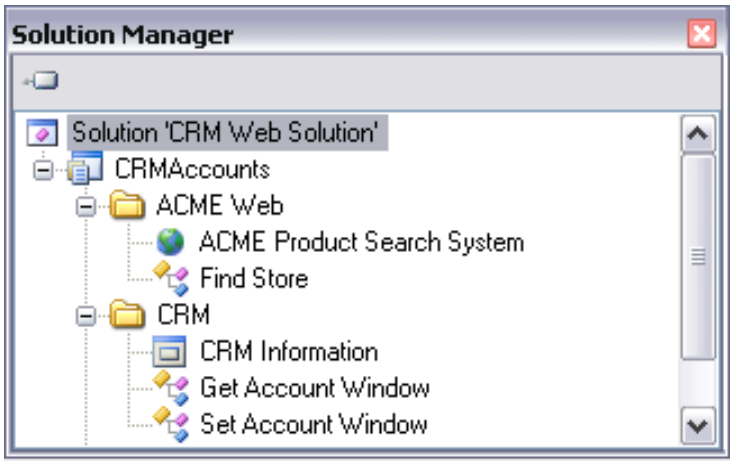
Comments within an automation can be extremely helpful when working in multi-designer environments. It will enable a designer to look at an automation and quickly discern the logic and flow or the process. For example, a standard Automation header comment such as the following can be used:



Keeping Automations with the Task or Adapter

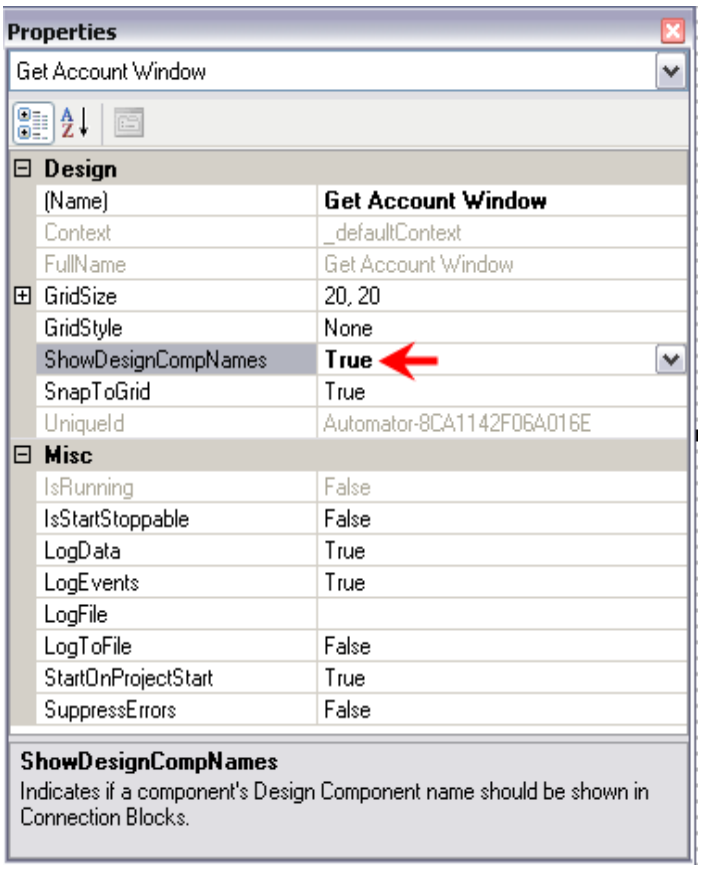
In a complex solution, you are likely have many automations. Keeping those automations with the task or adapter makes it much easier to locate when you debug or edit your solution.

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

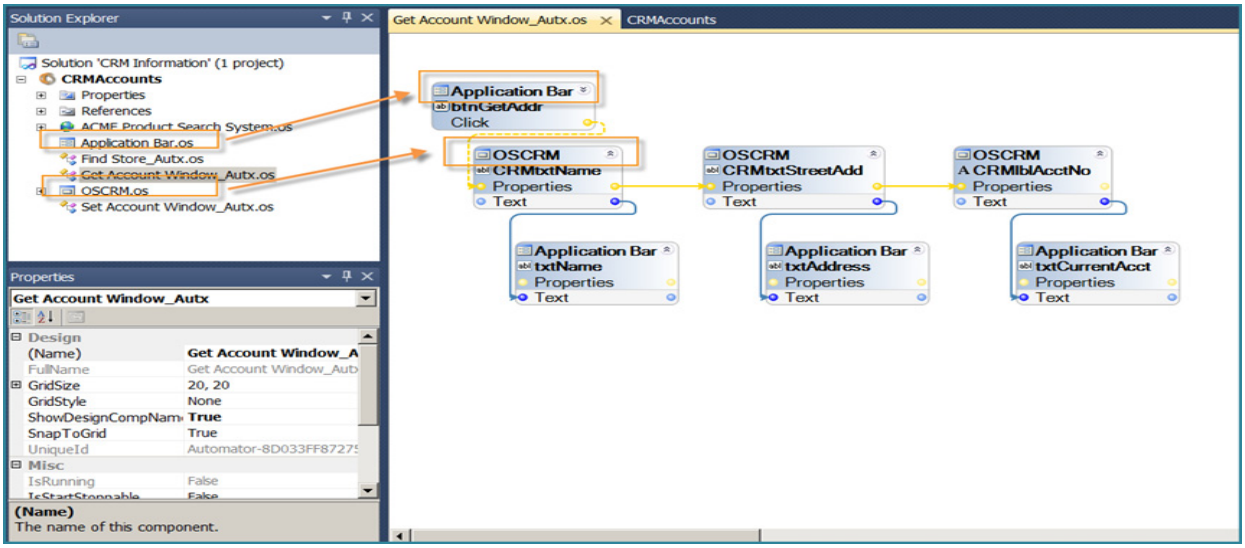


Setting the Automation ShowDesignCompNames Property to True

By setting the Automation property ShowDesignCompNames to True, OpenSpan 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.

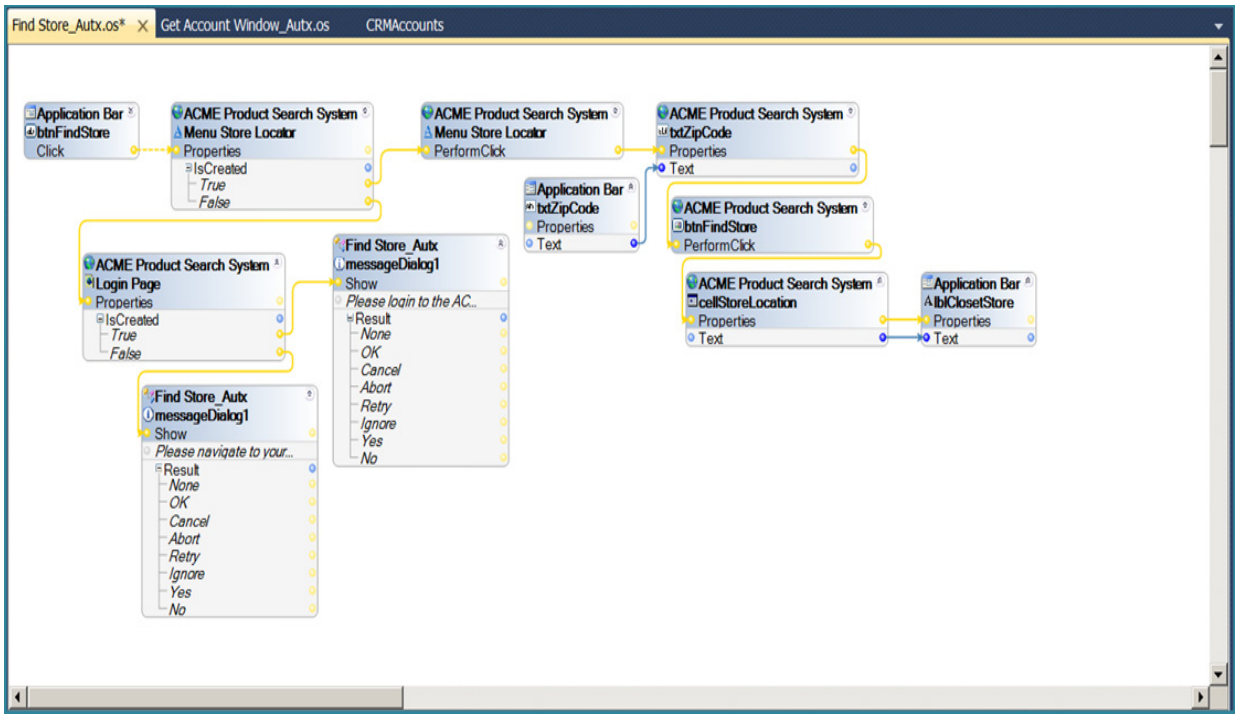
Using WaitForCreate and IsCreated

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:

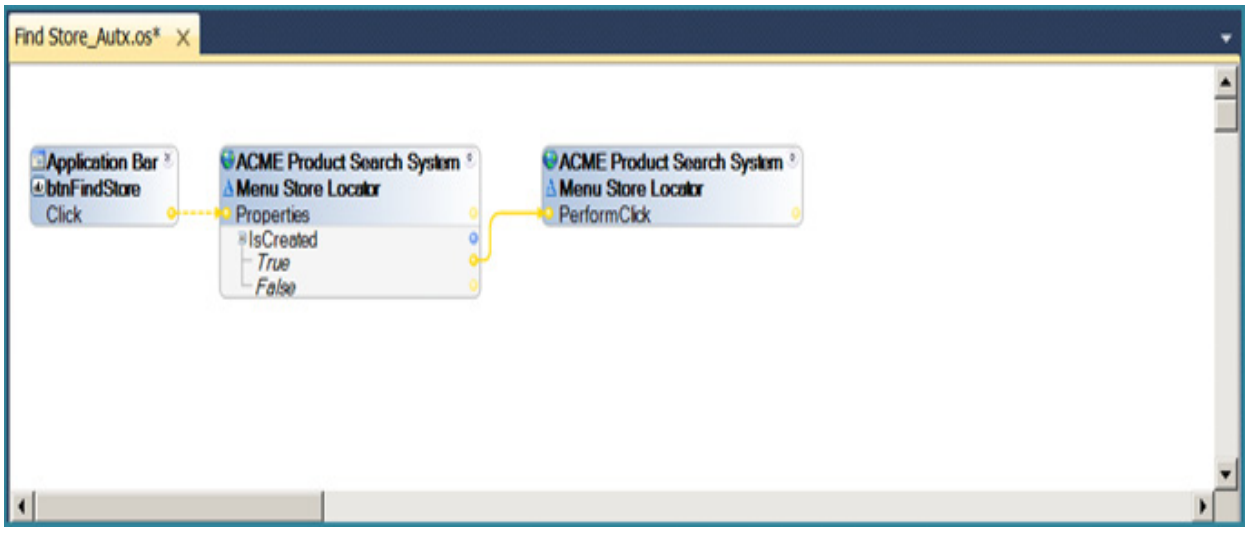
- **WaitForCreate Method -** waits for an object to be created. Returns True if the object is matched before the WaitTimeout period elapses. Otherwise, it returns False.
- **IsCreated Property -** returns True if the object is matched at a specific instance in time. Otherwise, returns False.

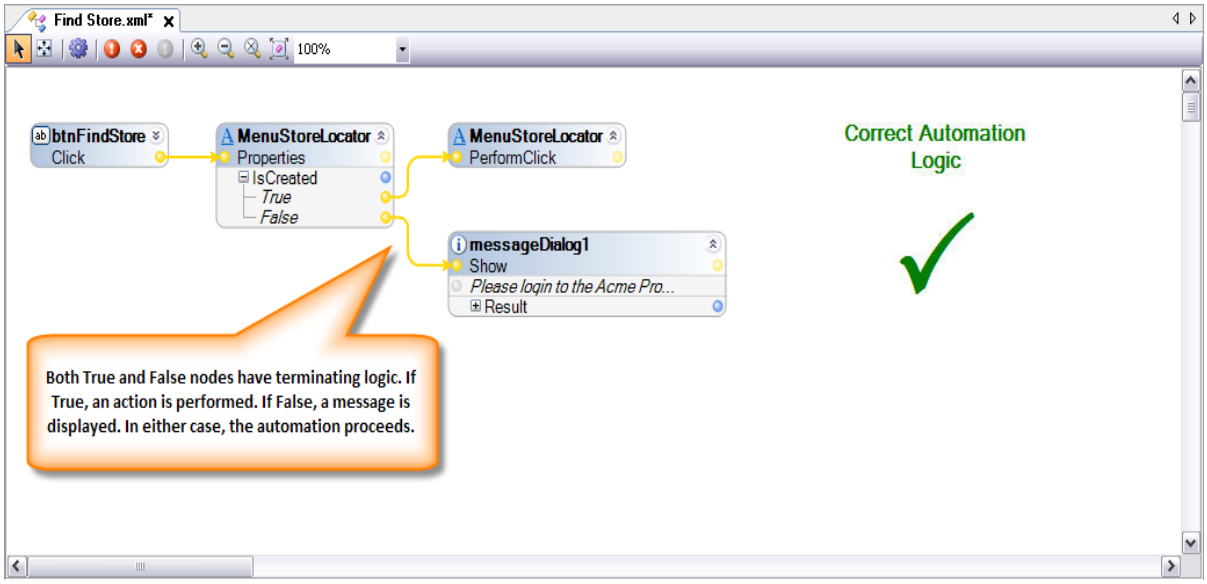
The following example of the Find Store automation from the CRM web solution in the OpenSpan Studio Basics Training course shows how the IsCreated property is used to ensure that certain web objects are created (matched) before being used by the automation logic.



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:



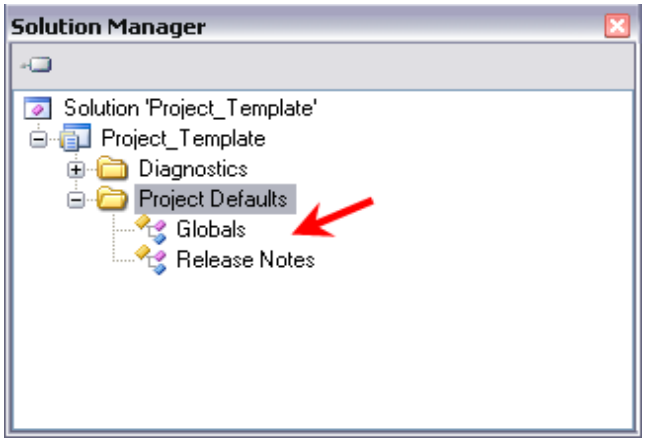


In the example shown above, a message dialog was used to terminate the False event path. If you do not want to alert the user or take any action based on the event, try using the DiagnosticLog component to capture the event in the Studio Log. See “Adding Diagnostic Logging” on page 18 for an example of logging automation that can be called from any automation in a solution.

Using Global Storage for Variables and a Release Notes Automation

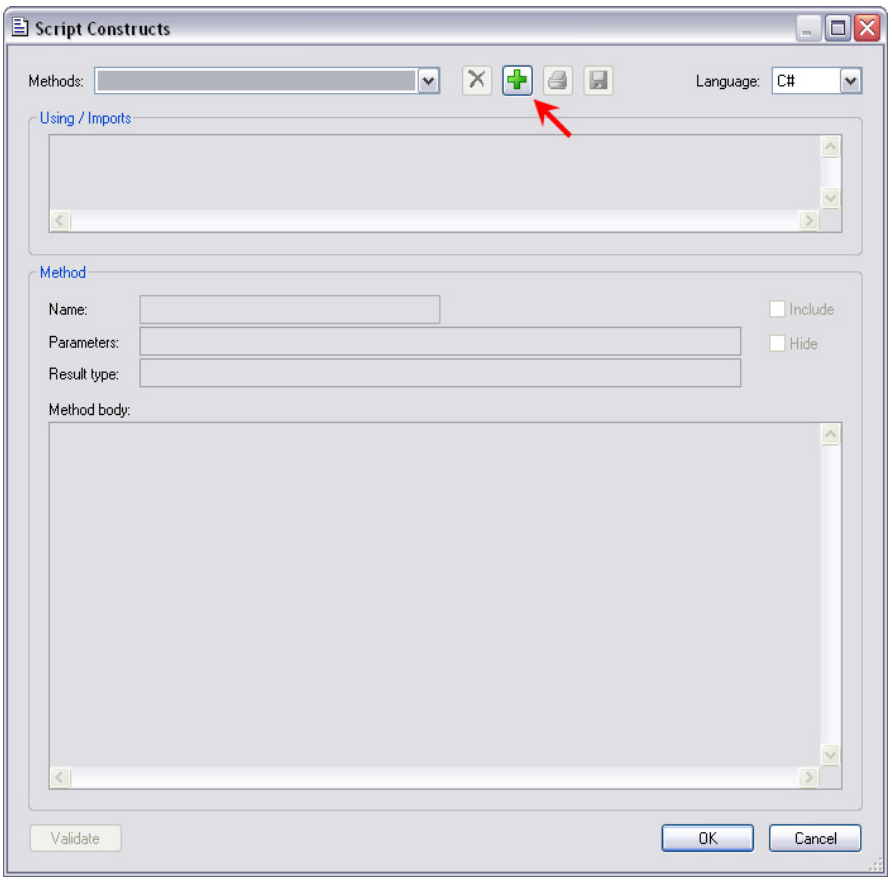
It is a good organizational practice to set up global storage for variables or components that can be called from multiple locations within the solution.

Another good practice is to include an automation for storing notes and tracking any revisions to the solution. The Project_Template solution available in “More Resources” on page 22 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.

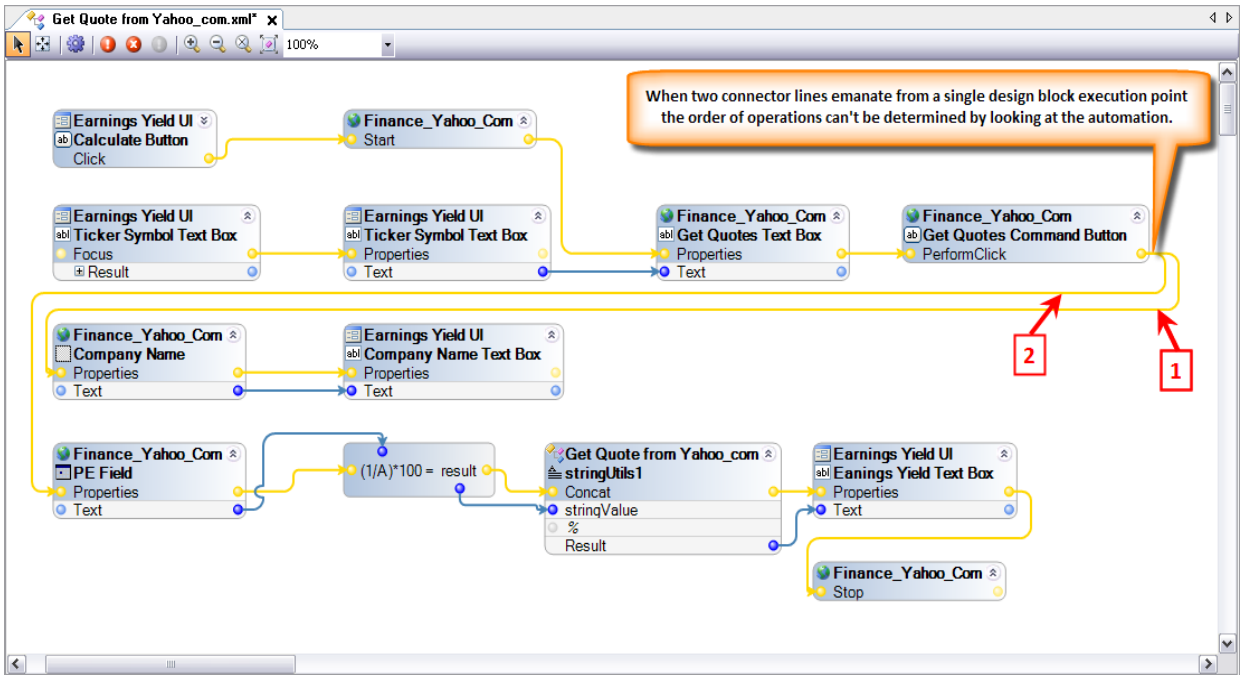


Following Threading Best Practices

Design your automations following the guidelines set forth in the OpenSpan *Best Practice: Threading* document. This document provides details on threading in the OpenSpan 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).

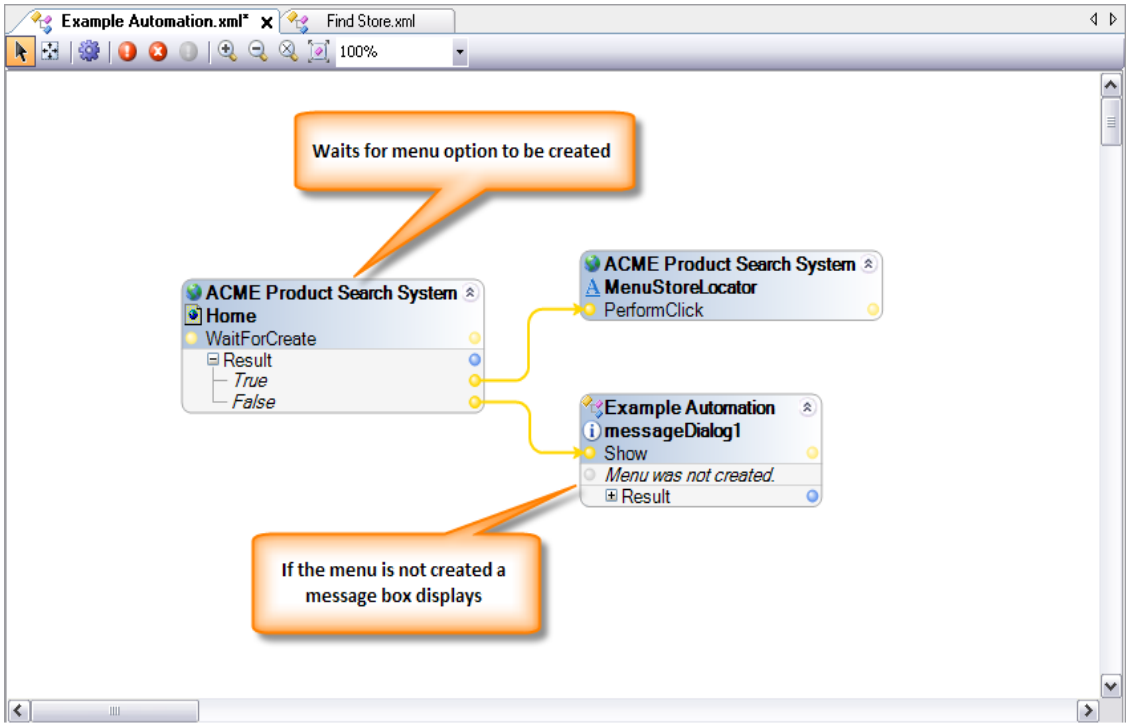
Debugging

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 box from a solution.

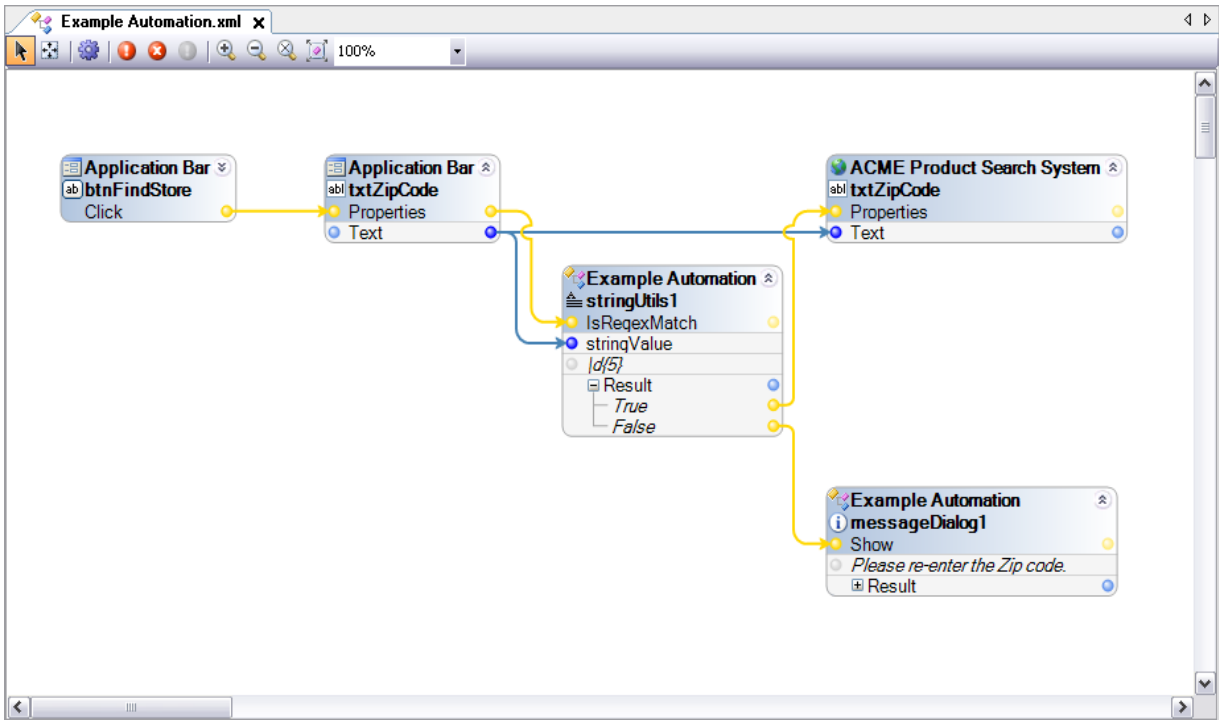
In this example, if the menu is not created, WaitForCreate will return a False result and the message dialog displays this message:

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 5 digits:



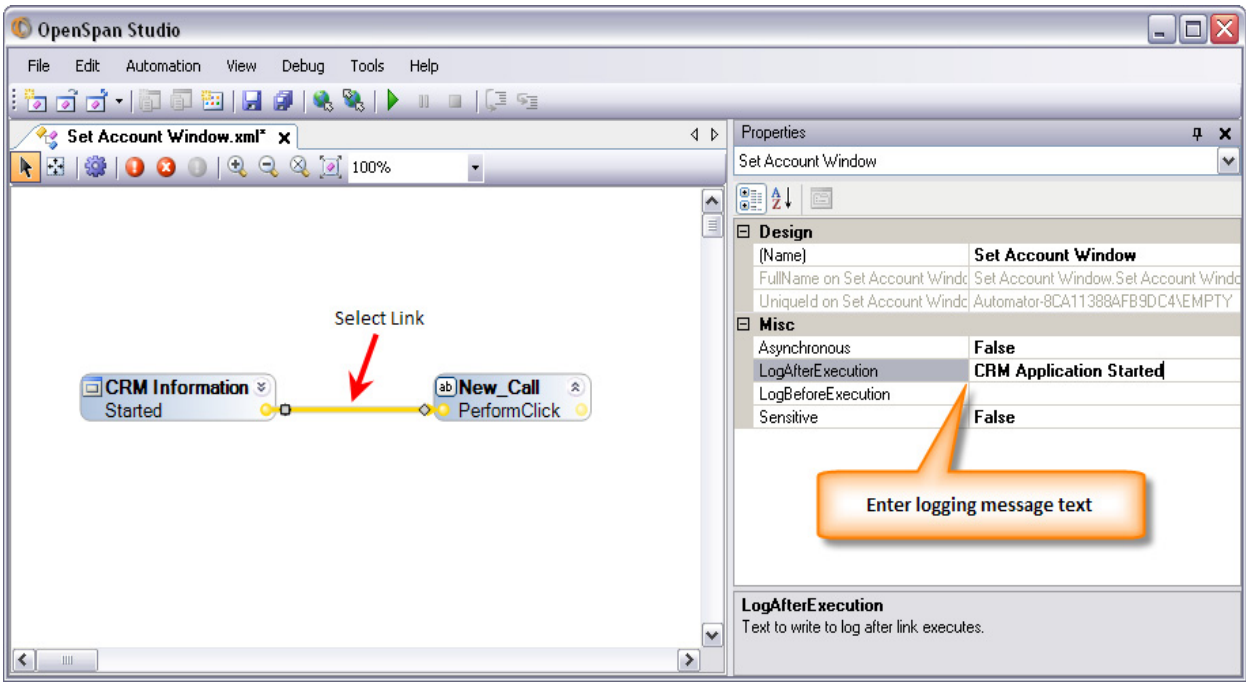
Adding Diagnostic Logging

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.

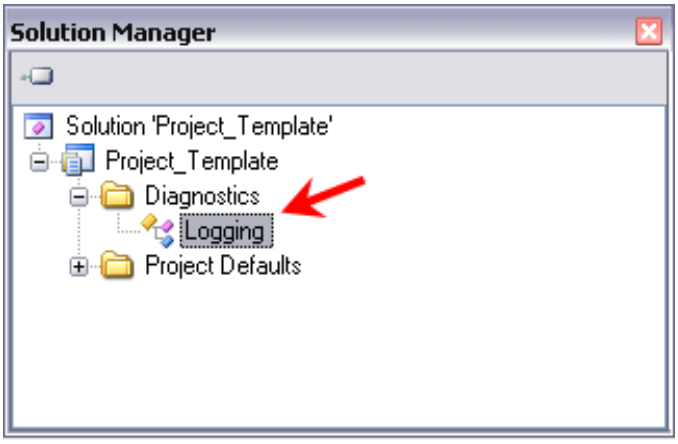
To invoke this type of diagnostics, select the link for which you want to enable messaging, the following properties display in the Properties Grid:

- Log Before Execution
- Log After Execution

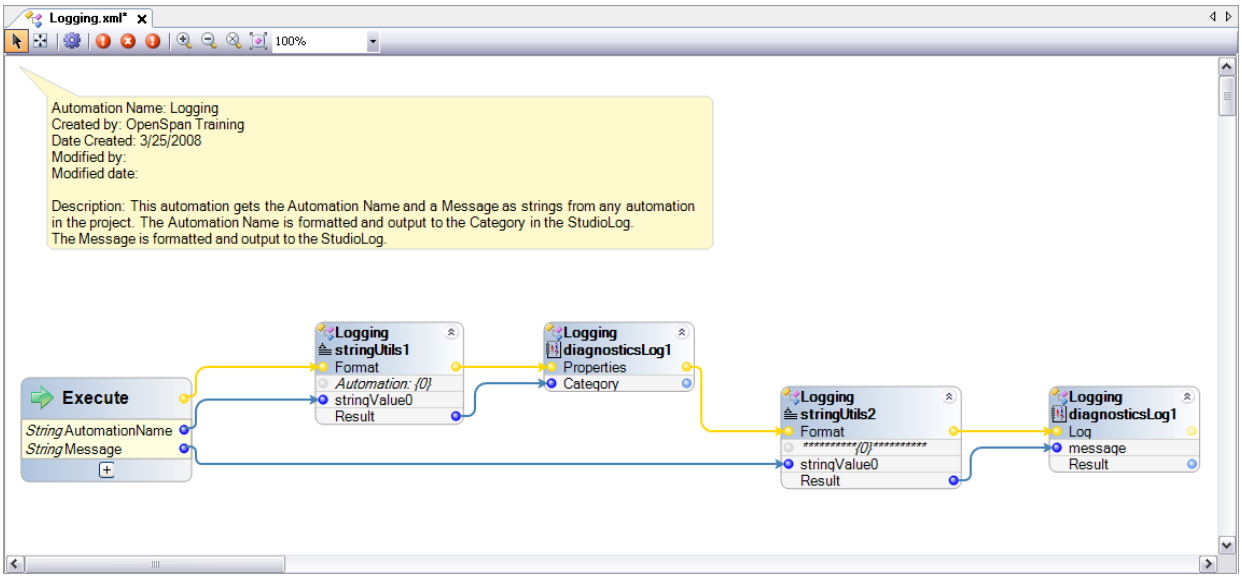


Setting up a Universal Diagnostic Automation

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



This 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“More Resources” on page 22) contains this logic.

Deployment

Using a Single Deployment Location

Setup a single location from which all OpenSpan deployment packages are deployed through OpenSpan Integrator. Options include a fileshare or web site. For details on deploying solutions, refer to the OpenSpan Integrator and Deployment training document.

Using a Common Extract Directory

By default, OpenSpan Integrator 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 IntegratorConfig.xml file from blank to the designated location.

Here is an example:

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

More Resources

Using OpenSpan Studio Help

To access material within OpenSpan Studio, choose Help | Help Contents.

Or, you can download the following solution to see the OpenSpan deployment files containing the sample automations used in this document.

Note This solution only works with OpenSpan Studio version 3.2 or newer.

Project_Template.OpenSpan