PEGA 7 PLATFORM

Pega Robotic Automation Architect Essentials

8.0

Student Guide

Pega®

ACADEMY

Trademarks

For Pegasystems Inc. trademarks and registered trademarks, all rights reserved. All other trademarks or service marks are property of their respective holders.

For information about the third-party software that is delivered with the product, refer to the third-party license file on your installation media that is specific to your release.

Notices

This publication describes and/or represents products and services of Pegasystems Inc. It may contain trade secrets and proprietary information that are protected by various federal, state, and international laws, and distributed under licenses restricting their use, copying, modification, distribution, or transmittal in any form without prior written authorization of Pegasystems Inc.

This publication is current as of the date of publication only. Changes to the publication may be made from time to time at the discretion of Pegasystems Inc. This publication remains the property of Pegasystems Inc. and must be returned to it upon request. This publication does not imply any commitment to offer or deliver the products or services described herein.

This publication may include references to Pegasystems Inc. product features that have not been licensed by you or your company. If you have questions about whether a particular capability is included in your installation, please consult your Pegasystems Inc. services consultant.

Although Pegasystems Inc. strives for accuracy in its publications, any publication may contain inaccuracies or typographical errors, as well as technical inaccuracies. Pegasystems Inc. shall not be liable for technical or editorial errors or omissions contained herein. Pegasystems Inc. may make improvements and/or changes to the publication at any time without notice.

Any references in this publication to non-Pegasystems websites are provided for convenience only and do not serve as an endorsement of these websites. The materials at these websites are not part of the material for Pegasystems products, and use of those websites is at your own risk.

Information concerning non-Pegasystems products was obtained from the suppliers of those products, their publications, or other publicly available sources. Address questions about non-Pegasystems products to the suppliers of those products.

This publication may contain examples used in daily business operations that include the names of people, companies, products, and other third-party publications. Such examples are fictitious and any similarity to the names or other data used by an actual business enterprise or individual is coincidental.

This document is the property of:

Pegasystems Inc.
One Rogers Street
Cambridge, MA 02142-1209
USA
Phone: 617-374-9600
Fax: (617) 374-9620
www.pega.com

DOCUMENT: Pega Robotic Automation Architect Essentials Student Guide
SOFTWARE VERSION: Pega 8.0
UPDATED: 10.16.2017

CONTENTS

COURSE INTRODUCTION

This lesson group includes the following lessons:

- Before you begin
- Completing the exercises

Before you begin

Pega Robotic Automation Architect Essentials Overview

In this course, you learn the basic functionality, process flow, terminology, interrogation, recommended best practices, and core building blocks of Pega Robotic Automation Studio to prepare architects to develop their first robotic project.

Objectives

After completing this course, you should be able to:

- Apply programming experience to develop a multi-project solution
- Create a standard Windows form using .NET window controls
- Integrate and interrogate a Windows application within the solution
- Integrate and interrogate a Web application within the solution
- Create a unified solution using project-to-project references
- Employ the use of Interaction Framework within the unified solution to seamlessly maintain and transfer data and activities across the solution
- Create a project and solution hierarchy structure to easily maintain and scale the unified solution
- Create automations to streamline the process by integrating the multi-project solution through programming logic
- Add and edit match rules on interrogated objects to create an understanding of the matching process for both Window and Web applications
- Generate build files and deployment packages of the unified solution
- Utilize the standard Visual Studio diagnostic and debugging tools to test the unified solution

Intended audience

This course is for software developers working on their first Pega Robotic Automation solution, or looking to update their skills on solution development projects. This course is also helpful to those who supervise business analysts or are responsible for business analysis activities.

Prerequisites

To succeed in this course, students should have:

- .NET and Java programming experience
- Knowledge of the business processes and policies in use at their company

- A basic understanding of business application development
- Exposure to Microsoft Visual Studio or another integrated development environment

WINDOWS INTEGRATION

This lesson group includes the following lessons:

- Getting started with Pega Robotic Automation Studio
- Developing solutions and projects
- Working with windows adapters

- Working with automations
- Working with multiple app instances

Getting started with Pega Robotic Automation Studio

Introduction to Pega Robotic Automation

You use Pega Robotic Automation Studio to automate manual processes, modify user interfaces, and add procedural and process guidance to workflows. This lesson provides instruction on common Studio terminology, understanding the Integrated Development Environment (IDE), organizing the workspace, and best practices for development.

After this lesson, you should be able to:

- Define the common terms of Pega Robotic Automation Studio
- Describe the use of the major components of the Integrated Development Environment
- Organize the workspace of the IDE

Overview of Pega Robotics

Pega Robotic Automation Studio provides a unique, intuitive, rapid visual development environment to create run-time solutions that enable end users to complete tasks more efficiently.

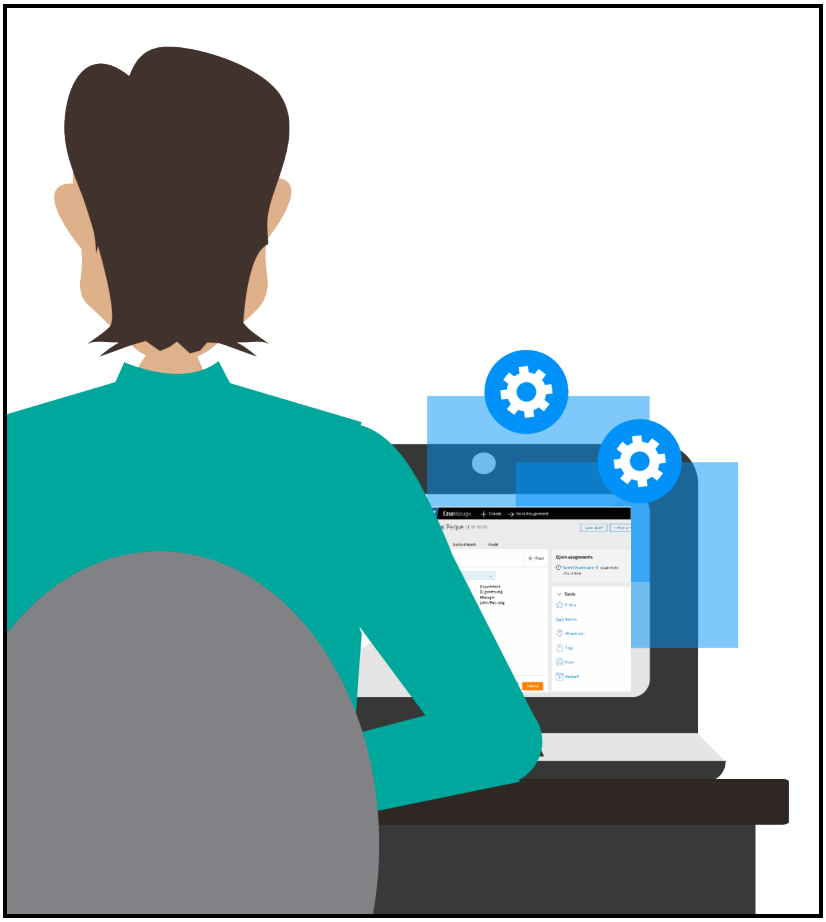
For example, you can create solutions that do the following things:

- Automate manual processes such as copying and pasting between applications
- Provide a single user interface for interaction with multiple applications
- Display process guidance tips to assist users in completion of workflow tasks
- Invoke a web service

The Pega Robotic Automation Studio application provides base adapters for integrating desktop applications without modifying the underlying application software. Using the base application adapters, you can select specific application controls such as text-entry controls and web links. You can then use the properties, methods, and events of the controls to design application automations and event monitoring solutions.

Pega Robotics provides two methodologies — **Robotic Desktop Automation (RDA)** and **Robotic Process Automation (RPA)** — using Microsoft Visual Studio as its development platform.

RDA deploys productivity robots to a user's desktop, providing a unified desktop experience by increasing productivity through task automations. RDA is typically deployed for front office and back office areas in call centers, retail, and branches.



RPA uses unattended robots replicating 100 percent of algorithmic work for back office, operations, and repetitive offload front office work. RPA services legacy applications and Pega workflows, and is typically deployed from a virtual server farm.

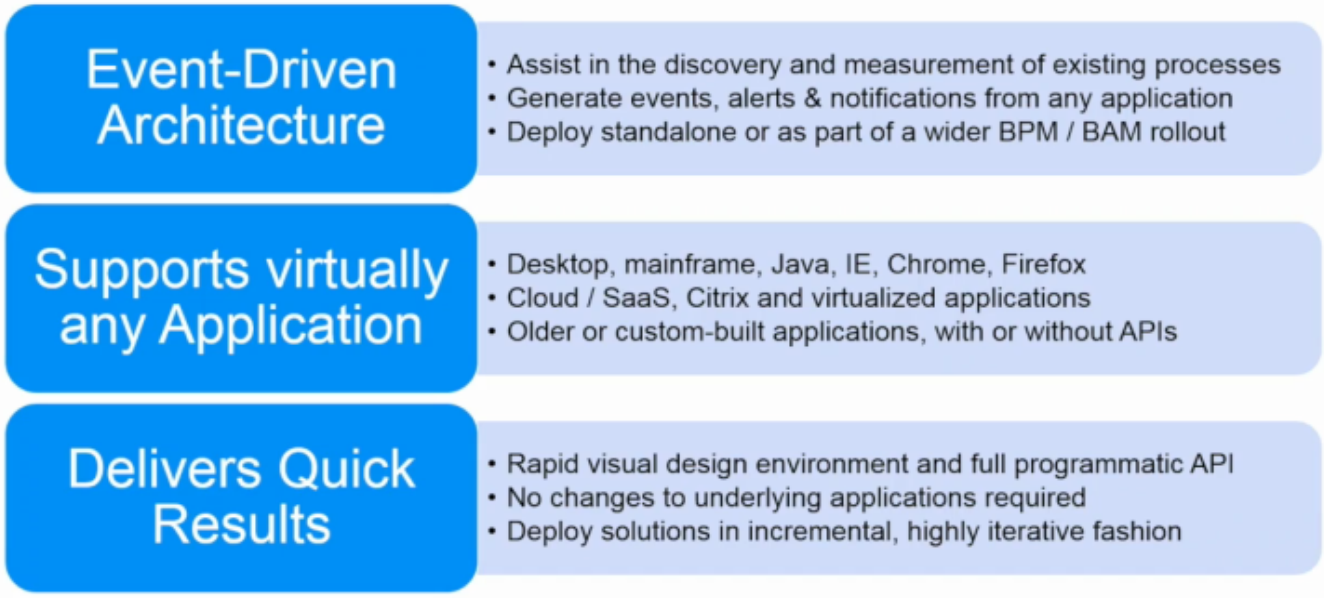
Implementing Pega Robotic Automations into the daily workflow helps:

- Reduce risk
- Increase compliance
- Reduce errors and rework
- Reduce redundancy and variability
- Reduce need for training and retraining

Pega Robotic Automation helps bridge technical gaps. For many legacy applications, APIs or services do not exist; any existing API or service integration is either too slow or expensive to see a large return on investment. Clients complete solutions through outsourcing, documentation, and more training.

Pega Robotic Automation works through three concepts:

How Does Pega Robotic Automation Work?



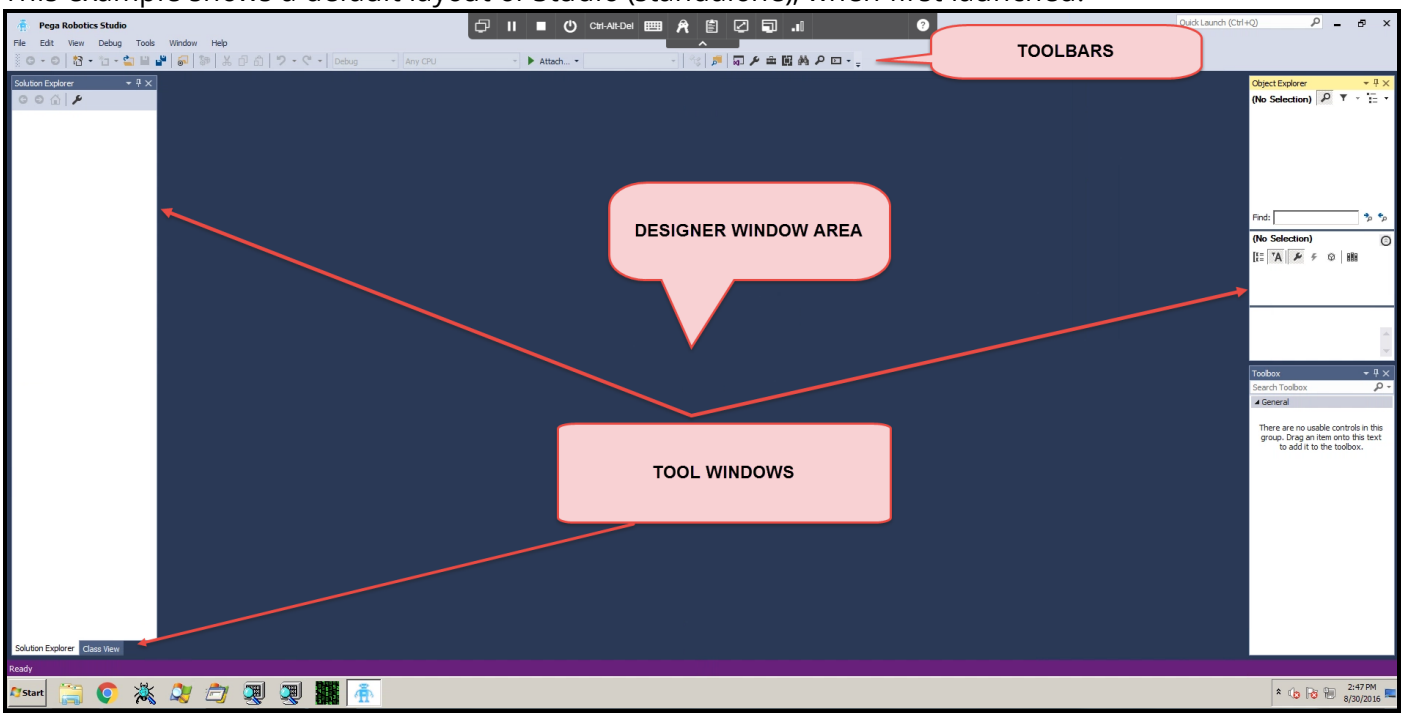
Studio IDE basics

Pega Robotic Automation Studio works within the Microsoft Visual Studio design environment. With either the standalone or plug-in version, the design experience provides streamline access to all the tools required for project design, debugging, and deployment.

Studio consists of these interface elements:

- Menu toolbar
- Standard toolbar
- Automation toolbar
- Tool windows
- Designer windows

This example shows a default layout of Studio (standalone), when first launched.



Toolbars

Toolbar buttons contain commonly used Pega Robotic Automation Studio functions. The toolbars are:

- Standard toolbar
- Automation toolbar

Descriptions of the Studio specific functions available from these toolbars follow. For information on the standard Visual Studio functions, refer to the Microsoft MSDN web site.

Standard toolbar

Here is an example of the Standard toolbar:

The Standard toolbar contains standard functions (File Open, Save, Cut, Copy, and Paste) as well as these Studio-specific functions:

Icon	Description
	Deploy a project
	Undo/Redo an action

Automation Toolbar

Here is an example of the Automation toolbar:

The Automation toolbar contains buttons for performing automation view and layout actions, such as grouping automation lines, adding a Try/Catch debugging function, and modifying the viewing size of the automation.

Tool windows

Studio's Integrated Development Environment (IDE) provides tool windows and document windows. The Tool windows are listed on the View menu and are standard within Visual Studio with the exception of Object Explorer. Tool windows provide access to the solution, project, and project items and related features.

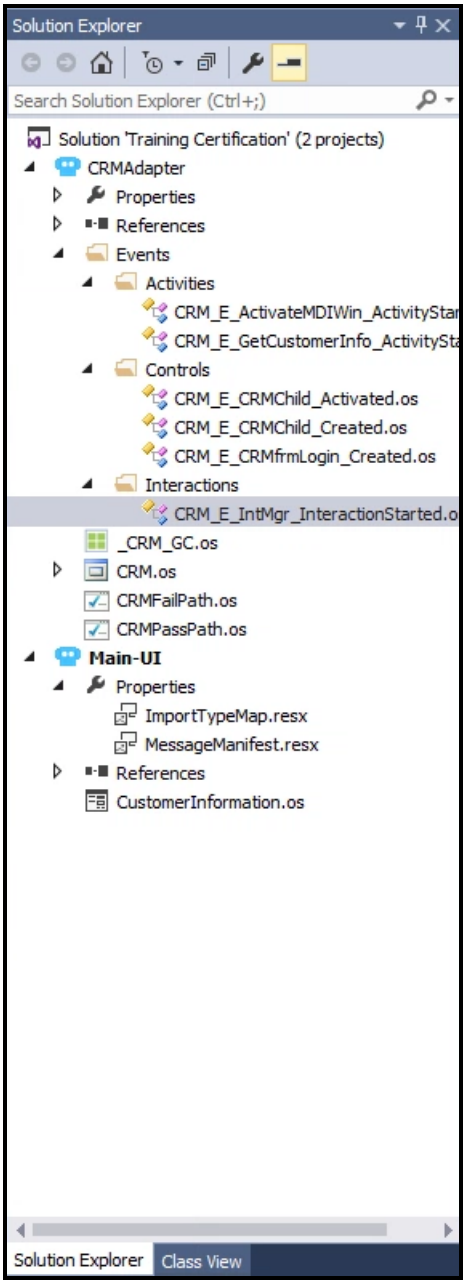
The Tool windows include:

- Solution Explorer
- Properties
- Object Explorer
- Toolbox
- Navigator
- Debugging windows

Document windows are dynamically created when you open a project item. For example, when you open an adapter, the Designer window for the specific adapter type appears. The Document windows contain functionality specific to the project item type.

Descriptions of the Pega Robotic Studio-specific features of these windows follow.

Solution Explorer



Solution Explorer lists the solution and project along with all of the project items added to the project such as:

- Adapters
- Automations
- Window forms
- Folders
- Global containers
- Configuration Project items

Solution Explorer also contains the properties and references used by the project. See the MSDN site for a description of Solution Explorer in Visual Studio.

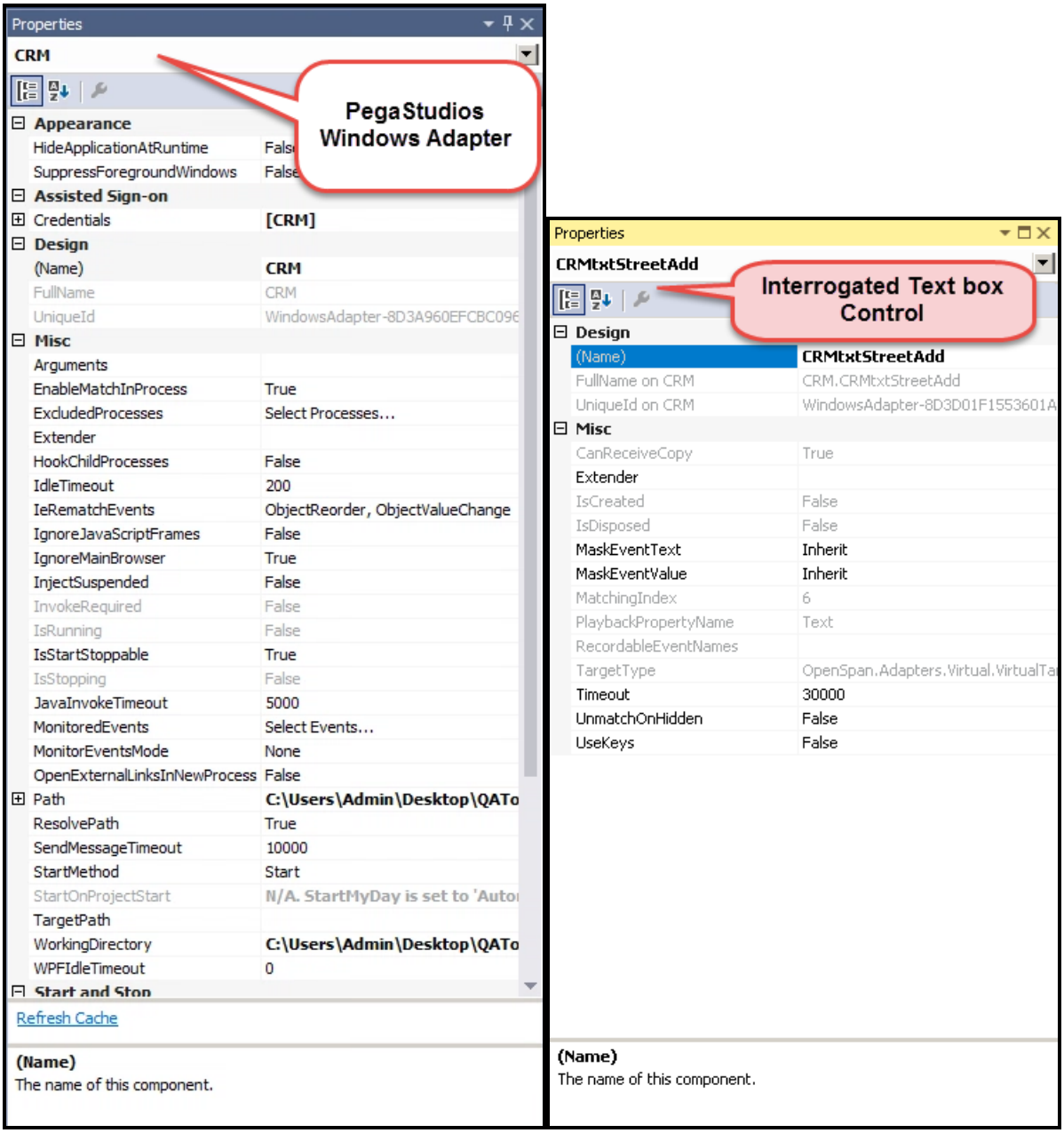
From Solution Explorer you can select project items for editing in the Designer window, add project items, and build the project.

You can right-click an item in Solution Explorer to display a menu for the selected item. The menu lists available functions for the item you selected in Solution Explorer.

Properties window

The Properties window displays design properties for project items (adapters, automations, and contexts) and controls. The Properties window displays different types of editing fields depending on the selected project item. These edit fields include edit boxes, drop-down lists, and links to custom editor dialogs.

Properties shown in gray are read-only. The drop-down list at the top of the Properties window shows the name of the selected object. Use the buttons to list the properties either categorically or alphabetically. In some cases, links to project item functions appear in the Properties window. Here are some examples of the Properties window for a few typical objects.



Object Explorer

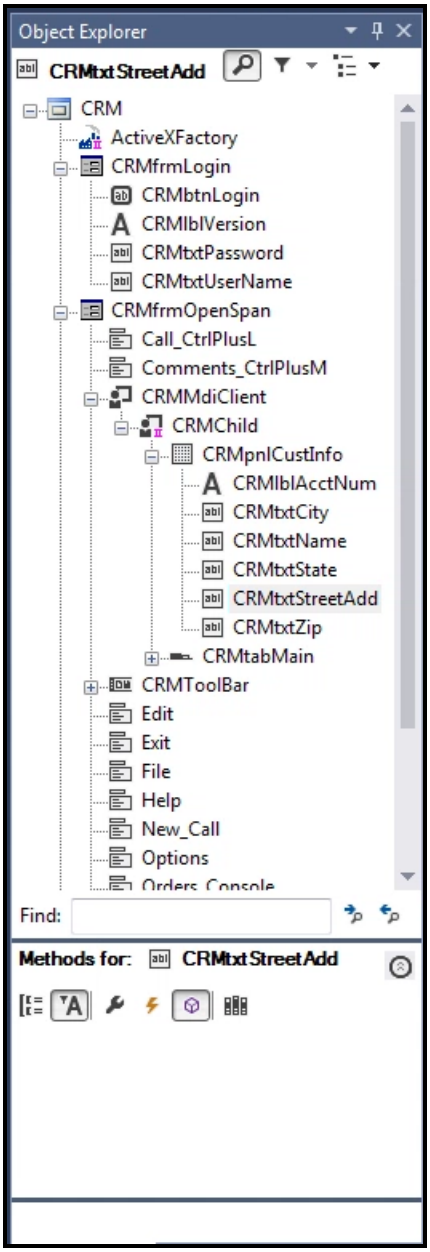
The Object Explorer window lists all adapters, forms, interrogated controls, automations, and components contained in a project and provide access to the properties, events, and methods of these items. Object Explorer has these panes:

- Object Hierarchy
- Object Inspector

The Object Hierarchy displays Project Items, such as adapters and automations, display in the Object Hierarchy. Interrogated controls appear beneath the corresponding adapter in a hierarchy, representing the control's parent/child relationship. The hierarchy changes its display of objects depending on which type of project item is currently active in the Designer window area.

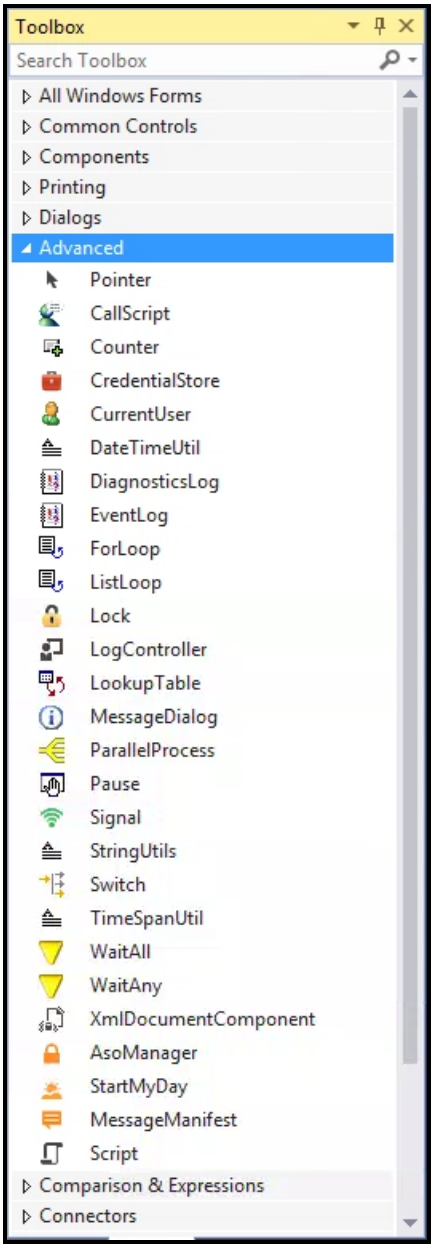
The Object Inspector, located beneath the Object Hierarchy, displays properties, methods, and events for the control selected in the Object Hierarchy.

Here is an example of Object Explorer showing adapters, components, and controls:



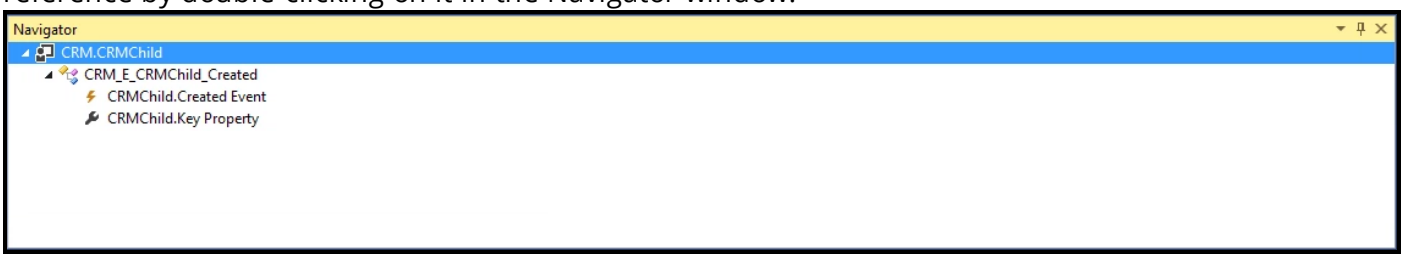
Toolbox

The Toolbox window contains .NET components and Studio components available for use in solutions. The standard .NET components are useful when creating Window forms and application bars in solutions. The Studio components are designed for use in automations when working with application adapters. These components are located on the additional tabs.



Navigator

The Navigator window shows the locations of a selected object within the project. To access the Navigator window, double-click on an object in the Object Hierarchy. You can easily locate the object's reference by double-clicking on it in the Navigator window.



Debug windows

Studio provides additional windows for design and debugging. The Debug windows are:

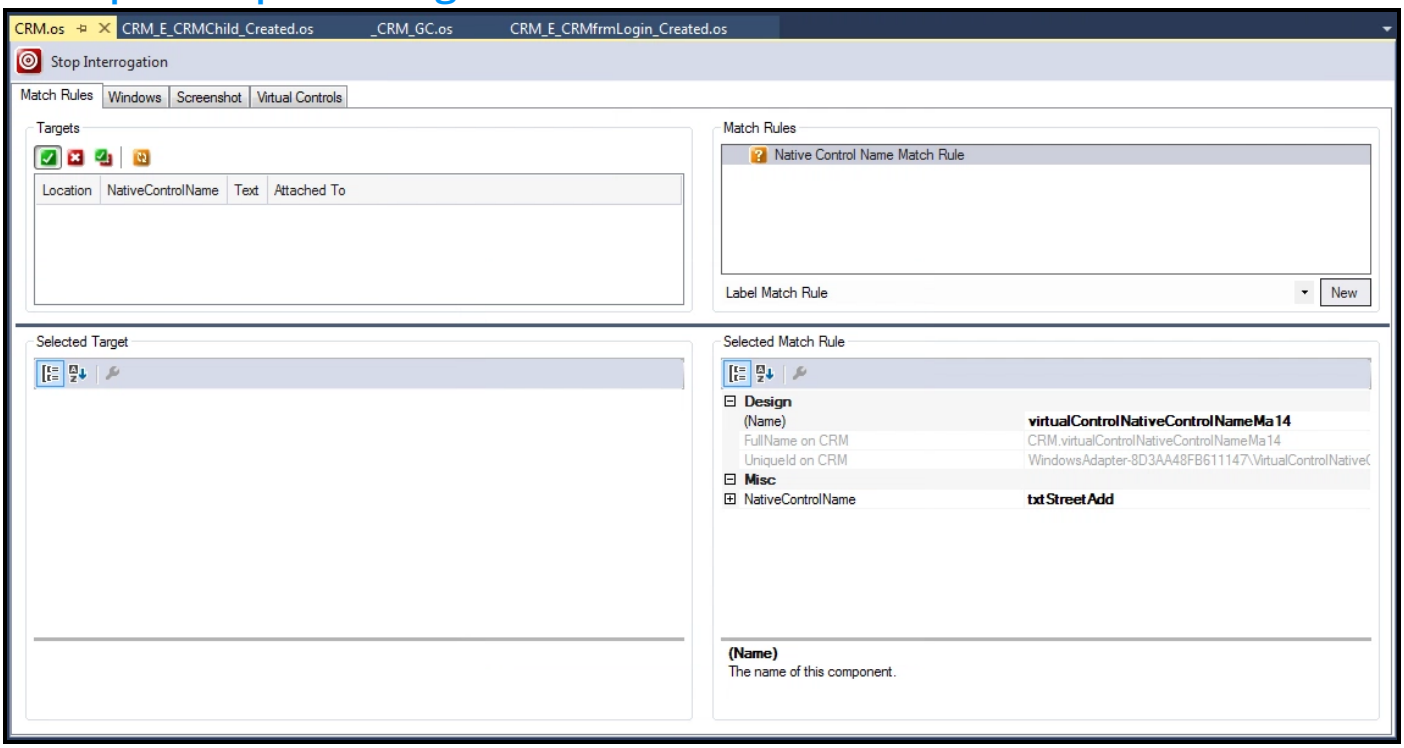
- Breakpoints
- Step into, over, and out
- Automation locals
- Automation watches
- Threads
- Output

Designer windows

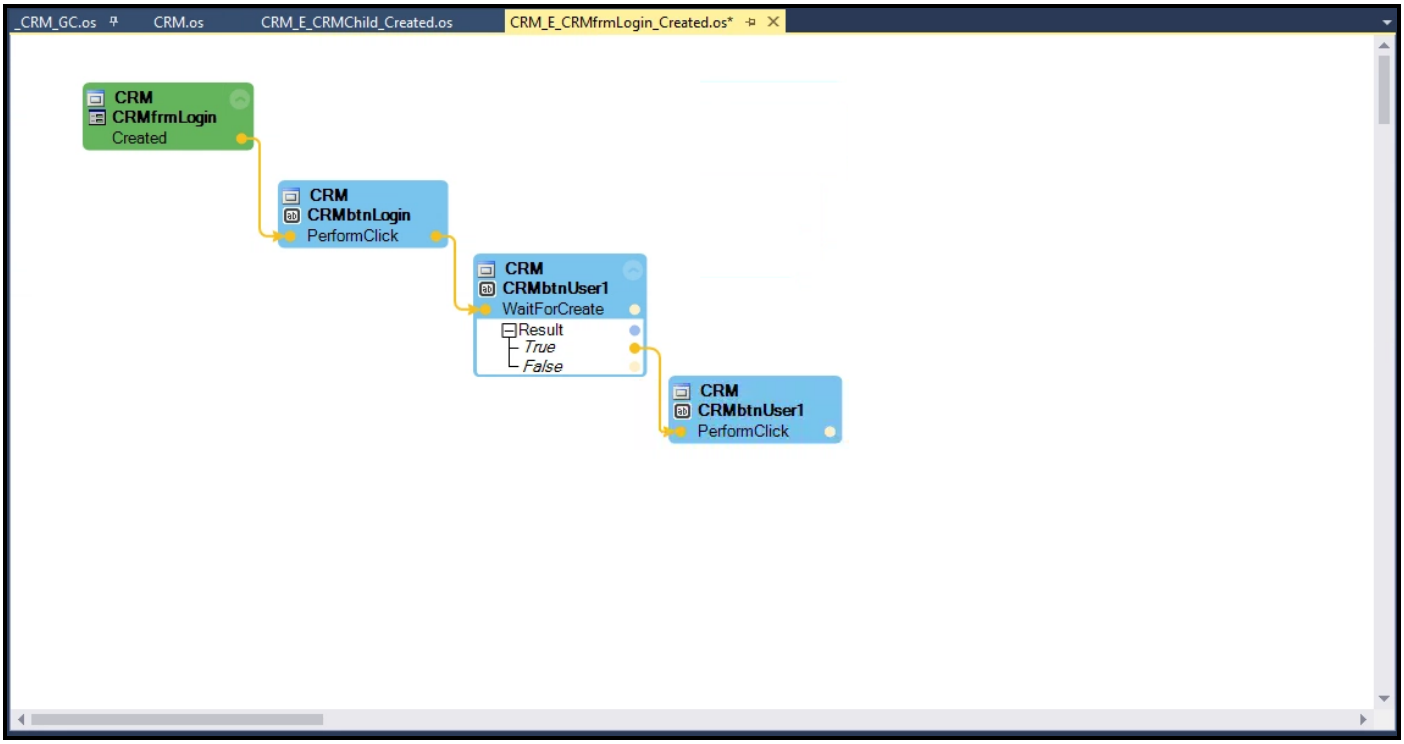
The Designer windows are document windows and therefore have different display options than Tool windows.

The Designer windows are created when you add or open a project item. These windows display as tabbed documents and are listed on the Windows menu. The Designer automatically opens a window for each project item you add to the solution. The Designer tab windows fill the center of the Studio IDE. Here are examples of Designer tabs for a web adapter and automation:

Example Adapter Designer window

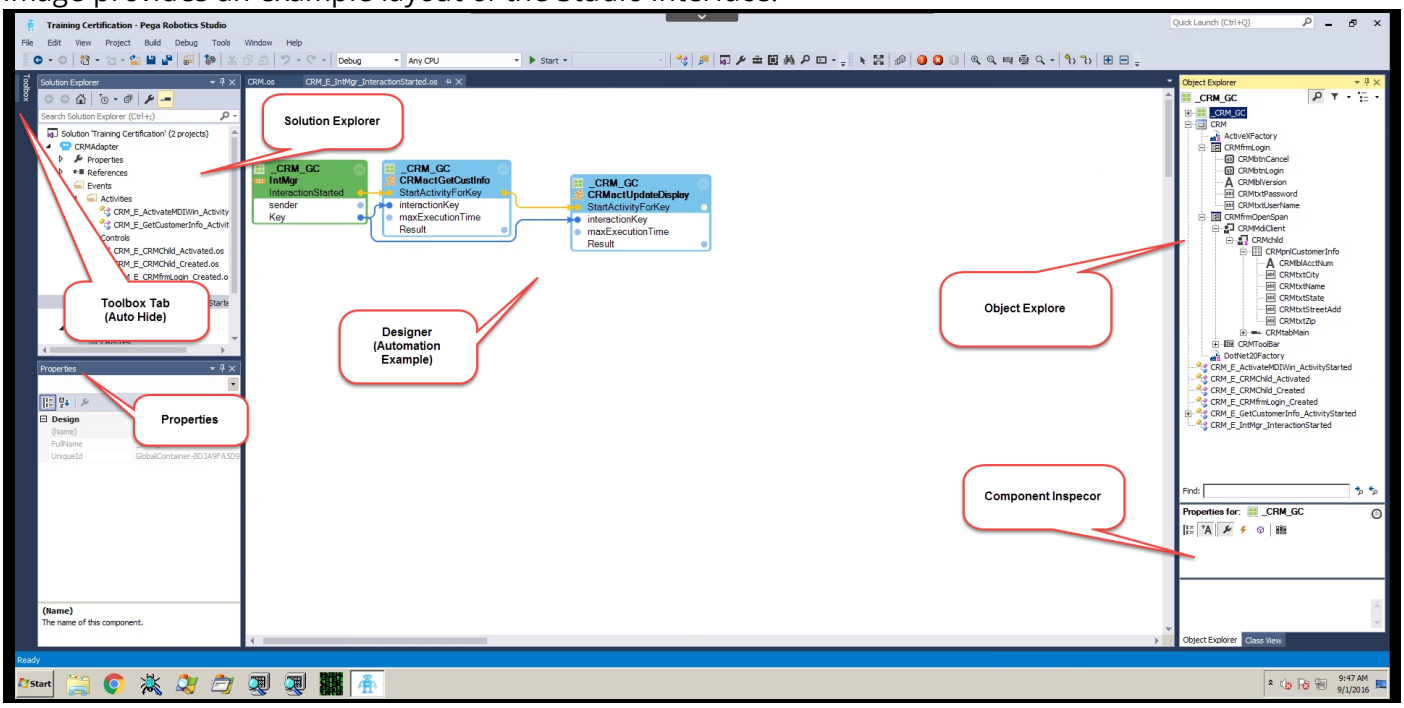


Example Automation Designer window



Organizing the workspace

Since Studio's Tool windows are movable/dockable, you can customize the Integrated Development Environment (IDE) layout. Tool windows can be stacked, docked, and set to Auto Hide. The following image provides an example layout of the Studio interface.



Docking Tool windows and Auto Hide

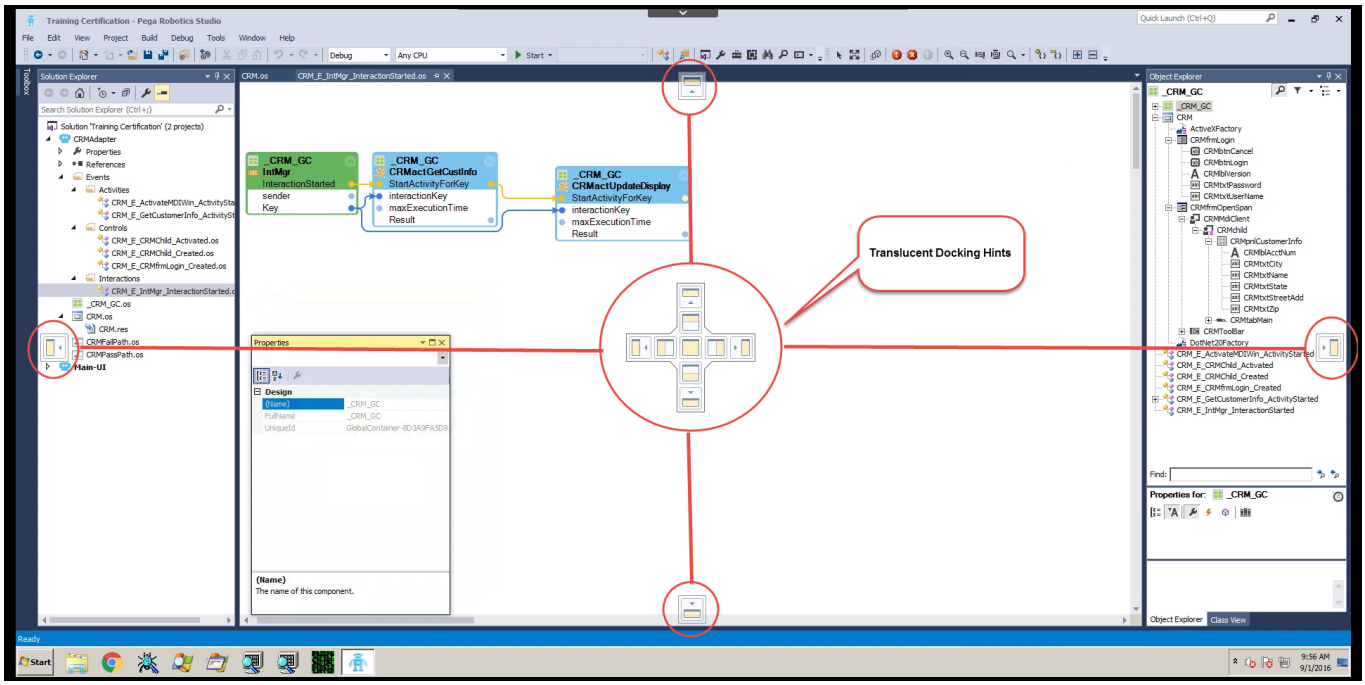
The following table provides the basic settings for arranging Tool and Debug windows in Studio.

Setting	Description
Docked/Floating	The window can be attached (docked) to any side of its parent window, or it can be detached and floating within the Studio application window.
Auto Hide	This feature lets you make the Studio work area larger by temporarily moving a Tool window out of view. A window hidden using the Auto Hide option is made available by moving your cursor over the tab with the window's name. The window slides back into view, ready for use.

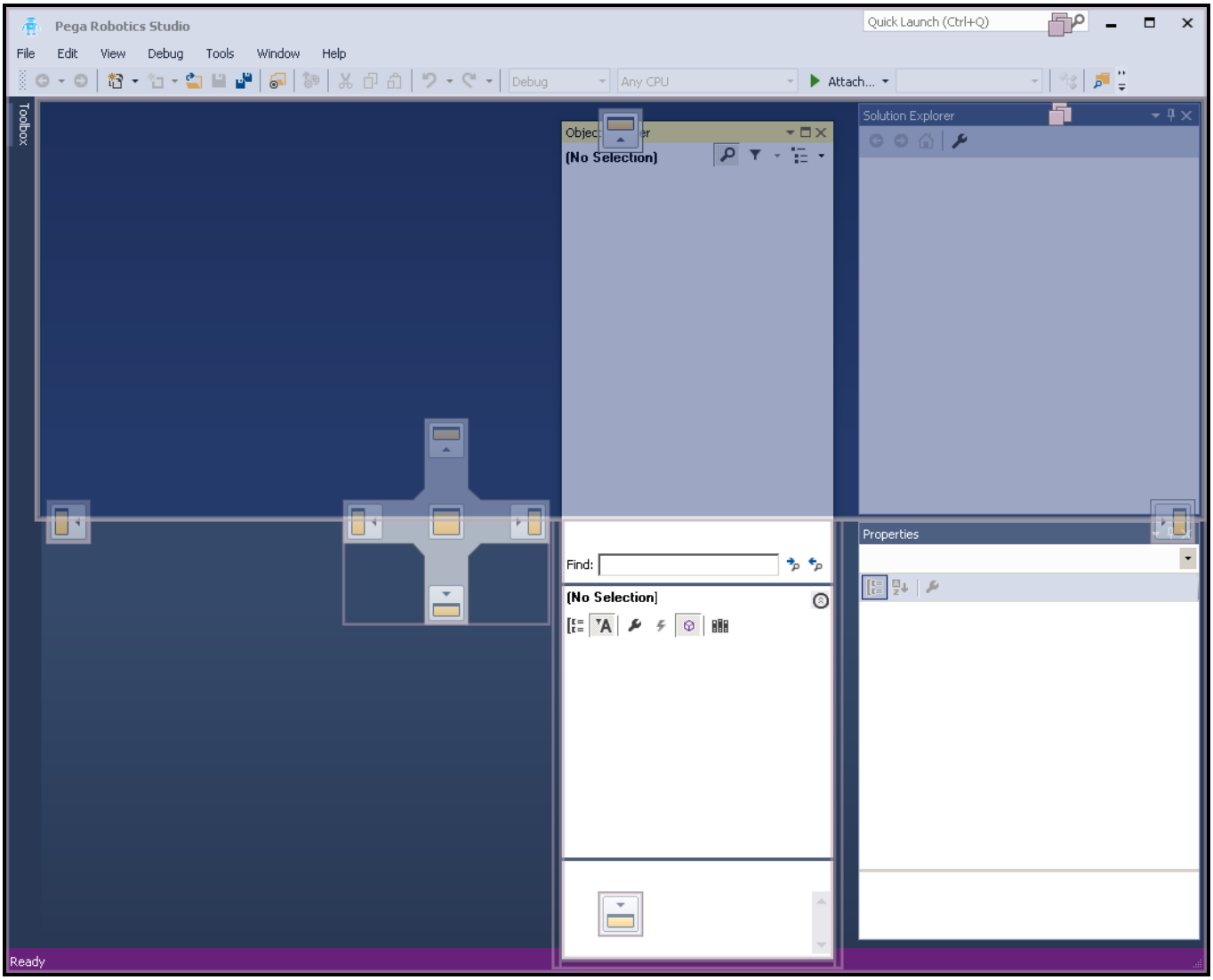
Float or dock Tool windows

All windows can be accessed and displayed from the View menu option. Once displayed, you have the option to dock, float, and auto hide docked windows.

1. Click the Tool windows title and drag the window. Translucent guide diamonds, representing various locations on the workspace displays.



2. While pressing the mouse button, place the cursor over the desired area of the translucent guide diamond. A darker, translucent representation of the docked window displays, allowing you to judge its placement. In the following image, the Object Explorer window is moved to the center to the workspace. The large, darker rectangle on the top half of the workspace is where the window is docked if the mouse button is released.

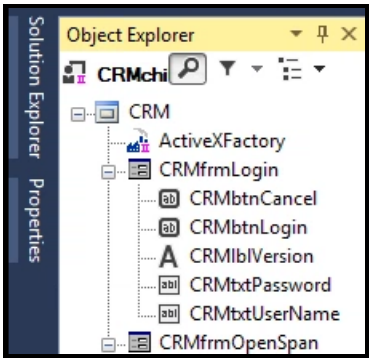


3. Release the mouse button to dock the Tool window.

Automatically hiding Tool windows

At the top of the Tool windows title bar, you can click the pushpin icon to enable and disable the Auto Hide feature. Auto Hide is only available when the Tool window is docked. If the window is in floating mode, you cannot Auto Hide it.

1. On a docked Tool window, click the **Pushpin** icon. The Tool window becomes tabbed on the side of the workspace where the window is docked and the window's name appears on the tab.



2. Click the tab to view a hidden window. Click anywhere in the workspace outside of the tool window to hide the window again.

Developing solutions and projects

Introduction to solutions and projects

This lesson covers the basic steps for creating a Pega Robotic Automation Studio solution and project, adding a Windows form for a user interface, adding interface .NET components, and renaming the objects in your project.

After this lesson, you should be able to:

- Describe the usage of projects and solutions
- Create a solution and a project
- Describe best practices when managing project items
- Add an item to a project
- Describe the purpose of the Toolbox
- Describe the purpose of the Object Explorer and Properties window
- Modify an object's properties

Solutions and projects

To create a good solution, it is important to understand the basic principles and relationship between solutions and projects.

Projects are the logical containers for all items you need to build your application. When you create the first project, Pega Robotic Automation Studio creates a solution that contains your project. During the creation of your project, you can add more new projects or existing projects to the solution . The location of the solution folder should be on the same computer as the Studio installation, not in a partition area or network drive. Studio requires access to the installation files when debugging and testing during development.

With Pega Robotic, a project should contain only one application. If your solution interacts with other applications, those applications should be included in your solution. For example if your solution interacts with four applications, there should be four projects within the solution – one for each application. Solutions should also contain a controller project, a central or main project that connects the application projects together as one working unit.

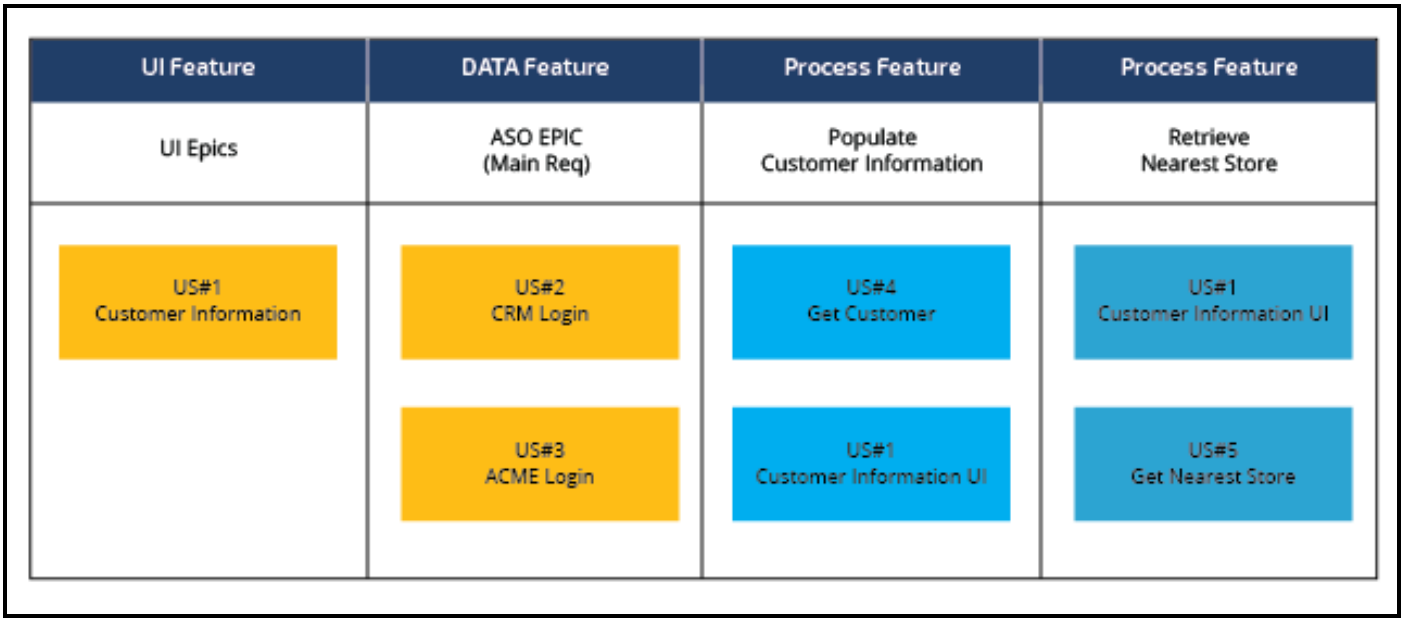
Training business case

It is important to consider the larger picture for the solution that you create to provide a roadmap to guide you through the development of the solution and the projects.

The solution for this course will be a multi-project solution, consisting of three projects. The solution's business case states that the end-users access the company's searchable web site to locate the nearest store to the customer's postal code. The reasons for this first case stems from the re-branding of stores because of a recent merger and customers are having difficulty locating the stores.

To accomplish this case, the solution will use a unified user interface (UI) that can grow when adding new tasks and processes from new business cases. The UI will display customer information from the CRM application when the call is accepted. Using the postal code of the customer's address, the agent clicks a button to access the ACME Search System to locate the nearest store address and displays it on the UI.

The image below outlines the business case process.



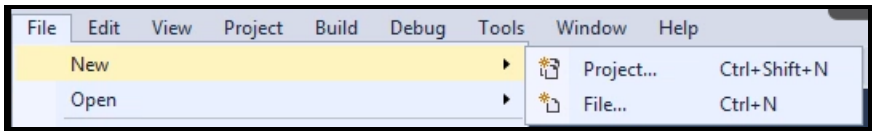
The controller project will be the user interface. Based on the business case, the user interface will be the central hub for this solution. There are two other projects: one for the CRM application and one for the ACME Search System application. We will complete the CRM application first, followed by the ACME Search System, and then unifying the two projects with the user interface controller project.

Creating a solution and project

When creating a new solution, you create a project and the solution together.

Follow these steps to create a new Studio solution and project.

1. Start Studio.
2. Select **File > New > Project**.



3. Select **Empty Pega Robotics Project**.
4. In the **Name** field, enter a project name.
5. Enter a folder location or accept the default.
6. In the **Solution Name** field, enter a solution name.
7. Click **OK**.

Tip: When naming solutions and projects, use names that represent the business case and application. The preferred nomenclature is camel case or a modified camel case format.

Understanding project items

Project items consist of specific functionality to integrate applications, invoke a web service, create customized forms or user interfaces, and add automations to execute program logic. The most common project items added to a project are:

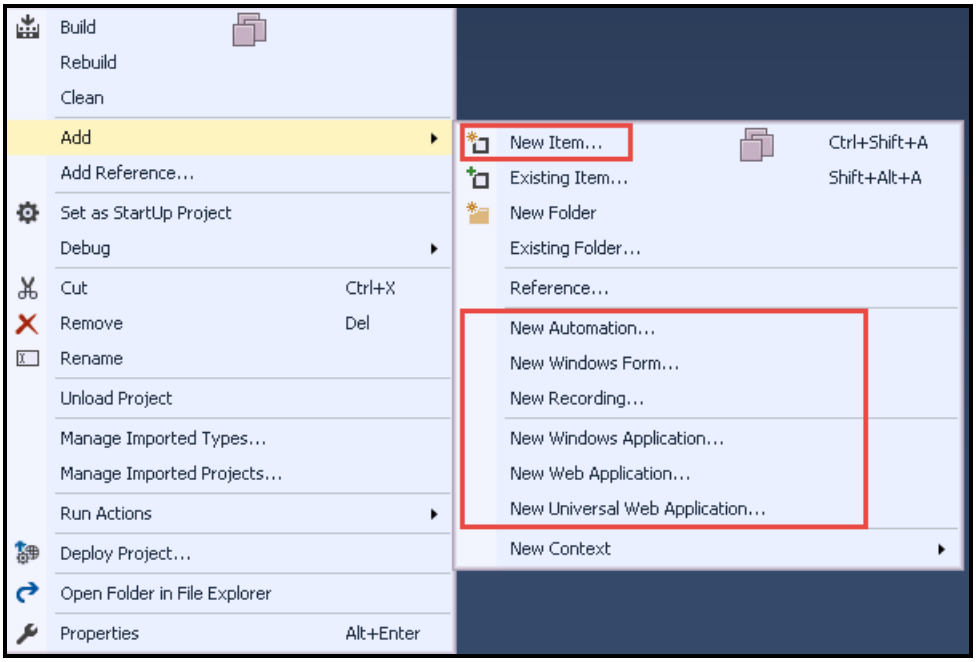
- Adapters
- Windows forms
- Automations
- Configuration Project Items
- Web Service

In the Solution Explorer, the Tool window displays the organizational structure of the solution and its projects. The projects should follow a standardized, defined structure to keep the project items organized. When working in a multi-developer department, this becomes important. All developers should use the same naming conventions and solution/project organizational structure to simplify the development process of merging single projects into the final solution. Developers should also use folders to help organize the project items into a meaningful manageable structure.

Adding an item to a project

For any project item, the steps to add an item are identical. Follow these steps to add an item to your project.

1. In Solution Explorer, right-click the project, select **Add > New Item**. Some of the most commonly used project items display on the context menu.
2. If you selected the project item on the context menu, Studio highlights the project item. If you selected **Add > New Item**, use the Installed Template frame to search for the necessary project item and highlight the project item.

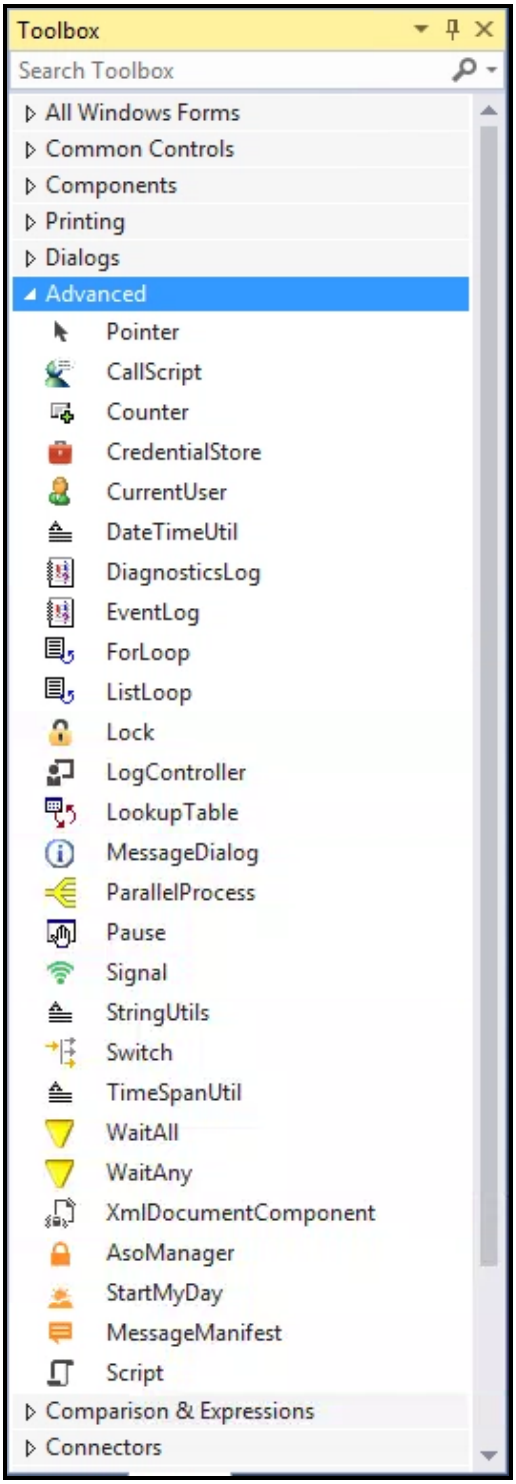


3. In the **Name** field, enter a project item name.
4. Click **Add**. The project item displays in the Solution Explorer, the Designer window opens, and the Object Explorer displays the project item at the root level.

Toolbox components

This topic covers the Toolbox components and their relationship to project items.

The Toolbox contains custom Pega Robotic Automation components that assist in the coding of automations as well as standard .NET controls to add to a Windows form to create a user interface. The toolbox contains common coding functions as components for use in automations, such as Variables, Counters, For loops, Look Up tables, Scripts, and Message Dialog boxes.



Some Toolbox components work only with specific project items, while others work with different project item types. After adding a component to a project item, you have access to that component's properties, events, and methods to use as needed for the project.

Object Explorer and the Properties window

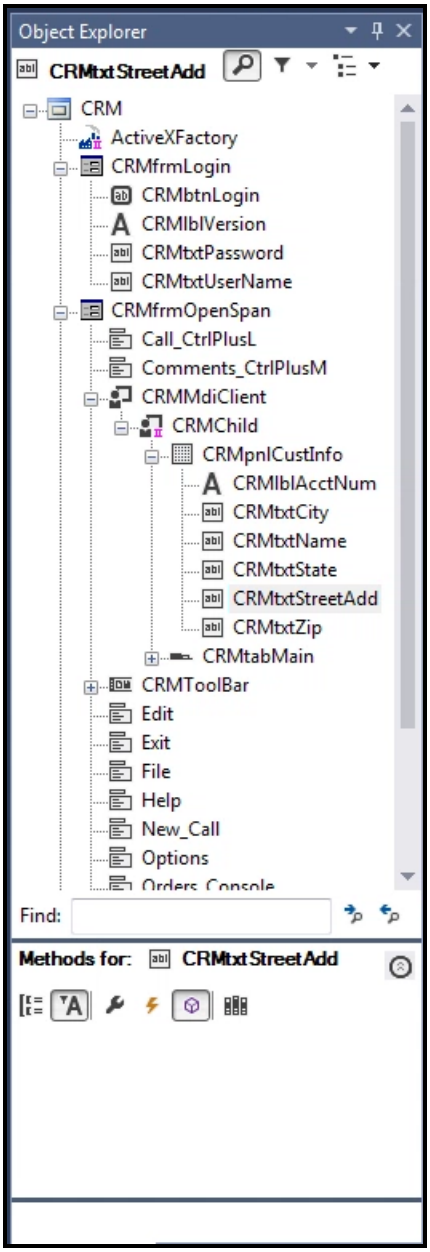
Understanding the relationship between the Object Explorer and the Properties window is important because it helps you to find objects in the project.

After adding an object to a project item, the object appears in the Object Explorer window in a hierarchy view. The hierarchy view shows the parent-child relationship of where the object exists in the project. If the object is an interrogated control from an adapter, the Object Explorer displays the matching status of the control during interrogation. The matching status of a control appears as a green checkmark.

The Object Explorer hierarchy view changes its structure depending on which project item is currently active in the Designer window area. If an adapter project item is active, the Object Explorer displays only those objects related to the adapter. When an automaton project item is active, all objects from all project items display in the Object Explorer.

When you select the object in the Object Hierarchy, you access the Properties window to change the object's default properties. Change the object's Design Name property to reference it throughout the project.

You can use automations to change properties of objects. Use the Object Explorer Inspector frame to select the property needing change. The Object Inspector provides access to an object's properties, events, and methods for use in an automation.



The tool/wrench, the lightning bolt, and the box icons represent the properties, events, and methods of the highlighted object in the Object Hierarchy. Click the corresponding icon to change an object's properties, respond to an event on the object, or perform an action on the object within an automation.

Modifying an object's properties

The following steps outline modifying an object's default properties before it is used in a project.

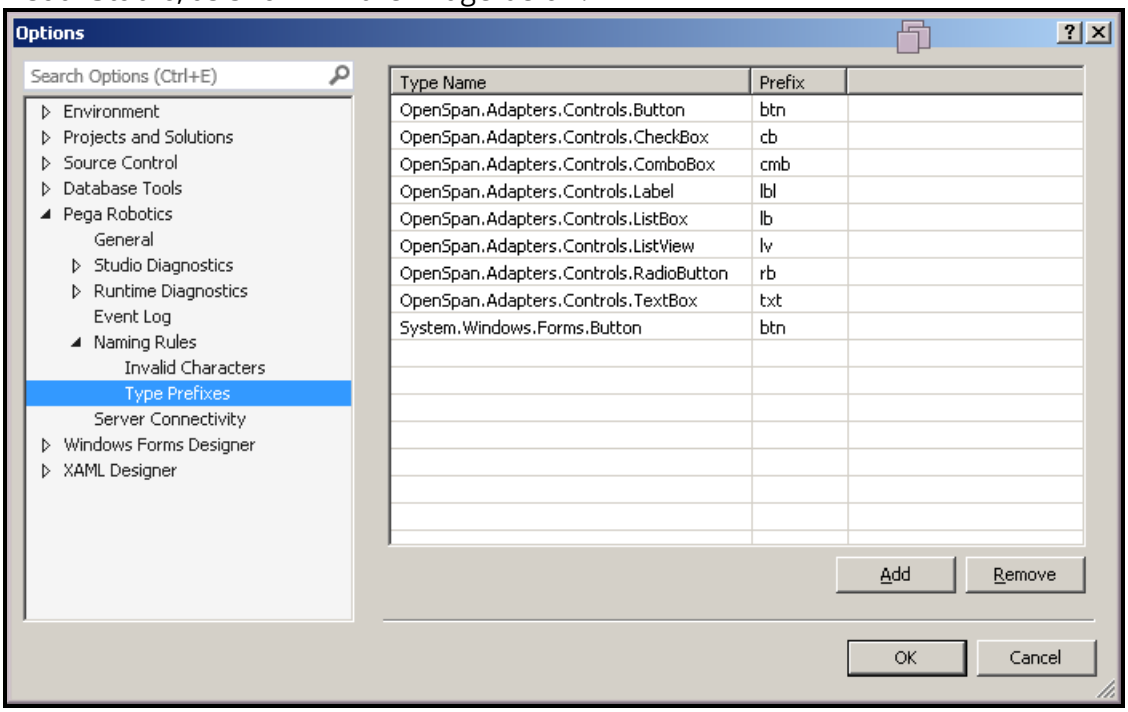
1. In the **Object Hierarchy**, click the object to select.
2. In the **Properties** window, scroll to find the property requiring editing. You cannot change property values in gray.
3. Click **File > Save**.

If a property contains a Boolean value, True/False, double-click the value to change it.

Sometimes a property value requires more modification. If this is the case, you will see an ellipses button. Click the ellipses button to display the property editor window.

Establishing a standardized naming convention for controls and components is important and a Pega Robotics Studio recommended practice. Whether you are building an automation, debugging a solution, or troubleshooting match rules, renaming the controls to something descriptive of their function makes it easier to identify the controls to complete the task.

For renaming controls, Pega Robotic Automation follows the standard object prefixes from Microsoft Visual Studio, as shown in the image below.



Use the following conventions when naming controls:

Object type prefix + Name of object

The object prefix is in lower case, and capitalized each word of the object's name. For example, btnSignIn and txtFirstName.

Working with windows adapters

Introduction to window adapters

This lesson introduces you to window adapters, their properties, and interrogation techniques.

After this lesson, you should be able to:

- Describe common adapter properties
- Define interrogation and alternative interrogation methods
- Interrogate a windows application

Window Adapters properties

When a developer is ready to add another application to a solution, the developer must first consider the type of adapter needed based on the type of application. The developer should add a new project for each new application required for the solution. Following this construct creates a reusable project for other solutions. One of the most common applications added for a new project is a Windows application.

A developer must understand how a Windows application performs and functions in order to configure the adapter properties for use in the project and the solution. The following adapter properties apply to a Windows application:

- Path
- TargetPath
- HookChildProcesses
- StartMethod
- StartOnProjectStart
- WorkingDirectory
- Arguments
- HideApplicationAtRuntime

Path property

Enter the full path in the **Path** property and the executable file name when the application runs from the same installation folder of the program. The adapter requires the Path property completed. Here is an example:

```
C:\Program Files\OpenSpan\CRM Setup\CRM.exe
```

If the application is in the system path, enter the application file name without the full path. Here is an example:

```
CRM.exe
```

The Path property does not support links (lnk), wildcards, or Regex text entries. If you are deploying a solution where the target application runs in different folders on the desktops, you can use:

- Different solution configurations
- The Folder sub-property under the Path property

The Folder property allows you to select a system folder and file location for the installed application directory.

TargetPath property

The **TargetPath** property references an application launched as the result of one or more other processes (applications) occurring first. The Path in the TargetPath property refers to the executable that starts the target application.

To use the Path/TargetPath relationship, you must change the StartMethod property from Start to StartAndWait. Using the Path/TargetPath configuration, you cannot interrogate the Path application. You can only interrogate the TargetPath application.

Note: Java-based applications use this configuration often.

HookChildProcesses property

The **HookChildProcesses** property configures the solution to integrate with an application and any application launched from that application. Set the property to True when:

- One application starts other applications
- Both parent and spawned applications are required for the solution

As a result, Studio must “hook” into, or, integrate with, each application process.

StartMethod property

The StartMethod property determines how the Path application starts. Options include:

Option	Description
Start	Tells Studio to launch and hook the application defined by the Path property after the adapter starts. This is the default.
StartAndWait	Tells Studio to launch the application identified in the Path property after the adapter starts, wait for the TargetPath application to launch, and then hook the TargetPath application. Note: Studio does not hook the Path application; thus, you cannot interrogate or automate the Path application.
MonitorAll	Tells Studio to wait for the Path application to run and then hooks the application. In this case, starting the adapter does not cause the Path application to launch. The Path application launches independently after the adapter starts (for example, from an external application, or manually). Once the application launches, Studio hooks the application. When using the MonitorAll, you can omit the full path when specifying the application executable in the Path property and enter the executable file name. Note: Using MonitorAll lets the adapter to remain running if the Path application shuts down because of an application error or the end-user closing the application. When re-starting the application through an automation or by the end-user, Studio will hook the application.

StartOnProjectStart property

Use the **StartOnProjectStart** property to start the adapter when the solution or project starts. The default setting for this property is True. The adapter may or may not launch the associated Path or TargetPath applications, depending on the StartMethod property.

If StartOnProjectStart is False, you must start the adapter first in an automation. When you set the StartOnProjectStart property to False, the Path and/or TargetPath applications do not launch when the project starts, regardless of the associated adapter StartMethod value.

Pega recommends leaving the default setting for those applications required to run when the solutions runs. A common usage of this property is to control the login process and application start up time for end-users.

Working directory property

Studio populates this field with the location of the **working directory** property. In this case, the working directory is the same value as the Path property value. Modify the working directory to point to the supporting installation files when different from the install directory of the executable file.

Arguments

Use the **Arguments** property to enter command-line arguments required for starting the application associated with the adapter. For example, you could enter the name of a file to launch when the application launches. You can also use this property to specify the Java class names for Java applications.

HideApplicationAtRuntime property

Set the **HideApplicationAtRuntime** property to true to hide the application during project runtime. To show the application, call the Show method on the adapter. You should use this method sparingly. Instead, develop options such as progress bars or grayed-out applications to notify the user when an automation or process completes.

Windows interrogation

Interrogation is a Pega Robotic Automation Studio function that exposes the properties of an application and its objects to Studio. Studio then uses system-defined matching rules to identify the object from other objects in the application. Studio saves the matched rules with the project. Each time the application runs within the solution, the solution compares the application's object property values to the matched rule values of the interrogated objects. If the two values agree, Studio matches the object for use in the solution.

The interrogation process does not change any application's underlying source code or property values. When interrogating an object, Studio creates a representation of the object in Object Hierarchy of the Object Explorer. Studio includes the parents of the interrogated object in the hierarchy as well, presenting a parent/child relationship with the adapter being the root of the hierarchy.

A green check mark on the object denotes a matched interrogated object. The green check displays only while interrogating an application.

Interrogating a windows application

Since application development is not consistent, you can interrogate an application in one of three ways. You can:

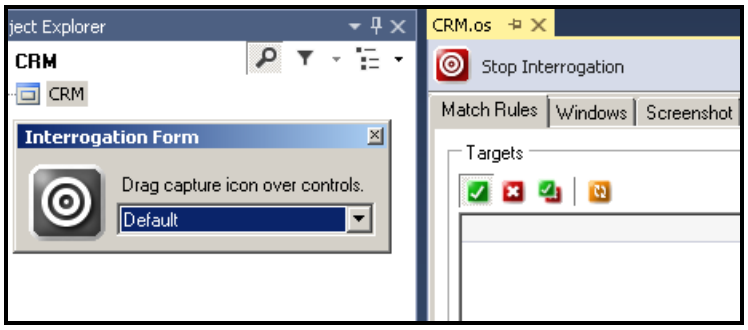
- Use Interrogation target
- Use Create Control
- Add Menu Items

Regardless of which method you use, starting and stopping the interrogation process is the same for all three.

Starting and stopping the Interrogation process

Follow these steps to begin the interrogation process for any adapter.

1. Double click the adapter project item in the Solution Explorer to ensure the adapter design window is active.
2. In the adapter designer window, click **Start Interrogation**. The adapter starts and the application launches. The **Interrogation Form** window displays.



3. Click **X** on the Interrogation Form window to stop the interrogation.

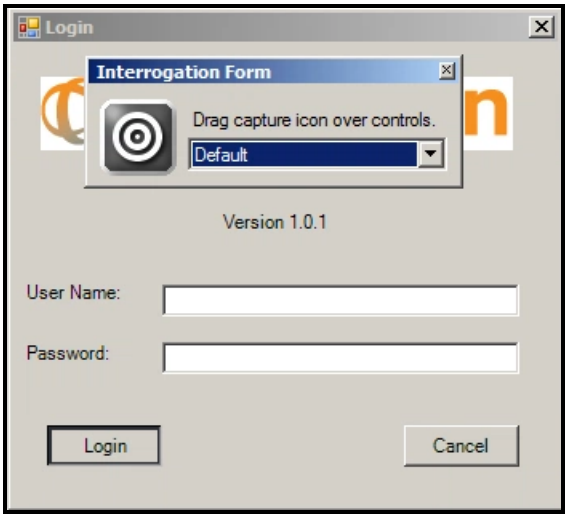
Note: You may also close the Interrogation by:

1. Close the application launched from the adapter.
2. Click **Stop Interrogation** on the adapter designer window.

Using the Interrogation target

The interrogation target is the most common method of interrogating an application. Follow these steps to interrogate an application.

1. Start the Interrogation process.
2. Click the **Target** icon.
3. Holding the mouse button, drag the mouse to the desired object. A black outline highlights the object.



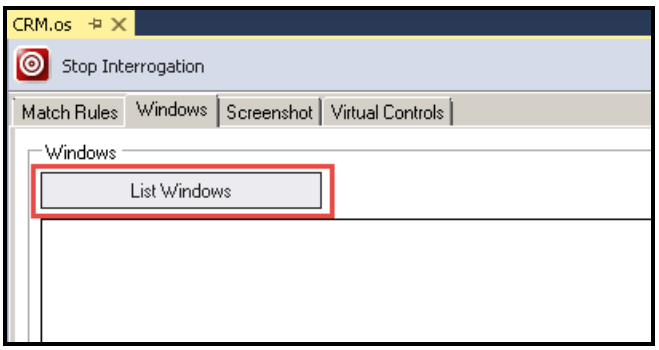
4. With the desired object highlighted, release the mouse button. The object and the object's parents display in the Object Hierarchy with the matching green checkmarks.

5. Repeatstep 4 for all needed objects on the application window.
6. Rename the interrogated objects in the Properties window.
7. Stop the interrogation process. The Application and Interrogation Form windows close.
8. Click **File > Save**.

Using the Create Control for windows applications

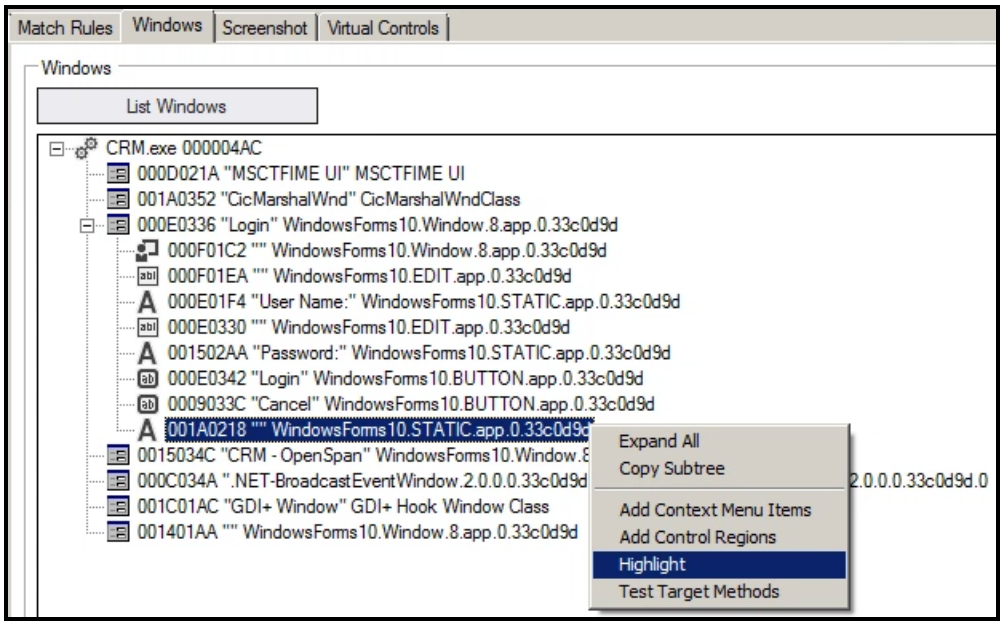
You may encounter a target that is visible but positioned so the bulls-eye icon cannot focus on the object. Follow these steps to use the Create Control interrogation method.

1. Start the Interrogation process.
2. On the adapter designer window, click the **Windows** tab.
3. Click the **List Windows** button to show all available processes within the selected adapter.

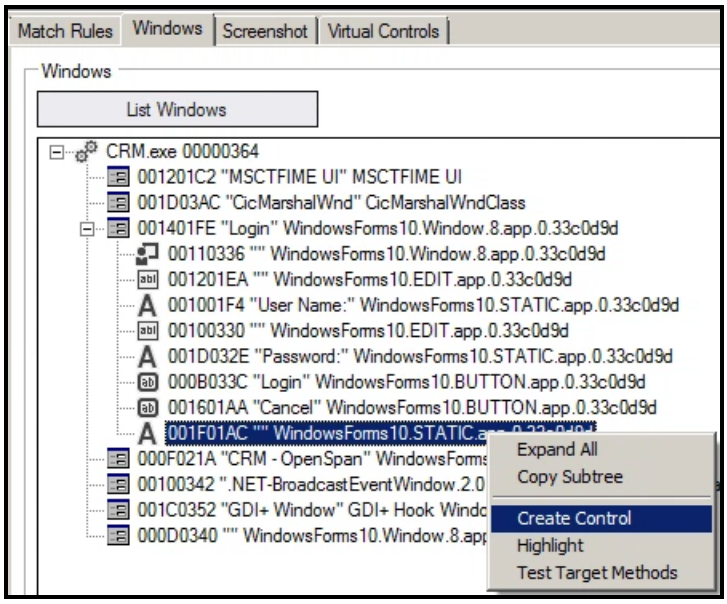


4. From the list of processes, click the **Plus** to expand the objects.
5. Continue expanding the different levels as needed.
6. Right-click on an object and select **Highlight**. The application appears with the selected object outlined and flashing.

   Note: Using the Highlight feature is helpful when you are trying to determine the location of a control within the application interface.



7. Return to the **Windows** tab on the designer window.
8. Right-click on the desired object and select **Create Control**. The control appears in the Object Explorer along with associated parents with the matching green checkmark.



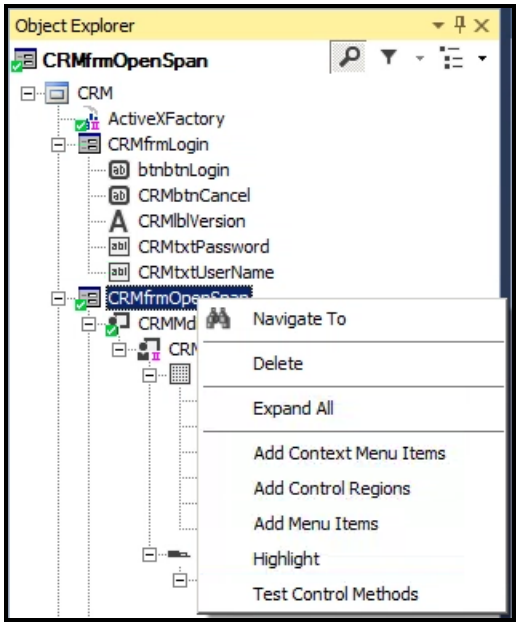
Repeat these steps for each object required for the solution and business case.

Using Add Menu Items

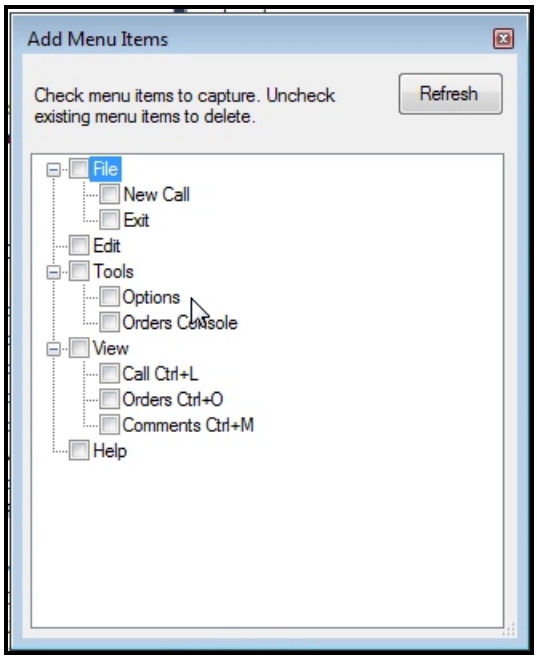
There are times that the business case requires a menu option for the solution. Using the interrogation target disables the menu options on the application window.

Follow these steps to use the Add Menu Items.

1. Start the Interrogation process.
2. Navigate to the main windows form that contains the menu.
3. In the **Object Hierarchy,** highlight the windows form.
4. Right-click the windows form object in the **Object Hierarchy**. The object must show as matched to display the **Add Menu Items** option.



5. On the context menu, select **Add Menu Items**. The Add Menu Items window displays.



6. Select the desired menu items from the list.
7. Click **Refresh** to display dynamically created menu items after navigating to a window that creates them.
8. Stop the interrogation process.

As long as the main window form matches, the menu items from the form match. Studio uses the Menu Item Path match rule to match these controls. The path refers to the hierarchy of the menu options.

For example, the path for the Exit menu option is **File > Exit**. When using sub-options, like **Exit**, in an automation, you only select the **Exit** option from the Add Menu Items window, the full path of **File > Exit** is not needed.

Working with automations

Introduction to automations

This lesson teaches you about automations and their relationships to the Object Explorer.

After this lesson, you should be able to:

- Explain the role of automations
- Describe the relationship between automations and the Object Explorer
- Describe the steps in using the Object Explorer to create automations
- Create an automation

Automation development

Automations are the backbone of Pega Robotic Automation Studio. Automations convert an end user's manual processes to consistent and repetitive operations. Because manual processes drive the development of automations, end user actions or application actions make automation development into event-driven project items. Whether using the click of a button, combo box selection, an application window's display, or a web page browser load, all events can trigger an automation to begin a process to meet the expected business goals.

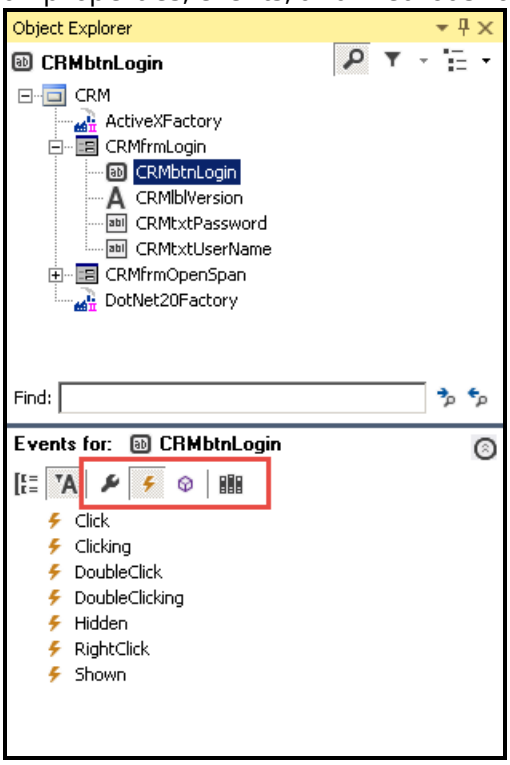
When attempting to mirror or automate a manual process, pay attention to how users interact with the application as well as how the application responds to the process. Watching for variations between users with mouse clicks versus keyboard shortcuts aids in the development and structure of your automations for the project and solution.

Object Explorer and automations

Automation development drives its structure from the Object Explorer. The Object Explorer displays interrogated application controls as well as .NET controls added to a windows form. Through this access, a developer has access to the object's properties, events, and methods to connect a logical flow that reproduces the manual process.

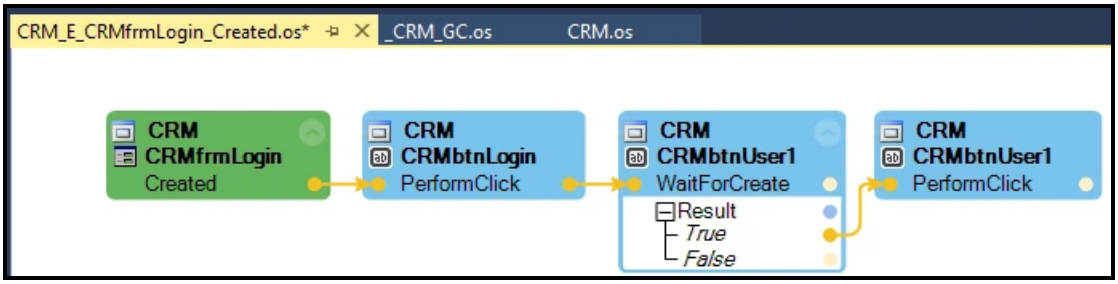
In automations, you can change the text value of a field, add and select an item from a combo box, determine if the end-user interacted with an application, or perform a click on a button. Studio provides these options to the developer in the Object Inspector of the Object Explorer.

The Object Inspector provides access to the object's properties, events, and methods. In the following Object Inspector image, the three buttons – wrench, lightning bolt, and box - display an object's properties, events, and methods when clicked. The fourth button, books, provides access to a library of all properties, events, and methods for a specific object type.



Automations usually begin with an object's event, meaning that the automation does not begin until a window displays or a user clicks a button. Design blocks are the items added to an automation.

When you add the logical sequence of the design blocks to an automation, the automation links denote the executable logic of the design blocks. Each design block has an input port and an output port. Green design blocks are events; blue design blocks are properties; and light blue blocks are methods. Studio provides linking to pass coding logic and data values. Studio denotes execution automation links in yellow and data automation links in blue. The following image shows an automation example.



The automation begins only when the application creates the CRMfrmLogin window. The automation performs the click on the CRMbtnLogin, waiting for the application to create the CRMbtnUser1, and when the button is available, the automation performs a click.

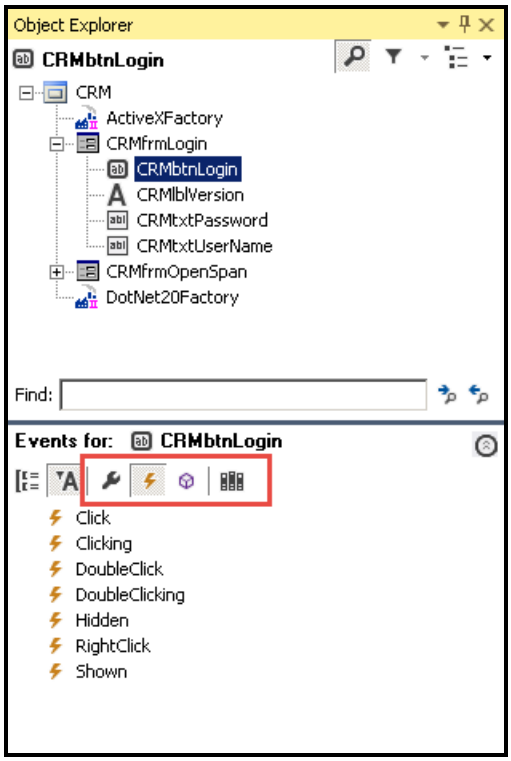
Object properties, events, and methods

Depending on the nature of the manual process, using the Object Explorer and its two sections to develop an automation is a simple process. While the order in which you add items to an automation does not matter, you should add them based in the order of the manual process steps.

Adding a property, event, or method to an automation

Follow these steps to add a property, event, or method design block to an automation. These steps assume the project contains an automation project item.

1. In the **Solution Explorer**, double-click on the automation project item to open it in the designer window.
2. In the **Object Hierarchy**, click to highlight the desired object or control.
3. In the **Object Inspector**, click either the property, event, or method button to filter the options.



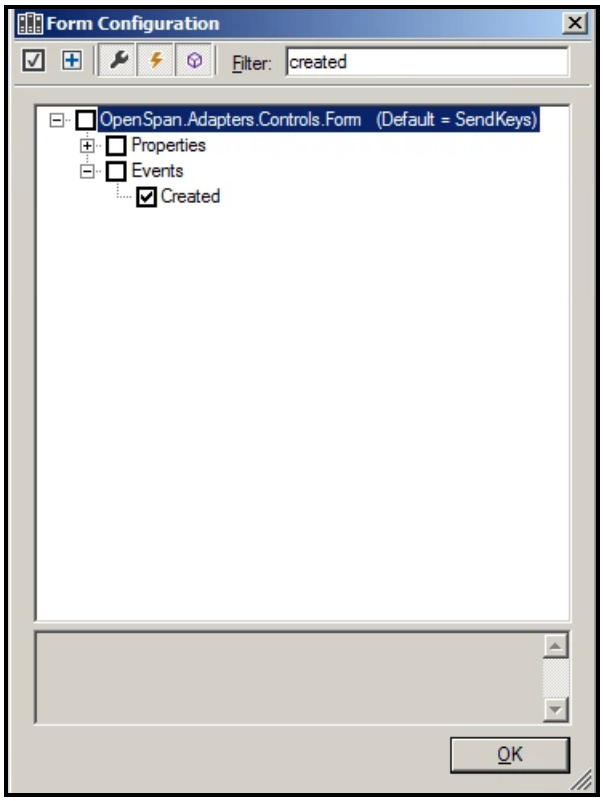
4. Click and drag the desired property, event or method to the automation designer window.
5. Release the mouse button to add the design block to the automation.

Adding properties, events, and methods to the Object Inspector

Often the desired property, event, or method does not display after clicking the associated toolbar icon in the Object Inspector.

Follow these steps to add a property, event, or method to the Object Inspector. An automation must be active in the designer window.

1. In the **Object Hierarchy**, click to highlight the desired object or control.
2. In the **Object Inspector**, click **Book** toolbar icon. The Object Type Configuration window displays.



3. Enter a name of a property, event, or method in the Filter textbox. The Configuration window begins to filter out all the options that do not contain the letters entered.
4. Click the **Plus sign** to expand and locate the needed property, event, or method.
5. Select the **check box** in front.
6. Click **OK**. The selected items display in the Object Inspector.
7. In the Object Inspector, click the appropriate toolbar icon. The selected property, event, or method from the Configuration window displays for use.

Working with multiple application instances

Introduction to MDI windows

In this lesson you learn how the UseKey property works in automations.

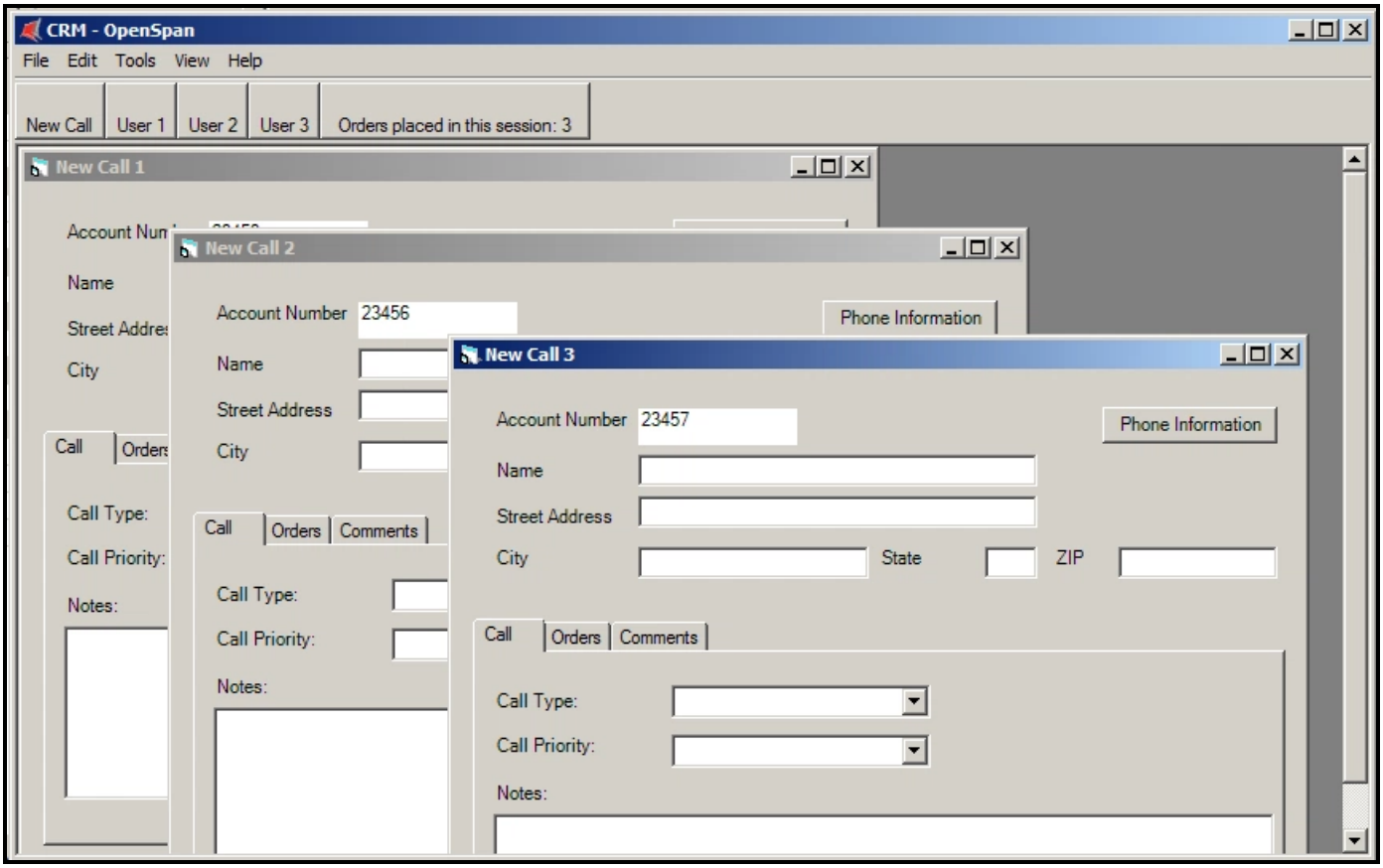
After this lesson, you should be able to:

- Define the UseKey property
- Explain how Studio applies the UseKey property in automations
- Describe the steps in setting the UseKey property on an object
- Configure the UseKey property

The UseKeys property

When building projects with Studio, you should pay attention to how an application responds and behaves to user interaction. One such concept to watch for is the creation of cloneable objects or multiple instances of the same object displaying simultaneously. One example is for an end user to log into the same application more than once concurrently.

Another example is our training solution CRM application. The CRM application uses a technology called multiple document interface or MDI. MDI allows the application to reproduce a template window that simply displays different values in the fields. The following image displays this occurrence.

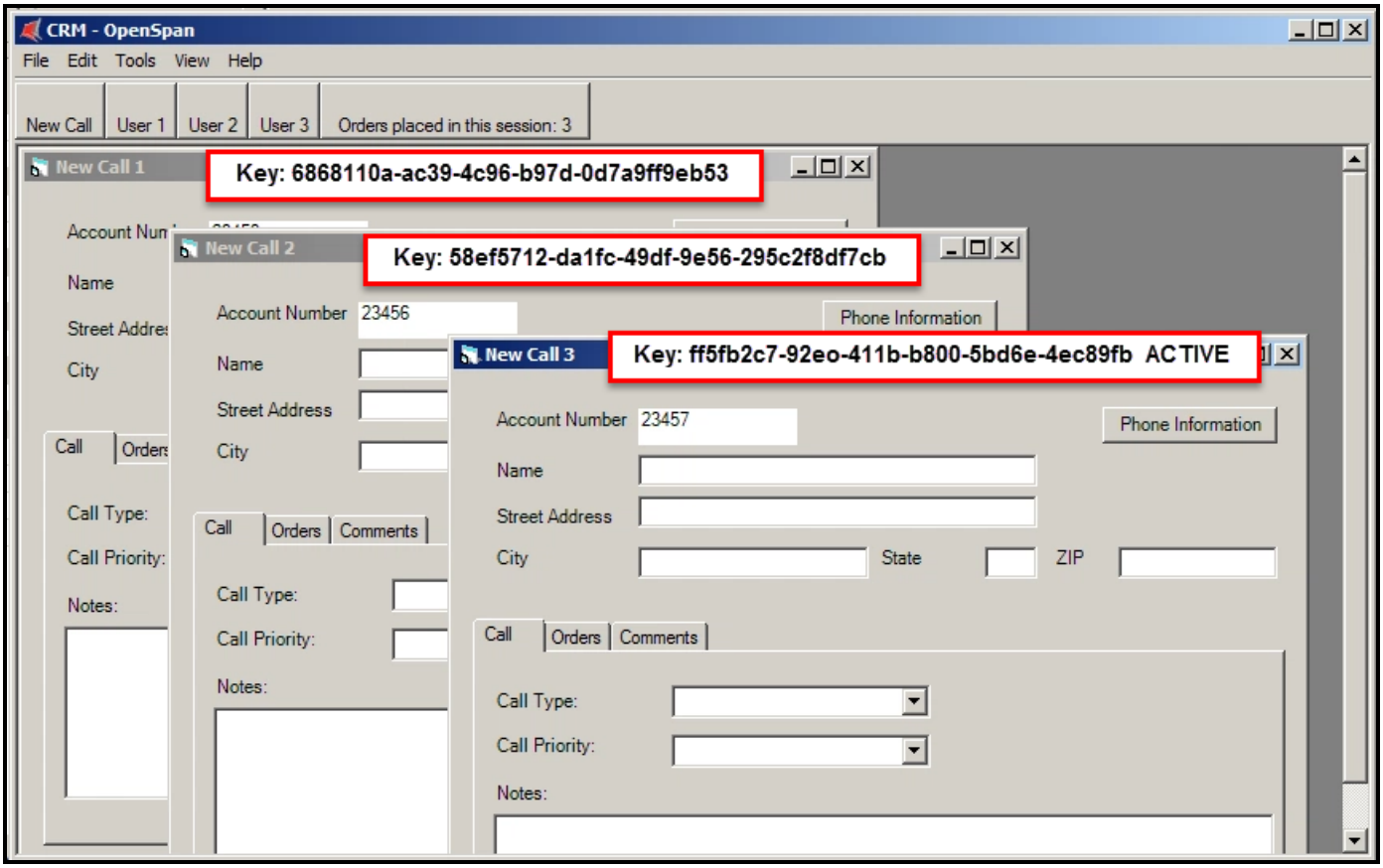


The CRM application produces the same window for all customers, and it allows the end-user to work on more than one customer at a time. Think about how this may influence your solution if the end-user can have multiple accounts opened at the same time across several different applications. How do you differentiate the accounts so that all applications reference the same account at the same time?

Studio creates a key property for all interrogated controls and uniquely identifies the object at runtime. The Key property has these two values:

Value	Description
Relative	Studio determines if the value is Active or Null
Absolute	Studio uses the GUID (Global Unique Identifier) of the object

For example, as each CRMchild window opens in the CRM application, Studio automatically assigns the absolute value to the key of each window. The key uniquely identifies the window and all of the objects on the window. Studio also designates a key as Active for the active window, as shown here:



However, using the GUID through a solution across several applications is difficult to maintain. Therefore, Studio provides you the ability to assign your own key to an object. In order to do this, you must change the UseKey property for the cloneable object.

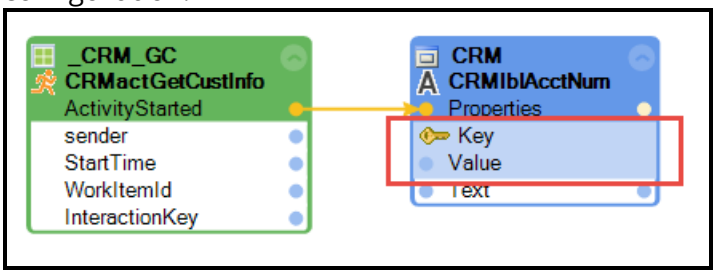
Application of keys

Studio creates and uses keys whenever a context exists for cloneable objects. Here, the term context defines any instance of an object for which multiple instances (clones) occur. For example, the CRMMDIchild object, as referenced in the Object Explorer, is a context.

A set of rules determines the creation of keys and the setting of keys for contextual objects.

- An event that creates the instance sets the context.
- An event from No Context to a Context requires a key assignment in an automation.
- An event from Context to Child Context requires a key assignment in an automation.
- An event from Context to Parent Context does not require a key assignment in an automation.
- Logic within the same context does not require a key assignment in an automation.

Studio assists you in applying the rules when developing automations. The glue that enables the rules to function are the automation links. You must connect the automation links between the design blocks so Studio can apply the key rules. If a key is required, the design block displays a new key option for configuration.

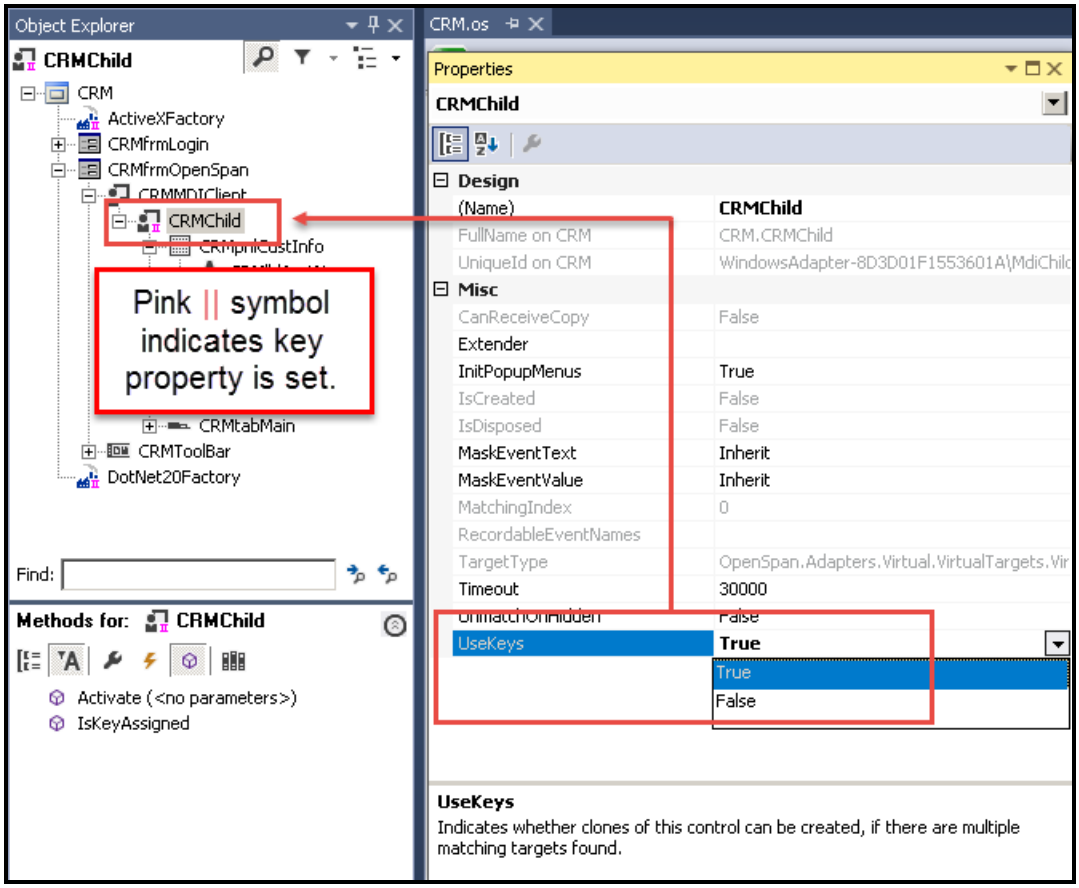


Setting the UseKeys property

The UseKey property should be set on the object that is the cloneable object or template. Use the interrogation process as an easy way to ensure you select the correct matched object to set the property.

Follow these steps to change the UseKeys property of an object.

1. In the Solution Explorer, double-click the adapter project item to open it in the designer window.
2. In the adapter designer window, click **Start Interrogation**.
3. In the launched application, navigate through the application until the cloneable object displays.
4. In the Object Explorer, validate the object's matching status. It should have a green checkmark on the object's icon in the Object Explorer.
5. In the Object Explorer, click the matched object to highlight it.
6. In the Properties window, locate the UseKeys property.
7. Double-click the UseKeys property value to change it from False to True. A pink double-pipe symbol displays on the object's icon in the Object Explorer.



8. Stop the interrogation.
9. Select **File > Save** to save the edits.

DEBUGGING AND DIAGNOSTICS

This lesson group includes the following lessons:

- Debugging
- Diagnostics
- Error handling

Debugging and Diagnostics

Introduction to debugging and diagnostics

This lesson teaches you basic debugging for Pega Robotic Automation Studio and how to use diagnostics within Studio.

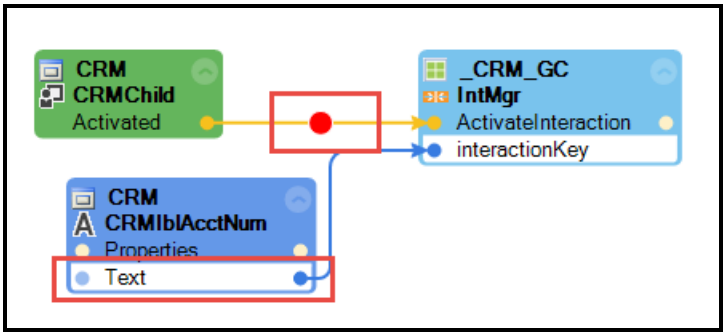
After this lesson, you should be able to:

- Explain Pega Robotic Automation debugging
- Describe the steps in adding, disabling, and deleting breakpoints
- Describe the diagnostic settings
- Describe diagnostic logging
- Explain how to implement diagnostic logging
- Complete the steps to add a diagnostic logging component

Overview of debugging

Now that you have built the first automation, it is time to discuss debugging with Pega Robotic Automation Studio. Because Pega Robotic Automation Studio uses Microsoft Visual Studio as its platform, the concept of debugging is similar. There are some slight variations with its implementation.

In Studio, you set breakpoints and review data values. Because an automation link or data link is similar to a line of code in other development tools, you place breakpoints on the automation and data links. In addition, Studio presents data values with the blue dot data ports on the design blocks. This is similar to variables in other development tools, where you can place your cursor on the variable during debugging to see its value. In Studio, you can place your cursor on the blue input or output dot to see the value in the design block.



How to set breakpoints

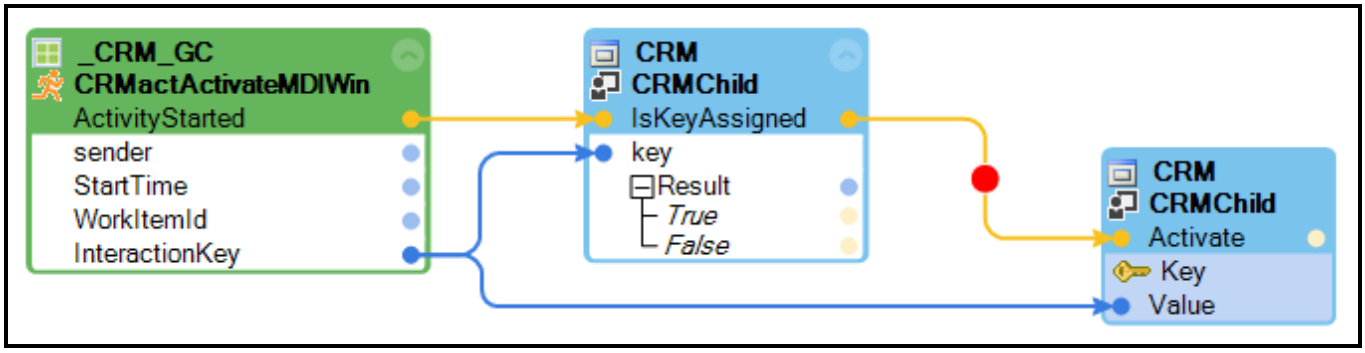
Setting and using breakpoints in Studio involves the following:

- Setting and removing breaking points.
- Stepping through automations.
- Viewing data values.

Setting and removing breakpoints

Follow these steps to set and remove breakpoints in an automation.

1. In the Solution Explorer, double-click on the automation to open it in a Designer tab.
2. Right-click on the automation link where you want to place a breakpoint and select **Toggle Breakpoint**. Once set, the breakpoint appears as a red dot over the execution link.

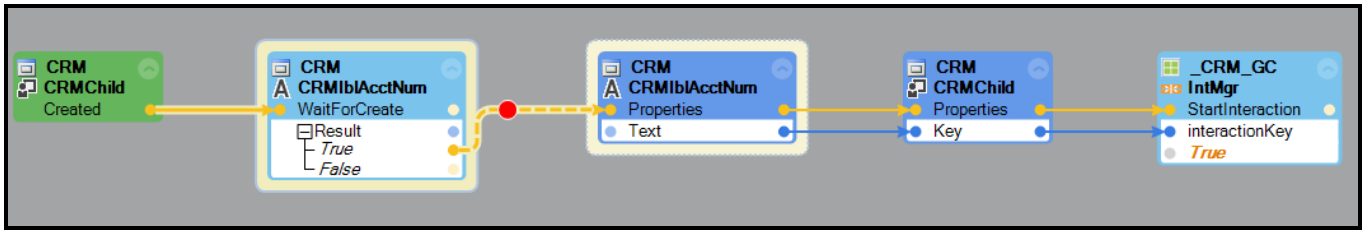


3. To remove the breakpoint, right-click on the automation link where you want to remove a breakpoint and select **Delete**.
4. To remove all breakpoints from a project, from the menu select **Debug > Delete All Breakpoints**.

Stepping through automations

Once you set breakpoints, you can step through the automation. Follow these steps to move through an automation.

1. On your keyboard, press **F5** to begin debugging.
2. Perform any steps necessary to reach the breakpoint. The execution stops at the breakpoint, and the automation displays the location of the breakpoint you reached. The automation link with the breakpoint displays as a flashing dotted line to indicate your current position in the execution.



3. Depending on the debugging you want to perform, step through the automation using the following options:

Option	Description
F5	Continues the execution until the next breakpoint.
F10	Moves the execution one automation link forward along the event path (Debug > Step Over).
F11	Moves the execution to next step along either the event or data path (Debug > Step Into).

Viewing data values

Follow these steps to view data values while debugging.

1. Set at least one breakpoint in an automation.
2. Press **F5** to start debugging.
3. Use the function keys to step through the automation.
4. Place your cursor over a blue data input or output port. A tooltip displays showing the current value of the item.

Diagnostic settings

Because Pega Robotic Automation Studio uses Microsoft® Visual Studio as its platform, the diagnostic components for Pega Robotic Automation Studio are similar. Within Pega Robotic Automation Studio, you can control the diagnostic settings for messaging for both Studio, during development, and Runtime, during debugging. The diagnostic settings are stored in the StudioConfig.xml. The Runtime settings are stored in the RuntimeConfig.xml. You can locate both files in this directory:

C:\Users\{userid}\AppData\Roaming\OpenSpan

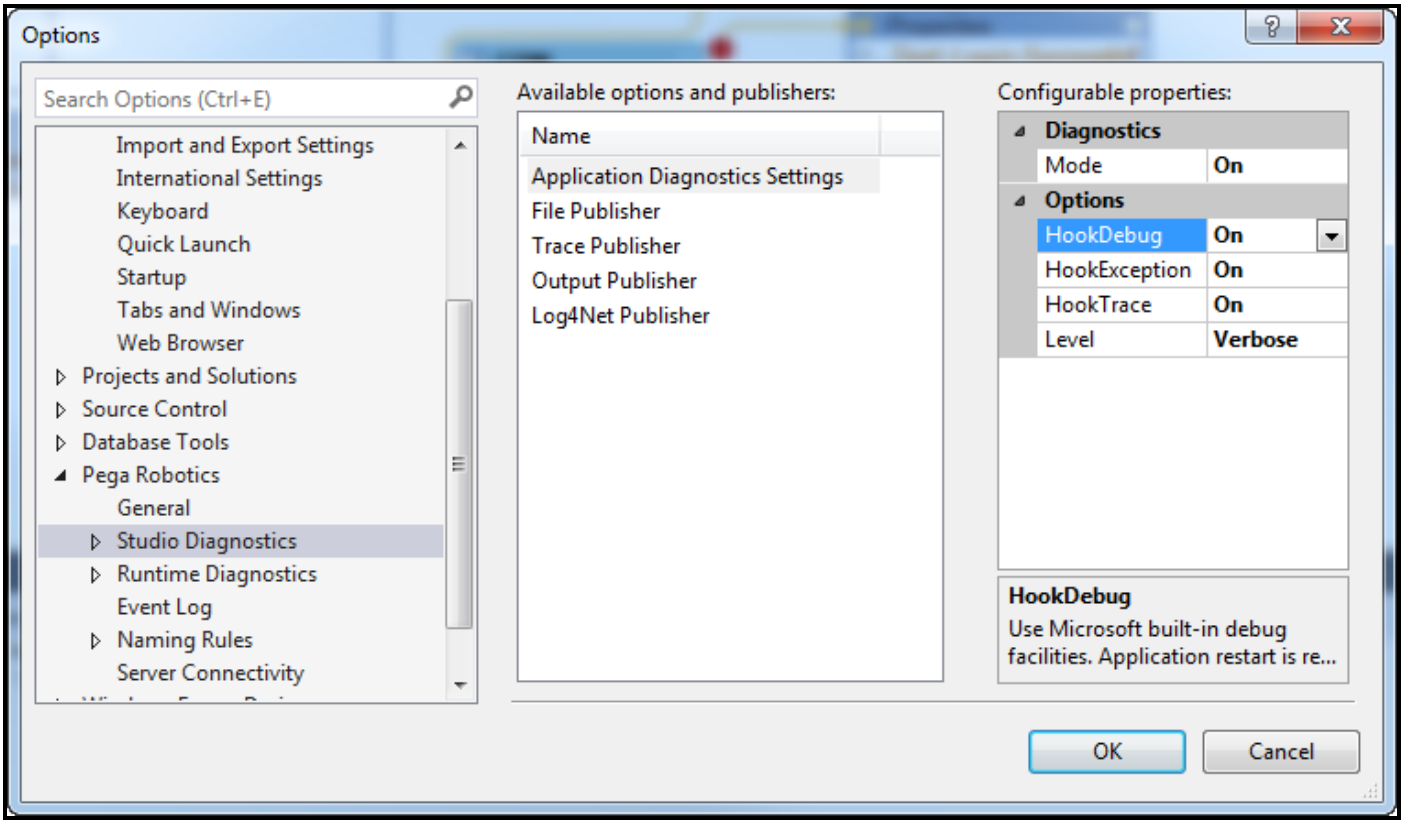
Note: You cannot modify the user's Runtime Diagnostic settings, only your own Runtime settings.

The diagnostic settings consists of two parts:

- Diagnostic message generation
- Publisher Options

Diagnostic message generation

The general category of Application Diagnostic Settings controls the message generation. The settings available for Studio are the same for the Runtime settings.

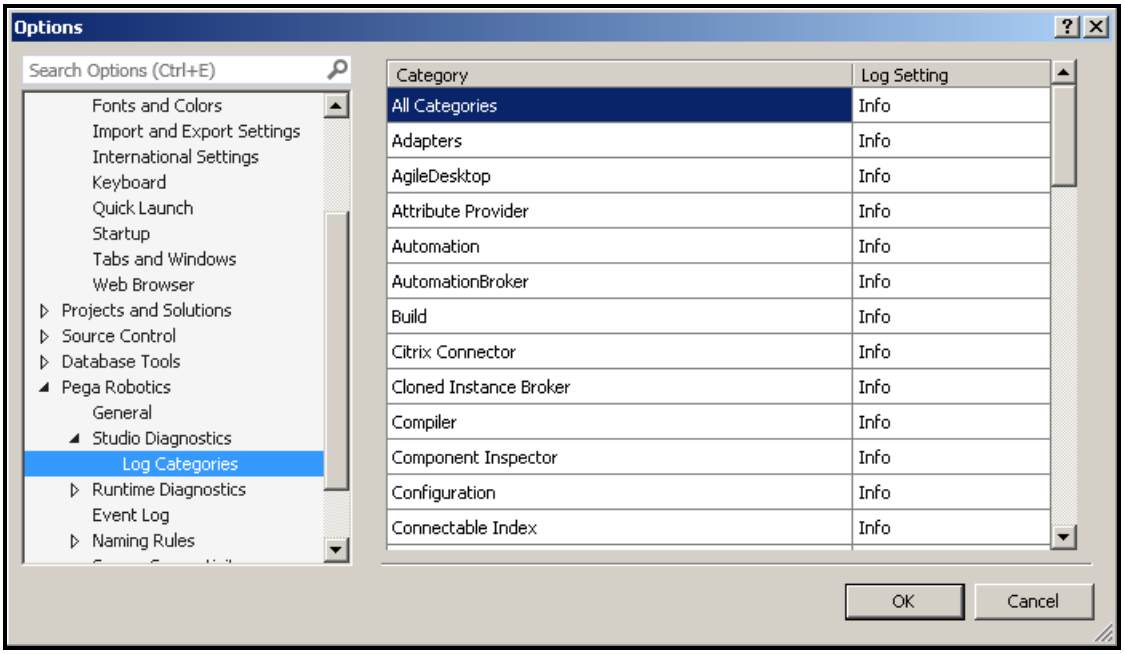


This first category determines if or how the other publisher items work. The properties for the Application Diagnostic Settings are:

Setting	Description
Mode	On or Off. The default is On. Set to On to enables diagnostic logging for OpenSpan Studio (or OpenSpan Runtime). This setting also determines if you can use any of the publishers. Setting this to Off turns off all publishers as well.
HookDebug	On or Off. The default is On. If you change this setting, you must restart the application for this change to take effect. This enables a listener for standard debug messages. This setting applies to development use only.
HookException	On or Off. The default is On. If set to On, this enables UnhandledExceptionHandlers for the main application domain. This feature shows the Final Curtain message if an error occurs that the system does not handle elsewhere in the code base. This also allows logging of the error to any available publisher. If disabled (Off) and such an error occurs a standard Microsoft error message for an unhandled error displays and no log would occurs because of the error. When set to On and an unhandled exception occurs in Pega Robotic Studio or Runtime, the exception information writes to the OpenSpan.Studio.exe.Exception.txt or OpenSpan.Runtime.exe.Exception.txt file.
HookTrace	On or Off. The default is On. If set to On, this enables a diagnostic listener to the standard .NET trace messages, so any Studio publishers messages are routed to system.diagnostic.trace(...) and include them in their log.
Level	This sets a logging level using a Microsoft TraceSwitch object. The TraceLevel sets the severity level for logging messages. The TraceLevel only applies to the startup messages in a diagnostic log such as when a publisher is initialized and the machine information is captured (machine name, version, and so on). All other messages use the severity set in the Log Categories. The default is Verbose. The levels are: • 0 - Off • 1 - Error. This setting records error messages, indicating the application was not able to perform a task as expected. The application is still running.

- 2 - Warning. This setting records both error and warning messages.
- 3 - Info. This setting records error, warning, and informational messages. It includes successful milestones of application execution, regardless of whether the application is working properly, and provides an overview of what happened.
- 4 - Verbose. This setting records error, warning, informational messages and verbose debugging output. This option generates a large number of messages and is not recommended when used with multiple trace source selections.

Log Categories, located under the Studio Diagnostic and Runtime Diagnostic levels, provide a more granular level to set error levels. You should set the error level on the Application Diagnostic Settings to Verbose to ensure that you write all messages regardless of the Log Category error level.



Publisher options

For both Studio diagnostics and Runtime diagnostic, there are 4 publisher types:

- File
- Trace
- Output
- Log4Net

The File Publisher stores messages into these text files:

File	Description
StudioLog.txt	Diagnostic messages from the execution of the Studio application.
RuntimeLog.txt	Diagnostic messages from the execution of the Runtime application.
OSCLog.txt	Messages generated during the build process of a Studio project.
OSDLog.txt	Messages generated during the process of creating a deployment package for a Studio project.

The Trace Publisher enables the generation of diagnostic traces for detailed application messages for Studio and Runtime. Only use the Trace publisher when advised by the Support team.

The Output Publisher displays the diagnostic messages to an Output window in Studio or Runtime.

The contents or types of messages depend on the source (Studio, Runtime, build, and so on) and the error levels set in the Log Categories.

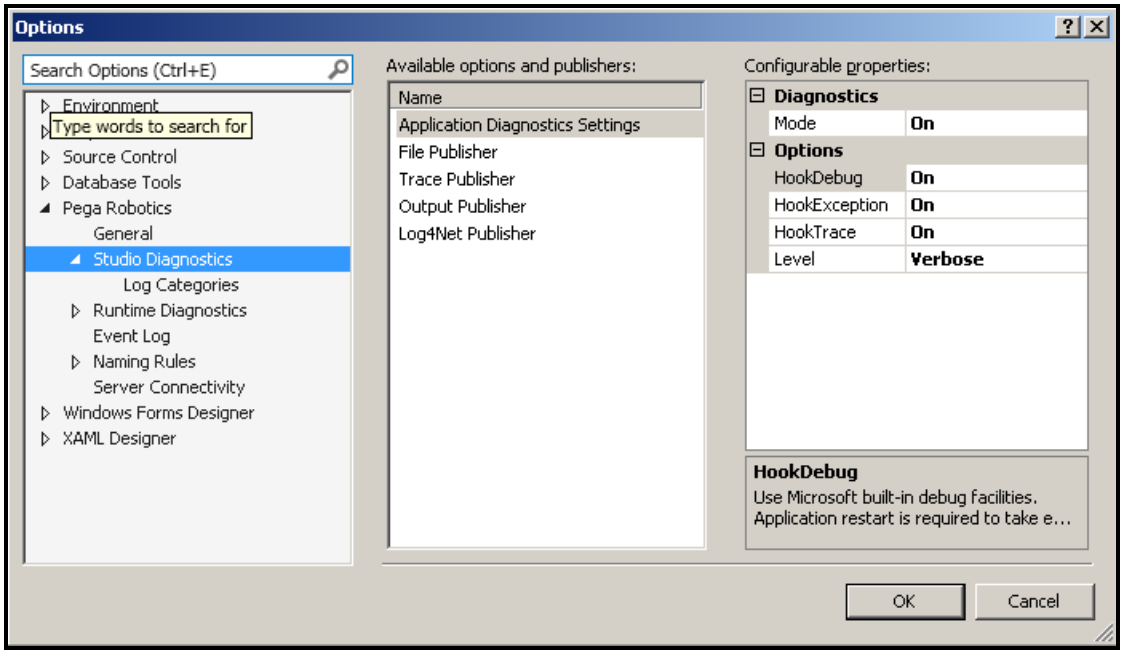
All publishers have the same configurable settings:

Setting	Description
Mode	On or Off. Set to On to enable diagnostic logging for OpenSpan Studio (or OpenSpan Runtime). If the Application Diagnostic Setting is Off, then all publisher modes are Off as well. The defaults vary depending upon the publisher.
ExceptionMode	The default is On. The corresponding publisher mode must be On for this setting to apply. If set to On, the publisher records log messages from diagnostic events that originate from an exception source (HookException). Note that this setting and the Hook Unhandled Exception setting are not the same. .
HookTrace	The default is On. The corresponding publisher mode must be On for this setting to apply. When set to On the publisher records log messages from diagnostic events that originate from a System.Diagnostic.Trace source. Note that this setting and the Hook Trace setting are not the same.

Modifying diagnostic settings

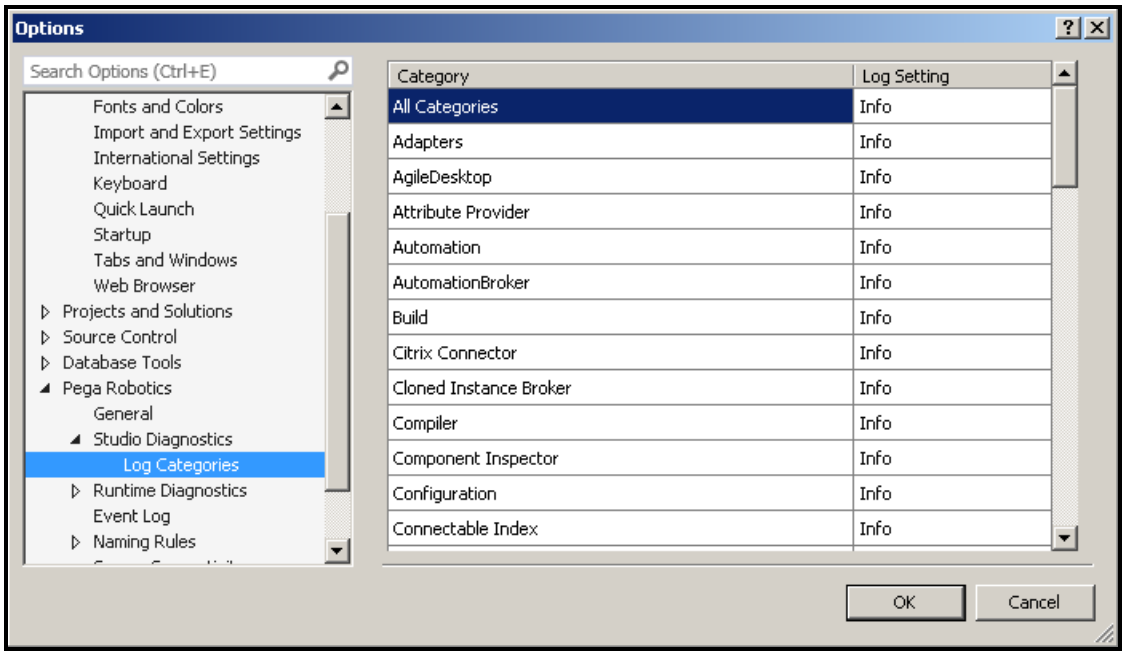
Follow these steps to modify the Studio Diagnostic settings.

1. From the menu, select **Tools > Options**. The Options window displays.



2. On the left frame, locate the Pega Robotics category and click the **Arrow** to expand the category.
3. Locate Studio Diagnostics, and click the **Arrow** to display the Log Categories.
4. Click **Studio Diagnostics** to highlight. The center and right frames update.
5. Depending on the need, select an available option or publisher in the center frame.
6. In the Configurable Properties frame (right), modify the properties as necessary.

7. In the left frame, click **Log Categories**. The window refreshes to display the log categories.



8. Use the All Categories option or individual option to set the error levels of the log categories.
9. Click **OK** to save the changes and close the window.

Note: You can repeat these steps to accommodate your local Runtime diagnostics by accessing the Runtime Diagnostics category on the same Options window.

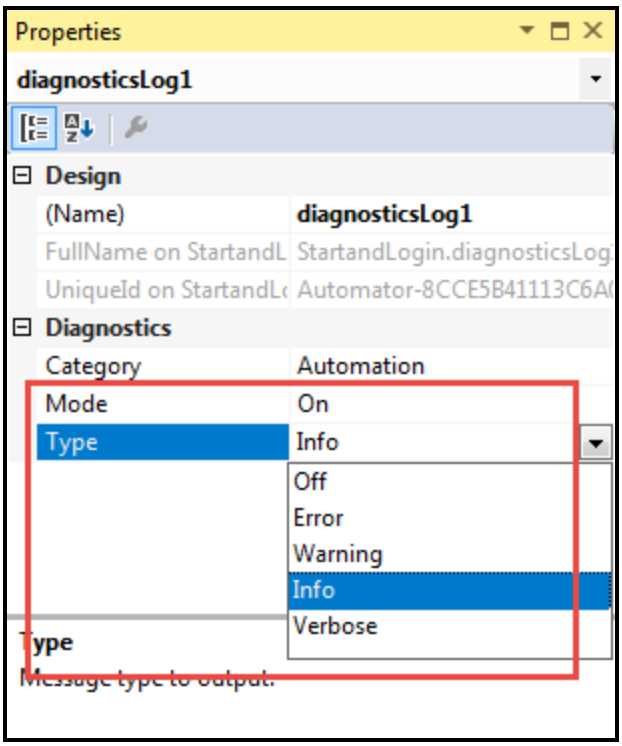
Diagnostic logging

Studio provides a mechanism to create custom log entries to the log files:

- Diagnostic log component.
- Automation link properties.

Diagnostic log component

Studio provides a Toolbox component that lets you write a custom log entry at specific points in a project's execution. You use the logging component in automations where you have access to the components properties to set the specifics of the log entry. To use a logging component, the Runtime Diagnostic File Publisher must be on.

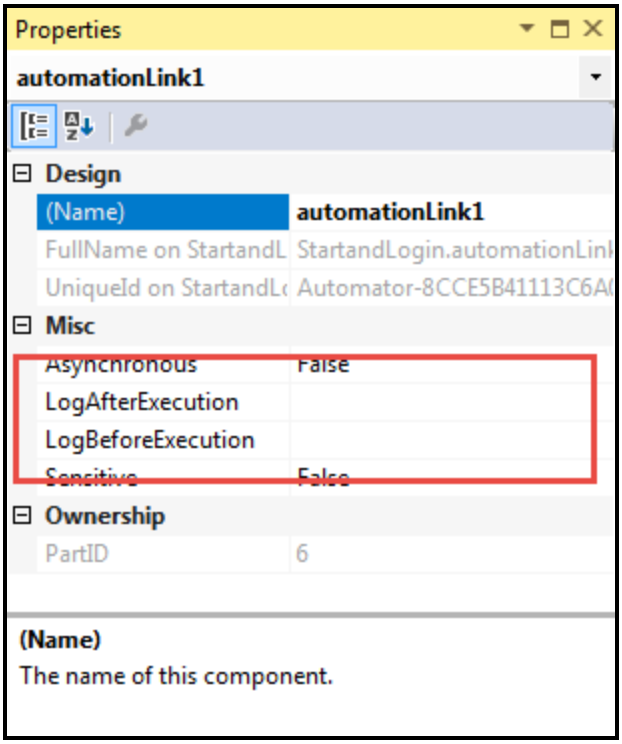


The two important properties of the logging component are:

Property	Description
Mode	On or Off. Set to On to enable diagnostic logging for Pega Robotic Runtime).
Type	This is the error level for the logging component. Select a setting equal to or less restrictive than the Application Diagnostic Setting level. When the type is set Off the logging of the componenet does not stop . The Mode setting manages the logging.

Automation link properties

Each automation link created in an automation has two settings that allow you to create custom messages in a log file.



Property	Description
LogAfterExecution	Writes the value to the log file after the thread completes processing.
LogBeforeExecution	Writes the value to the log file when executing the link.

The difference between the types of diagnostic logging mechanisms is the log component provides the ability to turn on or off the message to the log entry; whereas, the automation link properties always write to the log file until you delete the property value.

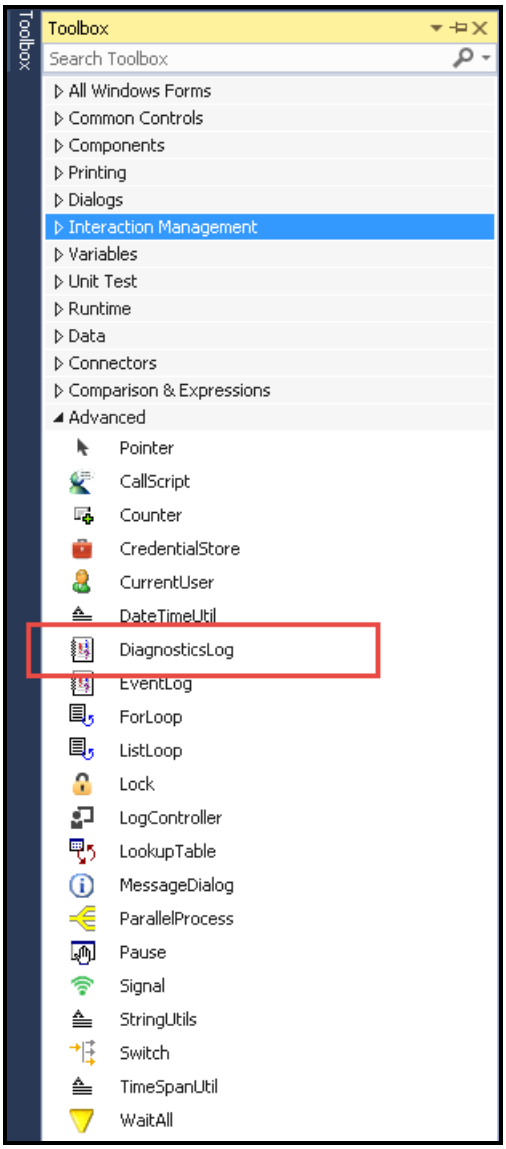
Note: Use unique or special characters to search the log files quickly for the custom entries.

Adding a diagnostic log and automation link property

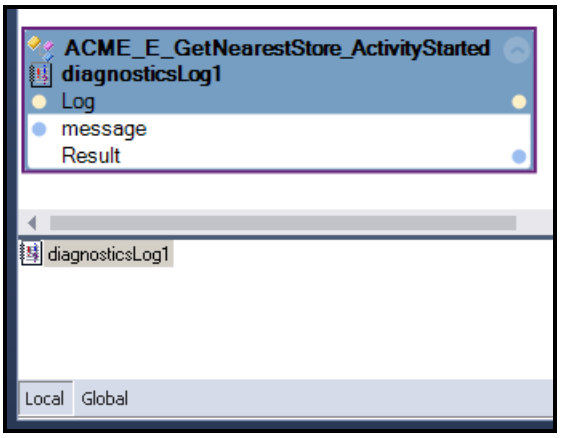
Adding a diagnostic log component

Follow these steps to add a diagnostic log component to an automation.

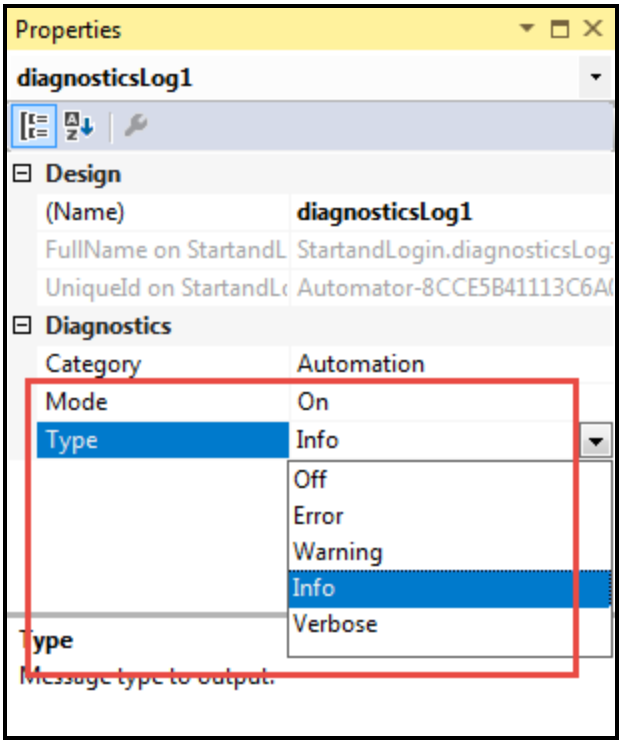
1. Ensure the necessary automation is open in a Designer window.
2. Determine between which existing automation design blocks the log component should be placed.
3. Delete the automation link connecting the two automation design blocks.
4. From the Toolbox window, locate and expand the Advanced category.



5. Click and drag the **DiagnosticsLog** component to the automation. A diagnosticsLog1 displays in the automation and on the Local variable tab.



6. On the diagnosticsLog1 design block, click the **Message** to enter the custom log file entry.
7. Connect the **diagnosticLog1** Design block with the automation links to the automation.
8. In the Object Hierarchy, click the **diagnosticLog1** object. The Properties window updates to display the log component.
9. In the Properties window, modify the Mode and Type properties as needed.

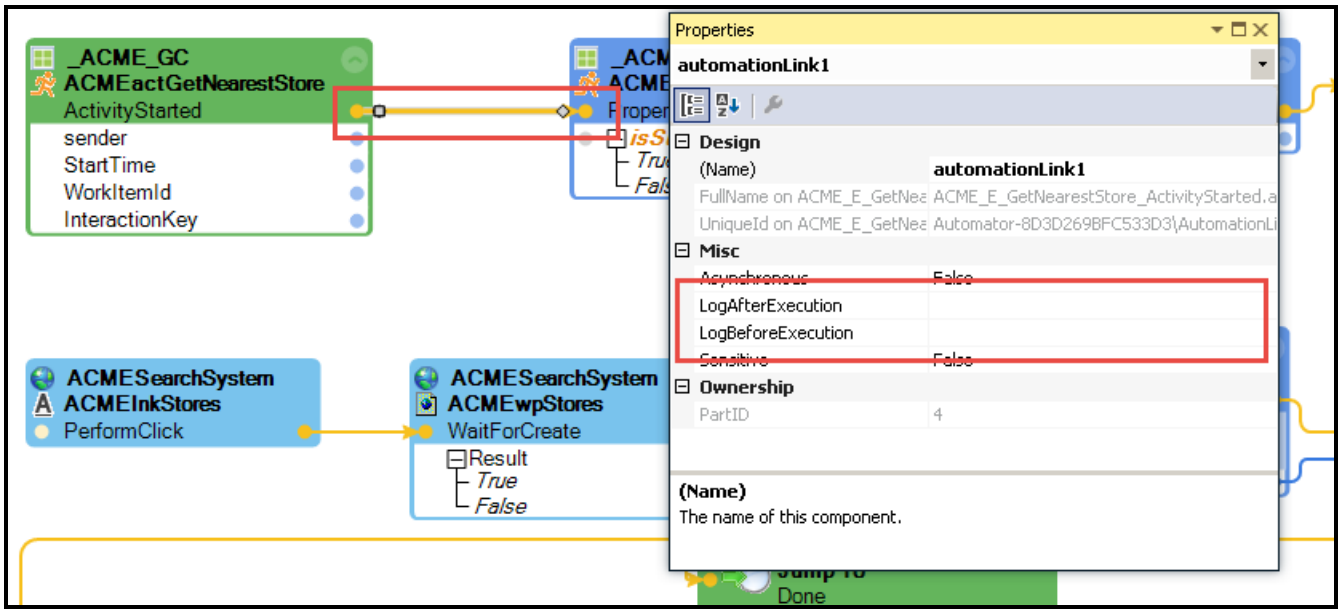


10. From the menu, click **File > Save All** to save your changes.

Modify the automation link properties

Follow these steps to modify the automation link properties to write a custom log entry.

1. Ensure the necessary automation is open in a designer window.
2. Locate the automation link needed for the custom log file entry.
3. Click the automation link to select it. The Properties window updates to display the automation link properties.



4. In the Properties window, click the **LogBeforeExecution** property and enter a custom log entry.
5. In the Properties window, click the **LogAfterExecution** property and enter a custom log entry.
6. From the menu, select **File > Save All** to save the changes.

Error Handling

Introduction to error handling

This lesson describes the different ways to capture and handle errors in a solution.

After this lesson, you should be able to:

- Explain error handling.
- Describe the Try and Catch component.
- Explain how to insert a Try and Catch component.
- Complete the steps to insert a Try and Catch component.
- Explain error suppression.

Error handling

Well-written solutions include error-handling code that allows them to recover from unexpected errors. When an error occurs, the solution may request user intervention, or it may recover on its own. In extreme cases, the application may log the user off or shut down the system.

Error handling enable you to receive and display error information for the solution. Ensure messages, when displayed to the user, contain the following information:

- What happened and why.
- What is the end result for the user.
- What can the user do to prevent it from happening again.

Studio provides at least two methods for handling runtime errors and exceptions.

- Try and Catch component
- Error Suppression

Try and Catch component

A Try and Catch component is a fail-safe logic to catch errors only if they occur within an automation. The Try block begins the logic to alert the system for an error. The Catch block performs actions depending on your configuration of the catch. The automation tool bar contains the Try and Catch blocks.

- The first icon is Insert a Try block.
- The middle icon is Insert a Catch block.
- The third icon is Wrap a Try.

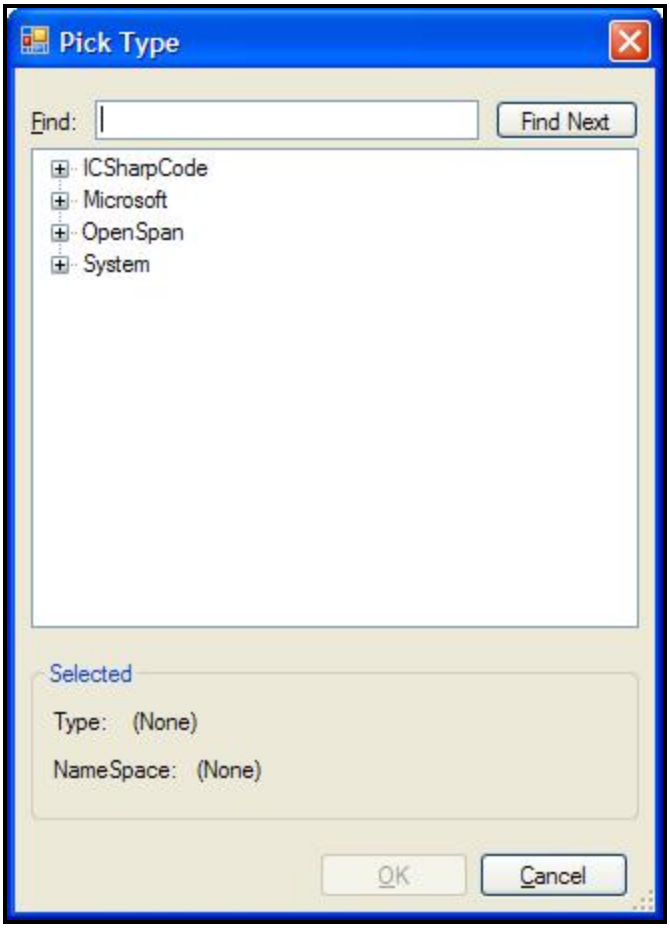
The Catch block provides various outputs to design alternative logical paths for an error. The Catch block has the following outputs:

Output	Description
Catch output	The output continues if no error occurs.
Exception output	The output continues on the exception path if the specified exception type occurs.
Exception Data output	Use this data output to capture the exception data, if necessary.
Message Data output	Use this data output to capture the message data, if necessary.

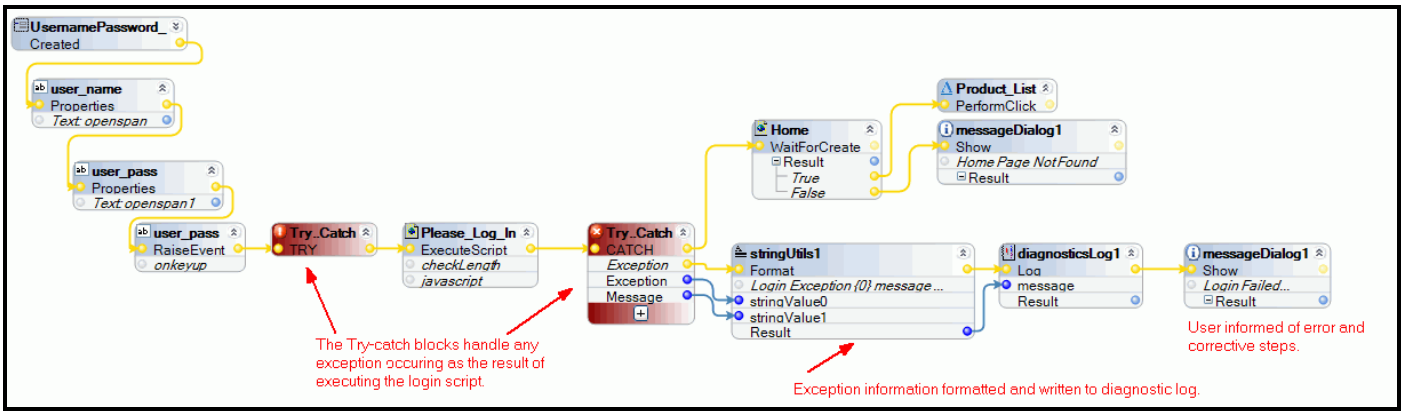
Exception types

The Catch block allows you to define specific exception types to catch. The default option, if not modified, is System exceptions. Click the exception to change its type. The Catch portion also allows you to include more than one exception type as well. Click the plus to add exceptions.

Note: If you have more than one exception type configured on the Catch block, the first error raised causes the automation to follow that output path. The system does not catch any subsequent configured errors if they occur. For example, if you configure three exception types on one Catch and the first error raised matches the second type, the Catch block executes. If a raised error matches exception one or three, the Catch block does not catch those errors.



Below is an example of using a Try and Catch component in an automation.



This automation uses the Try-Catch block to handle an exception that could occur if executing the CheckLength Java script fails for a web login page.

Inserting a Try and Catch component

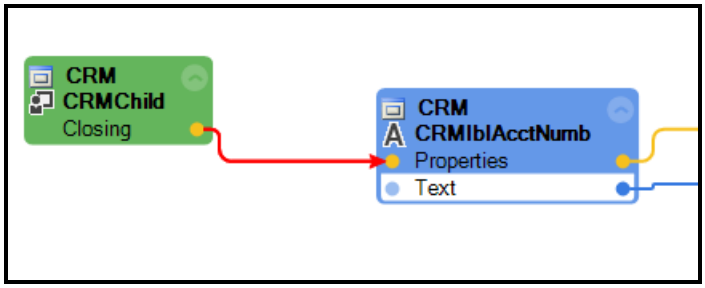
Inserting the Try and Catch components consists of two possible processes.

- Inserting a Try block and a Catch block.
- Wrapping Try and Catch blocks.

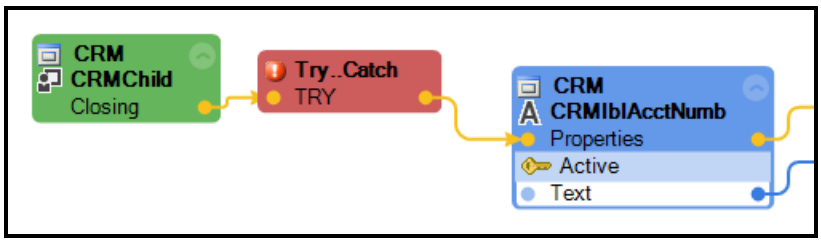
Inserting a Try block and a Catch block

Follow these steps to insert a Try block. The steps below assume you have an automation open in a Design window.

1. On the Automation toolbar, click the **Insert Try block** icon. The button does not change, but the cursor, when placed in the automation Design window, changes to a hand with a Try block icon.
2. In the automation, place the cursor over the automation link where you want to add the Try block. The automation link turns red.



3. Click the **mouse button**. The Try block appears in the automation with the automation links connecting the Try block.

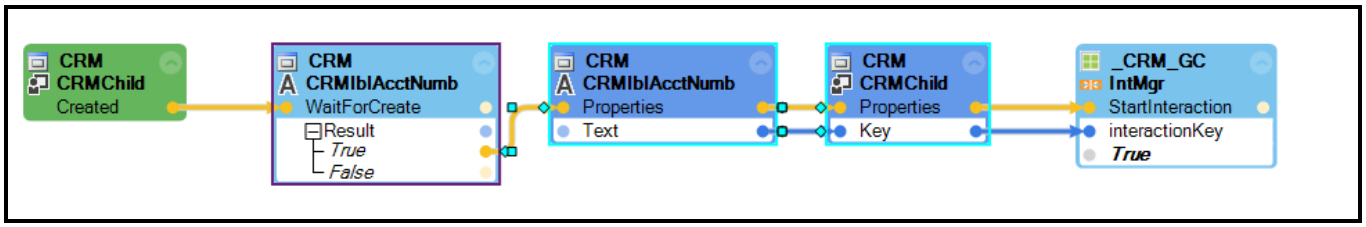


4. On the Automation toolbar, click the **Insert Catch block** icon. The button does not change, but the cursor, when placed in the automation Design window, changes to a hand with a Try block icon.
5. In the automation, place the cursor over the automation link where you want to add the Try block. All automation links from the Try block to the Catch insertion turn red, displaying the section of code to watch for an error.
6. Click the **mouse button**. The Catch block appears in the automation with the automation links connecting the Catch block.

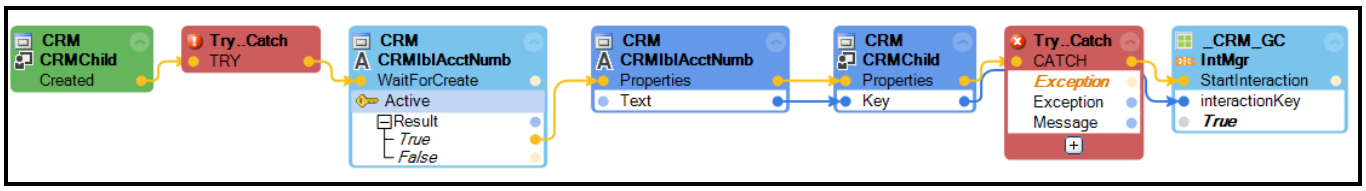
Wrapping a Try and Catch block

Follow these steps to wrap a section of automation code in a Try and Catch block. The steps below assume you have an automation open in a Design window.

1. In the automation, click at least one design block or select a group of design blocks. The highlighted design blocks indicate where the Try and Catch blocks go.



2. On the Automation toolbar, click the **Wrap in a Try..Catch Block** icon. The automation updates placing the Try block in front of the first design block highlighted and the Catch block at the end. All automation links connect.



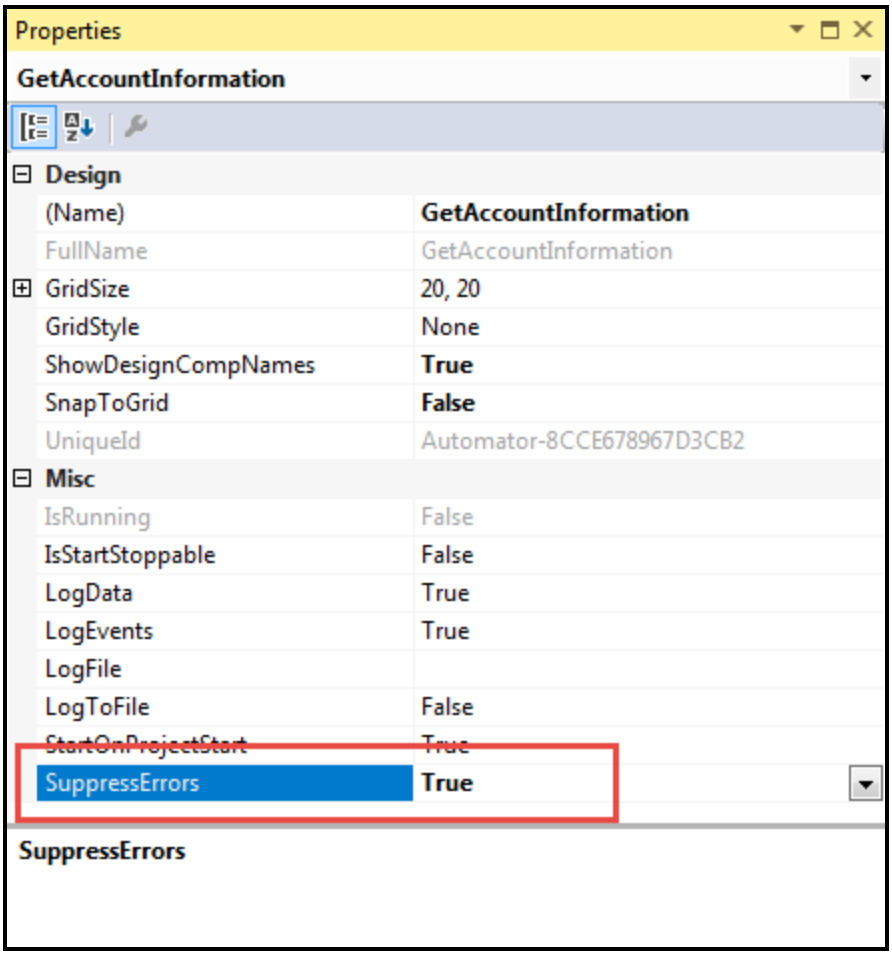
Error suppression

Often, you may encounter an error that consistently appears without causing issues to the automation. This could arise from a known error in the application. For errors that appear but do not cause issues , Studio can suppress these errors for automations and for the adapter. By suppressing the automation and/or adapter errors, users do not see an error message dialog in the event that an exception occurs.

Despite error suppression, you can track the exceptions by using the diagnostics reporting to log errors to the necessary log files.

Automation errors

Suppressing automation errors occurs on the Properties window of the automation. Select the automation in the Object Explorer and change the **SuppressErrors** property to **True**.



Adapter errors

You suppress adapter errors in the RuntimeConfig.xml file located on each user's workstation. To suppress adapter errors, change the associated key value to **true**, as shown below:

<add key="SuppressAdapterExceptions" value="true" />

Note: Change the adapter error suppression key to True suppresses all adapter errors in the solution.

WEB INTEGRATION

This lesson group includes the following lessons:

- Working with web adapters
- Working with match rules
- Navigating through a web application

- Working with Interaction Framework
- Integration with Pega 7 Platform

Working with web adapters

Introduction to web adapters

In this lesson you learn about web adapters, web adapter properties, interrogating web applications, and global web pages.

After this lesson, you should be able to:

- Describe the web adapter properties
- Complete the steps to add a web project and web adapter
- Explain different interrogating web applications methods
- Explain how global web pages influence web application interrogation
- Interrogate a web application

Web adapter properties

A second widely used adapter is the web adapter. Web and window adapters share some properties. Developers should be able to recognize the most used web properties to understand how they influence the application's use in the solution.

The following adapter properties apply to a web application:

- StartPage
- IgnoreMainBrowser
- StartOnProjectStart

StartPage property

The **StartPage** property specifies the first HTML page to open for the application. The property accepts the site location in several formats. Here are some examples:

- www.pega.com
- http://216.215.233.67/
- Z:\Inetpub\wwwroot\index.htm

The StartPage property is synonymous with the Path property for the windows application. However, the StartPage property is optional for a web application. If left blank, the adapter opens the browser and navigates to the users home page. The StartPage property sets the page to load when the adapter starts.

IgnoreMainBrowser property

The **IgnoreMainBrowser** property controls how Studio interacts with the browser target. If you set this property to True, Studio ignores the browser control while the project is running. The Browser events or methods are not available for use in automations (with the exception of the Created and Destroyed events).

Note: Only set the IgnoreMainBrowser property to True when an additional web application windows open, such as Alert Panels and the web application freezes while running your project. .

Important: The IgnoreMainBrowser propety only applies to the main (top-level) browser control. If the application contains browser controls within frames, all events and methods for these controls are available.

StartOnProjectStart property

The **StartOnProjectStart** property applies to all Studio adapters. This property controls when an adapter starts during the course of loading and running the project. The default is True, which means the adapter starts when the project runs. If you set this property to False, specific automation logic is required, such as executing the Adapter.Start method, to start the adapter.

Note: Starting the adapter does not automatically launch the associated adapter application. The application launches depending on the following:

- Web Applications: When the adapter starts, Studio launches the web browser, navigates to the application identified in the StartPage property, and hooks the application. The interrogated controls are available to the project automations or event monitoring.
- Windows Applications: When the adapter starts, the StartMethod determines whether Studio launches the associated Path application.

Target Creation

One important aspect of web applications that you must consider is the amount of time a web page spends to load. Some web applications load quickly, while others do not. Numerous variables can cause this variance, such as time of day, number of users, and background application processing.

When automating actions that interact with web pages and their controls, you should make sure the target pages and controls are available before your automation sends data or events to the controls. Studio provides properties, events, and methods to make sure targets are available and matched.

Created event

When Studio matches a target, the Created event for the object triggers. For example, in the CRM_E_CRMChild automation you created in the CRMAdapter project, when the CRM child is fully loaded, a Created event triggers. Remember the Created event for any object triggers when the object matches, meaning navigating back to a previous page retriggers the Created event for all interrogated objects on that page.

IsCreated property

This property is False until the object's Created event triggers. Returning to the example of the CRM_E_CRMfrmLogin automation you created in the CRMAdapter project, the CRMfrmLogin window IsCreated property turns True when it matches the Studio match rules for the object. When you click the Login button, the window un-matches, and the IsCreated property returns to False. The IsCreated property is one property that a developer cannot modify through an automation.

WaitForCreate method

The **WaitForCreate** method sets up automation logic to wait until the Created event triggers for an object before proceeding to the next step of the automation. The method provides a Boolean result of the Created event. If the Created event occurs within the wait time, the automation continues through the True logic. If it does not occur within the wait time, the automation continues through the False logic.

If the WaitForCreate method is not used, all objects have an implicit wait of 30000 milliseconds (30 seconds) by default. This means that Studio waits until the Created event triggers for an object. After 30 seconds, the next step in the automation is attempted.

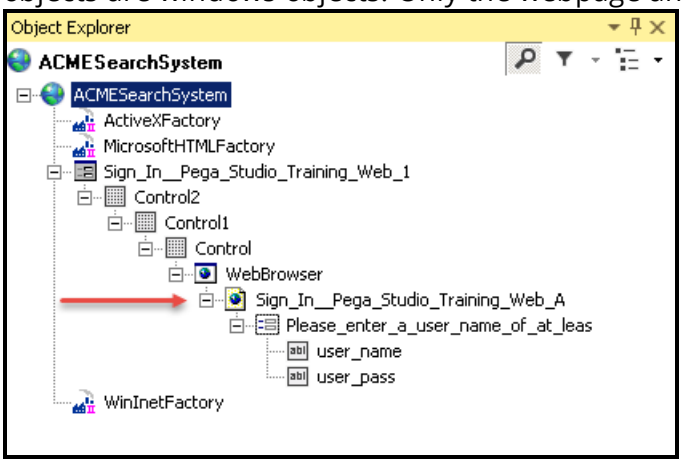
Using the implicit wait time is not the preferred application of the logic. You should use the method because it provides logic to the automation through the method's Boolean result.

Web application interrogation

Web application interrogation is similar to windows application interrogation. Without modifying the web page source code, Studio evaluates an object's properties against match rules to ensure the object is a unique control every time the application runs within the solution. During interrogation, a green check mark displays when an object matches.

The major difference between the interrogations of web applications and windows applications is the code used for matching. Since web applications use HTML code, the interrogation process reviews the HTML tags to determine the object type. If Studio can use the object type to match the control, the process completes. If Studio cannot use the object type, Studio uses the attributes found inside the HTML tag to differentiate between similar object types.

When reviewing the Object Hierarchy of an interrogated web application, the objects differ from window-based objects to web-based objects. The first web-based object shown in the image below is the Sign_In webpage. From there all webpage's children are web objects. All objects above the webpage are windows-based object. The web browser is a windows application in structure, so those objects are windows objects. Only the webpage and its children are web objects.



Note: When interrogating a web application, set the Zoom property to 100%.

Global web pages

Global web page is a function of Studio that allows the interrogation process to ignore all window based objects of the browser and interrogate only the web-based objects. This functionality arose from two reasons:

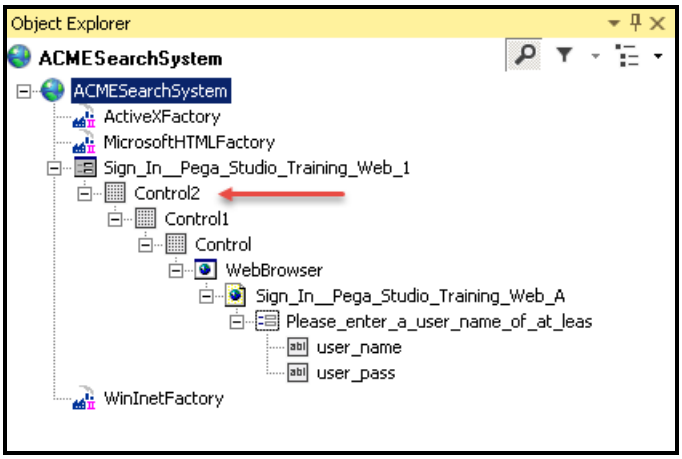
- Internet Explorer changes between versions 7 and 8.
- Use of web page frames.

The Create Global Web Page function is the default option when starting the web interrogation process.

Internet Explorer versions

When Internet Explorer version 8 was released,, a new feature was available – tabs. No longer did a user have to launch a new browser window to surf a second website. But, the new tab structure caused issues with existing solutions when upgraded.

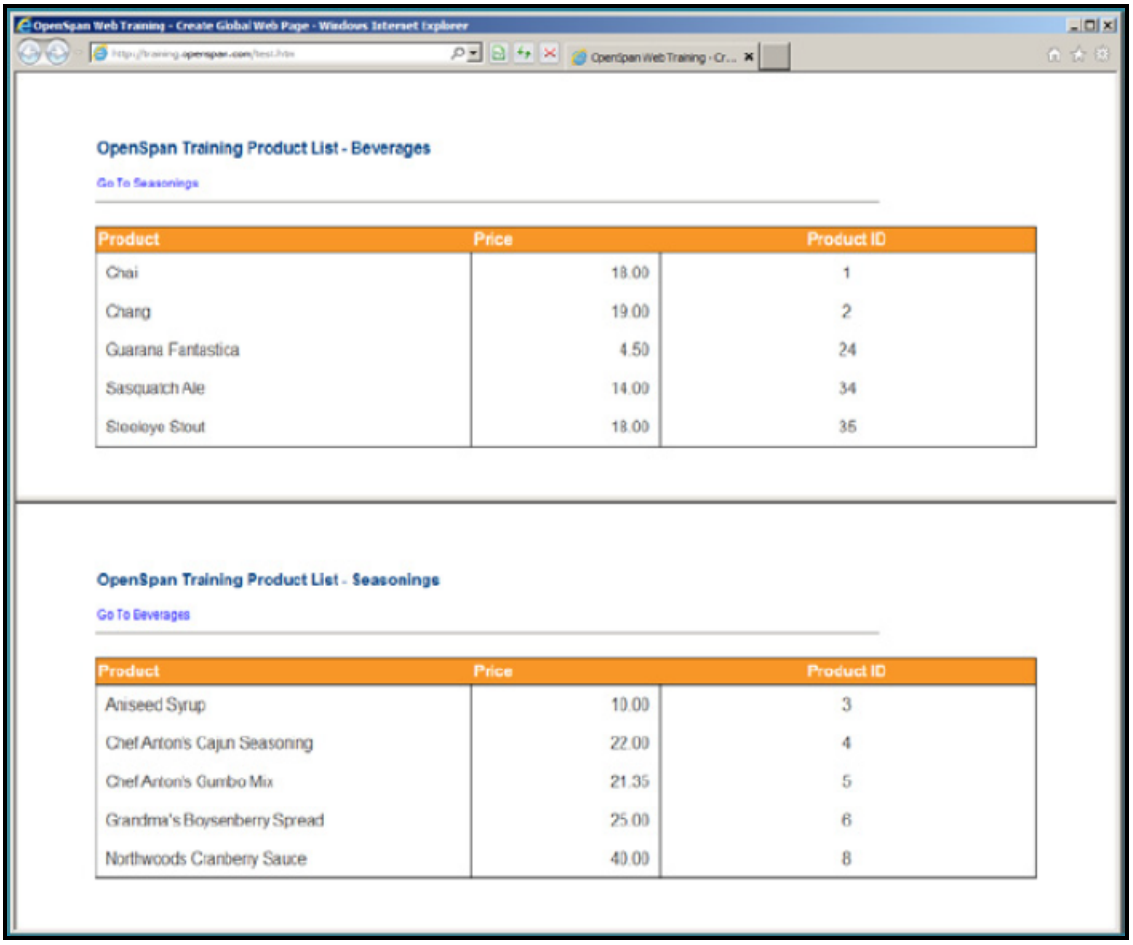
With the new tab, a new control displayed in the hierarchy – a control that did not exist on previously interrogated web applications. Therefore, all solutions failed because the parent/child relationship was different. In the image below, the Control2 level is the version 8 Tab object. No Control2 level children matched because Control2 was not part of the initial interrogation.



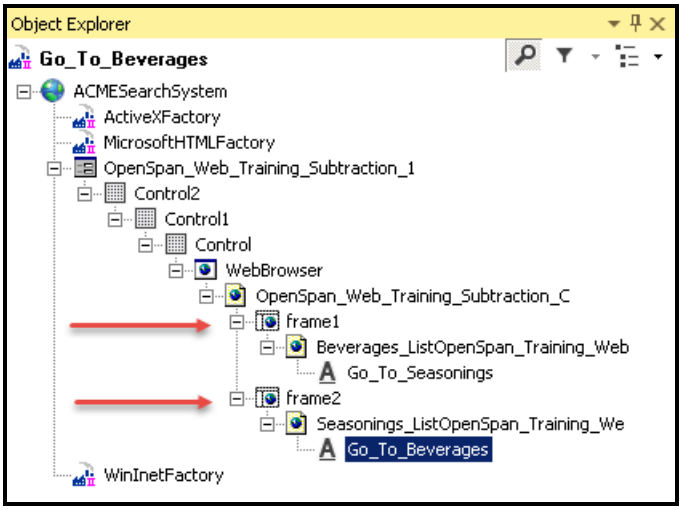
Using the Create Global Web Page function you can make the Object Hierarchy and its objects browser version independent.

Web page frames

Another feature used often on web pages is multiple frames. Multiple frames is a structure that allows a developer to use one web page to display other website pages all on one page. The following image shows the multiple frame web page.

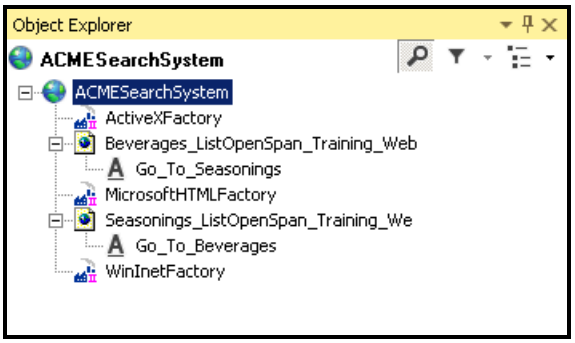


When interrogating the webpage shown above without the Create Global Web Page function enabled, the Object Hierarchy resembles the following image:



Notice the use of the two frame objects below the main web page. Each frame object also contains a web page. Suppose you also used the Beverages pages by itself in the solution. When the Beverages webpage is interrogated from this structure, the solution fails when accessing the Beverages webpage without the frame. To get the framed webpages to work regardless of their use in a multiple framed

webpage or as their own web page, use Create Global Web Page. Interrogating the same multiple frame webpage using the Create Global Web Page, the hierarchy resembles the following image:



Now, regardless of how users access the pages, they match each time because the abbreviated hierarchy structure only contains the webpage object and its children.

Interrogating a web application

The standard interrogating process using the Target Icon still exists. Two alternative processes are available as well.

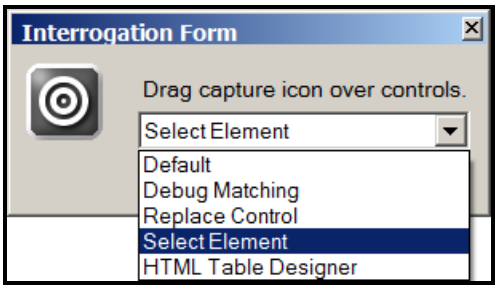
1. Using Select Element.
2. Using Web Controls.

Using Select Element

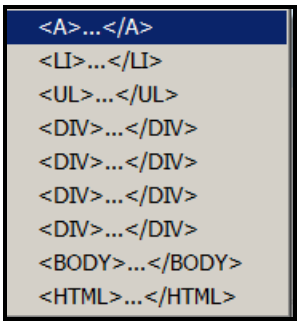
Dragging the Interrogator target (bulls-eye) icon over a target on a web page may not always provide enough precision to highlight the target needed for the solution. When this occurs, use the **Select Element** option.

The Select Element option displays a list of HTML tags encasing the target selected on the web page. You can then select the target based on the HTML tag surrounding the target.

1. In the Solution Explorer, double-click on a web adapter project to open it in the design window.
2. In the design window, click **Start Interrogation**. The Interrogation form window displays and the browser opens.
3. From the **Interrogation Form** window, select **Select Element** from the combo box.



4. Using the Target icon, click and drag the Target icon to highlight the required object.
5. When the object highlights, release the mouse button. A Context menu displays.



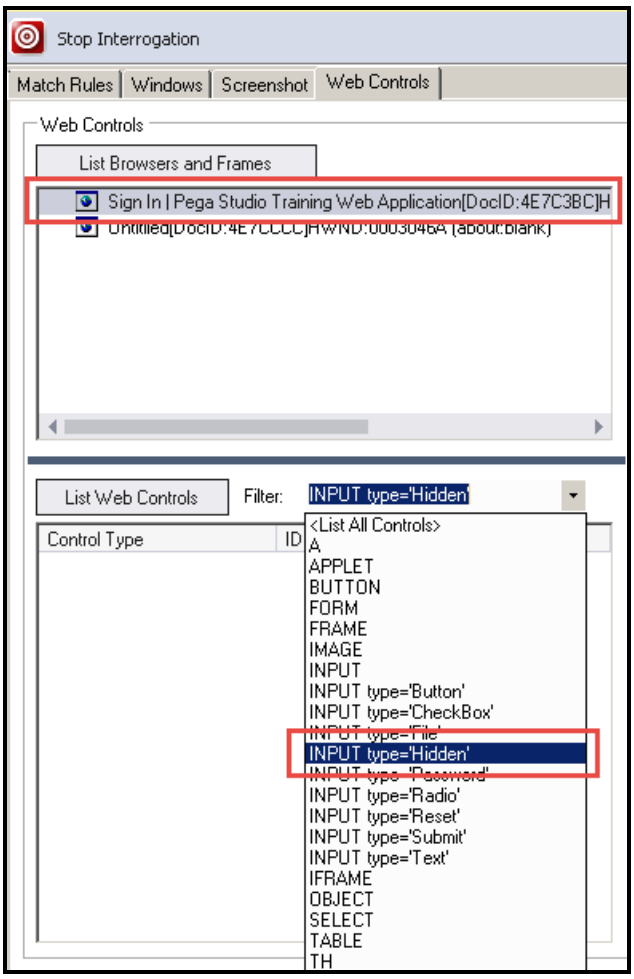
6. On the Context menu, move the cursor over each HTML tag. Each object highlights to ensure the correct selection.
7. Click the corresponding HTML tag on the context menu to interrogate the object. The matched object displays in the Object Hierarchy.

Using Web Controls

Developers often use hidden controls in websites to store data or important information needed for the application but not for the eyes of the user. Since you cannot see the label, you cannot drop the target icon on it.

Use the Web Controls tab on the Adapter Design window to locate the hidden object and add it to your solution. The Web Controls tab also provides a convenient way to list all web targets from a page currently open and matched in the interrogator.

1. In the Solution Explorer, double-click on a web adapter project to open it in the design window.
2. In the design window, click **Start Interrogation**. The Interrogation form window displays and the browser opens.
3. Navigate in the web application to the desired web page.
4. Minimize the browser window to bring the Adapter Design window into view.
5. On the Adapter Design window, click the **Web Controls** tab.
6. On the Web Controls tab, click **List Browsers and Frames**. The browser list populates.
7. In the browser list, select the webpage to highlight it.
8. In the Filter combo box on the Web Controls tab, select I**nput type = 'Hidden'**. You can use this method to search for any web object from the list.



9. On the Web Controls tab, click **List Web Controls**. A list populates.
10. If needed, right-click on the object in the web controls list to highlight it on the web page.
11. On the web control list, right-click on the control and select **Create Control**. The matched object appears in the Object Hierarchy.
12. Complete the interrogation process.

Working with match rules

Introduction to match rules

In this lesson, you will learn what a match rule is and how match rules work in Studio.

After this lesson, you should be able to:

- Describe matching and match rule concepts.
- Describe web-based match rules.
- Describe the steps to modify a match rule.
- Complete the steps to use the Attribute Value match rule.
- Explain situations where ambiguous matching may occur.
- Complete the steps to use the Element Index match rule.

Web-based default match rules

For web applications, the persistent data available to identify an element or control are the HTML tags. The tags provide a standardized format that allows Studio to match the element consistently. While there exists web-based default match rules that are uniquely associated to each object type, there also exists default web control match rules. A developer can use any of these rules if the object match rules do not work.

Below is a sample of the web object default match rules.

Object Name		
Anchor • Anchor Target • Anchor URL	Image • Image Alternate Text • Image Name • Image Source	Table • Table Border • Table Height • Table Width
Form • Form Method • Form Name • Form Target • Frame Name • Frame Source	Input • Image Button Alternate Text • Image Button Source • Input Index • Input Name • Input Size • Input Type • Input Text	Table Data Cell (TD) • Table Cell Column Index • Table Cell Height • Table Cell Padding • Table Cell Row • Table Cell Row Index • Table Cell Spacing • Table Cell Width

If Studio cannot match the control based solely on the web object rules, it uses standard default match rules:

- Attribute Value
- Control Children
- Element ID
- Element Index
- Element Inner Text
- Element Path

Studio may use one or more of the above match rules to match a target in combination with the web object match rules as well.

Element ID

The most used default match rule is the **Element ID** match rule. HTML designates every object on the web page with a unique identifier, thus Studio uses this rule the most. However, there are instances where it may cause trouble.

If a developer lets the web page assign the element ID for all objects, the web page creates a new element ID for the objects each time the pages loads. Therefore, on the initial interrogation, the Element ID rule may have an initial value. When testing the solution, the object fails to match when the application runs because the element ID is now different.

Attribute Value match rule

All HTML tags have pre-defined properties or attributes for a developer to use to distinguish one similar tag from another. Since web developers can decide which attributes to use, when to use the attribute, as well as what values to use for the attributes, a Studio developer can use the **Attribute Value** match rule to define a unique attribute and value.

Element Index match rule

The **Element Index** match rule simply generates the index of the HTML tag of the control relative to the other tabs of the same type. The index numbering starts at zero. If three <p> tags exist on the web page, the first tag has an index of 0, the second of 1, and the third of 2. Use this rule sparingly and as a last quick fix. Web pages are too dynamic and the index can change too often.

Element Path match rule

The **Element Path** match rule is the routing of the HTML tags surrounding the interrogated object. Using the Select Element function during interrogation displays the element path of the interrogated object on the context menu.

Modifying match rules

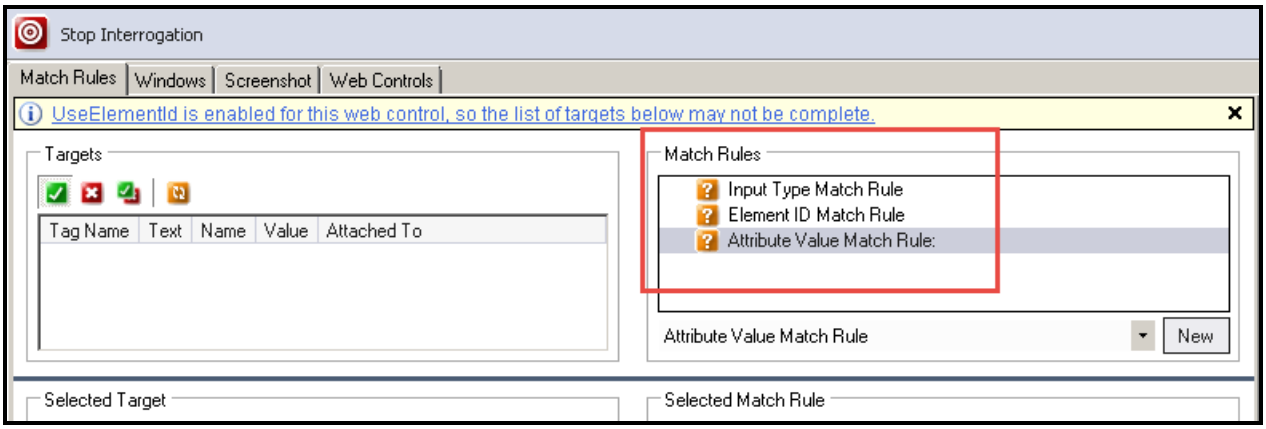
Due to the dynamic nature of web applications, rules can easily and quickly be changed without notice, which creates more opportunities to modify and update match rules. However, the process and logic in editing and changing match rules applies for both windows and web applications. Before any match rule modification can occur, you must have the interrogation process running.

The first steps to consider are adding and removing match rules.

Adding match rules

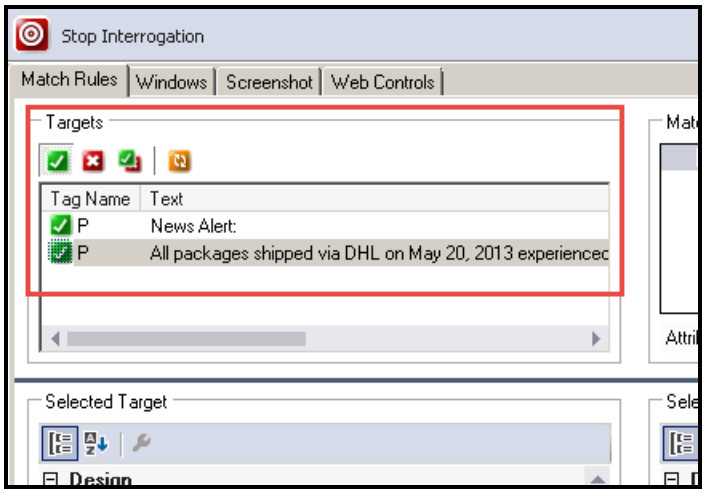
Follow these steps to add a match rule.

1. In the Adapter Design window in the Targets frame, select the control to highlight it.
2. In the Match Rules frame, select a new match rule from the combo box. Only the match rules available for the object type display in the combo box.
3. Click **New** to add the new match rule to the Match Rules frame. You can use a match rule once per object. Once added, Studio removes it from the combo box list. If the rule requires manual input, the green check marks turn to orange question marks.



4. Make any necessary adjustments to the rules.

5. Click **Refresh** to accept the manual changes.



Removing a match rule

Follow these steps to remove a match rule.

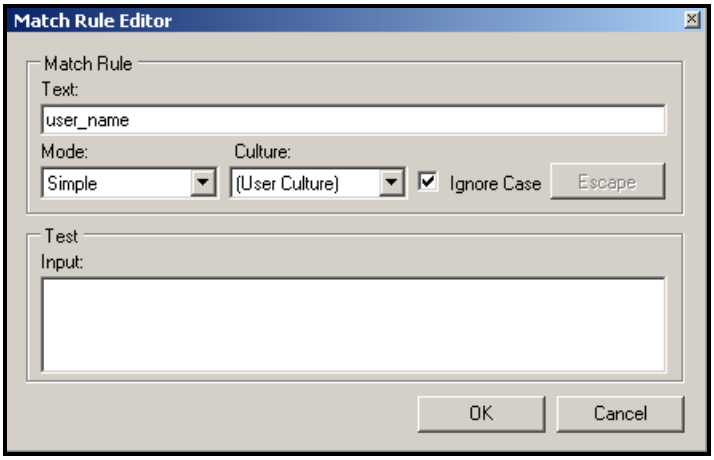
1. From the Match Rule frame, highlight the match rule you want to delete.
2. Right-click on the match rule and select **Delete**.

Using the Match Rule Editor

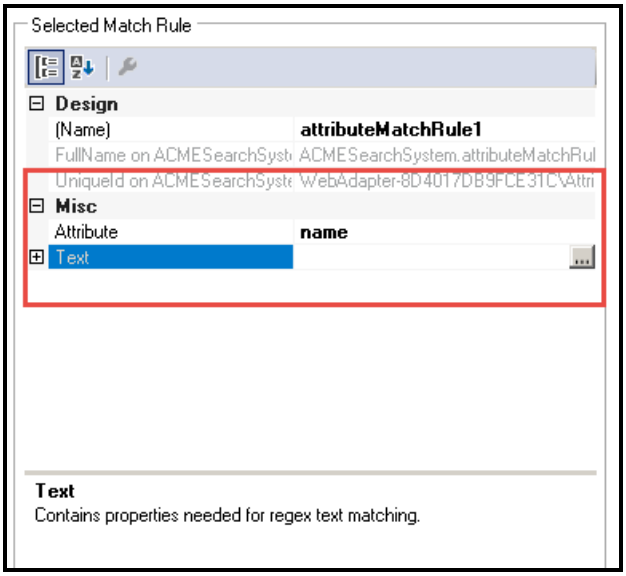
Often, Studio uses match rules that validate against any text associated with the object. Windows forms and web pages may contain text that changes with each user or each account accessed. This dynamic text causes issues because the match rule considers the text static or persistent text. You must modify text-based match rules to eliminate dynamic text and evaluate the object solely on the static text. Studio provides a Match Rule Editor window to perform this task.

The Match Rule Editor consists of the following options:

Property	Description
Text	The target object label text, such as the window title bar text or a control button label. For the sample CRM.exe application, the Login window text is Login. Modify the text to support the Mode selection.
Culture	Language (culture) used by target application users. The default is User Culture, which uses the operating system setting (see region and language options under Control Panel).
IgnoreCase	The default is True, which indicates that text can appear in either upper- or lowercase.
Mode	The default is Simple, which means the text must match literally. Mode types available include Regex, StartsWith, EndsWith. Regex mode indicates that the Text property contains Regex syntax. Select the mode in conjunction with the Text property value.



If the match rule requires manual text edit, in the Selected Match Rule frame an ellipses button displays that allows access to the editor for modifications of the rule parameters.



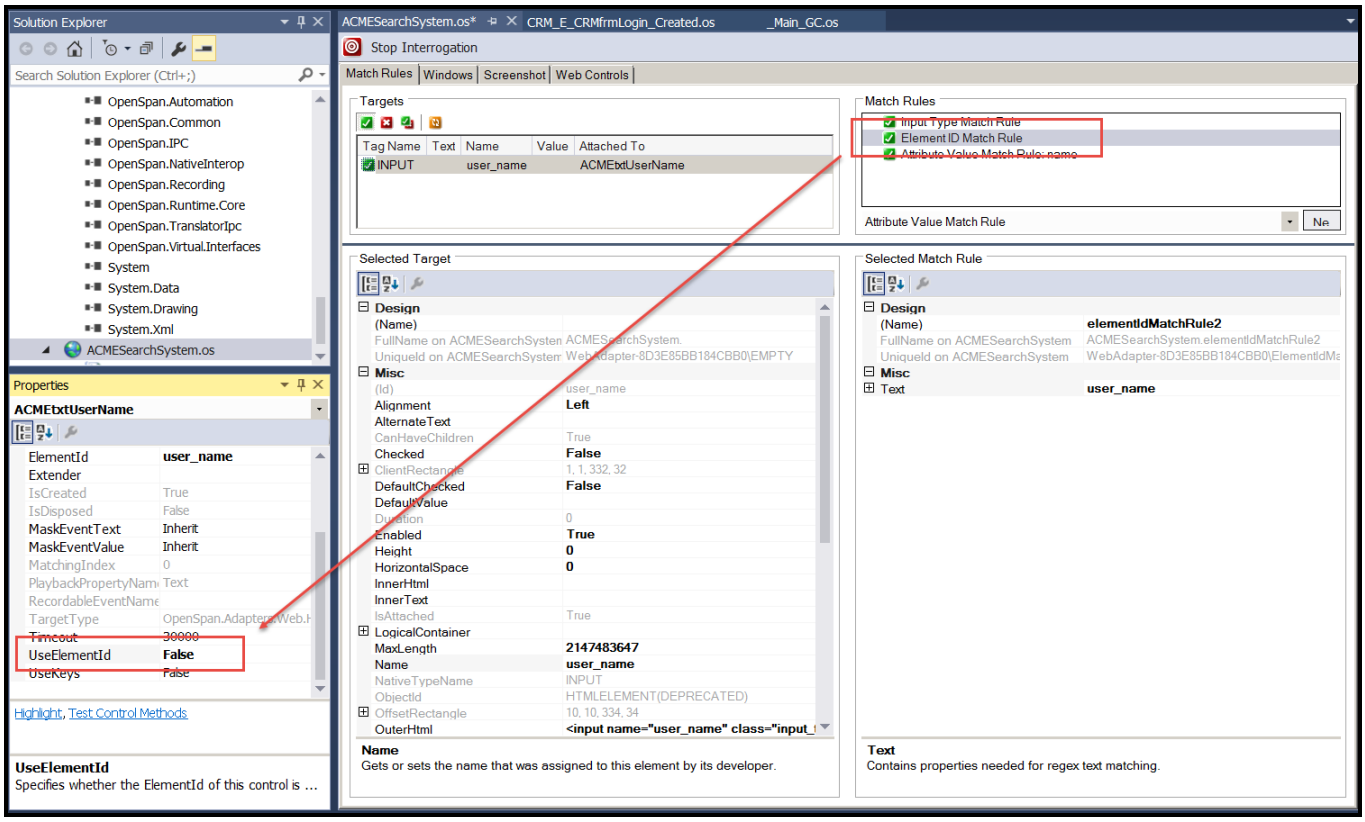
Removing with Element ID

With web applications, the Element ID match rule causes objects not to match. Studio presents a message to you if it believes the Element ID match rule is the problem. Use the Debug Matching option on the Interrogation Form to complete the process.

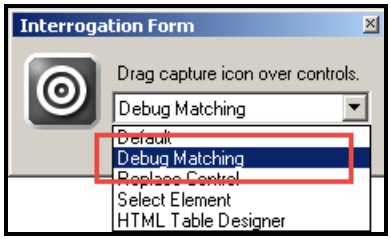
Follow these steps to remove Element ID match rule from the control.

1. In the Object Hierarchy, highlight the control using ElementID match rule.
2. In the Properties window, locate the UseElementID property.

3. Change the **UseElementID** property to **False**.



4. In the Adapter Design window, in the Targets frame, select the correct control to modify match rules.
5. In the Match Rules frame, right-click and delete the **Element ID Match Rule.**
6. In the Interrogation Form, change the combo box from **Default to Debug Matching**.



7. Bring the browser back to focus from the Taskbar.
8. Re-interrogate the same object selected in the Object Hierarchy. Studio now ignores the Element ID rule, and searches for other match rules to match the control.

Match rules

Every time an application runs a new user interface, the underlying program renders the interface. To users, the interface appears the same because certain characteristics, such as the type and position of application interface elements (labels, buttons, lists), remain the same. However, from a technical perspective, the application interface elements are different for every instance of the application.

To automate an application, a Studio developer must identify an application interface element across multiple application instances, just as a user would. Additionally, the developer must distinguish between unchanging or persistent data used to identify an element and the changing or transient data the automation should ignore.

Studio provides an advanced matching system that uniquely identifies application interface elements across multiple instances of an application using a set of rules that capture the necessary persistent data. Developers can customize matching behavior by adding, removing, or modifying these rules.

At runtime, as applications run and the targets within the applications display on and remove from the user interface, Studio dynamically applies the match rules for the controls you interrogated. If a target is not currently displaying, then the match rules for the target have failed.

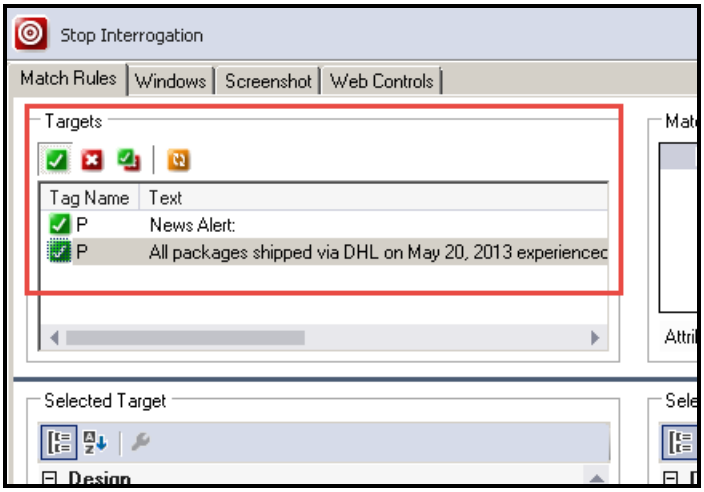
When the target displays, Studio uses the properties of the target and compares them to the match rules associated with the corresponding control in Studio. At runtime, if the properties of the target satisfy the requirements of the control's match rules, the target matches, and the control is available for the project automations.

Since controls exist within a hierarchy, when a control fails to match, all associated child controls also fail to match. Studio applies the rules in the most efficient way so that the matching process is quick and runtime maintains the application performance.

Recognizing ambiguous matching

Ambiguous matching occurs when the default match rules fail to identify a single target. This failure usually occurs because the information available about the target is insufficient to differentiate it from other targets in the application. If ambiguous matching occurs, a message displays during the interrogation if an object does not match uniquely.

In addition, more than one target displays in the Target frame of adapter design window.



There are a couple common situations for ambiguous matching: generic controls and objects in the Button class. The two ambiguous matching instances may happen more frequently and serve as good examples.

Generic controls

Studio makes every attempt to identify each target. For example, when you interrogate a text box in a Windows application, Studio tries to identify the object as a text box with all of its inherent properties,

methods, and events so you can use the object fully in automations. In some cases, Studio can only define the targets as a base Windows or generic control. These controls have a base set of properties, methods, and events that are often a subset of the full functionality available within the application.

Button class

When interrogating web applications, HTML classifies most buttons as an <input> object. So regardless of button, radio button, or checkbox, the <input> is the only defining item. The only difference may be a text value or ID value of the object to differentiate.

Navigating through a web application

Introduction to web navigation

This lesson teaches you about the types of automation links, imitating events in an automation, and navigating a web application.

After this lesson, you should be able to:

- Describe the difference between synchronous and asynchronous links.
- Explain issues to consider when automating a web application.
- Explain Label/Jump To components.
- Describe procedure automations.
- Complete steps on creating a procedure automation.
- Describe raising an event in an automation.
- Complete steps on creating project structure and organization.
- Complete steps on adding a raised event in an automation.
- Complete steps on completing the web navigation.

Types of automation links

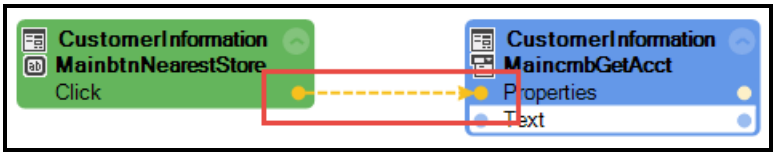
Studio projects use threads to paint the screen with the project's windows, application bars or Windows forms and the associated controls, execute automations, and integrate with the project applications. At project runtime, when users interact with a Windows form or its child controls, Windows sends messages describing the users activity to the user interface thread.

The user interface processes hundreds of messages every second. In response to these messages, the user interface thread raises Windows form events such as Click, KeyPressed, or TextChanged.

All Windows form events raise synchronously and execute on the user interface thread. Since the user interface thread is responsible for painting the screen, any long-running activity that occurs on the user interface thread can block both painting and message processing.

This gives users the impression that the user interface is not responding. To avoid this, any automation triggered by a Windows form event should execute an asynchronous link to release the user interface thread before it interacts with any adapter controls or non-Windows form components, such as web services or data access. Use the event link Design properties to set asynchronous/synchronous execution. You may also right-click on the automation link and select Asynchronous.

The asynchronous link resembles a yellow dash line in the automation.



If you are working in a solution using the Interaction Framework, remember that an activity, when queued, automatically creates its own thread. Therefore, if a control on a Windows form starts an activity, the activity does not need the asynchronous link.

Web navigation concepts

When automating a process that incorporates a web application, you should always perform a check to determine the user's location within the application before attempting to perform another process as well as the running of the application adapter. A best practice is to use a consistent starting point in the application to begin the process. Typically, this is the hoIFme page or landing page in the web application. Using a consistent starting point allows you to create a procedure automation that you can call at any time you need to begin a new process.

You should not create an automation that initiates on a control created event. At any point in time when navigating a web application, users may mistakenly initiate an automation because the automation runs when the page matches.

Label/Jump To components

While creating an automation you may find that you want the automation to complete many processes and tasks. But it is important that you keep the number of tasks and processes that you want the automation to complete small. Keeping the automations focused on specific smaller processes aids in the debugging of the solution by making it easier to locate the bug or error without affecting other automations. However, there are times where the structure of an automation defies that recommendation.

If this occurs, Studio has a component that allows you to organize and keep the automation neat and legible – the **Label/Jump To** component. Label - Jump To functionality lets you connect one part of an automation to another part of the same automation. The Label/Jump To component cannot navigate the process to another automation. For example, if you have a very large automation in which you need to connect a variable and method, instead of a long execution line or data path, you can connect them with a Label-Jump To pair.

Automations as procedures

An old adage about coding is to reduce programming redundancy. While developing, if you notice that you copy the same lines of code throughout the program, you may have stumbled upon redundant code. Maintaining redundant code is cumbersome. Remembering where each reference to the same code is located for future updates or edits wastes valuable development time. You can streamline development and reuse by creating the function at the start of the program with a referenceable function name. Now as you develop, any time that you need that same coding logic, you reference or call the function. Maintaining one section of code versus many sections streamlines the development.

Studio provides the same logic with automations. While developing automations, if a logical process is reusable, create the automation process as a procedure that you call or execute when needed. You may declare parameters to pass into the procedure as well as pass parameters out of the procedure.

Using automations as procedures allows you to nest automations within automations, creating a level of hierarchy. This structure improves the debugging and testing process. Testing the lower level processes first allows you to review the higher levels with a degree of confidence, knowing the lower repetitive automations function correctly. This structure also emphasizes the importance of having short automations, focused on one specific task. .

Automations created as procedure should following a naming convention that describes what the function does. Keep function automations in a separate folder in the Solution Explorer. Per coding standards, procedure automations generally have a success and failed exit points to pass the necessary parameters out of the procedure automation.

Creating an automation as a procedure

Creating an automation as a procedure has two steps. First you configure the automation and then you execute the automation.

Configuring the automation

Follow these steps to configure an automation as a procedure. The process assumes an automation project item exists in a project.

1. In the Solution Explorer, double-clicking on an automation to open it in the designer window.
2. In the automation designer window, right-click to access the context menu.
3. From the context menu, select **Add Entry Point**. An Execute design block displays.
4. If required for the automation, click the **+** on the design block to add parameters to pass into the automation. A parameter displays in the design block.
5. Click the data type to change the data type.
6. Click **param1 text** and enter a name of the parameter.
7. Repeat steps 4-6, to add and configure the required automation parameters.
8. In the automation designer window, right-click to access the context menu.
9. From the context menu, select **Add Exit Point**. An Exit1 design block displays with an exit parameter added. You cannot delete this exit parameter. You can change the data type and its name. You can also change the name of the Exit1 block by clicking on the Exit1 text.
10. If required for the automation, click the **+** on the Exit1 design block to add more exit parameters. A parameter displays in the design block. You can change the data type and name of the parameter.
11. Complete the automation logic as needed.
12. From the menu, select **File > Save All**.

Executing the automation

Follow these steps to execute an automation as a procedure. The process assumes a procedure automation exists in a project.

1. From the Solution Explorer, right-click on the project and select **Add > New Automation.**
2. In the Add New Item window, enter the name of the automation.
3. Click **Add**. The automation appears in the Solution Explorer and the automation designer window opens.
4. From the Object Hierarchy, select the function automation to highlight it.
5. In the Object Inspector, select the **Box** icon to display the methods. An Execute method appears.

6. From the Object Inspector, **click and drag the Execute method** to the automation. The Execute design block appears. If you added entry and exit parameters to the procedure automation, they appear on the Execute design block. Blue data ports for the parameters display for data links.

7. Complete the automation logic as needed.
8. From the menu, select **File > Save All** to save the changes.

Raising events

Depending upon the application development, certain events that happen with a user interaction may not occur when automated. Certain events trigger another application control to respond, such as tabbing away from a field after entering a value prompts the application to evaluate the entry and perform a separate action. In an automation, when moving data into that field, the tab event does not occur and then the evaluation check does not occur either.

Studio provides a method for some controls that enables you to mimic user interactions in the application. This RaiseEvent method allows you to keep the integrity of the coded application processes while still automating the full process.

Automation data proxies

Despite coding standards and best efforts to make automations simple and focused, sometimes the process and subprocesses needed to complete a task are complex. When developing automations, the concept of drawing automation links and data links to generate the necessary workflow adds complexity and legibility not found in written code.

In Robotic Automation Studio, creating proxies within automations meets the concise and legible coding standards. A proxy is an agent of a real object where you can either forward the proxy or include additional logic on the proxy.

Proxies in automations allow the developer to minimize the automation design complexity and reduce the crisscrossing execution and data links throughout an automation.

The two basic concepts of proxies within an robotic automation are:

- Extracting a proxy from a data result on a design block
- Using a parameter proxy from the Object Explorer when the parameter is needed within a procedure automation

Either concept provides the same access to the proxy's properties and methods for use within an automation. A difference between the two approaches is that extracting a proxy from the design block creates a global proxy variable, making the proxy available to other automations within the project. Using the Object Explorer method defines the use of the proxy to be local to the automation only.

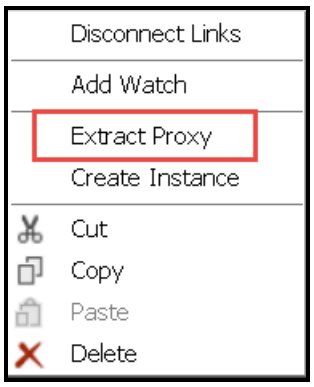
Extracting data proxies

The two ways to extract a data proxy in a robotic automation are to either use the design block data output or use the Object Explorer.

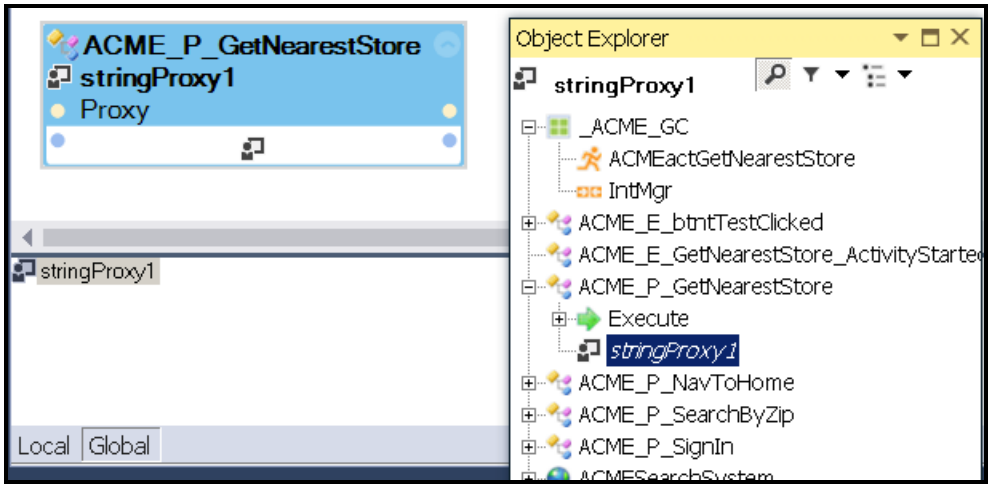
Using the design block data output

A design block that produces data or passes data for an automation provides an option to extract a proxy on the data result for use within the same automation or other automations within the same project. The following steps assume that an automation is in development with a design block.

1. On a design block with a data output (blue dot), right-click the data output. A context menu is displayed.



2. On the context menu, select **Extract Proxy**. A new design block is displayed in the automation connected to the original data output with a data link. At the bottom of the automation, Studio creates a global automation variable defined named by the proxy's data type, {datatype}.proxy1. The Object Explorer updates the automation hierarchy and displays the proxy.

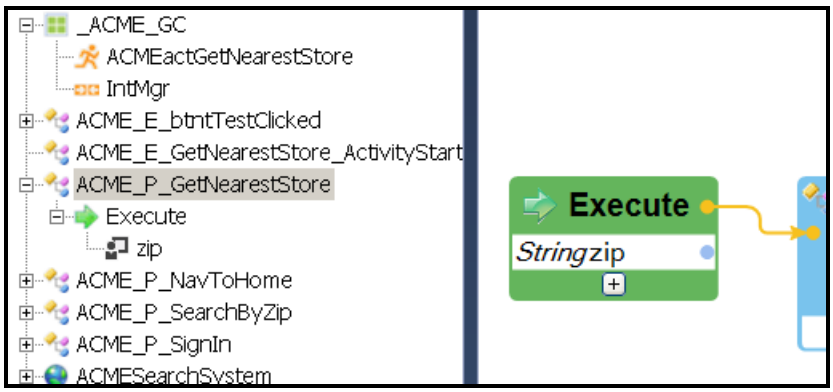


3. If needed, use the Properties window to uniquely name the proxy.
4. From the Object Inspector, click **Configure Type** to access the proxy's properties and methods for use in the automations.
5. In the Object Inspector, click **Configure Type** to access the parameter's properties and methods. The Configuration window is displayed.
6. On the Configuration window, check the needed properties and methods for the proxy.
7. Click **OK**. The Configuration window closes and selected options are displayed in the Object Inspector.
8. In the Object Inspector, click and drag the property or method needed to the automation.

Use the Object Explorer

A procedure automation that contains input parameters is the best use for this method. The following steps assume that a procedure automation using an input parameter is in development.

1. In the Object Explorer, click the **Plus** icon to expand the automation currently in development, and on the Execute note, click the **Plus** icon. The input parameters for the automation are displayed.



2. In the Object Explorer, select the input parameter needed for the proxy.
3. In the Object Inspector, click **Configure Type** to access the parameter's properties and methods. The Configuration window is displayed.
4. On the Configuration window, check the needed properties and methods for the parameter.
5. Click **OK**. The Configuration window closes and selected options are displayed in the Object Inspector.
6. In the Object Inspector, click and drag the property or method to the automation.

INTERACTION FRAMEWORK

This lesson group includes the following lessons:

- Interaction Framework structure
- Working with multiple-project solutions
- Interacting between applications and projects

Interaction Framework structure

Introduction to Interaction Framework

In this lesson, you learn about the components of the Interaction Framework, how to edit the interaction.xml, local and global variables, and how to add framework components to a project.

After this lesson, you should be able to:

- Describe the Interaction Framework
- Explain the components of the Interaction Framework.
- Explain the use of the interaction.xml.
- Explain the difference between local and global variables.
- Edit the interaction.xml file.
- Define the Interaction Manager component.
- Add the framework components to a project.

Elements of the Interaction Framework

The Interaction Framework is a systematic approach that provides Pega Robotic Automation Studio developers with a collection of tools to incorporate into the development of projects and solutions. By project-to-project references, Interaction Framework allows projects to communicate with each other. The Interaction Framework simplifies development, improves project reusability, and streamlines solutions for implementing new features.

The Interaction Framework consists of four basic elements:

- Interaction
- Interacation.xml
- Interaction Manager component
- Activity component

The basis of the framework is an interaction. An interaction is any client or account engagement, for examples:

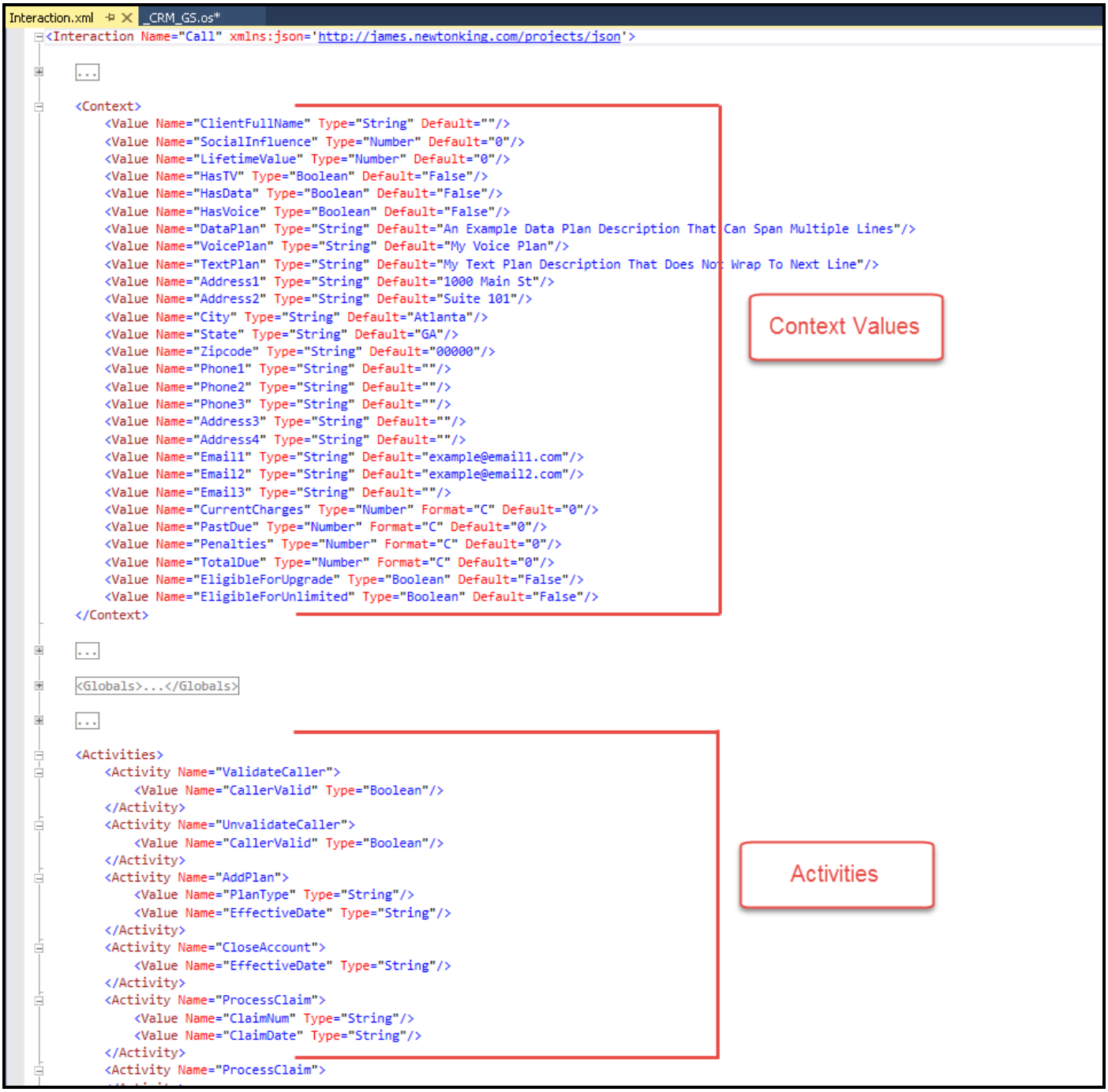
- A call center agent answers an incoming call by a customer.
- A bank teller greets the next banking customer in line.
- The loan officer receives the loan documents to begin processing the loan.

All of these examples expose only one interaction with a client or customer; however, the customer interaction may have several activities to accomplish before the interaction is complete. Without an interaction, there can be no framework created within the solution and referencing projects.

The Interaction Framework components require access to all other project items within a project and the solution. The global container is a project item that provides project-wide access for the components.

The interaction.xml file

The interaction.xml, or binding contract, pulls all the referenced projects together and allows them to communicate information and events between them. The interaction.xml configuration file defines the context values and activities used in the framework. The following image shows an example of an interaction.xml file.



The interaction.xml is the only framework component that is solution wide, meaning a solution requires only one interaction.xml. By using other framework components, scoped to each project, you reference the one interaction.xml file for the solution. Visually, you can consider the XML file as the center or hub of the solution, and each project, through the other framework components, read and communicate to each other using the XML file.

Studio provides a base interaction.xml during installation. You should use it as a base file for development. Once you create a base file, use it for all subsequent solutions moving forward to minimize development time. You can change the name of the file, if desired. The interaction.xml file can also be a living file so that it grows as the solution grows with new business cases.

Note: The base file is located in the following directory: C:\Program Files\OpenSpan\OpenSpan Studio for Microsoft Visual Studio 2015\AgileDesktop\Samples\Config.

Interaction Manager component

The Interaction Manager component connects all projects and customer interactions through the XML file by accessing the XML through their properties, events, and methods in the Object Explorer. The Interaction Manager maintains a direct link to the contents of the interaction configuration file and stores the context values to share them across projects and to complete interaction. These context values appear as the Interaction Manager properties in the Object Explorer. When developing a solution, the Interaction Manager must be the first framework component added to the Global Container for each project.

Each referenced project in the solution has only the Interaction Manager — it connects all the projects to the framework's contexts and activities configured in the XML file. You start the framework by starting an interaction. Once an interaction starts in any referenced project within the solution, all other projects can share the context values and perform any activities for that interaction.

Local and global variables

The two types of variables used in Studio when working with a solution are:

- Local
- Global

You can designate local variables in automations. Studio exposes the local variables only to their automation. When each automation begins its process, Studio creates and uses the local variables only when the automation runs. Studio adds variables at the automation level to the Local tab by default. However, you can change a local variable to a global variable if its scope changes during the course of development.

If you change a local variable to global in an automation and use the component in other automations, do not delete the automation to which you originally added the component. Deleting the automation with the global variable, deletes all of the variable references in other automations. You can avoid this by using a Global Container project item and adding any variables and components requiring global project scope.

When you create global variables they are project-specific. Studio creates them when the project starts, and they exist while the project runs. You can use them in any automation at any time specific to each project.

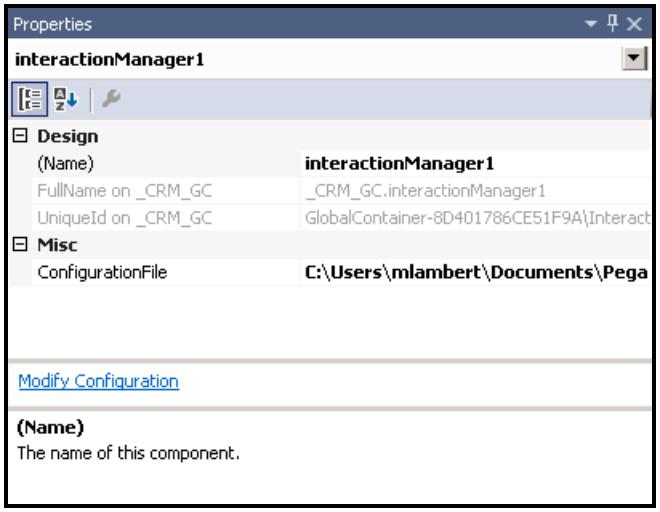
You create a global container for each project to add your global variables. The Object Explorer displays the global container and its objects, making them available to all automations in a project. You can use more than one global container for a project to help organize the types of variables and components in each.

Adding the Interaction Manager component

The Toolbox contains the Interaction Manager and Activity components. You add the component to each project's global container. The process assumes a global container exists in the project.

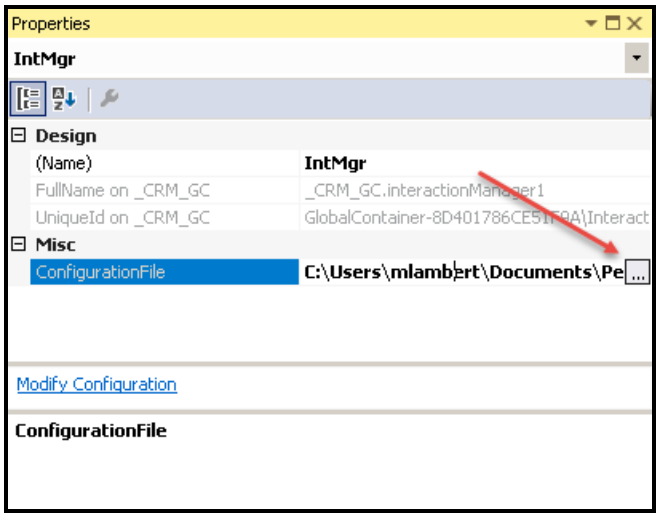
Follow these steps to add and configure the Interaction Manager component.

1. In the Solution Explorer, double-click a project's global container to open it in the designer window.
2. In the IDE, click the **Toolbox** window.
3. In the Toolbox, click the **Interaction Management** category to expand it.
4. Click the **Interaction Manager** component and drag it to the Global Container designer window. The Properties window refreshes to display the Interaction Manager component. By default, a path pointing to the base XML file is displayed.



5. In the Properties window, click the **(Name)** property field.
6. Enter a name for the Interaction Manager component.
7. In the Properties window, click the **Configuration File** field. An ellipses button is displayed.

8. Click the **ellipses** button to display the Browse window.



9. In the Browse window, navigate to the solution folder to locate the interaction.xml file for the solution.
10. Select the interaction.xml file to highlight it.
11. Click **Open** to associate the XML file. The Browse window closes.
12. From the menu, select **File > Save** to store the changes.

Modifying the interaction.xml

You can edit the interaction.xml file within Studio. The structure and syntax of the xml document is case sensitive, requiring precision with spacing as well. When editing the interaction.xml document, you must:

- Edit the interaction.xml file
- Update the solution in Studio

Editing the interaction.xml within Studio

You can edit the interaction.xml file for the solution at any time during development, but you should attempt to create the file for the solution prior to development.

Follow these steps to edit the interaction.xml file within Studio.

1. In the Solution Explorer, double-click a project's global container to open it in the designer window.
2. In the Global Container designer window, right-click the Interaction Manager component.
3. Select **Modify Configuration**. The interaction.xml file opens in a designer window showing color-coded text.
4. Scroll through the file to locate the section.
5. Enter the needed context or activity to its corresponding section in the file.
6. From the menu, select **File > Save** to save the edits.

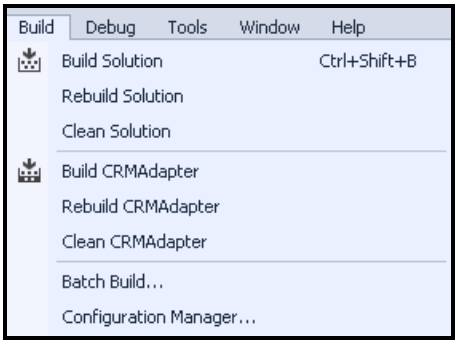
Note: XML files are case sensitive and syntax specific. For the best editing options, use copy/paste to ensure that syntax stays intact.

Updating the solution

Refresh the solution to remove any old entries and add new entries to the memory cache. If you edit the interaction.xml with a different editor than Studio, you still update the solution.

Follow these steps to update the solution to use the edited interaction.xml file.

1. From the menu, select **Build > Clean Solution**. The Output window displays the clean messages.



2. Once the clean process completes, from the menu, select **Build > Rebuild Solution**. The Output window refreshes showing the rebuild messages.

Working with multiple-project solutions

Introduction to multiple-project solutions

In this lesson, you learn about project-to-project references, context values, interactions, activities, and using automation development concepts.

After this lesson, you should be able to:

- Explain project-to-project references
- Create a project-to-project reference between projects
- Establish the start up project in a multiple-project solution
- Describe context values
- Describe the relationship of interactions and activities
- Complete steps on adding combo boxes to an automation
- Complete the steps working with context values in automations

Project-to-project references

Pega Robotic Automation relies upon the Microsoft Visual Studio platform, therefore some functions of Visual Studio are available to developers. One of these functions is the project-to-project reference.

When building a solution or project, Studio creates assembly files. When you have a project that produces an assembly, use project-to-project reference. The advantage of a project-to-project reference is that it creates a dependency between the projects in the build system. Studio builds the dependent project if it has changed since the last time Studio built the referencing project.

When working in an environment where many Studio developers are creating projects that involve the same application, you should create and maintain a single project dedicated to the interrogation of the application and the automation of basic application functionality.

For example, this application project includes:

- The full interrogation of the application.
- The modification of match rules as required.
- Automations to login and navigate the application to the initial application window used by the intended end users.

Other Studio developers within your organization may use this application project as a base for developing specific business automations or event data collection or both for other solutions.

The benefits of this approach include:

- Maintenance: If the primary application changes, new interrogated targets added or match rules modified, Studio updates the changes made to the primary application project to all other referenced application projects.
- Reusability: A single developer can interrogate the main application and develop the login/navigation logic. Other Studio developers can then focus on building projects that perform specific business automations and event data collections.

In the Solution Explorer, use the startup project as the main or controller project. Studio denotes the startup project by changing the font to Bold.

Interaction Framework

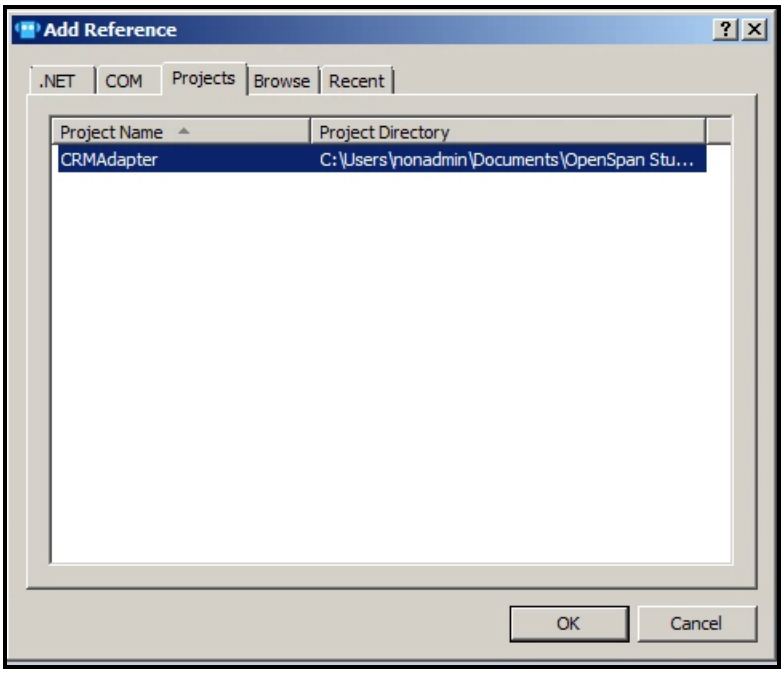
Interaction Framework can work with or without creating a project-to-project reference; however, typically a multi-project solution is deployed to the user as one package containing all projects. Therefore, using the project reference with Interaction Framework ensures that the framework works as one unit for the solution and not by project.

Creating a project-to-project reference

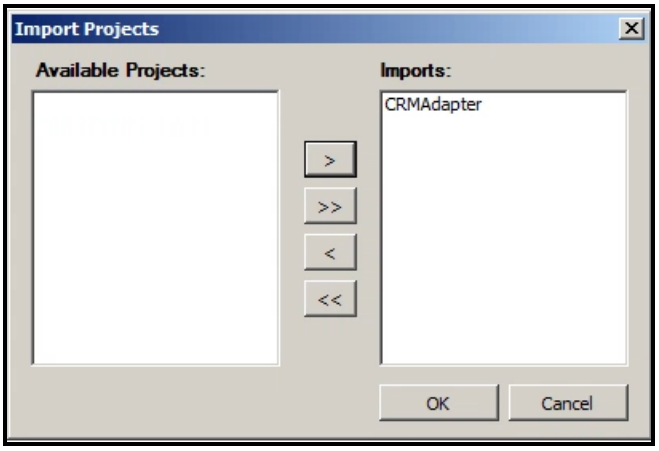
When you create a project-to-project reference in your solution, the end user works with a single unit rather than multiple units.

Follow these steps to create a project-to-project reference.

1. In the Solution Explorer, **right-click** the controller project and choose **Add Reference**. The Add Reference window opens.
2. On the **Projects** tab, select a project.
3. Click **OK**. The Add Reference window closes and the project is displayed in the Reference folder under the controller project in the Solution Explorer.



4. In the Solution Explorer, right-click the controller project and choose **Manage Imported Projects**. The Import Projects window opens.
5. From the Available Projects list, highlight the referenced project and click **>**. This moves the project into the Imports list.



6. Click **OK**. The window closes and imported project is saved to the controller project.

Setting the StartUp project

You may need to modify which project in a multiproject solution is the controller project.

Follow these steps to set a project as the controller project in a multiproject solution.

1. In the Solution Explorer, right-click the project to use as the controller project.
2. From the context menu, select **Set as StartUp Project**. The selected project name becomes bold in type.

Framework context values

The interaction.xml file contains the configuration of the context values that represent the data referenced in a solution. The context values are displayed as properties of the Interaction Manager component in the Object Explorer. Each project has its own Interaction Manager component, configured to reference the interaction.xml. This configuration allows for storing, editing, and sharing of the context values across all solution projects through the Interaction Manager.

When using the framework, you must initiate the framework by starting an interaction. Once the interaction starts, a virtual data table creates a structure column to represent each context value. Every interaction started receives a new row in the data table. The data table also has an interaction key column. The interaction key is similar to a database primary key, storing a unique identifier of each table row or interaction available in the framework. When an automation closes an interaction in the framework, the framework deletes the data row and values of the interaction.

The interaction.xml file is the hub of the framework, so the basic data flow is for the data source applications to store the context values in the framework first. Once stored, all other applications can update or use the context values to complete specific activities. If data does not store correctly into a context value, the context value uses its configured default value.

Interactions and activities

An interaction is a customer contact through a phone call, a face-to-face meeting, or a received document. Any of these things initiates the call to perform one or many activities to meet the need of the customer.

The Interaction Framework provides the flexibility for a solution to manage one interaction or many interactions simultaneously. The framework creates a data table to store the interaction context values. An end user may have several interactions opened at the same time, but can work on only one interaction at a time. Compare this situation to your computer desktop: you may have several applications open and running at the same time, but you can work in only one of the applications at a time.

Once an interaction starts in any project in the framework, all other projects in the framework can complete work or activities for the interaction. Activities run with the same logic. Once an activity starts in one framework project, the activity can complete in any framework project, including the one in which it started.

You must start and stop interactions and activities so that the framework can communicate the success or failure of the process across the projects.

Working with combo boxes

Combo boxes can cause difficulties when used in automations. Think about the combo box control as having two levels — the first is the control level, and the second is the items or options in the combo box. Accessing the right properties, events, and methods for this second level is not intuitive.

Adding combo box options to the Object Explorer

The process assumes that the combo box control exists in the Object Explorer.

Follow these steps to add the properties, events, and methods for combo boxes.

1. In the Object Explorer, locate the combo box control.
2. In the upper-right corner of Object Explorer, click the **Explore Component Properties** toggle button. The Object Explorer window refreshes.
3. Scroll down the list of components and click **Items** to highlight it.
4. In the Object Inspector, click **Configure Type**.
5. In the Object Configuration window, search for the needed property, event, or method. Use the Filter field to minimize the search results matching the entered text.
6. Select the check box corresponding to the needed property, event, or method.
7. Click **OK** to close the Object Configuration window.
8. In the Object Inspector, click the **Wrench**, **Lightning bolt**, or **Box** to locate the newly added options.
9. Click and drag the needed option to the automation.

Note: Click the **Explore Component Properties** toggle button to return to the original Object Explorer view.

Interacting between applications and projects

Introduction to application interaction

In this lesson, you learn about user interactions with solution applications and framework logic with interactions and activities.

After this lesson, you should be able to:

- Explain the Activity component
- Describe how to develop for user interactions
- Complete steps on adding an activity to the XML
- Explain activating interactions versus activities
- Complete automations on activating interactions and activities

The Activity component

The Activity component represents any work process for an interaction. Activities can reference actual processes such as add customer, close account, and update customer information. Activities could also reference back-end work to allow the solution to flow and function, such as refresh windows and search database.

The Activity component may contain parameters that allow you to pass needed information through to each project to complete the activity successfully. You add the Activity component to the global container where you have access to manage the activity through its methods in the Object Explorer.

Studio queues each activity when started within an automation. The activity completes once all processes across the solution complete the activity. You develop the automations to queue and time the activities using the activity methods and parameters.

The following table lists common methods for activities.

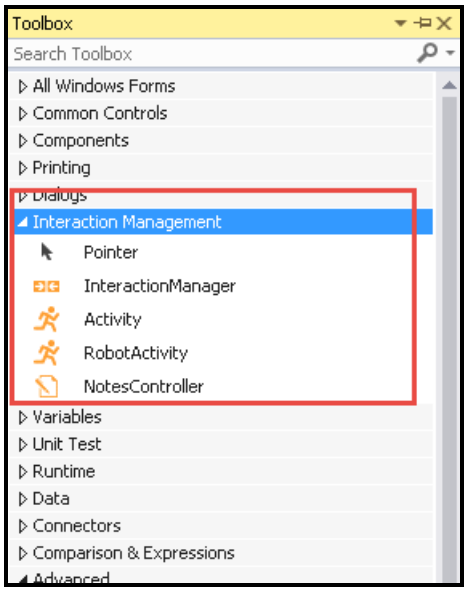
Method	Description
CancelActivity	Use this method to stop an activity if the activity hangs for a WaitForCreate method within an automation.
Start	Use this method to queue the activity for the current interaction.
StartNow	Use this method to add the activity to the top of the queue. Any currently running activity does not stop.
StartAndWait	Use this method to place an activity at the bottom of the activity queue and then block the automation from proceeding until the activity has reached the top of the queue and has executed completely.
StartNowAndWait	Use this method to place an activity at the top of the activity queue and then block the automation from proceeding until the activity has executed completely. Any currently running activity does not stop.

Adding an Activity component

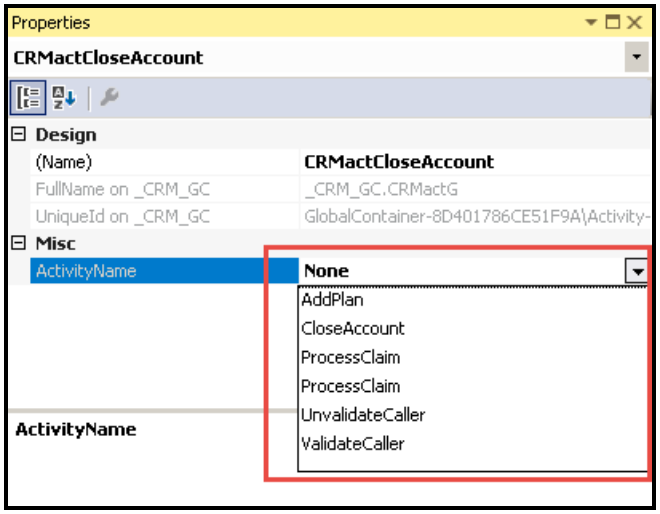
Follow these steps to add and configure the Activity component.

1. In the Solution Explorer, double-click a project's global container to open it in the designer window.
2. In the IDE, access the Toolbox window.

3. In the Toolbox, click the **Interaction Management** category to expand it.



4. Click the **Activity** component and drag it to the Global Container designer window. The Properties window refreshes to display the Activity component.
5. In the **Properties** window, click the **(Name) property** field.
6. Enter a name for the Activity component.
7. In the **Properties** window, click the **Activity Name** field. A drop-down box enables.



8. From the drop-down box, select the activity. The activities listed in the drop-down come from the interaction.xml file configured with the Interaction Framework component.
9. From the menu, select **File > Save** to store the changes.

Note: If the activities in the drop-down box are incorrect or missing, check the Configuration File property of the Interaction Manager component and confirm the path directory. If you need an activity that does not exist in the drop-down, you must add it to the XML file used for the solution.

Comparisons and expressions

Quite often during programming, you must evaluate and judge two values against each other to determine the next step in a logical path. Since Studio does not have standard coding, in order to perform this evaluation, you must access the Toolbox. All comparisons and expressions, along with variables, reside in the Toolbox in the Comparison and Expressions section.

Studio provides a standard comparison functions such as, greater than, equals, does not equal, and so on. The expression objects allow you to enter customized expression statements for Boolean, numerical, and string datatypes.

Considering user interaction with a solution

When working with Studio to automate and streamline processes, you should consider how the solution changes the user's daily work processes. Users perform specific steps to complete their required tasks. When an automated solution streamlines those tasks, their specific steps change, possibly causing issues when users interact with the solution.

Keeping applications hidden during runtime may not help the user. Most users prefer a message or identifier to alert them when automations run and complete. Hide the applications at runtime as a last resort. Other items to consider for user interactions consist of using mouse-clicks versus keyboard shortcuts. Not all users interact the same if given multiple options to perform the same task. Most error handling and trapping stems from not considering how users may interact with the application.

In terms of solution business cases, the cases should provide recommendations to account for user interactions. For the framework, consider using the Activities portion of the XML to drive not just standard work processes but also to drive user interaction activities.

Interaction versus activity activation

The Framework allows for concurrent interactions; however, the end user can only work on one interaction at a time. Understanding this makes development decisions easier.

Studio provides a method to activate an interaction. When you invoke this method in an automation, Studio makes all other interactions inactive. Activating an interaction can initiate other activities to refresh the supporting project applications to match the new active interaction. This helps ensure data integrity, but automations should also check to ensure the interaction matches the account information in all project applications.

You can consider this logic similar to a relational database or your computer desktop. A database contains several to thousands of rows of data. You can only edit one row of data at a time.

Activities do not have an Activate method. They have a Start method. Similar to interactions though, once an activity starts in the project in the Framework, all other projects can respond to the activity started.

Working with Interaction Framework

Introduction to working with Interaction Framework

This lesson teaches you about updating automations to work with the Interaction Framework and responding to context value changes within a project.

After this lesson, you should be able to:

- Complete steps on updating automations to use the Interaction Framework
- Explain how to respond to updated context values in the Framework.
- Complete steps on updating an interface based on updated context values.

Update Framework context values

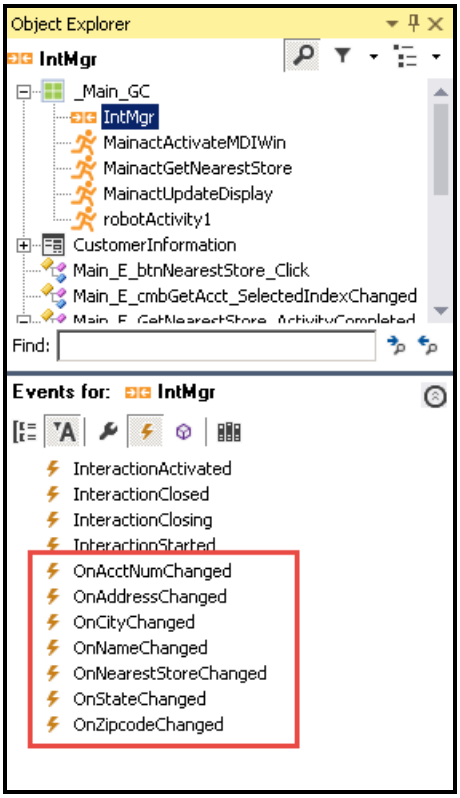
When implementing the framework within a solution, it is common to have different processes or activities update an interaction's context values. A context's value may also force an automation to perform another series of activities based on its value. Because of these common occurrences, the framework provides a mechanism that allows a developer to respond to a change in a value.

The framework's flexibility creates a dynamic event for the Interaction Manager component. For every context value added to the Context section of the interaction.xml, the framework creates an event based on that context value.

Using the solution's interaction-call.xml, the Context section contains seven context values:

- Account Number
- Name
- Address
- City
- State
- Zip Code
- Nearest Store

In the Object Inspector for events, the IntMgr displays seven specific events– one for each context value changed.



A single automation may contain a line to handle each specific context change event, not a separate automation for each context value. Remember to use function automations to perform the actual process of updating any interface or applications.

Integration with Pega 7 Platform

Introduction to Pega 7 integration

This lesson teaches you about the integration of Pega Robotic Automation Studio with Pega 7 platform applications.

After this lesson, you should be able to:

- Explain the integration of Pega Robotic Automation Studio with Pega 7.
- Explain the Robot Activity.
- Describe the steps to configure a Robot Activity.

Overview of Pega 7 Integration

Compiled desktop automations designed and built in Pega Robotic Automation Studio run in Pega Robotic Runtime on the local desktop. The Pega client invokes the automations directly through a REST service running in Pega Robotic Runtime. After receiving a request, Pega Robotic Runtime starts the automation, performs the automation logic, and returns an appropriate response to your Pega 7 application. Because of the structure of Pega 7 applications, you must consider certain parameters with the integration.

The Pega Robotic Runtime requires a SSL-enabled REST service because browsers do not allow calls from an SSL Pega 7 application to a non-SSL service. In addition, the REST service guarantees that only designated domains can invoke a request to run a desktop automation.

The invoked automations from the REST Service, execute synchronously, requiring your Pega 7 application to wait for the automation completion before proceeding to the next flow step. Design your automations for speed and maintenance with one single focus or purpose. Break down larger functionality blocks into smaller automations to avoid slow response or the appearance of being unresponsive.

The automations, with the use of the Robot Activity component, can invoke from any flow action that runs as part of the standard flow, modal dialog box, or overlay. You can invoke an automation as a pre-action or post-action, depending on the business requirements. The flow action must reference the automation identifier assigned to the automation within the Robot Activity.

Robot Activity

Pega Robotic Automation Studio 8 has a Robot Activity component, based on the Interaction Framework activity component. This component works similar to an activity that does not require an active interaction.

Before designing an automation and configuring a Robot Activity, you need the following items to coordinate between Pega 7 Platform Designer Studio and Pega Robotics Studio development environments:

- The automation identifier used in Robotics Studio to identify your automation.
- The fully qualified class name of the case the automation invokes (for example, MyOrg-MyApp-Work-OpenTicket).
- The format of the data received by the automation.
- The required format of any data that returned from an automation to your Pega 7 application.

Automations handle input such as a customer name and account number, and return output, such as the amount of a transaction. At this time, Pega 7 applications and automations can pass only scalar Pega 7 Platform properties. The Pega 7 integration does not support hierarchical structures and lists of data. You can only access an activity's parameters or properties when the activity is active – from the start of the ActivityStarted event until its completion.

Configuring a Robot Activity

The Toolbox contains the Robot Activity component. To use the Robot Activity, an Interaction Manager component is not required.

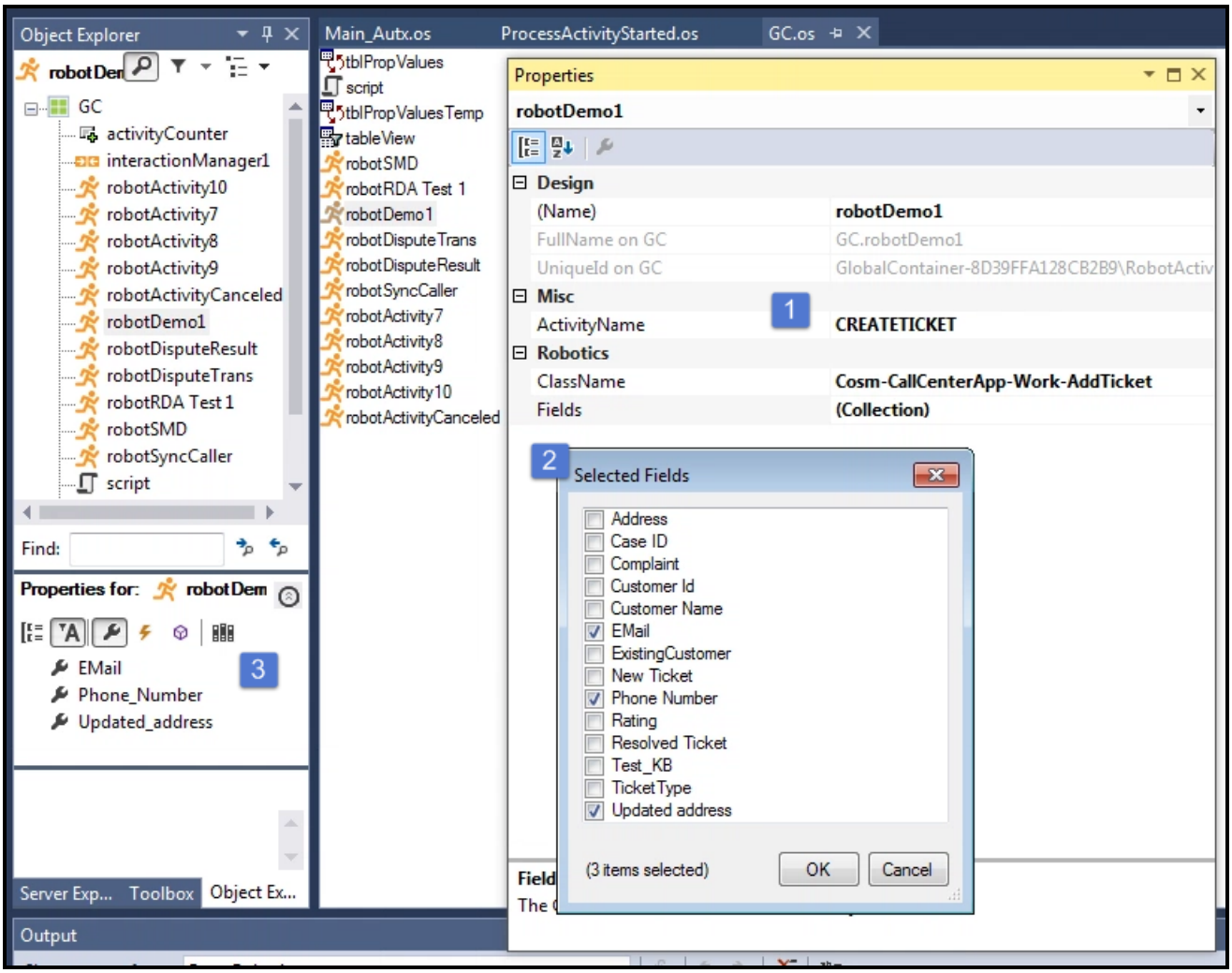
Configuring the Robot Activity

Follow these steps to add and configure the Robot Activity component. The process assumes a global container exists in the project and the Pega Robotics Studio developer has been configured access to the Pega 7 application.

1. In the Solution Explorer, double-click on a project's global container to open it in the Designer window.
2. In the IDE, access the Toolbox window.
3. In the Toolbox, click the **Interaction Management** category to expand it.
4. Click the **Robot Activity** component and drag it to the Global Container Designer window to add it. The Properties window refreshes to display the Robot Activity component.
5. In the Properties window, update the following properties:
   a. Design Name
   b. ActivityName
   c. ClassName

d. Fields

Property	Description
ActivityName	The activity name is a string used to identify the automation, and is the automation identifier referenced in the Pega flow action as the robotic automation name.
ClassName	The Pega case type that supplies data to the automation. The Pega developer communicates the class name to the automation developer. Examples: MyOrg-MyApp-Work-OpenTicket or MyOrg-MyApp-Work-OpenTicket-OpenSubTicket.
Fields	With a class name provided, Pega Robotics Studio programmatically retrieves a list of all available relevant record fields available for automations. An automation developer can choose a subset of these fields to appear as properties on the Robot Activity component.



6. With the ClassName provided, click the **Fields** property. An Ellipses button appears.

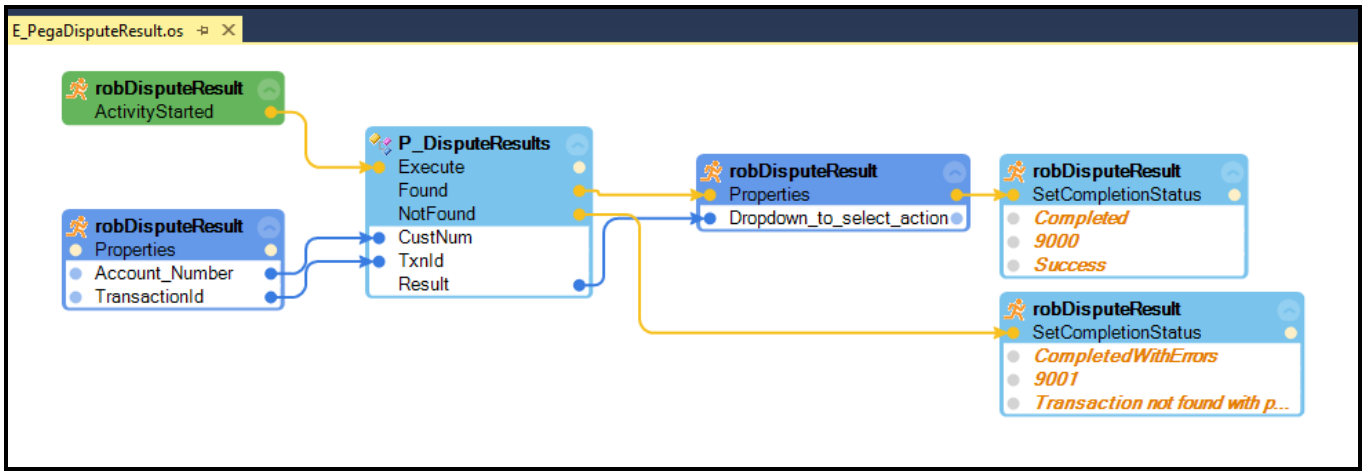
7. Click the **Ellipses** button to access the Fields window to select the needed fields for the automation.
8. Click **OK** to close the window and save the selection.
9. Select **File >Save All** to save your changes.

Using the Robot Activity

Follow these steps to implement the Robot Activity component in an automation. This process assumes an automation and a configured Robot Activity already exists.

1. From the Object Inspector for the Robot Activity, click and drag the **robotActivity.ActivityStarted** event to the automation.
2. Complete the automation logic as the business requirements dictate.
3. From the Object Inspector for the Robot Activity, click and drag the **robotActivity.SetCompleteStatus** method to end the processing of the activity and return the appropriate result.

Note: The return status is for informational purposes only and does not affect the processing flow in the Pega 7 application.



DEPLOYMENT

This lesson group includes the following lessons:

- Solution deployment

Solution deployment

Introduction to solution deployment

This lesson covers the deployment of Pega Robotic Automation Studio solutions.

After this lesson, you should be able to:

- Explain the Studio deployment.
- Describe Studio project deployment properties.
- Explain how to deployment a solution.
- Complete the steps to deploy a solution.

Deployment

In order to deploy a solution using either the RDA (Robotic Desktop Automation) or RPA (Robotic Process Automation) Studio methodology, each workstation employing the solution must have Pega Robotic Runtime installed. Runtime processes the Studio deployment files for use on the workstations. See the *Pega Robotic Runtime Installation* article in the Related Content section article for more information regarding Pega Robotic Runtime.

Once you have created a project and set the Project Property pages, you are ready to begin the process of deploying the project to the user Runtime environment. Follow these guidelines to create the Runtime project:

- Test the project in Pega Robotic Automation Studio
- Create a Project Deployment Package for the project
- Install and configure Runtime on pilot systems for testing the project
- Deploy the project to the pilot workstations
- Run and test the project in a pilot Runtime environment
- Establish a file deployment strategy or use the company's deployment strategy

For deployment, make sure all user workstations contain the same drivers or supporting files used by Pega Robotic Automation Studio and the applications. In general, the computers running Runtime must be able to run the project applications the same way that the Pega Robotic Automation Studio designer workstation does.

Testing in Studio

Before preparing a deployment package for use with Runtime, you should thoroughly test the solution in Pega Robotic Automation Studio. As a best practice, verify the following:

- Login authentication works (if required)
- Application navigation and target matching is accurate
- Data entry validation works
- Error trapping works
- Application shutdown and restart works

Using deployment package files

Use the Studio's Deploy Project option to create the files required by standalone Runtime installations. Pega Robotic Automation Studio deployment can use either a local deployment or a server deployment. Local deployment requires the developer to transfer the files manually to the party responsible for disseminating the deployment files to the users' workstations. Server deployment uses a Deployment Portal product for disseminating the solution to the user.

The deployment package consists of two deployment files:

File Type	Description
.openspan	The compiled project file along with all referenced assemblies and translators needed to run the project in Runtime.
.manifest	A list of the .openspan file contents along with project version information.

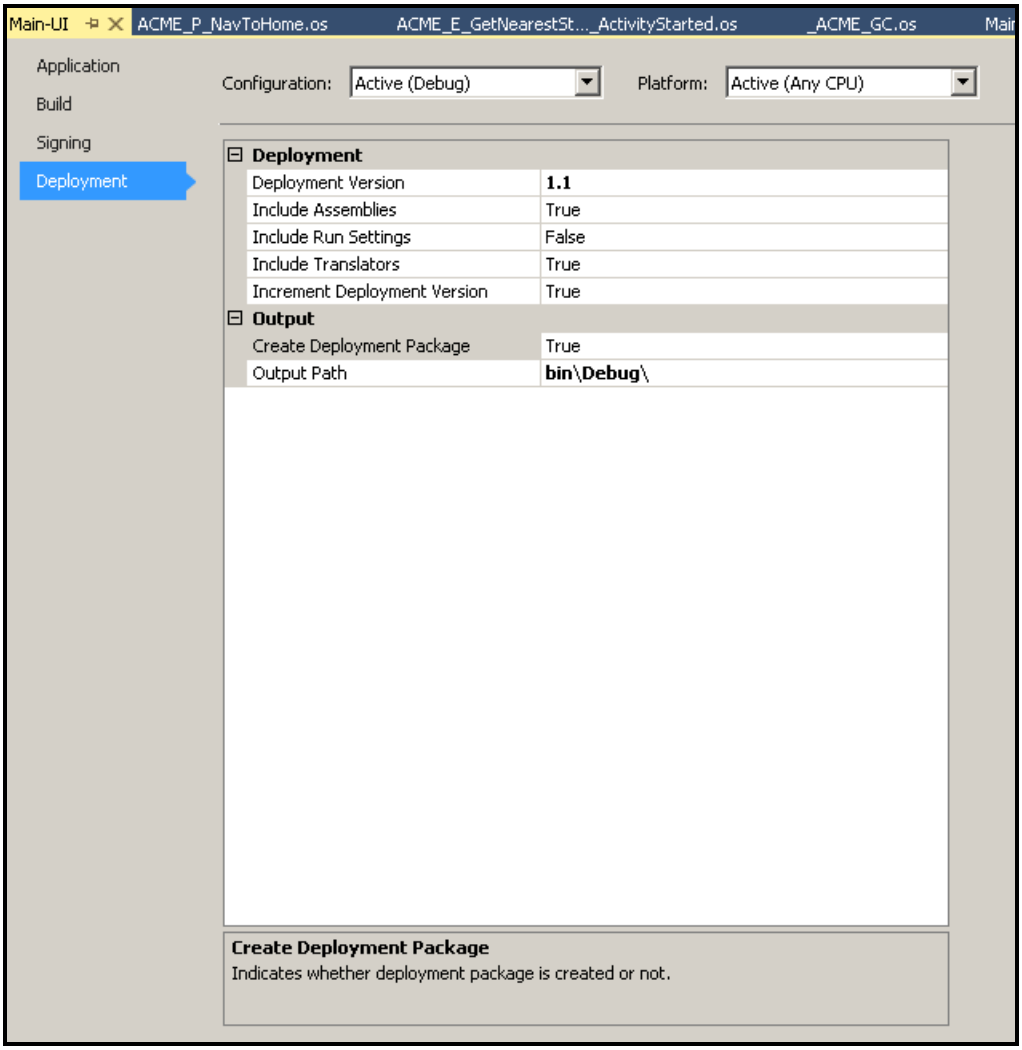
For example, the CRMAdapter project, when deployed, results in these files:

- CRMAdapter.manifest
- CRMAdapter.openspan

Regardless of the local or server deployment strategy used, the general process of deploying a new or updated solution to a user focuses on the Manifest file. The Runtime services compares the local version of the Manifest file to the deployment location of the Manifest file. If there are any differences between the files, the deployment location file wins and overwrites the local version of the solution files. This logic allows for an update to a new solution version or a rollback to a previous solution version.

Project deployment properties

Prior to deploying a solution, you should review each project's deployment properties. The type of the solution or the business need determines which project to modify. For our training solution which has a multi-project solution with a controller project, Main-UI, you only need to modify the controller project properties.



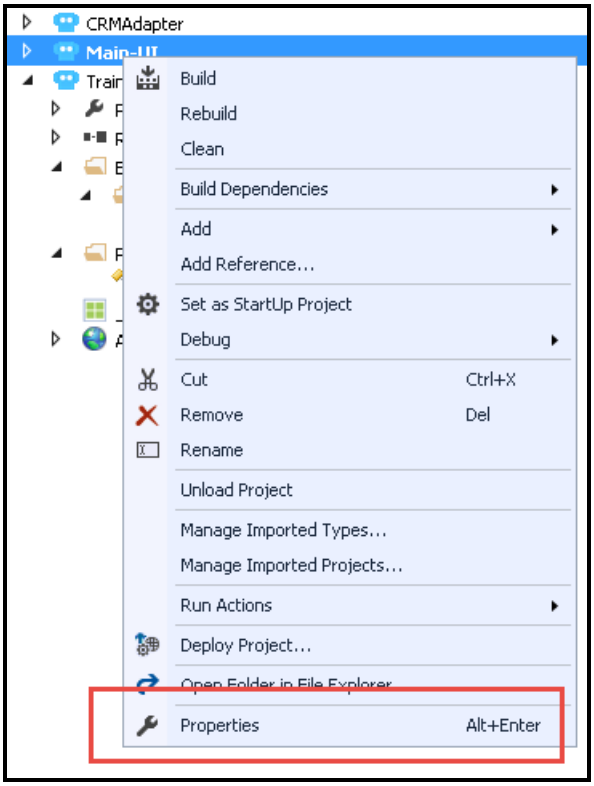
Property	Description
Deployment Version	This is the version of the deployed project. The version number starts at 1.0 for a new project. Each time you deploy a project, the deployment version is incremented. The Manifest file contains the version number. Runtime uses the version number to determine if the package differs from the last version run. If the versions are different, Runtime downloads and extracts the updated project files. You have the option of deploying a project without

	updating the deployment version. See the Increment Deployment Version property below for more information.
Include Assemblies	This option lets you control whether to include assemblies required for the project in the deployment file (.openspan). Usually, you want to set this to True so all project assemblies are included in the package. However, if you want to limit the size of the Studio project deployment file, you can set this to False. **Note:** If you set this to False, the deployed project uses the assemblies that currently exist on the Runtime computers for the project. If any of the assemblies have changed or new assemblies added between the deployed versions of the project and this setting is False, the project may not run correctly or may fail to load.
Include Run Settings	This option lets you deploy the project with the Run Actions currently set for the project in the Solution Explorer.
Include Translators	This option controls whether to include translators required for the project in the deployment file (.openspan). Usually, you set this to True to include all project translators in the package. However, if you want to limit the size of the Studio project deployment file, you can set this to False. **Note:** If you set this to False, the deployed project uses the translator files that currently exist on the Runtime computers for the project. If any of the translators have changed or new translators added between deployed versions of the project and this setting is False, the project may not run correctly or may fail to load.
Increment Deployment Version	This property defines whether to increment the deployment version each time you create a project deployment package. Select True to increment the Deployment version.
Create Deployment Package	Set this property to True to create a deployment package for the project as part of the build process. If the project is strictly a reference within the solution, set this property to False.
Output Path	Set the location for the deployment package files for a local deployment. Use the relative project path.

Deploying a solution

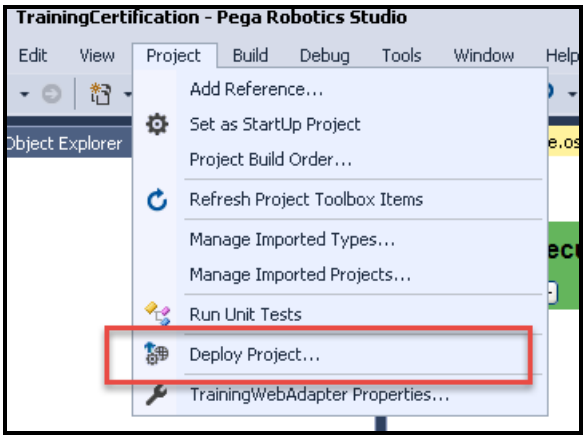
Follow these steps to deploy a solution. The process the developer has completed all necessary debugging and configurations needed for deployment.

1. In the Solution Explorer, right click on the project to deploy.
2. On the context menu, select **Properties**. The Project Properties window displays.

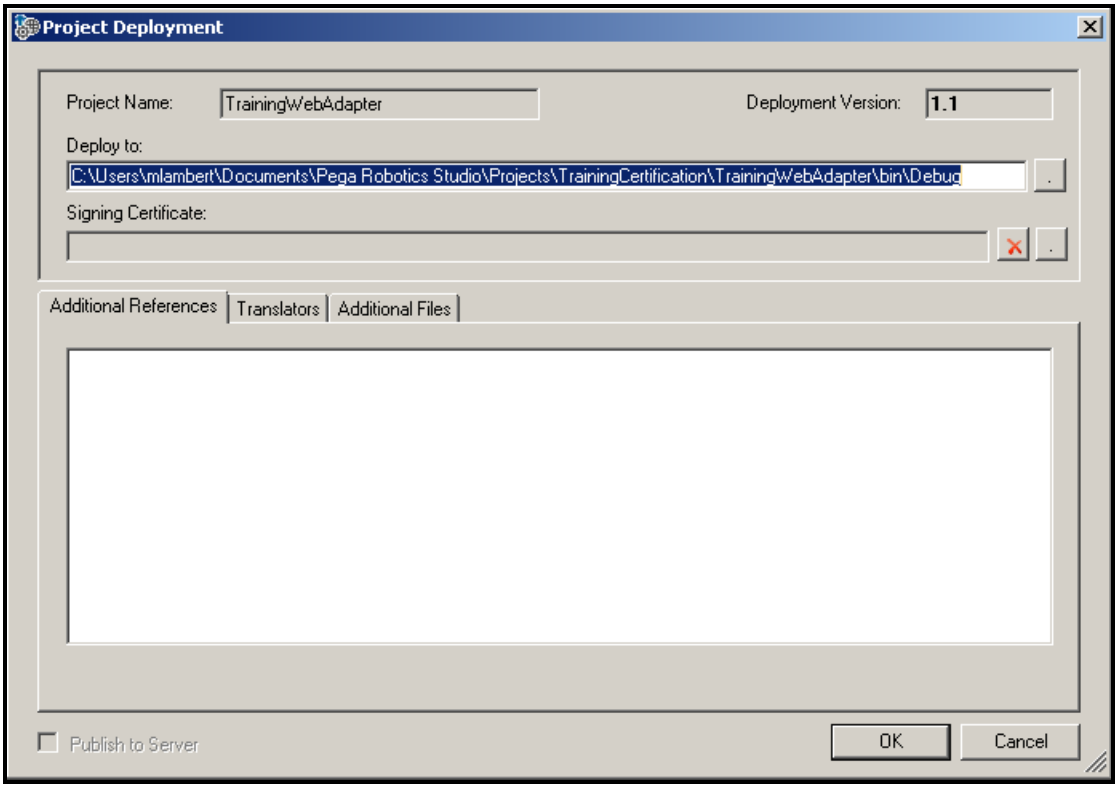


3. On the Project Properties window, select the **Deployment** tab. The Deployment tab displays.
4. On the Deployment tab in the Configuration combo box, select the Solution Configuration used for deployment.
5. On the Deployment tab, modify the deployment properties as required.
6. Select **File > Save All** to save your changes.

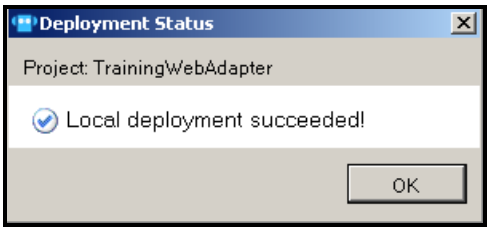
7. From the menu, select **Project > Deploy Project**. The Project Deployment window displays.



8. On the Project Deployment window, you can change the deployment path and add a digital certificate, if necessary.



9. Click **OK.** The deployment package process begins. When completed the Deployment Status window displays.



10. Click **OK** to clear the window. The Deployment Status window closes.

COURSE SUMMARY

This lesson group includes the following lessons:

- Course summary

Course summary

Pega Robotic Automation Architect Essentials summary

Now that you have completed this course, you should be able to:

- Apply programming experience to develop a multi-project solution
- Create a standard Windows form using .NET window controls
- Integrate and interrogate a Windows application within the solution
- Integrate and interrogate a Web application within the solution
- Create a unified solution using project-to-project references
- Employ the use of Interaction Framework within the unified solution to seamlessly maintain and transfer data and activities across the solution
- Create a project and solution hierarchy structure to easily maintain and scale the unified solution
- Create automations to streamline the process by integrating the multi-project solution through programming logic
- Add and edit match rules on interrogated objects to create an understanding of the matching process for both Window and Web applications
- Generate build files and deployment packages of the unified solution
- Utilize the standard Visual Studio diagnostic and debugging tools to test the unified solution

Next steps

For more information on implementing automations and using them in Pega 7 applications, access the Robotic automation page on the PDN.