



---

## OpenSpan Core Training

### OpenSpan Studio – Diagnostics and Debugging

- **INTRODUCTION:** Training Module Overview
- **CHAPTER 1:** Debugging OpenSpan Projects
- **CHAPTER 2:** OpenSpan Studio Diagnostics Publishers
- **CHAPTER 3:** Error Handling and Suppression

© Copyright 2010 OpenSpan, Inc. All Rights Reserved

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

OpenSpan, Inc.  
4501 North Point Parkway  
Suite 140  
Alpharetta, Georgia 30022

OpenSpan<sup>®</sup> is a registered trademark of OpenSpan Inc., a Georgia Corporation.

SuperTrace<sup>®</sup> is a registered trademark of Green Hills Software, a Delaware Corporation.

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

<b>Introduction .....</b>	<b>4</b>
Prerequisites .....	4
Conventions .....	5
Chapter Overviews .....	6
Chapter 1: Debugging OpenSpan Projects.....	6
Chapter 2: OpenSpan Studio Diagnostic Publishers .....	6
Chapter 3: Error Handling and Suppression .....	6
<b>Chapter 1. Debugging OpenSpan Projects .....</b>	<b>7</b>
Objectives .....	7
OpenSpan Studio Debugging Features .....	7
Breakpoints .....	9
Running a Project with Breakpoints .....	10
Breakpoints Window .....	10
Exercise: Add Breakpoints and Step Into Project.....	12
Automation Locals .....	15
Automation Watches .....	16
Exercise: Using Automation Locals and Automation Watches .....	17
Threads.....	20
Call Stack .....	20
Exercise: Viewing Thread and Call Stack Information.....	21
<b>Chapter 2. OpenSpan Studio Diagnostic Publishers .....</b>	<b>23</b>
Objectives:.....	23
Application Diagnostic Settings.....	23
Log Categories.....	25
Log Settings.....	25
Log Settings Definitions .....	26
Diagnostic Publishers .....	26
File Publisher.....	26
SuperTrace Publisher .....	26
Event Log Publisher.....	26
Output Window Publisher .....	27
Exercise: Enabling Publishing to Output Window.....	28
DiagnosticsLog Component .....	30
LogBeforeExecution / LogAfterExecution .....	31
Exercise: Using the DiagnosticsLog Component .....	32
<b>Chapter 3. Error Handling and Suppression .....</b>	<b>37</b>
Try - Catch Exception Handling .....	37
Adding Try – Catch Exception Handling.....	37
Exception Types .....	38
Exercise: Adding Try..Catch Exception Handling.....	39
Testing the Project .....	40
Catching Project Error.....	40
Error Suppression.....	43
Automation Errors .....	43
Adapter Errors.....	43

## INTRODUCTION

This course describes how to use OpenSpan Studio and OpenSpan Runtime diagnostic functions to debug OpenSpan projects. In addition to Visual Studio's debug functions, OpenSpan provides a custom set of debugging tools specifically designed for OpenSpan projects.

Key topics covered in this course:

- OpenSpan Debugging Functions: break points on execution links, step into and step over, view locals window
- Setting Up OpenSpan Studio Diagnostic Publishers
- Using the DiagnosticsLog component
- Setting Error Suppression

### Prerequisites


This course requires successful completion of the **OpenSpan Studio Basics** training module. You will need a working knowledge of OpenSpan Studio and the ability to create basic Windows and Web-based projects.

The course requires the following:

- OpenSpan Studio 4.5
- Sample CRM application (installation file provided in OpenSpan Studio installation Extras folder)
- Sample solutions (downloaded from the OpenSpan Community website) and in Extras folder
- Training project files available for download from the [OpenSpan Community website Learning and Certification page](#) (download and then extract Zip file to a folder location of your choice)  
**Note:** If you are using the OpenSpan Studio plug-in, change the extensions of the sample solution files from .ossln to .sln.

## Conventions

This document uses the following typographical rules and conventions:

Text Appearance	Meaning
<b>Black bold characters</b>	<b>Names of program elements that require emphasis, such as command buttons, menus, and dialog boxes, are shown in black bold text.</b>
<b>Blue Bold Characters</b>	<b>Text that you are supposed to type or data selections, such as from drop-lists, appear in blue boldface characters.</b>
<b><u>Remember</u></b>	<b><u>Definitions of terms and important concepts that bear remembering.</u></b>
	Next to the Tip icon, you can find best practices and shortcuts to use OpenSpan Studio more effectively.

## **Chapter Overviews**

### **Chapter 1: Debugging OpenSpan Projects**

Describes the OpenSpan Studio debugging functions, including: Setting and using Breakpoints, Viewing Automation Locals, Setting Automation Watches, Viewing Threads and Call Stacks.

### **Chapter 2: OpenSpan Studio Diagnostic Publishers**

Describes how to enable application diagnostic message reporting through the OpenSpan publishers: File and Output Window. This chapter describes the Log Categories for use with diagnostics message reporting. Steps are provided for including diagnostics reporting within automations– the DiagnosticsLog and event link LogBeforeExecution/LogAfterExecution

### **Chapter 3: Error Handling and Suppression**

Describes how to use Try..Catch functions to handle errors incurred during automation execution. In addition, steps for suppressing automation and adapter exceptions are included.

## CHAPTER 1. DEBUGGING OPENSPAN PROJECTS

OpenSpan Studio debugging tools are specifically designed to debug runtime and logical errors in OpenSpan projects. The tools provide functions for diagnosing runtime errors such as applications not starting and/or controls not matching. In addition, OpenSpan debugging tools provide functions for diagnosing logical errors in automations and thread handling errors.

### Objectives

By the end of the chapter you will be able to:

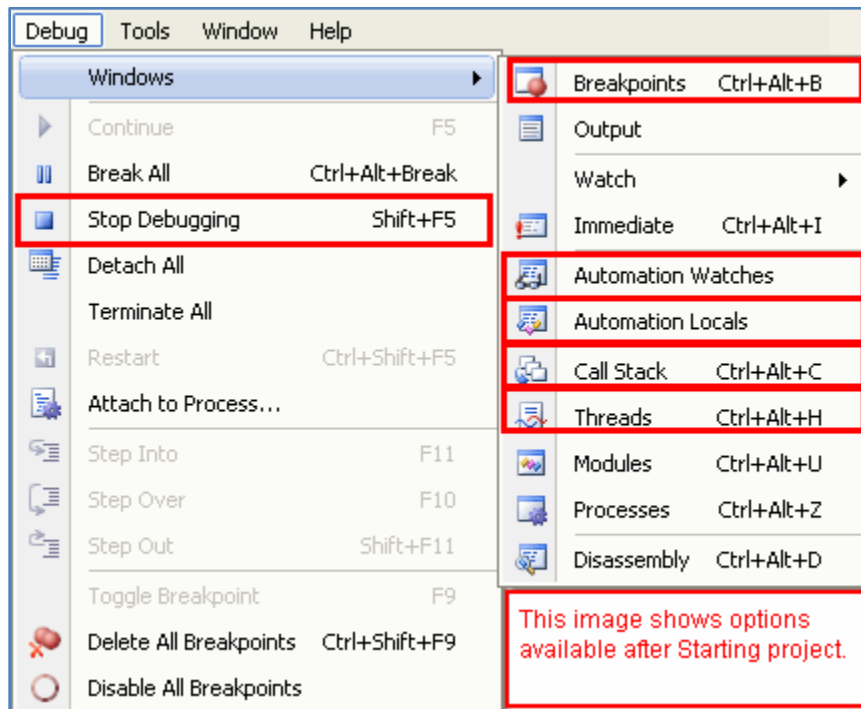
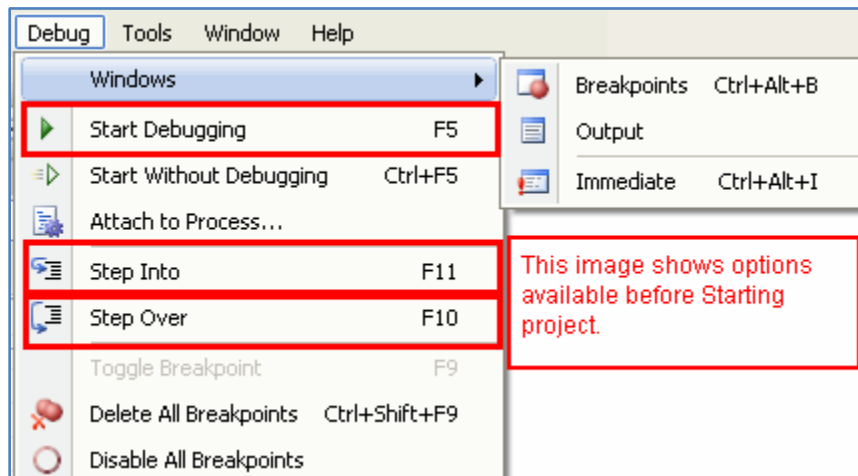
- Debug a project by using the Step Into, Step Over and Step Out commands
- Add Breakpoints to automations for stepping through OpenSpan project execution.
- Use Automation Watches and Automation Locals to view data values and types for OpenSpan project controls during project execution.
- View Threads and associated Call Stacks active while running an OpenSpan project

### OpenSpan Studio Debugging Features

OpenSpan Studio provides the following debugging functions:

- **Breakpoints:** Enable you to define locations along automation event paths (Breaks) which you can use with the Step Into, Step Over, and Step Out functions to cause the project halt when the Break location is reached.
- **Step Into, Over and Out:** Enables you to run a project halting at each Breakpoint for viewing project information, such as automation locals and watches, and threads/call stacks.
- **Automation Watches:** Enable you to define data ports on automation connection blocks which display value and type information for any automation in the project, including automations that are not currently in scope (not running at a particular point during the project debugging process).
- **Automation Locals:** Enables you to view value and type information for automation objects while the automation is in scope (actively running in the project).
- **Threads:** Enables you to view the active threads.
- **Call Stack:** Enables you to view a list (in order) of the calls being made for a selected thread.

These functions are located on the **Debug | Windows** menu:



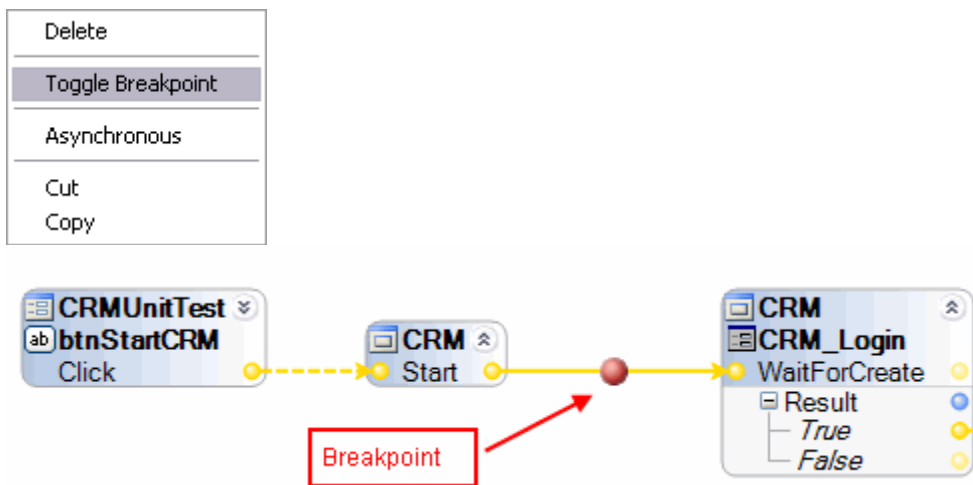


## Breakpoints

Use Breakpoints along with the Step Into function to halt the execution of automations at specific event execution path locations so that you can observe runtime behavior and note the values of automation watches, locals and threads. As you are debugging a project, OpenSpan displays all of the Breakpoints in the Breakpoints window. You can view information about the Breakpoints and toggle Breakpoints on/off while the project is running.

The steps below outline how to add, disable, and remove Breakpoints:

1. Open the automation you want to run in Break mode.
2. Right-click on the event link where you want to place a Breakpoint and select Toggle Breakpoint from the local menu. A red sphere appears on the event link.

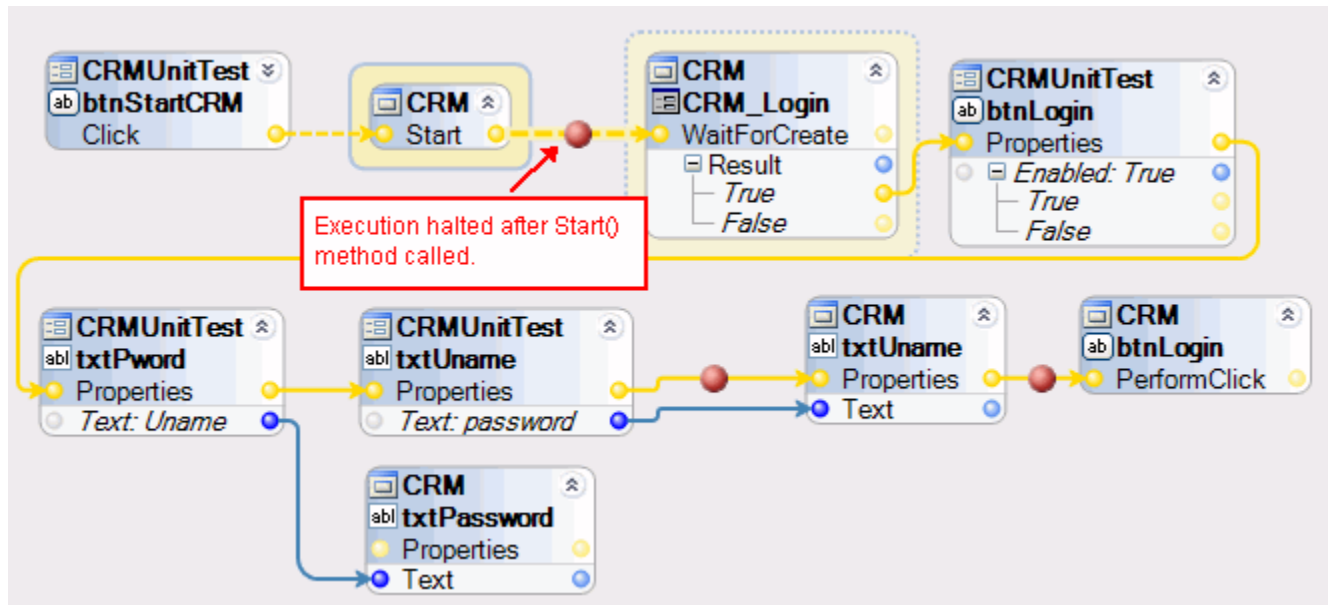


3. Once set, the Breakpoint appears as red dot over the execution link.
4. To completely remove a Breakpoint from an automation, right-click on the Breakpoint and select Delete from the local menu.
5. You also have the option of Disabling a Breakpoint. Disabling a Breakpoint leaves the Breakpoint at the current position; however automation execution does not halt at the location when you run the project. You can toggle the Breakpoint back to Enabled if you need to continue debugging the automation and require the execution to halt at the Breakpoint location.
6. To remove all Breakpoints from a project, select Debug | Delete All Breakpoints.

## Running a Project with Breakpoints

After setting Breakpoints within your project automation(s), select **Start Debugging** from the **Debug** toolbar. OpenSpan will launch the Runtime application, load and start the project. When a Breakpoint is reached, the project execution stops. The automation shows the Break location and the Debug windows display.

An example of an automation halted at the first Breakpoint follows:

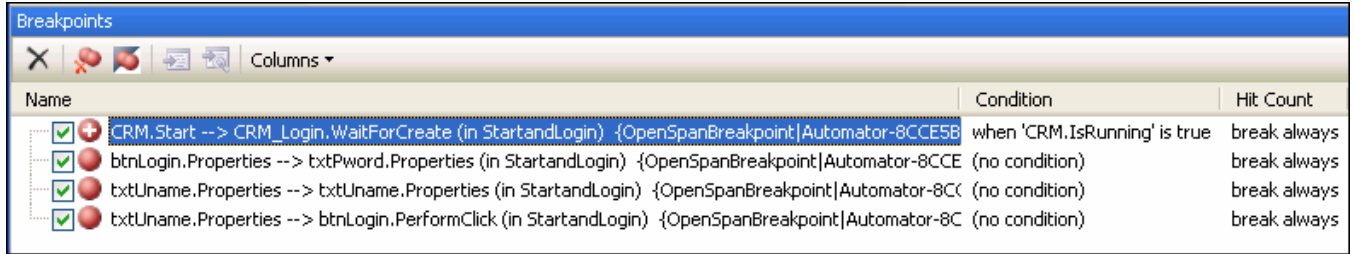


Continue the execution by:

- **F5** key continues the execution until the next Breakpoint.
- **F10** key moves execution one step forward along event path (**Step Over** command on the Debug menu).
- **F11** key moves execution to next step along either the event or data path (**Step Into** command on the Debug menu).

## Breakpoints Window

The **Breakpoints** window lists all the Breakpoints in the project. While modifying a project you can view the Breakpoints list by selecting **Debug | Windows | Breakpoints**. From this window you access Breakpoint functions, such as setting a Breakpoint Condition. While running a project which contains Breakpoints, the Breakpoints window displays by default. You cannot modify Breakpoint properties while the project is running. An example of the Breakpoints window follows:



Name	Condition	Hit Count
<input checked="" type="checkbox"/> CRM.Start --> CRM_Login.WaitForCreate (in StartandLogin) {OpenSpanBreakpoint Automator-8CCE5B}	when 'CRM.IsRunning' is true	break always
<input checked="" type="checkbox"/> btnLogin.Properties --> txtPword.Properties (in StartandLogin) {OpenSpanBreakpoint Automator-8CCE}	(no condition)	break always
<input checked="" type="checkbox"/> txtUname.Properties --> txtUname.Properties (in StartandLogin) {OpenSpanBreakpoint Automator-8C}	(no condition)	break always
<input checked="" type="checkbox"/> txtUname.Properties --> btnLogin.PerformClick (in StartandLogin) {OpenSpanBreakpoint Automator-8C}	(no condition)	break always

### Breakpoints Window Columns

The Breakpoints window displays the following for each Breakpoint in the project:

**Enabled/Disabled Checkbox:** A check appears next to the Breakpoint name if the Breakpoint is enabled. While modifying a project, you can clear the checkbox to disable the Breakpoint.

**Name:** The Breakpoint is named according to the connection blocks between which the break is located, the name of the automation, and an OpenSpan GUID.

**Condition:** Displays information about conditional Breaks. These are Breakpoints that halt execution only if a specific expression is true (for example, in a tree view with the automations at the top level and breakpoints the second level. Use the Breakpoint functions to set the Condition.

**Hit Count:** Displays the hit condition required for enforcing the Breakpoint. Using the Breakpoint functions, you can set a Breakpoint to be active based on the number of times it has been 'hit' during the project execution.

### Breakpoints Window Buttons

The following **Breakpoint Window Buttons** are applicable to OpenSpan automation Breakpoints:



- Deletes the selected (highlighted) Breakpoint



- Deletes all Breakpoints in the project



- Disables/Enables the selected (highlighted) Breakpoint



- Lists additional properties to display in the window

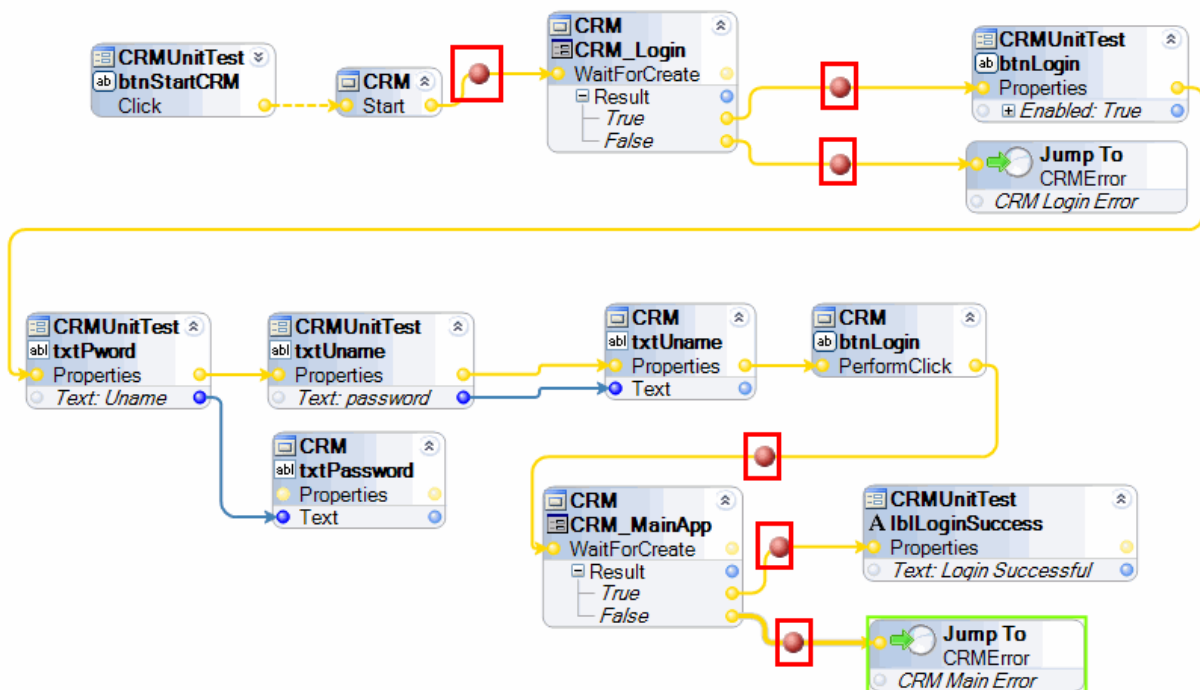
### Exercise: Add Breakpoints and Step Into Project

Use the steps below to set add Breakpoints to a project and then debug the project by using the Step Into function. This exercise requires the following setup:

- OpenSpan Extras - Training solutions and CRM.msi setup application must be installed.
- CRM.exe must be installed on your computer to the following location: C:\Program Files\OpenSpan\CRM Setup (the installation file CRM.msi is included in the Extras folder for OpenSpan Studio).

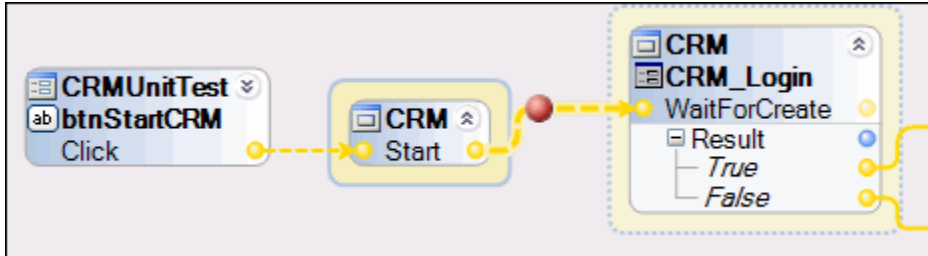
Begin this exercise by unzipping the Debugging\_DiagTr1.zip file to a Project folder named: Debugging\_DiagTr1. This file can be downloaded from the OpenSpan Community website.

1. In OpenSpan Studio, open the Debugging\_DiagTr1 solution from the Debugging\_DiagTr1 folder.
2. Open the **StartandLogin** automation.
3. Add **Breakpoints** on the following event links:
  - Between the **CRM Start** and **CRM\_Login.WaitForCreate** methods.
  - Immediately after the **True** and **False** events of the **CRM\_Login.WaitForCreate** method
  - Between the **CRM.btnLogin.PerformClick** method and the **CRM\_MainApp.WaitForCreate** method
  - Immediately after the **CRM\_MainApp.WaitForCreate** method on both the **True** and **False** output event links.
4. To add a **Breakpoint**, right-click on the event link and select **Toggle Breakpoint**. Your automation should look like the following:

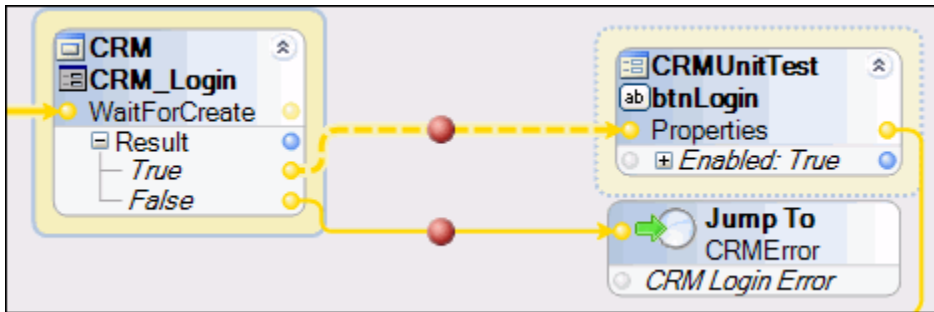


5. Save the solution.

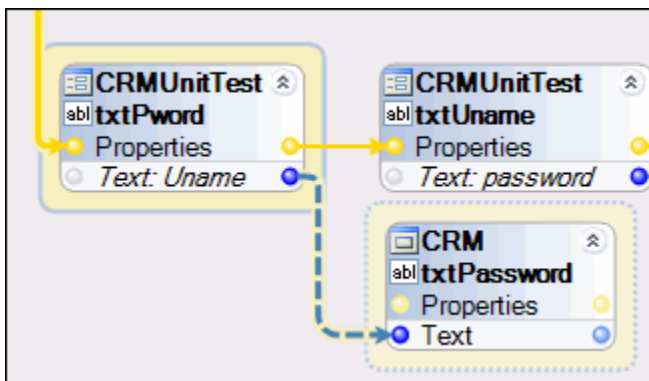
6. Select **Start Debugging** from the **Debugging** toolbar. After the project build process completes, the **CRM Unit Tester** window displays.
7. Click the **Start CRM** button on the CRM Unit Tester window. OpenSpan starts the CRM application and the project execution moves to the first **Breakpoint**. Bring OpenSpan Studio into focus as the top-most window. The **StartandLogin** automation should show a box surrounding the **CRM\_Login.WaitForCreate** method and the event link as a dashed line.



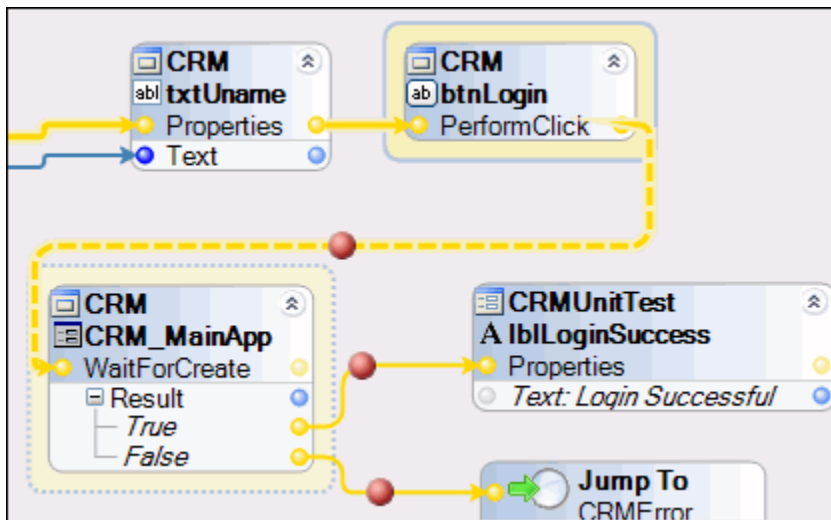
8. Press **F5** to move to the next **Breakpoint**. Execution moves past the **CRM\_Login.WaitForCreate** method. If the Login window has opened and is matched, the True event link displays as a dashed line:



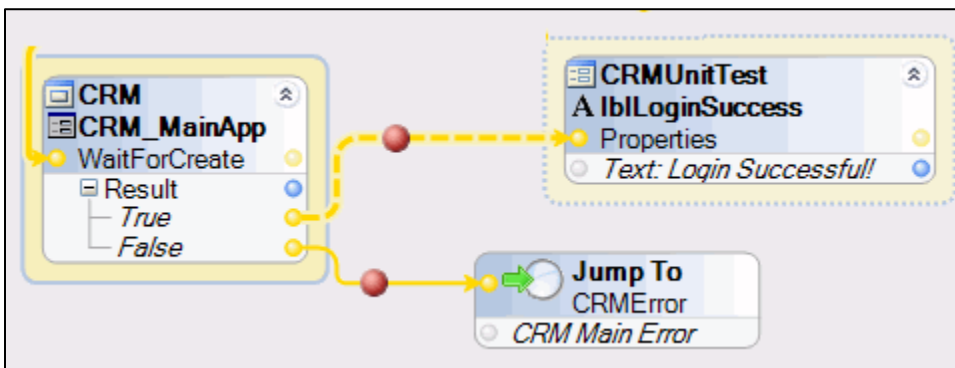
9. Press **F11** or select **Step Into** to move execution to the next event link. Press **F11** again to move to the next **data** link (CRMUnitTest.txtPword.Text property and CRM.txtPassword.Text property) as follows:



10. Press **F5** to move to the next **Breakpoint** as follows:



11. Press **F5** to move to the next **Breakpoint**. If the **CRM\_MainApp.WaitForCreate** method has a True result (i.e., the Main CRM window is open and matched), the True event link appears as a dashed line:



12. Press **Shift+F11** or **Step Out** from the **Debug** menu to continue the project execution until completion.
13. Stop the project.

## Automation Locals

The **Automation Locals** window displays the values and types of all data ports for the active automation. To view the Automation Locals your automation must include at least one Breakpoint. The automation locals are updated as you Step Into, Step Over, and/or move to each Breakpoint. To open the Automation Locals window, select **Debug | Windows | Automation Locals** from the main menu.

An example of the **Automation Locals** window follows. This window displayed at the `btnLogin.PerformClick --> CRM_MainApp.WaitForCreate` Breakpoint in the solution from [Exercise: Add Breakpoints and Step Into Project](#).

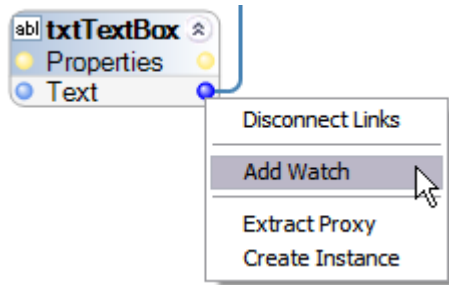
Automation Locals		
Name	Value	Type
CRM_NewCall.Result	True	System.Boolean
txtCallAccNo.Text	23453	System.String
CRM.IsRunning	True	System.Boolean
messageDialog1.Result	ConnectableWorker - message...	System.String
lblAccNo.Text	23453	System.String
messageDialog1.Result	ConnectableWorker - message...	System.String

You can also use the automation locals displayed to navigate while the debugger is active. By double-clicking an item in the Automation Locals window focus is placed on the associated link in the automation.

## Automation Watches

Automation Watches display data for selected data ports on any project automation. Automation Watches enable you to select the objects for which you want to view runtime values and types. In order to view Automation Watch information, you must first designate the data Watch ports in your automations.

To designate an Automation Watch, right-click on the data port and select Add Watch from the local menu.



An example of the Automation Watches window follows:

Automation Watches		
Name	Value	Type
txtCallAccNo.Text	23453	System.String

To remove a watch, select it in the Automation Watches window and press Delete.



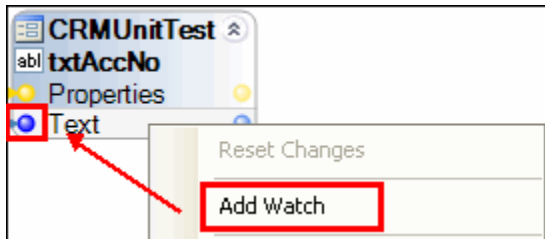
### Exercise: Using Automation Locals and Automation Watches

Use the steps below to set Automation Watches and view both Automation Locals and Automation Watches. This exercise requires the following setup:

- OpenSpan Extras - Training solutions and CRM.msi setup application must be installed.
- CRM.exe must be installed on your computer to the following location: C:\Program Files\OpenSpan\CRM Setup (the installation file CRM.msi is included in the Extras folder for OpenSpan Studio).

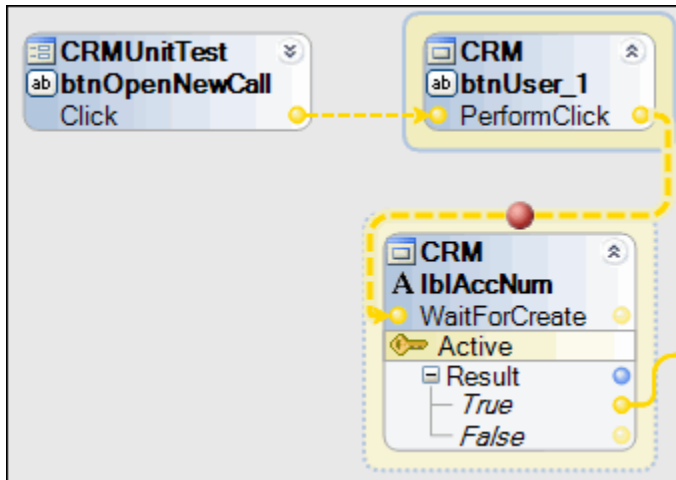
Begin this exercise by unzipping the Debugging\_DiagTr2.zip file to a Project folder named: Debugging\_DiagTr2. This file can be downloaded from the OpenSpan Community website.

1. In OpenSpan Studio, open the Debugging\_DiagTr2 solution from the Debugging\_DiagTr2 folder.
2. Open the OpenNewCallWindow automation.
3. Add an Automation Watch for the Text input data port of the CMRUnitTest.txtWinKey.Text property:



4. Save the solution.
5. Select Start Debugging from the Debugging toolbar.
6. On the CRM Unit Tester window, click the Start CRM button. The CRM application starts and the Login process completes.

7. On the CRM Unit Tester Window, click the New Call button. The first Breakpoint in the OpenNewCall automation is hit.



8. Press F5 to move to the next Breakpoint. Press F5 to complete the automation logic. The Key for the New Call window displays on the CRM Unit Tester Window.
9. Open the Automation Locals window. Values and Types display for the properties of the controls on the automation.

Name	Value	Type
txtAccNo.Text		System.String
lblAccNum.Result	True	System.Boolean
lblAccNum.Text	23453	System.String

10. Open the Automation Watches window. Information displays only for the CRMUnitTest.txtWinKey.Text property. Press F5 to pass the Breakpoint and set the txtAccNo.Text property. The value and type display on the Automation Watches window:






The screenshot displays the OpenSpan Studio interface. The top section shows a workflow diagram with several steps: 'CRMUnitTest btnOpenNewCall Click', 'CRM btnUser\_1 PerformClick', 'CRM A lblAccNum WaitForCreate', 'CRM A lblAccNum Active', and 'CRMUnitTest txtAccNo Properties Text'. A red arrow points from the 'Text' property of the 'txtAccNo' control to the 'Automation Watches' window at the bottom. The 'Automation Watches' window contains a table with the following data:

Name	Value	Type
txtAccNo.Text	23453	System.String

11. Stop the project.


## Threads

The Threads window lists active threads by Thread ID. Double-clicking a thread takes you to the call stack for that thread.

Threads						
	ID	Category	Name	Location	Priority	Suspend
 	36	 Worker Thread				0
	14	 Worker Thread				0

## Call Stack

The Call Stack window displays the stack trace for the current thread. It only monitors the currently selected thread, allowing you to watch for changes to that thread and not interfere with the normal execution scheduler.

Call Stack	
Name	Language
 CRM_NewCall.WaitForCreate	
btnUser_1.PerformClick	

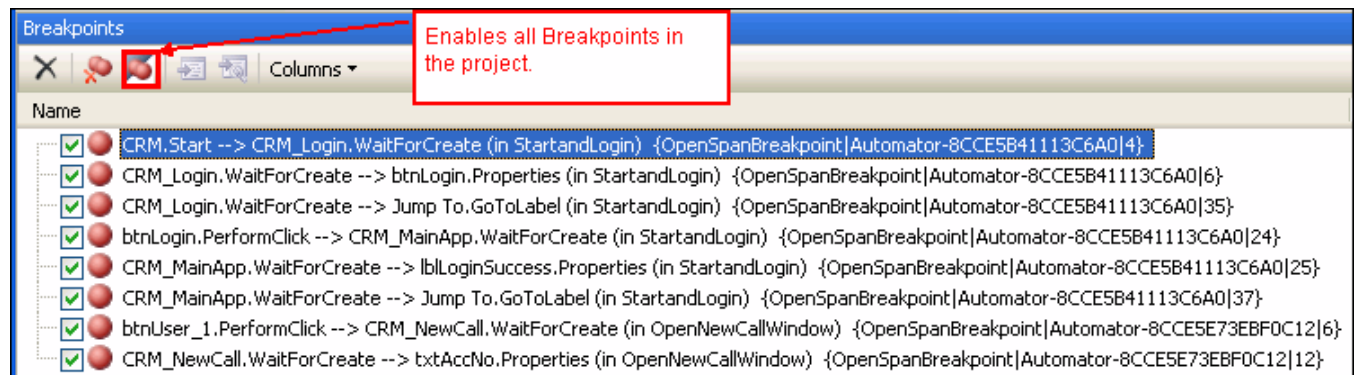
**Exercise: Viewing Thread and Call Stack Information**

Use the steps below to view Thread and Call Stack information. This exercise requires the following setup:

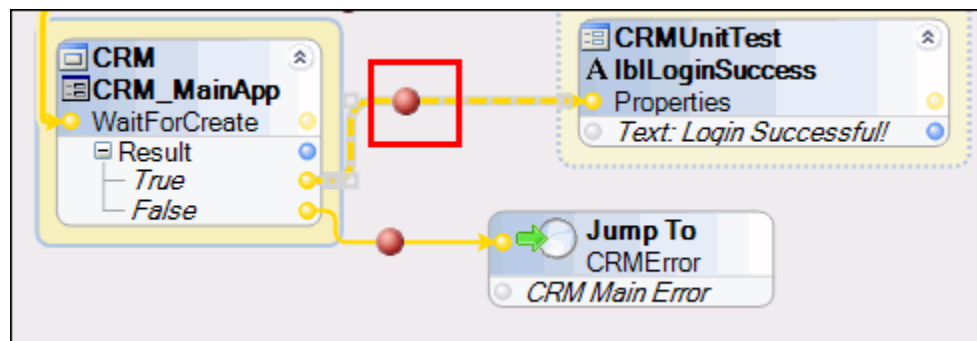
- OpenSpan Extras - Training solutions and CRM.msi setup application must be installed.
- CRM.exe must be installed on your computer to the following location: C:\Program Files\OpenSpan\CRM Setup (the installation file CRM.msi is included in the Extras folder for OpenSpan Studio).

This exercise uses the same solution as the Using Automation Locals and Watches exercise. Continue using the Debugging\_DiagTr2 solution in the Debugging\_DiagTr2 folder. If you have not completed the Using Automation Locals and Watches exercise, download the file from the OpenSpan Community website and unzip the Debugging\_DiagTr2.zip file.



1. In OpenSpan Studio, open the Debugging\_DiagTr2solution from the DebuggingEx2 folder.
2. Open the Breakpoints window (Select Debug | Windows | Breakpoints from the main menu).
3. In the Breakpoints window, verify that all Breakpoints are enabled. If not, click on the Enable All Breakpoints button.




4. Save the solution.
5. Select Start Debugging from the Debugging toolbar. The CRM Unit Tester window opens.
6. Click the Start CRM button on the CRM Unit Tester window. The first Breakpoint in the StartandLogin automation is hit.
7. Press F5 repeatedly until the CRM\_MainApp.WaitForCreate --> lblLoginSuccess.Text Breakpoint.



8. Open the Threads window. A single Main thread displays:

Threads						
	ID	Category	Name	Location	Priority	Suspend
	12	 Main Thread				0

9. Open the Call Stack window. A list of connection blocks displays showing all of the properties, methods, and events executed on the Main Thread.

Call Stack	
	Name
	CRM_MainApp.WaitForCreate
	btnLogin.PerformClick
	txtUname.Properties
	txtUname.Properties
	txtPword.Properties
	btnLogin.Properties
	CRM_Login.WaitForCreate
	CRM.Start

10. Stop the project.

## CHAPTER 2. OPENSPAN STUDIO DIAGNOSTIC PUBLISHERS

OpenSpan Studio includes diagnostic tools to help you in both fault finding and performance evaluation of your OpenSpan projects.

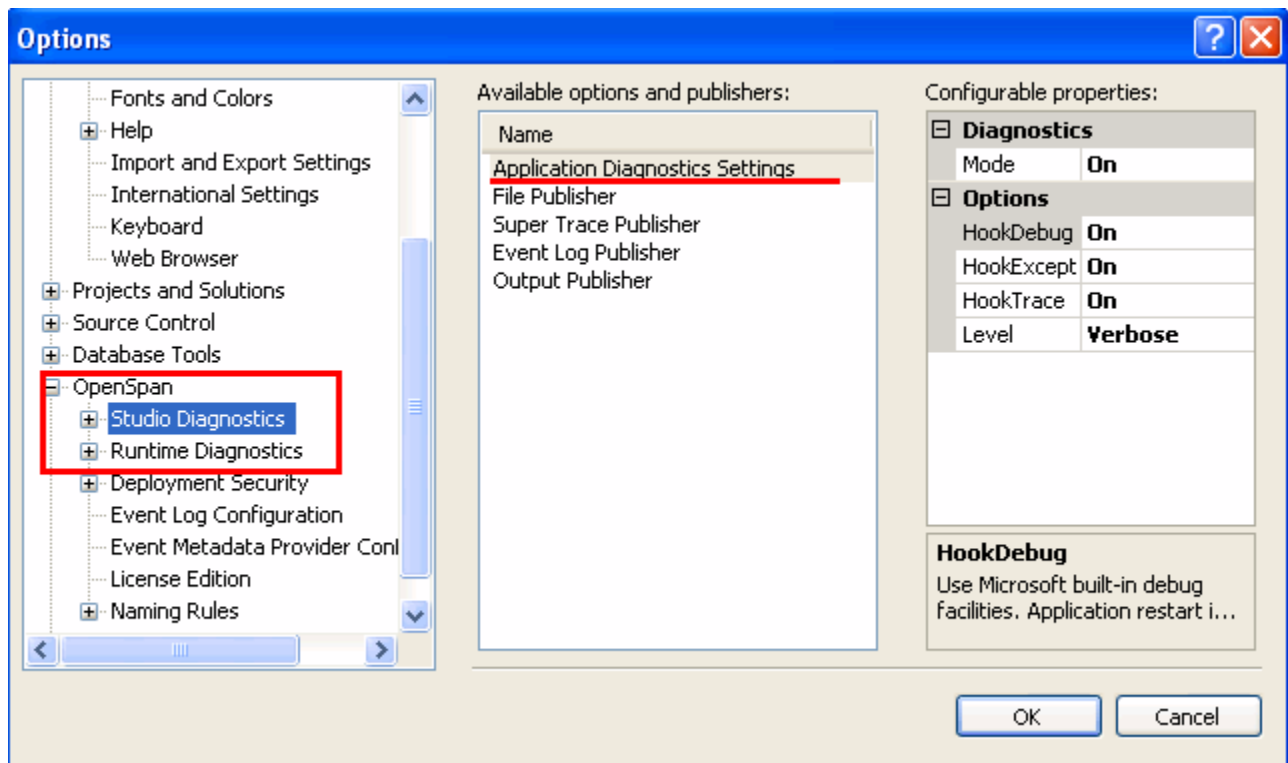
### Objectives:

By the end of this chapter you will be able to:

- Set the OpenSpan Studio Application Diagnostic Settings to enable diagnostic publishing.
- Enable the File Publisher and use the DiagnosticsLog component to write messages to the StudioLog.txt file.
- Set the Log Levels and Categories to control the types of diagnostic messages published.
- Understand how to enable Output Window publisher.

### Application Diagnostic Settings

Diagnostic settings are managed in the Diagnostics Configuration window (**Tools | Options | OpenSpan**). You set diagnostics for OpenSpan Studio separately from OpenSpan Runtime. The settings are saved in the StudioConfig.xml and RuntimeConfig.xml files, respectively.

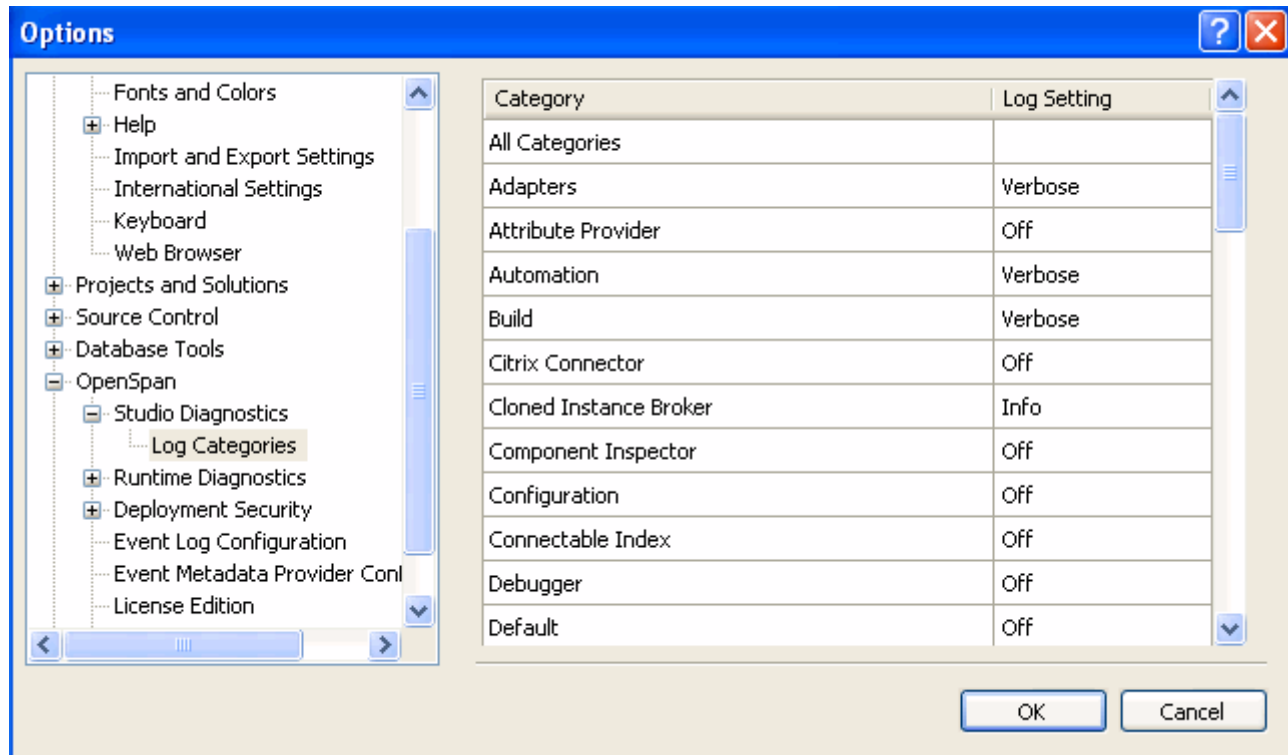


Application Diagnostics Settings	
<b>Mode</b>	Set to On to use any of the individual diagnostic tools (e.g., File Publisher). Alternatively, turning on any individual tool automatically turns on Application Diagnostics Settings. Turning off Application Diagnostics Settings turns off all of the individual diagnostics tools.
<b>HookDebug</b>	Enables Listener service for standard debug messages. Debug messages are removed from release versions of OpenSpan software making this option of limited use to end-users.
<b>HookException</b>	Enables UnhandledExceptionHandler that will show an OpenSpan custom message in the event of an error not handled by the OpenSpan code base. If disabled (Off) and such an error occurs, the standard Windows <i>Unhandled Exception</i> error message is issued, and the error is not logged.
<b>HookTrace</b>	Setting HookTrace to On enables a diagnostic listener to the standard .NET trace messages and any enabled OpenSpan publishers will log messages routed to <code>system.diagnostics.trace(...)</code> . This tool is of limited value to end-users since OpenSpan Studio is now largely standardized to use its internal diagnostic API for outputting diagnostic messages.
<b>Level</b>	Sets the logging severity level for a small number of initialization messages, including when the Diagnostic Publisher is started and the OpenSpan assembly version. It does not affect the message logging for individual log categories or for the DiagnosticsLog automation component.



## Log Categories

Log Categories are predefined values that filter the quantity and type of events that are logged. OpenSpan provides separate log categories for different components (e.g., Windows Adapter) and operations (e.g., Matching). An example of the Log Categories dialog follows. In this sample, Adapters, Automations, and Build categories are set to Verbose reporting while the other categories are set to “Off” – no diagnostic reporting.



A complete list of categories and the associated messages can be found in the OpenSpan documentation. **The selections on the Log Categories dialog are saved in the StudioConfig.xml and RuntimeConfig.xml files, respectively.**

## Log Settings

Log Settings are used to assign a desired level of message detail to log categories. Every event that potentially can be written to the log has an associated event level that cannot be changed. Every log category is also assigned an event level; however category event levels can be altered by changing the Log Setting.

By default, the log setting is *Info* for all categories. OpenSpan will log an event only if its level is equal to or more restrictive than the log setting for the associated log category. For example, if an event level is *Info* and the log setting for its associated category is the less restrictive *Verbose*, then the event will be logged.

## Log Settings Definitions

Descriptions of the event levels (or Log Settings) follow:

Level	Description
<b>Error</b>	Logs only Error events. This is the most restrictive option (i.e., logs the fewest events).
<b>Warning</b>	Logs Warning and Error events.
<b>Info</b>	Logs Info, Warning, and Error level events.
<b>Verbose</b>	Logs Verbose, Info, Warning, and Error event. This is the least restrictive option (logs the most messages).
<b>Off</b>	Nothing will be logged for the category.

## Diagnostic Publishers

### File Publisher

The File Publisher creates output files of diagnostic messages. OpenSpan creates the following files:

- **StudioLog.txt** - diagnostic messages from execution of the OpenSpan Studio application.
- **RuntimeLog.txt** - diagnostic messages from execution of the OpenSpan Runtime application.
- **CompilerLog.txt** - messages generated during build process of an OpenSpan project.
- **DeploymentLog.txt** - messages generated during the process of creating an OpenSpan deployment package for a project.

The type of messages written to the file depend on the source (Studio, Runtime, build, etc.) and the error levels set by the Log Categories.

The File Publisher also publishes messages generated within the execution of automations through use of the DiagnosticsLog component and link properties (LogBeforeExecution/LogAfterExecution).

In order to see the DiagnosticsLog and link messages, the FilePublisher Mode setting must be ON and the Default and Automation log categories must be set to a value other than “Off”. See the description of the [DiagnosticsLog Component](#) for more information.

### SuperTrace Publisher

The SuperTrace Publisher writes detailed messages at the ProcessID and ThreadID level for OpenSpan Studio and/or OpenSpan Runtime. Generally, you would only enable SuperTrace logging when troubleshooting issues with OpenSpan support.

### Event Log Publisher

This publisher records logs messages to the Windows Application log. See the OpenSpan on-line Help for topics detailing the use of the application event log.

## Output Window Publisher

OpenSpan Studio and/or OpenSpan Runtime diagnostic messages can be written to the Output window. The content of the messages written to the Output window is the same as the content written by the File Publisher. Descriptions of the diagnostic message information follows:

### Output window column descriptions

Column	Description
<b>Level</b>	Describes the output level of the item message.
<b>Time</b>	Time the event was logged.
<b>Thread</b>	Thread ID being executed.
<b>Category</b>	Log category of the logged item.
<b>Design Component</b>	Project item associated with the logged item.
<b>Component</b>	Integrated component that data is coming from, going to, or is being used.
<b>Text</b>	Description logged event.

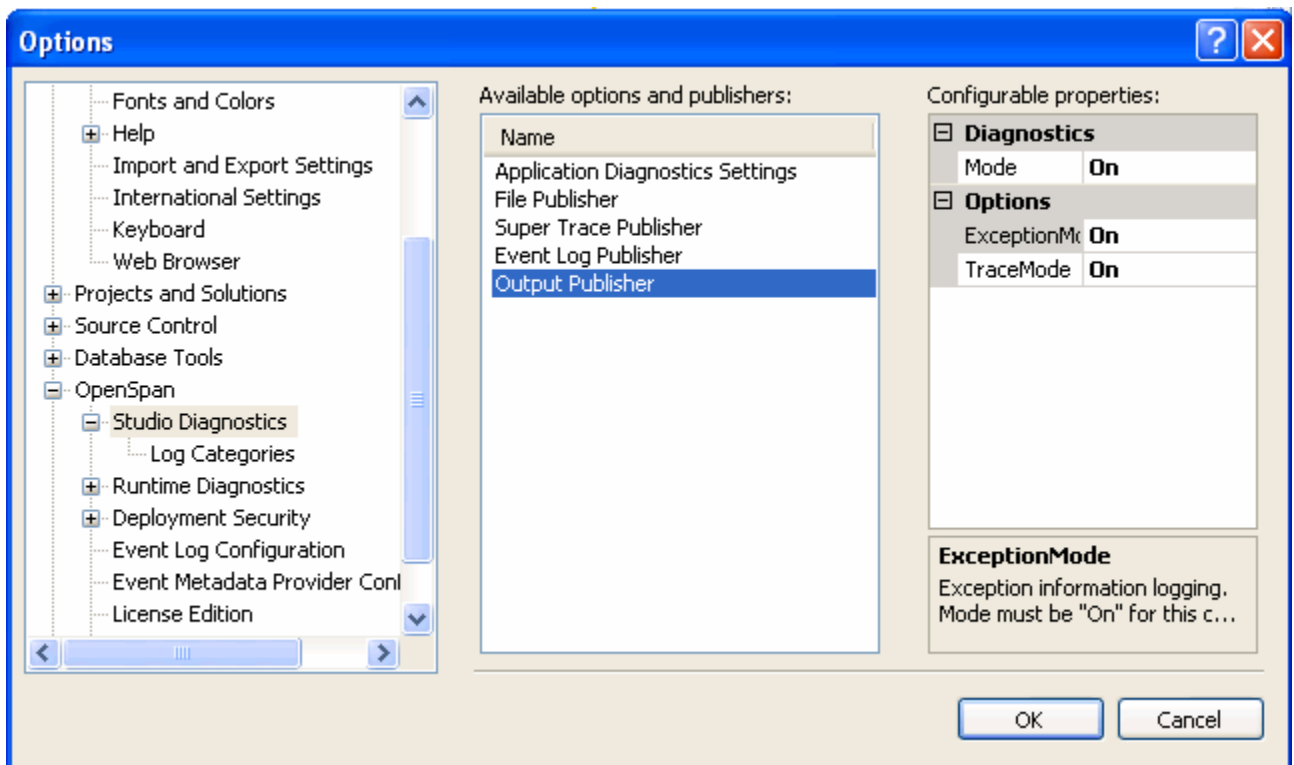
### Exercise: Enabling Publishing to Output Window

Follow these steps to display and view diagnostic messages in the OpenSpan Studio Output window. This exercise requires the following setup:

- OpenSpan Extras - Training solutions and CRM.msi setup application must be installed.
- CRM.exe must be installed on your computer to the following location: C:\Program Files\OpenSpan\CRM Setup (the installation file CRM.msi is included in the Extras folder for OpenSpan Studio).

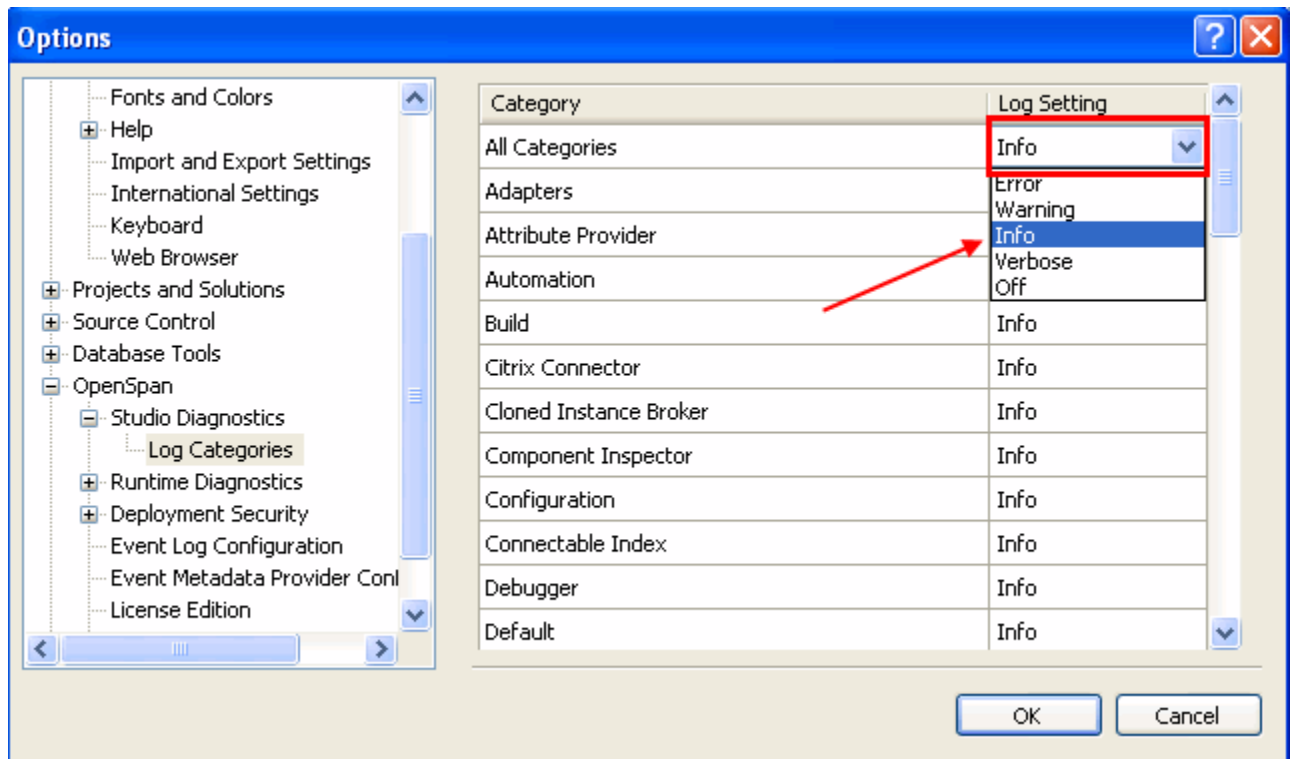
**Note:** You can set the OpenSpan Diagnostic options without having a solution open in OpenSpan Studio.

1. In OpenSpan Studio, select **Tools | Options**.
2. From the **Options** window, select **OpenSpan | Studio Diagnostics**.
3. Select the **Output Publisher** option from the **Available options and publishers** list box.



4. Set all **Configurable Properties** to **On**.
5. Expand the **Studio Diagnostics** node and select **Log Categories**.

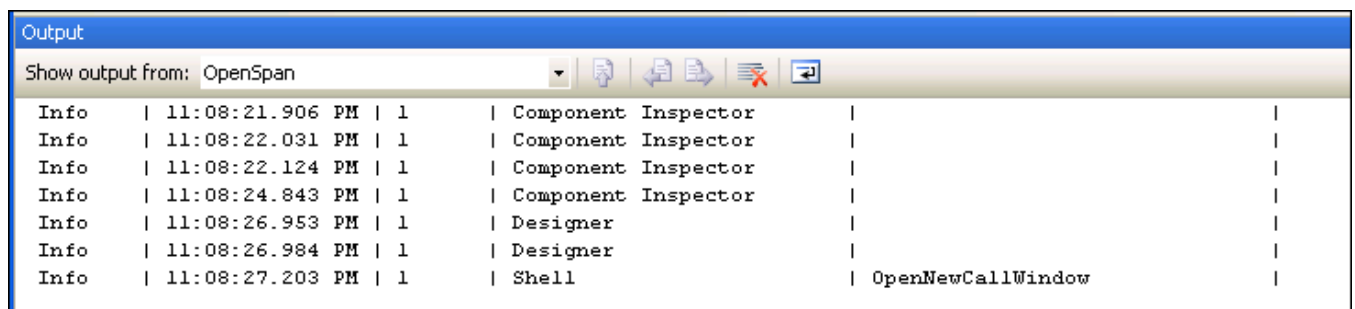
- For the **All Categories** Category, set the **Log Setting** to **Info**. Click **OK** to save the Category settings.



- To have the Category setting changes take effect, you must toggle the Application Diagnostics Settings Mode from On to Off, and back to On again. After toggling the Mode back to On, click OK to save all settings.
- Open the Debugging\_DiagTr2 solution, or any current solution.
- Select **View | Output** to open the **Output** window. To show the log information in the output window, make sure the dropdown box has **OpenSpan** selected.

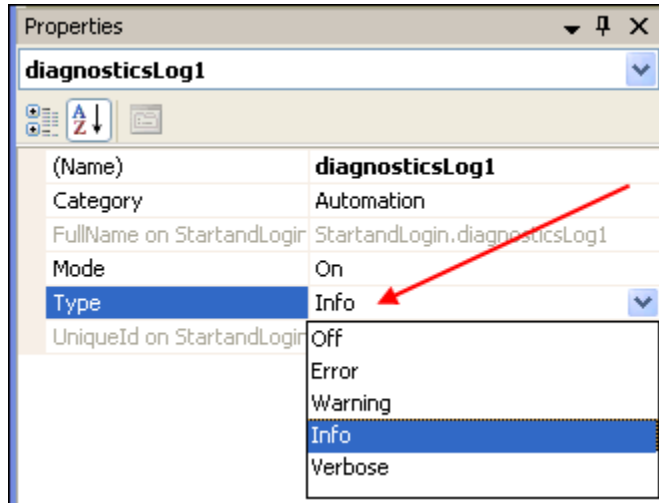


- Close all open project items.
- Right-click anywhere in the Output window and then select **Clear All** from the context menu.
- In the Solution Explorer, open an automation. Note that messages appear in the Output window related to opening the automation:



## DiagnosticsLog Component

The DiagnosticsLog component allows you to write custom log entries from specific points in a project's execution by using the component in OpenSpan automations. Use the **Type** field in the Properties window to select the diagnostic level for the **DiagnosticsLog** component:

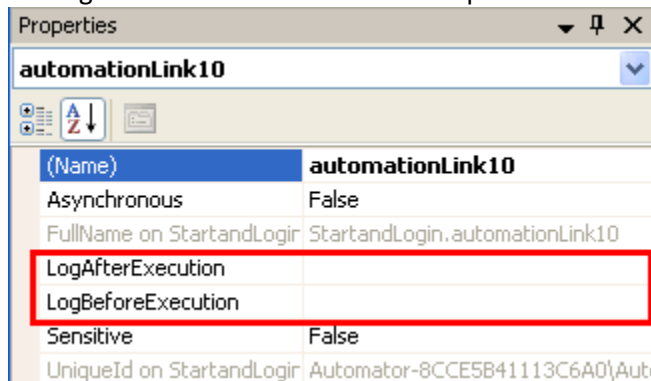


### Notes:

- The DiagnosticsLog component is dependent on the Diagnostics Configuration window's *Default* and *Automation* log categories. The DiagnosticsLog Type property must be set to a level that is equal to or less restrictive than the Application Diagnostics Level. For example, if the Application Diagnostics Level is set at Error and the DiagnosticsLog Level property is set at Verbose, OpenSpan will not log messages for the DiagnosticsLog component.
- The Type property *Off* option does not disable logging for the DiagnosticsLog component. The Mode property turns logging on or off. The Type property is compared to the Diagnostics Configuration window Default log category (as described above), and the Off option is the most restrictive setting for that comparison.

## LogBeforeExecution / LogAfterExecution

The LogBeforeExecution and LogAfterExecution properties on data and event links enable you to publish messages to the File Publisher and Output window before/after the execution of link.



LogAfterExecution writes the message after a thread has completed. In the log, you might see the LogBeforeExecution entry at the time a link is executed, but the after message appears much later. In terms of execution, OpenSpan Studio writes the LogBeforeExecution message, executes the component, and the rest of the thread, which potentially includes other automations, and then finally writes the LogAfterExecution message when the thread is terminated.

To use the LogBeforeExecution and LogAfterExecution properties, select the automation link for which you want to enable messaging, and then enter the message text to be written to the log file/output window.

### Exercise: Using the DiagnosticsLog Component

Follow these steps to publish diagnostic messages during the execution of an automation by using the DiagnosticsLog component. This exercise requires the following setup:

- OpenSpan Extras - Training solutions and CRM.msi setup application must be installed.
- CRM.exe must be installed on your computer to the following location: C:\Program Files\OpenSpan\CRM Setup (the installation file CRM.msi is included in the Extras folder for OpenSpan Studio).

This exercise uses the same solution as the Using Automation Locals and Watches exercise. Continue using the CRMDebugging solution in the Debugging Ex2 folder. If you have not completed the Using Automation Locals and Watches exercise, download and unzip the Debugging\_DiagTr2.zip file to a Project folder named: Debugging\_DiagTr2. This file can be downloaded from the OpenSpan Community website.

1. Open the Breakpoints window and click the Disable All Breakpoints button.
2. Run the project. Click the New Call button **without clicking the Start CRM** button on the CRM Unit Tester window to start the CRM application. The project does not display the account number in the Account textbox.
3. With the project running, open the Output window and select OpenSpan Runtime as the output type. Note the automation exception message stating that the btnUser\_1 is not matched which is followed by the exception:

Info | hh:mm:ss.nnn PM | 11 | Automation | OpenNewCallWindow | btnUser\_1 | PerformClick: Exception invoking method - The control btnUser\_1 is not matched.

When writing using a control in an application, it is a good practice to ensure that the control is created (matched) before using the control in automation logic. This way, if a project application closes unexpectedly (e.g., an end-user closes the application while the project is running), logic can be added to the automation to restart the application or execute alternative logic.

In this exercise, the following steps lead you through adding the **IsCreated** property for the btnUser\_1 control and then using a DiagnosticsLog component to write an error to the diagnostic log. To recover from the error, a MessageDialog is used to inform the user that the application is not running and instructing the end-user to click the Start CRM button on the CRM Unit Tester window.

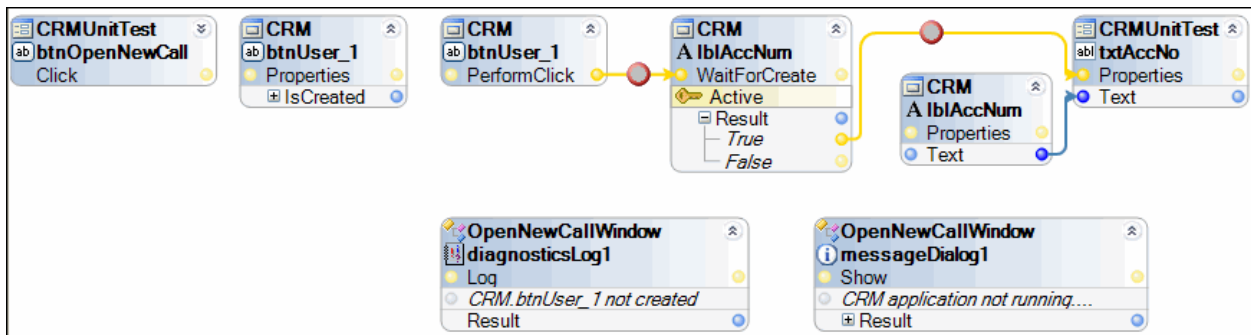
1. Delete the link between the CRMUnitTest.Click event and the CRM.btnUser\_1.PerformClick method.



2. Add the following components and controls to the OpenNewCall automation:

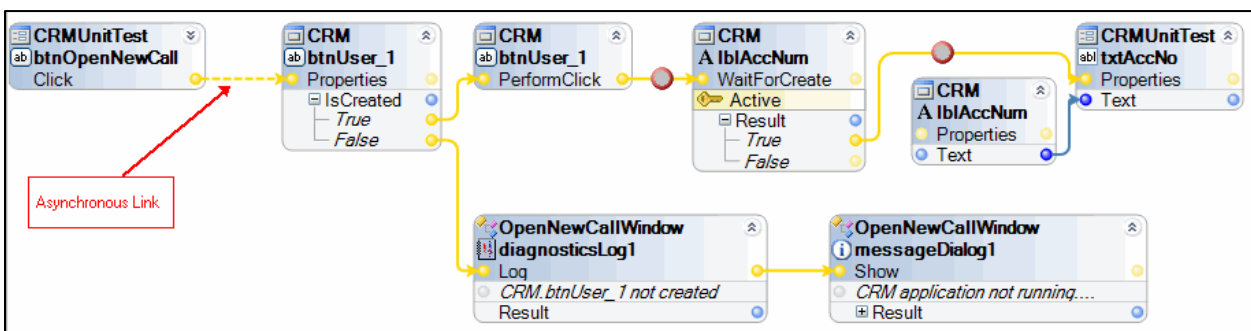
- CRM.btnUser\_1.IsCreated property
- DiagnosticsLog component, Log method. Set properties as follows:
  - Message: “CRM.btnUser\_1 not created”
  - Category: CRM Application Matching
  - Level: Info
- MessageDialog component, Show(1 parameter - message) method. Set message to: “CRM application not running. Click the Start CRM button.”

3. Your automation should look like the following:



4. Complete the execution path. Start by connecting the CRMUnitTest.btnOpenNewCall.Click event to the CRM.btnUser\_1.IsCreated property. Right-click on this link and select Asynchronous (so that the main thread is released after the button click, otherwise the Windows form will appear non-responsive until the automation completes).

5. Connect the execution path for the remaining automation controls/components as follows:

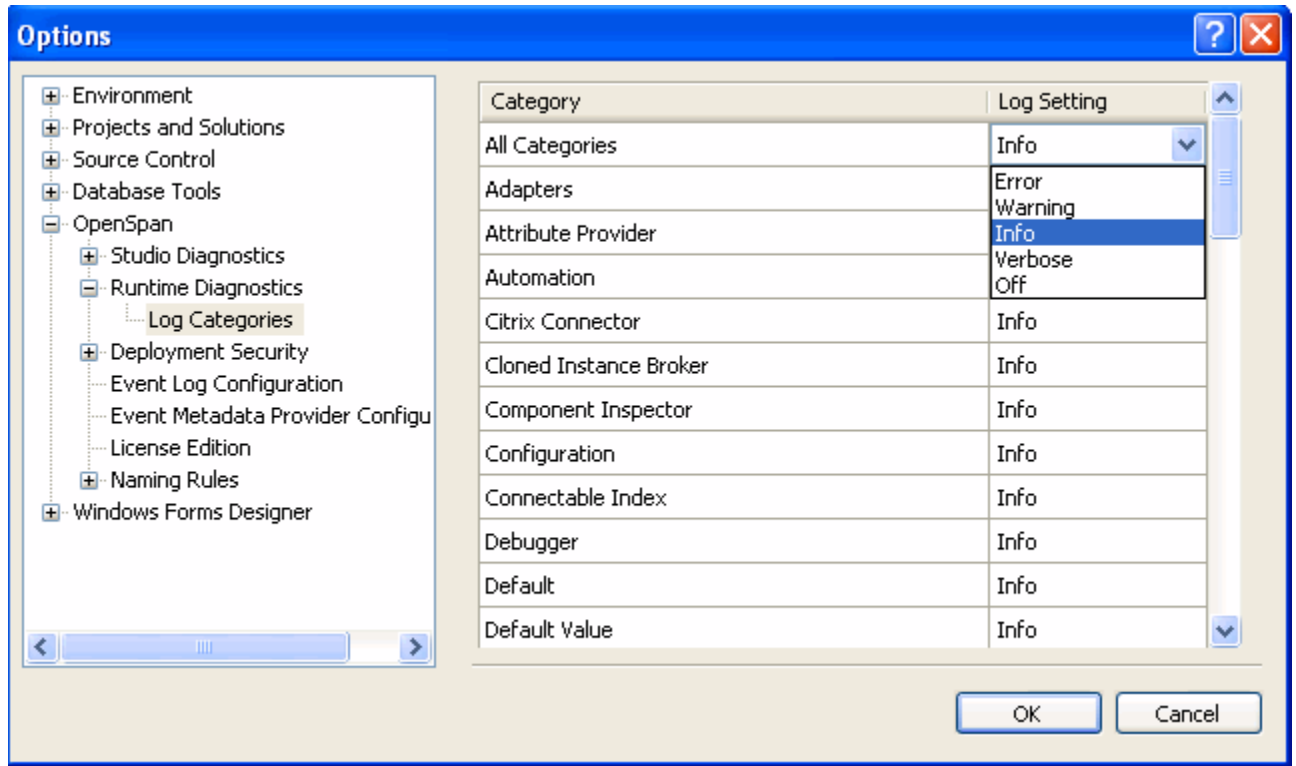


6. Save the solution.

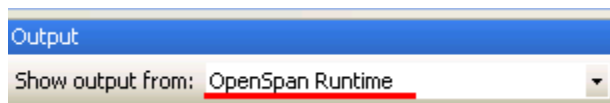
7. Select **Tools | Options | OpenSpan | Runtime Diagnostics | Log Categories**.

8. Change the All Categories log setting to **Info**, if it is not already set as such.

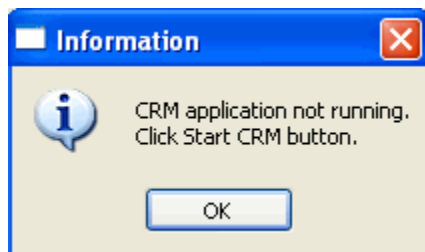
**Note:** The **DiagnosticsLog** component is tied to the **Automation** Category.



9. Select **OK** to save your selections.
10. Select **Start Debugging** to run the project.
11. In the Output window, select OpenSpan Runtime in the **Show Output From:** drop-down list:



11. Do not click the Start CRM button. Instead, click the New Call button on the CRM Unit Tester window.
12. A message displays informing you that the CRM application is not running:



13. Click OK to dismiss the error message.

14. Review the Output window (make sure that OpenSpan Runtime is selected as the output type). Note that the DiagnosticsLog message appears:

Info		hh:mm:ss.nnn PM		12		CRM Application Matching				CRM.btnUser 1 not created
------	--	-----------------	--	----	--	--------------------------	--	--	--	---------------------------

15. Click the Start CRM button. The CRM application starts and the login completes.
16. Click the New Call button on the CRM Unit Tester window. The New Call window opens in the CRM application and the account number displays in the Account textbox on the CRM Unit Tester window.
17. Stop the project and save the solution

**This page intentionally left blank.**

## CHAPTER 3. ERROR HANDLING AND SUPPRESSION

Even after you have completed thorough testing and debugging as part of creating OpenSpan projects, it is still possible for projects to encounter runtime errors. OpenSpan Studio provides functions for handling runtime errors and exceptions. These functions range from using special automation blocks (Try-Catch) to setting error suppression properties.

By the end of the chapter you will be able to:

- Use Try-Catch exceptions handling in automations
- Set error suppression for adapters and automations

### Try - Catch Exception Handling

Automations contain Try and Catch error handling functions. To use Try-Catch blocks, wrap any automation execution logic within the Try-Catch blocks. The Catch blocks handle any exceptions that may occur when running the automation logic. The Catch block provides output events and data for processing the error/exception:

- **Catch output event** – This event is triggered after the Catch block is called. Use this event to execute an alternate execution path based on the error/exception.
- **Exception output event** – This event is triggered if an exception is raised.
- **Exception data** – Outputs the exception type.
- **Message data** – Outputs the exception message.

### Adding Try – Catch Exception Handling

To add the Try and Catch blocks you have two options:

1. Manually add a Try block and a Catch block, and then connect the blocks to the appropriate event links.

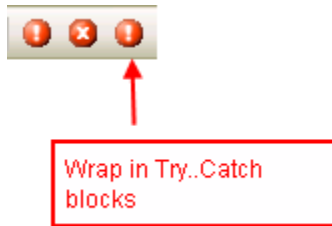


- Insert Try Block



- Insert Catch Block

2. Highlight one or more existing connection blocks and select the Wrap in a Try – Catch block button



## Exception Types

The type of exception that you want to handle with the Catch block can be selected by browsing the Exception event.



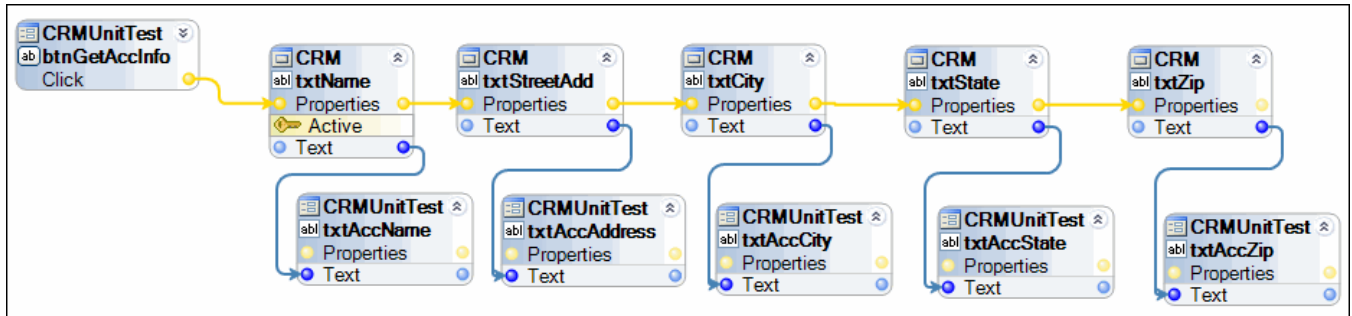
If you browse the Exception event, the Pick Type dialog displays which enables you to choose exception types from the following namespaces:

- Microsoft
- MS
- OpenSpan
- System

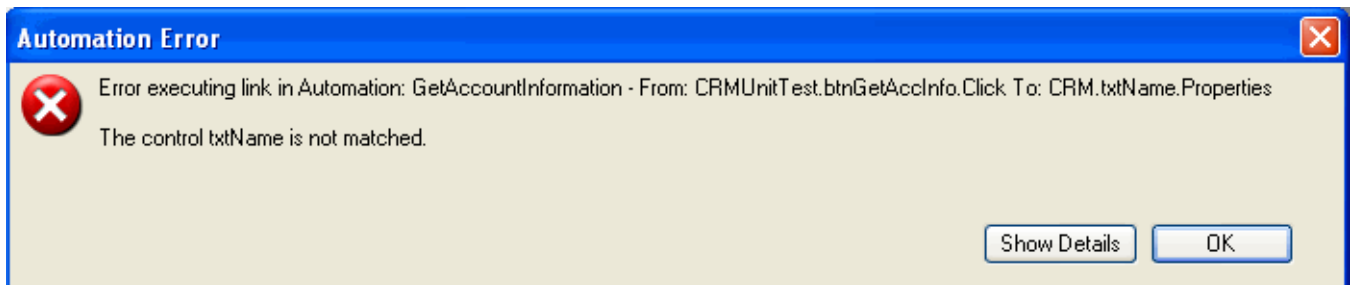
The default "Exception" corresponds to the System.Exception type.

### Exercise: Adding Try..Catch Exception Handling

Use the steps in this exercise to add Try-Catch blocks to an automation for handling an exception that occurs when the automation attempts to use a control that is not created (matched). In the sample solution for the exercise, the GetAccountInformation automation retrieves Name and Address information from the CRM application for the current New Call window and copies the information to the CRM Unit Tester window. An example of the automation follows:



If the end-user closes the New Call window or CRM application before attempting to retrieve the account information, an error occurs:



There are two ways to handle this error:

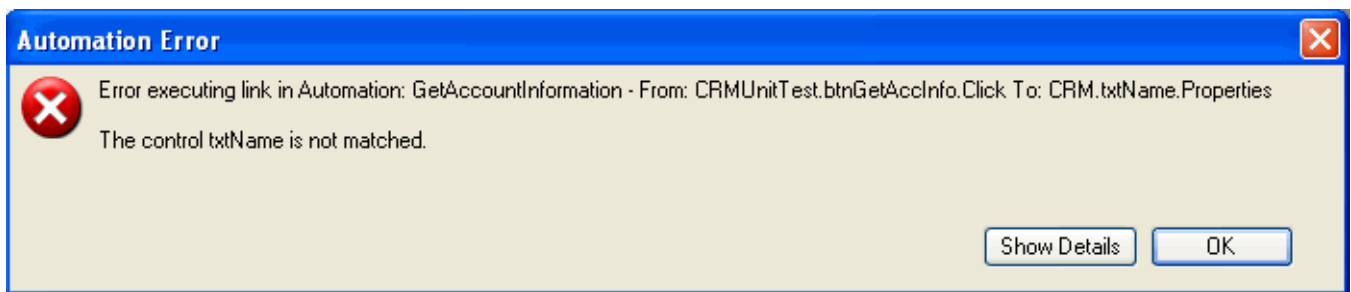
1. Use the **CRM.txtName.IsCreated** property to determine whether the control is matched and use a **DiagnosticsLog** in the case where the control is not created. This method was used in the Using the DiagnosticsLog Component exercise.
2. Use **Try-Catch** blocks to handle an exception caused by attempting to use a control that is not created (matched). This exercise uses the Try-Catch blocks.

This exercise requires the following setup:

- CRM.exe must be installed on your computer to the following location: C:\Program Files\OpenSpan\CRM Setup (the installation file CRM.msi is included in the Extras folder for OpenSpan Studio).
- Debugging\_DiagTr3 .zip file unzipped to Projects\Debugging\_DiagTr3 folder.

## Testing the Project

1. In OpenSpan Studio, open the Debugging\_DiagTr3 solution from the Debugging\_DiagTr3 folder. Run the project to see how an error can be generated by an end-user. The project begins by opening the CRM Unit Tester window.
2. Click the Start CRM button on the CRM Unit Tester window to start the CRM application. The CRM application starts and the Login is completed.
3. Click the New Call button on the CRM Unit Tester window. A New Call window opens in the CRM application and the account number displays on the CRM Unit Tester window.
4. **Close the New Call window in the CRM application.**
5. Click the Get Acc Info button on the CRM Unit Tester window. After a 30 second timeout period in which OpenSpan waits for a control to be matched, an error displays informing you that a control is not matched:

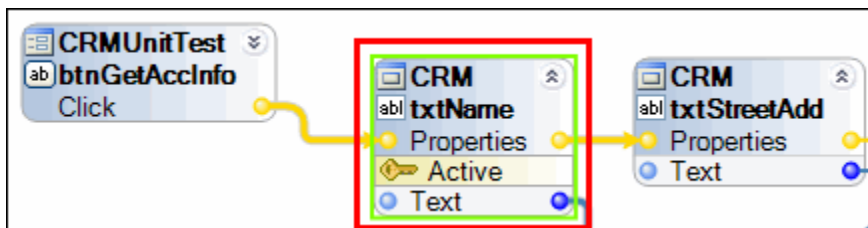


6. Click OK in response to the Automation Error and Stop the project.

## Catching Project Error

To catch an error caused by using a control that is not matched, surround the CRM.txtName.Text property in Try—Catch blocks.

1. Begin by highlighting the CRM.txtName.Text property on the GetAccountInformation automation.

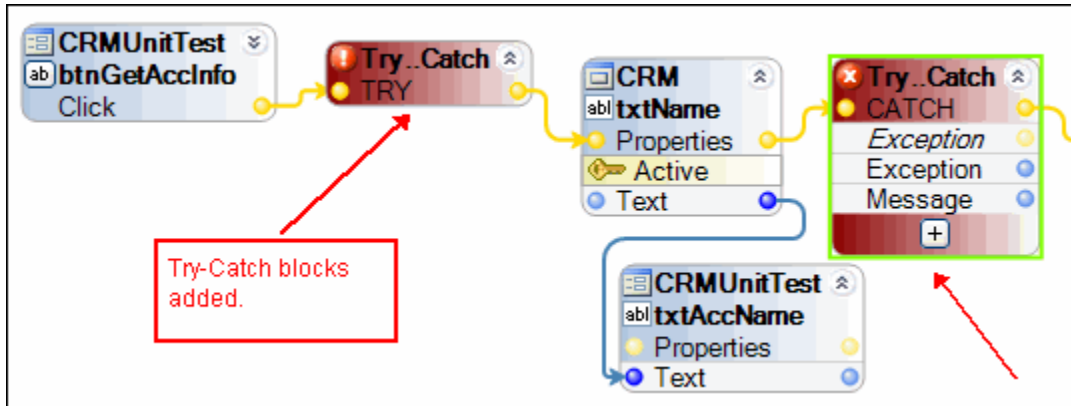




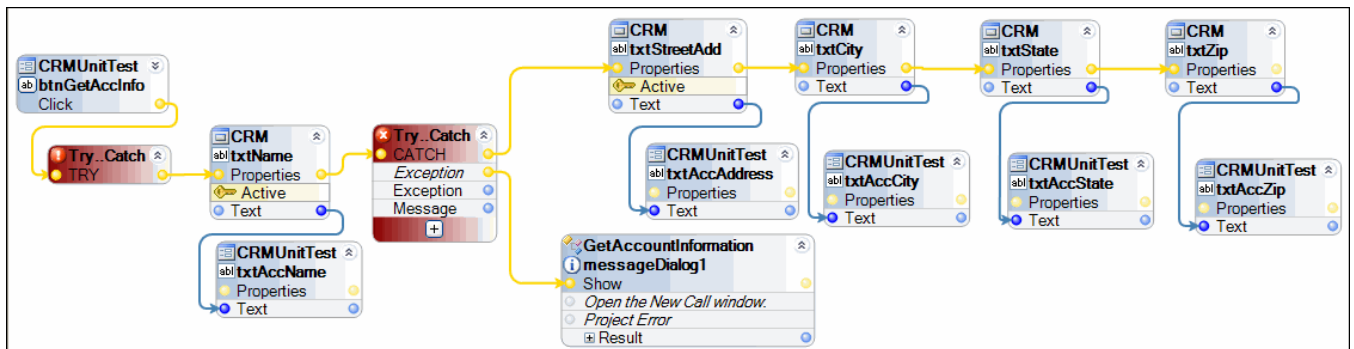
- Click the Wrap in a Try – Catch block button:



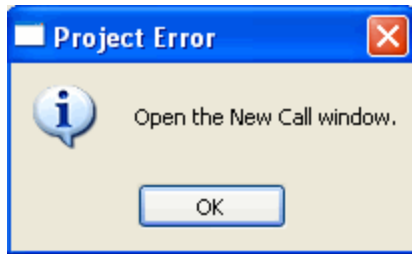
A Try connection block is inserted on the event link between the CRMUnitTest.btnGetAccInfo.Click event and the CRM.txtName.Text property. A Catch connection block is inserted on the event link after the CRM.txtName.Text property.



- Add a MessageDialog component to the GetAccountInformation automation:
- Set the message to: Open New Call window.
- Set the caption to: Project Error
- Connect the Exception output event port from the Catch block to the input event port of the MessageDialog.
- An example of the completed automation follows:



- Save the solution.
- Repeat the process listed in the [Testing the Project](#). This time when you close the New Call window and then click the Get Acc Info button, OpenSpan attempts to match the CRM.txtName control. After the 30 second default timeout, the Project Error message displays.



10. Click OK to clear the message.

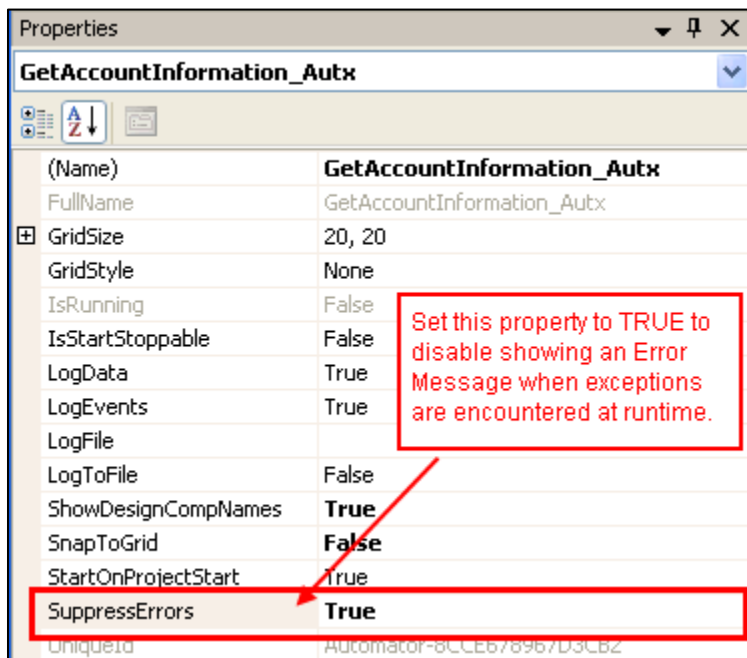
11. Stop the project.

## Error Suppression

OpenSpan Studio can suppress automation and adapter errors. By suppressing automation and/or adapter errors, an end-user will not see an error message dialog in the event that an exception occurs. However, you can track the exceptions by setting the diagnostics reporting to log errors to the RuntimeLog.txt file. See the [OpenSpan Studio Diagnostic Publishers](#) chapter for details on enabling Application Diagnostics and the File publisher.

## Automation Errors

Suppress automation errors by selecting the automation in the **Solution Explorer**, and then change the **SuppressErrors** value to **True** in the Properties window.



**Note:** You can use Configuration project items to set this property to True for a “Runtime” configuration and False for a “DesignTime” configuration. This way you can see errors when debugging the project in OpenSpan Studio and then suppress the error notification(s) for the deployed Runtime version of the project. See the *OpenSpan Runtime and Project Deployment Training Module* for details on using Configuration project items.

## Adapter Errors

Adapter errors are suppressed at the application level. That is, a single Runtime configuration file setting suppresses errors for all adapters. To suppress adapter errors, edit the RuntimeConfig.xml file and change the following key value to **true**:

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

Save the updated RuntimeConfig.xml file and restart the Runtime application to apply the setting change.

**This page intentionally left blank.**