­­



**Fest 2018 Workshop**

**Building Applications for Relativity and RelativityOne**

September 14, 2018 - Version 1.0

Contents

[1 Course Overview 3](#_Toc524527449)

[1.1 What You’ll Learn 3](#_Toc524527450)

[1.2 Required Development Software 3](#_Toc524527451)

[2 Getting Started 4](#_Toc524527452)

[2.1 Visual Studio Project 4](#_Toc524527453)

[2.2 DevVM Setup 6](#_Toc524527454)

[3 Starting Development 11](#_Toc524527455)

[3.1 Application Set Up 11](#_Toc524527456)

[3.2 New Event Handler 14](#_Toc524527457)

[3.3 Publish 17](#_Toc524527458)

[4 Logging 21](#_Toc524527459)

[4.1 Adding Logging 21](#_Toc524527460)

[4.2 Viewing Logs 22](#_Toc524527461)

[5 Customize New Button Override & Override Edit Link URL Object Rules 25](#_Toc524527462)

[5.1 Custom page 25](#_Toc524527463)

[5.2 Visual Studio Post-Build Event & Publish to Relativity Console 35](#_Toc524527464)

[5.3 Object Rules 42](#_Toc524527465)

[5.4 ObjectManager API 49](#_Toc524527466)

[5.5 Gravity API 50](#_Toc524527467)

[6 Unit Tests 57](#_Toc524527468)

1. Course Overview

As a developer, you want a solid process for building and testing your apps, so you can focus on delivering value quickly. The Application Deployment System (ADS) is the standard way to build applications on the Relativity platform. Relativity Test Helpers is our new, open source testing framework.  
  
In this advanced session, we'll review both of these standard approaches, covering common patterns for setting up effective ADS development solutions and integration tests. Topics will include: setting up your development environment using DevVM's; Visual Studio developer tools including templates, NuGet packages, and the "publish to Relativity" extension; and GitHub open source solutions such as Gravity, Test Helpers, and more. We'll also highlight ways to ensure your applications make a smooth transition from the on-premise world to RelativityOne. You will create tests and learn how to run those test from scripts as well as gain practical experience writing tests and potentially automating tests for your solutions, ensuring your customization will be ready run with the latest release of Relativity and RelativityOne.

* 1. What You’ll Learn
* Faster Development
* Publish to Relativity
* Logging
* How to Override New Button Link for a RDO
* How to Override Edit Link for a RDO
* Publish to Relativity Console
* Visual Studio Post-Publish Event
* Object Manager API
* Gravity API
* Unit Tests
* Versioning

1.2 Required Development Software

This workshop and associated development resources require a development environment that includes the following items:

* Relativity 9.5
* Visual Studio 2017
* Relativity Visual Studio Templates
* .NET Framework 4.6.2
* SQL Server 2012

We have provided the required software for you for this workshop.

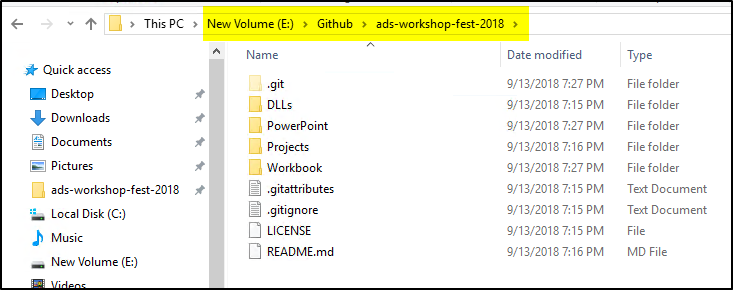
1. Getting Started

2.1 Workshop Projects

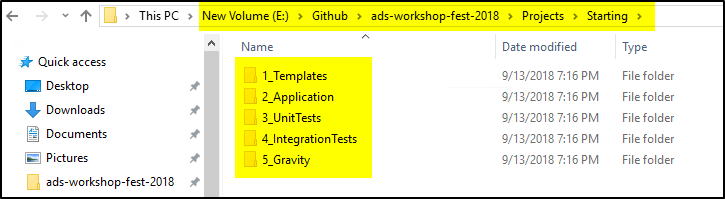
1. Make sure you have a folder shortcut on Desktop with name **ads-workshop-fest-2018**.



1. Double click on the **ads-workshop-fest-2018** folder to open it. Make sure you see the files as shown in the below screenshot.



1. Go to the **Projects/Starting** folder to make sure you see the folders as shown in the below screenshot. These folders contain Visual Studio projects for different sections in the workbook.

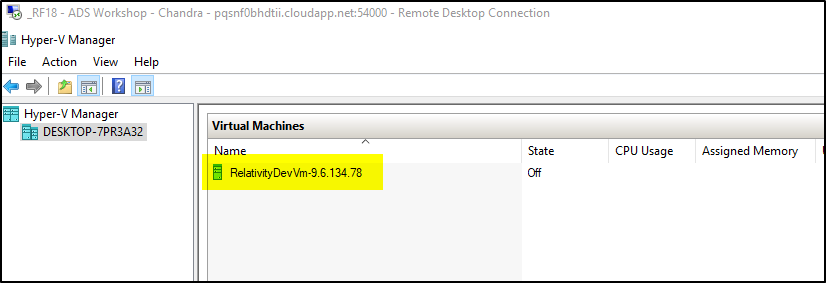


2.2 DevVM Setup

1. Open the **Hyper-V Manager** application by clicking the blue icon in the taskbar.



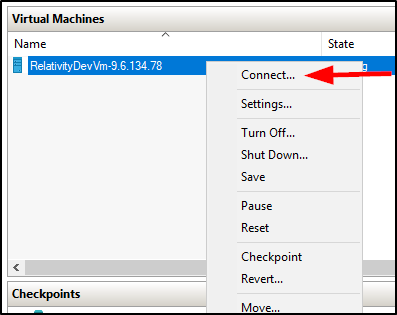
1. You will see a **Relativity 9.6.134.78 DevVM** listed in the Hyper-V Manager **Virtual Machines** section.



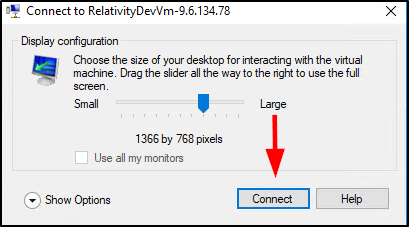
1. Right click on the **DevVM** and select the **Start** option.



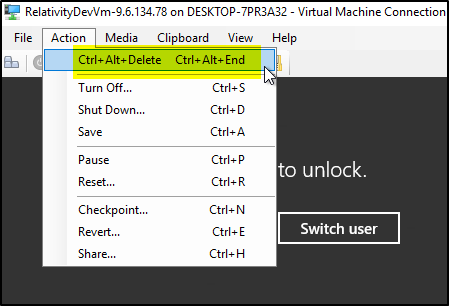
1. Right click on the **DevVM** and select the **Connect** option.

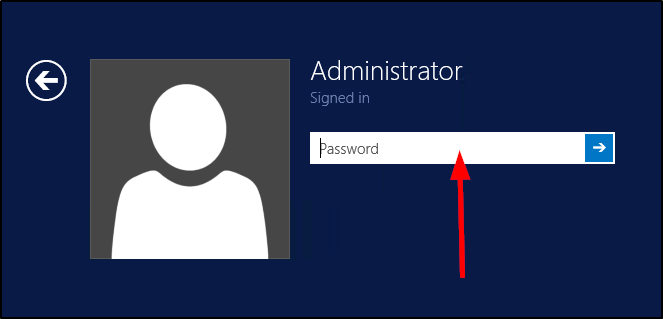


1. If you get a Display configuration pop-up, click the **Connect** button.



1. Select the **Action** Menu item and then select **Ctrl+Alt+Delete** option.





1. Use the below DevVM credentials highlighted in yellow to login to the VM.

**DevVM:**

Windows Admin login: Administrator

Windows Admin password: Test1234!

**Relativity:**

Admin login: relativity.admin@relativity.com

Admin password: Test1234!

**SQL:**

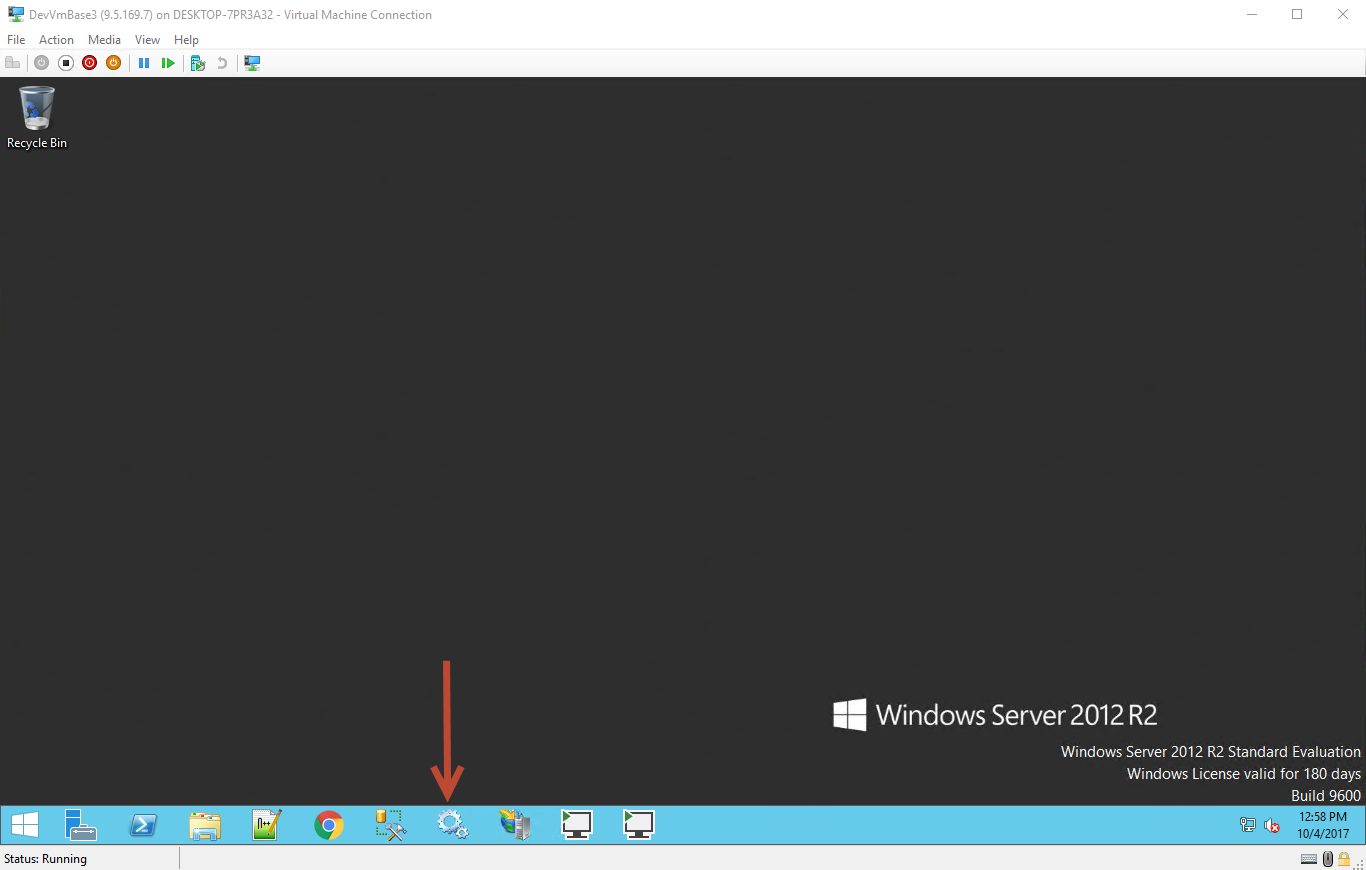
login: eddsdbo

password: Test1234!

Admin login: sa

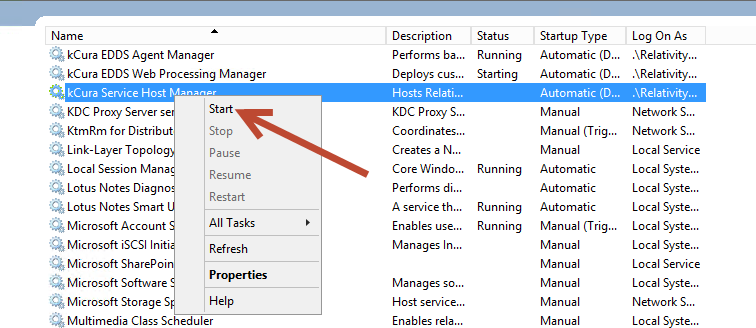
Admin password: Test1234!

1. Once you log in to the DevVM, you should see the Desktop with few applications pinned to the taskbar, which you will be using throughout the workshop.
2. Click on the **Services** program icon  in the taskbar as shown in the below screenshot.



1. Make sure all the **services** listed below are **running**. If they are not already running, right click on the service and select the **Start** option,

* Service Bus Gateway
* Service Bus Message Broker
* Service Bus Resource Provider
* Service Bus VSS
* kCura Service Host Manager
* kCura EDDS Web Processing Manager
* kCura EDDS Agent Manager

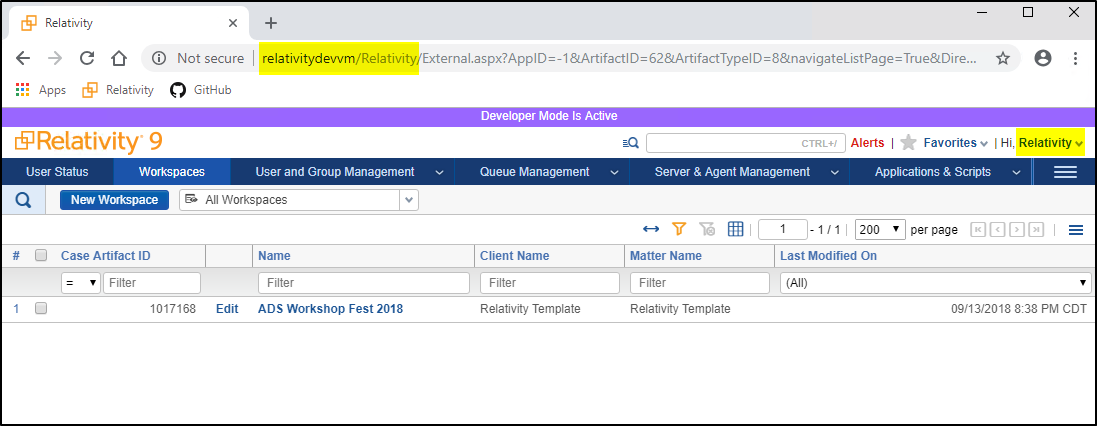


1. Open **Chrome** on your Workshop machine (Not the DevVM) and go to [**http://RelativityDevVm/Relativity**](http://RelativityDevVm/Relativity). Use the below Relativity admin credentials to login. Verify that you can login to Relativity.

**Relativity:**

Admin login: relativity.admin@relativity.com

Admin password: Test1234!



1. Starting Development

Relativity is an extensible platform that allows an administrator/developer to create customizations. Development on the Relativity Platform is made easier with the use of some tools such as Visual Studio Templates and the Publish to Relativity GUI. This section demonstrates utilizing the visual studio templates and Publish to Relativity tool to quickly iterate on application development.

3.1 Application Set Up

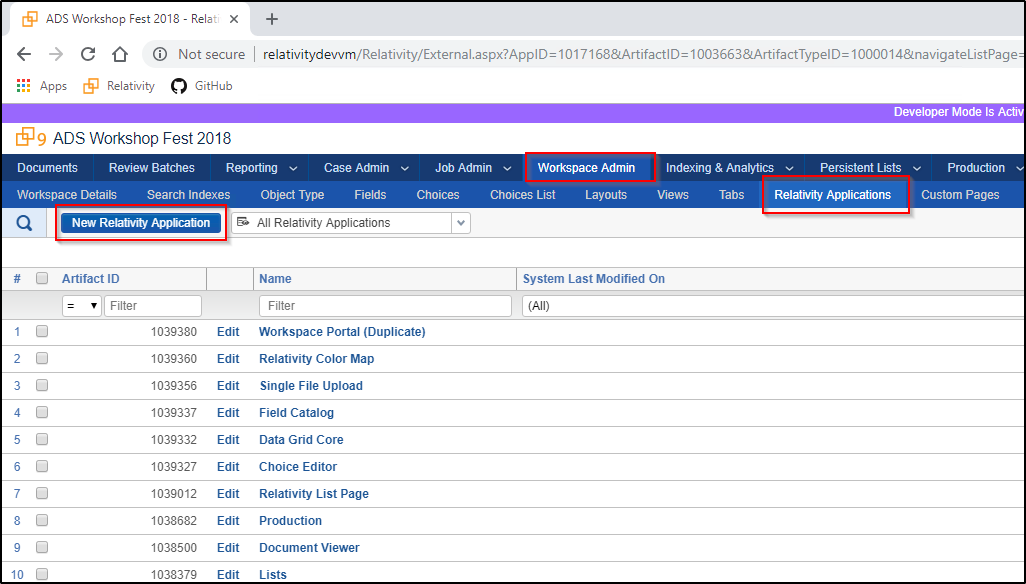
1. Open **Chrome** on your Workshop machine (Not the DevVM) and go to [**http://RelativityDevVm/Relativity**](http://RelativityDevVm/Relativity). Use the below Relativity admin credentials to login.

**Relativity:**

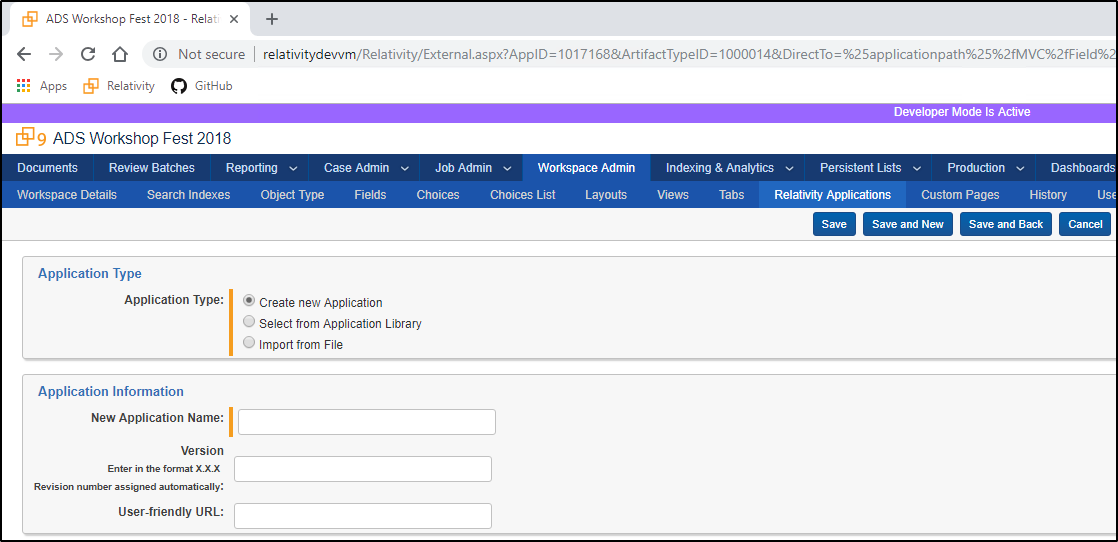
Admin login: relativity.admin@relativity.com

Admin password: Test1234!

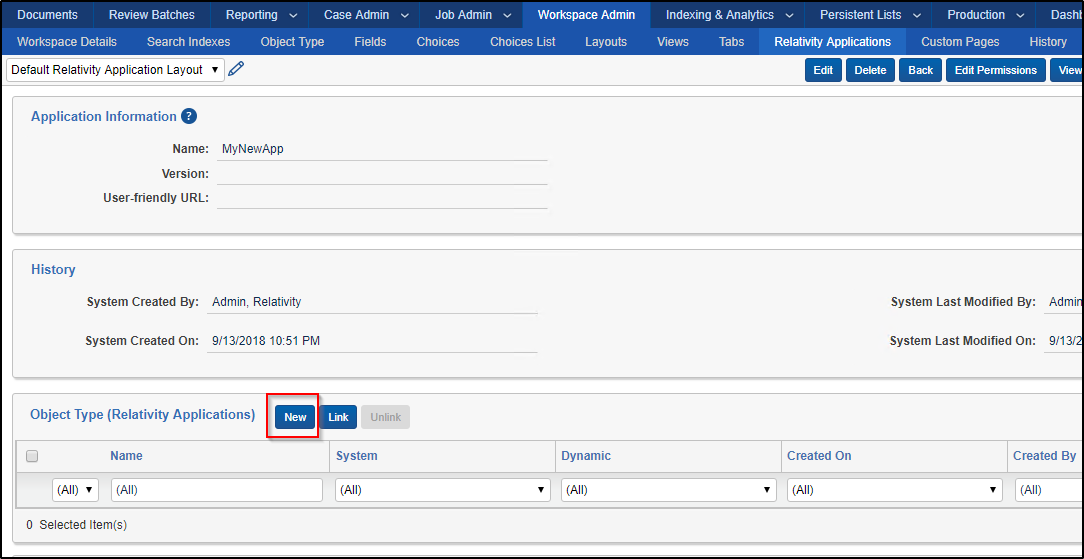
1. Enter [**ADS Workshop Fest 2018**](http://relativitydevvm/Relativity/RedirectHandler.aspx?defaultCasePage=1&AppID=1017168) and navigate to the **Relativity Applications** Tab.

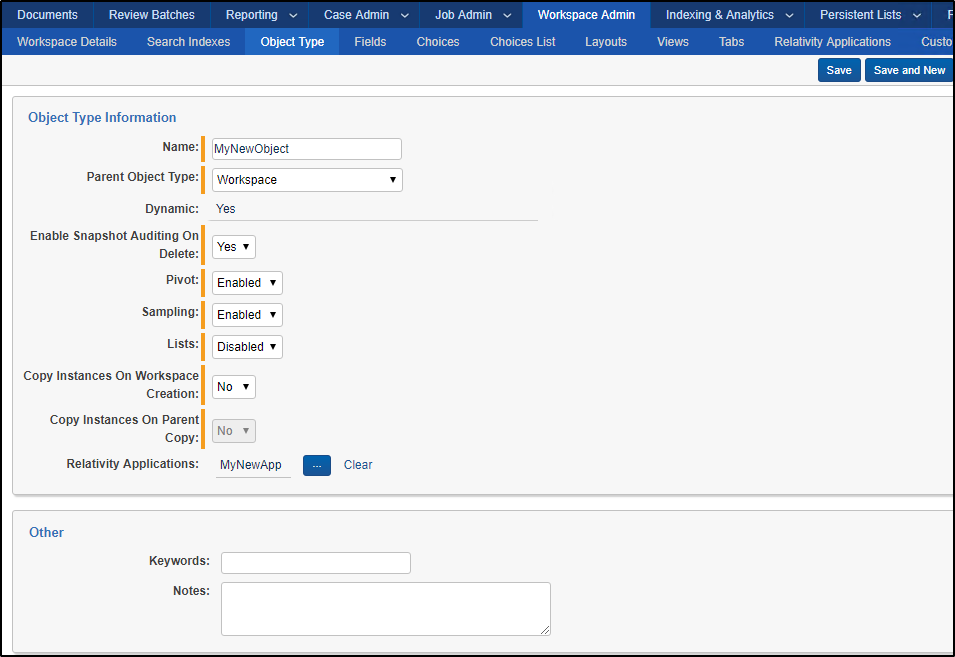


1. Create a new **Relativity Application** with the name of your choice.



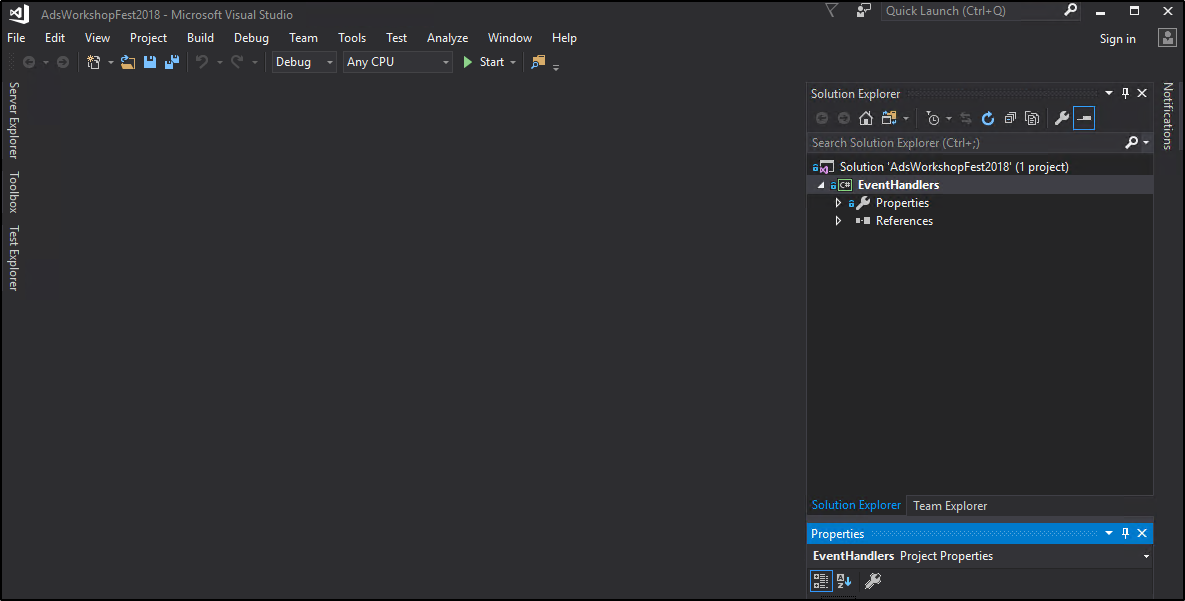
1. Create a new **Object Type** to the application from the view page. This auto-associates the objects initial views, fields, and layouts.





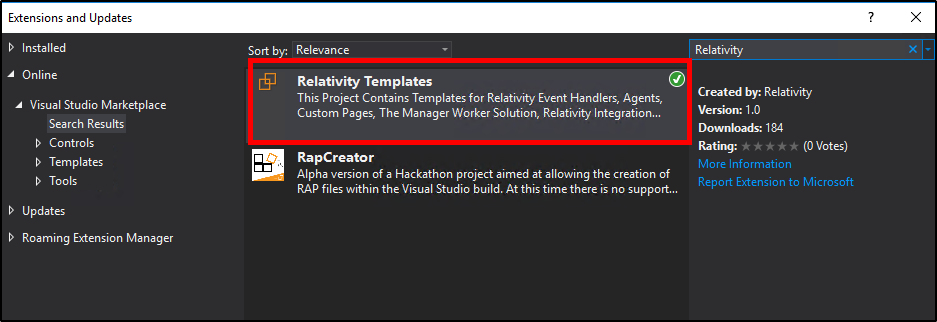
3.2 New Event Handler

1. Open the Visual Studio solution **AdsWorkshopFest2018.sln** in the **E:\Github\ads-workshop-fest-2018\Projects\Starting\1\_Templates\Project** folder.

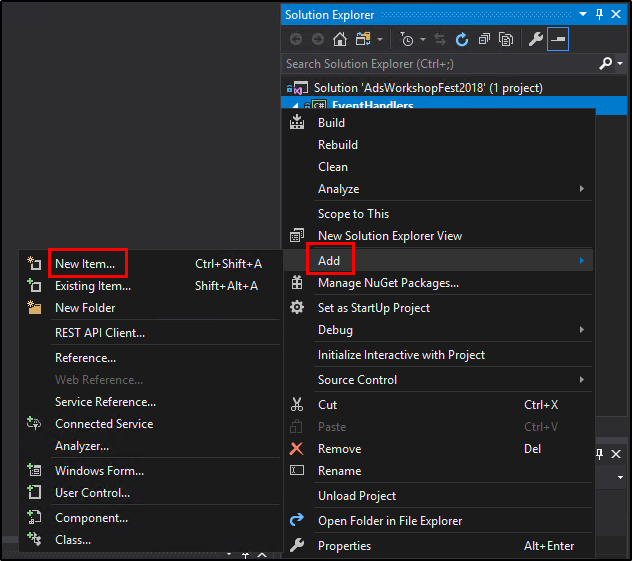


1. Verify the **Relativity Templates** are installed. You may need to **restart** Visual Studio.
2. **Tools** — **Extensions and Updates** — Search "**Relativity**"

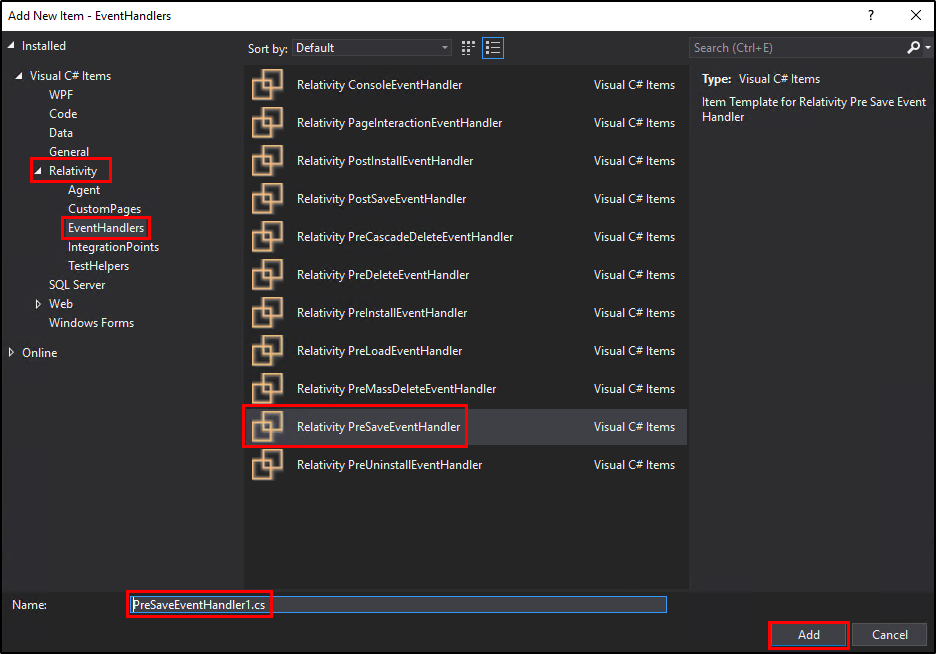




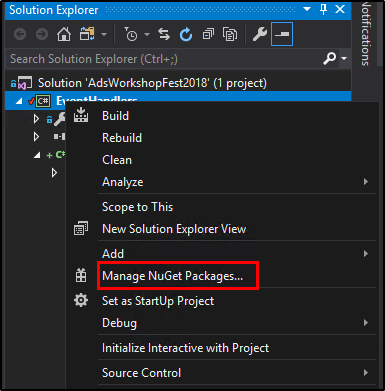
1. Right click the **EventHandlers** Project in the solution explorer. Select Add — **New Item**.



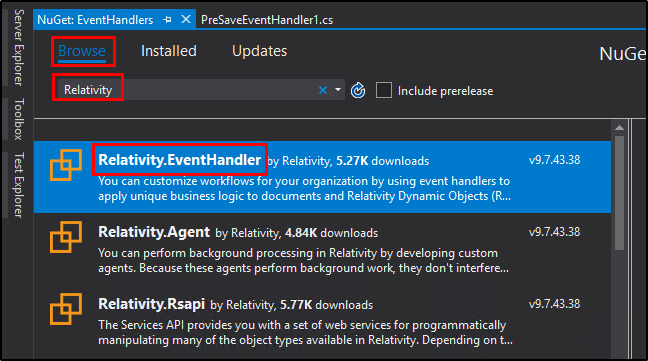
1. Under the Relativity Section, select the **pre-save event handler template.**



1. Click **Add**.
2. Install the 9.6.134.78 **Relativity.EventHandler** NuGet package.
   1. Right click on the Project.
   2. Select **Manage Nuget Packages** option.



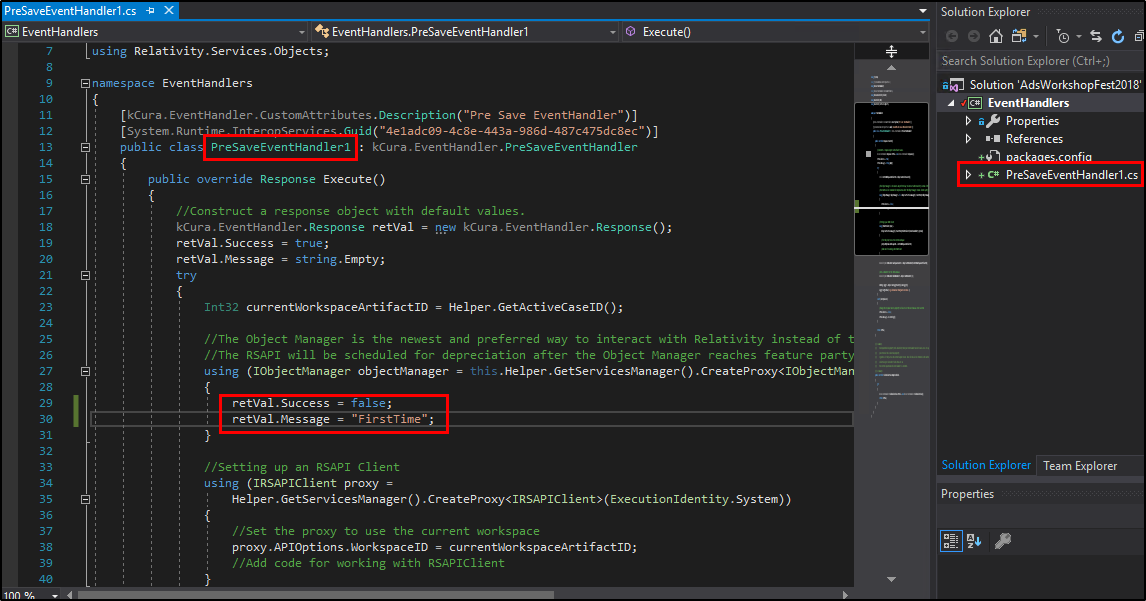
* 1. Switch to **Browse** tab.
  2. Search for “**Relativity**”.
  3. Select the **Relativity.EventHandler** NuGet package.



* 1. From the **Version** drop-down select the **9.6.134.78** option.
  2. Click **Install**.

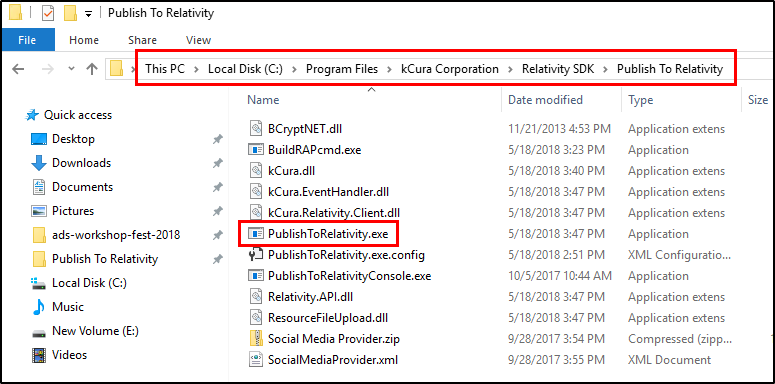


1. Update the event handler response.
   1. *Success* – false
   2. *Message* – Message of your choice

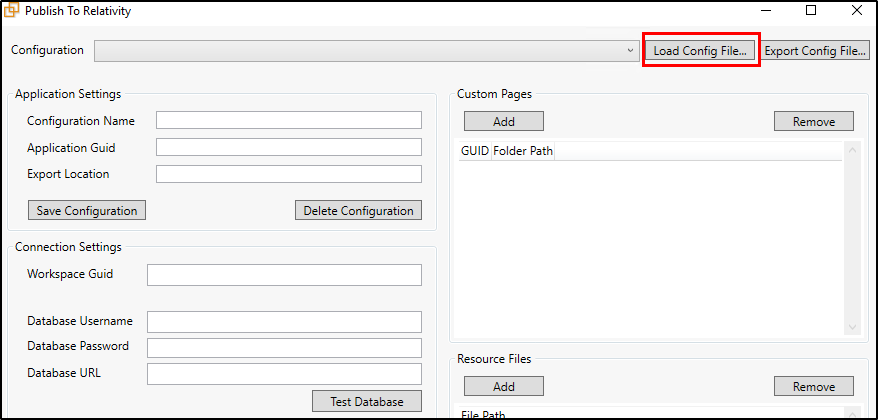


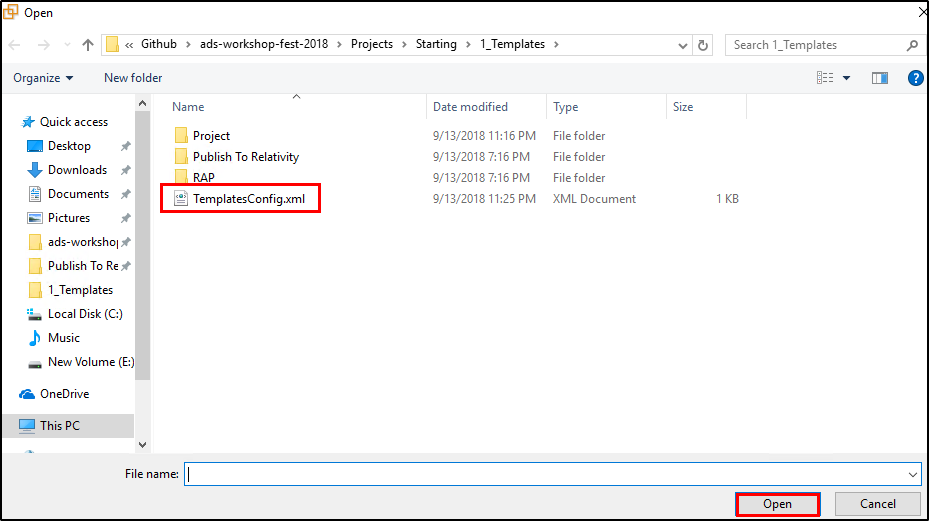
3.3 Publish

1. Build the solution.
2. **Build** — **Build Solution**
3. Run the Publish to Relativity executable.

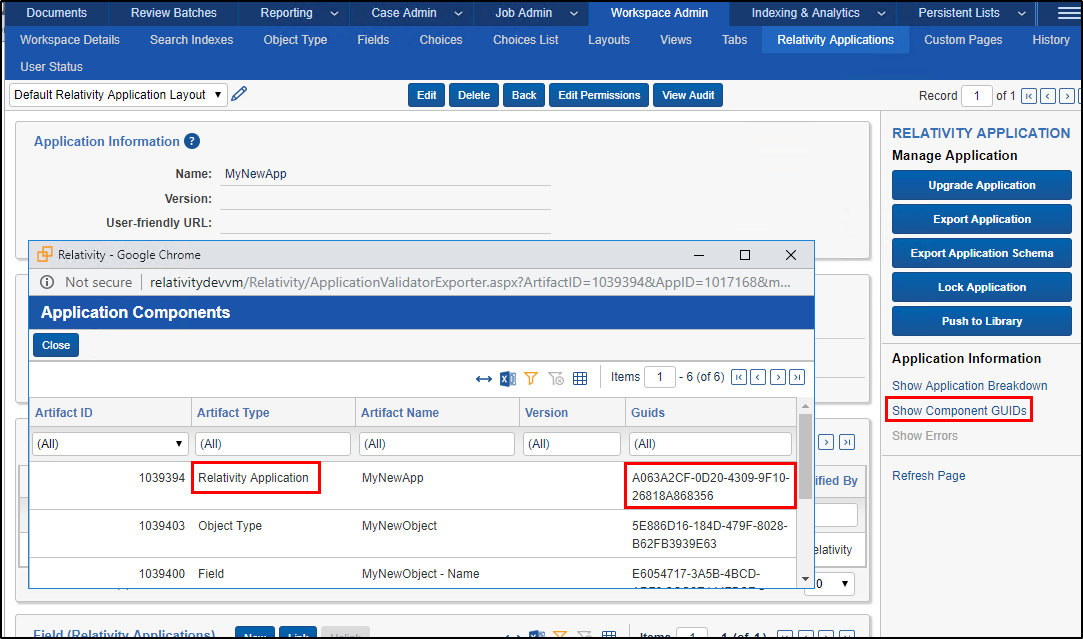


1. Click the **Load Config File** button and select the **TemplatesConfig.xml** file in the **E:\Github\ads-workshop-fest-2018\Projects\Starting\1\_Templates** folder.

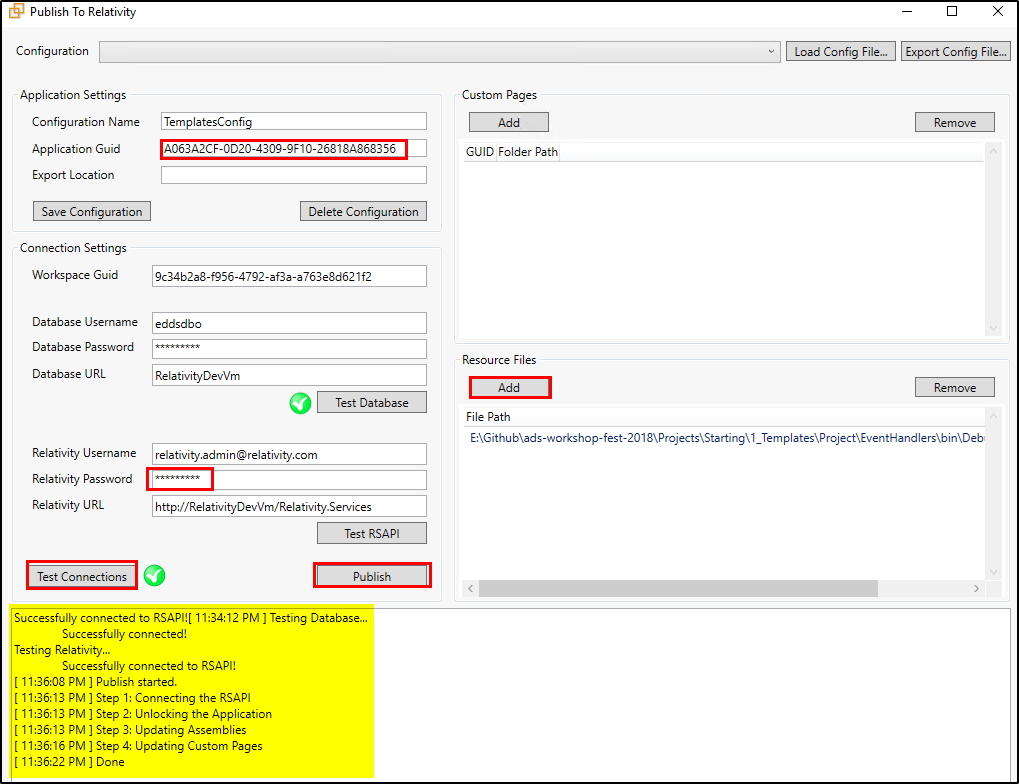




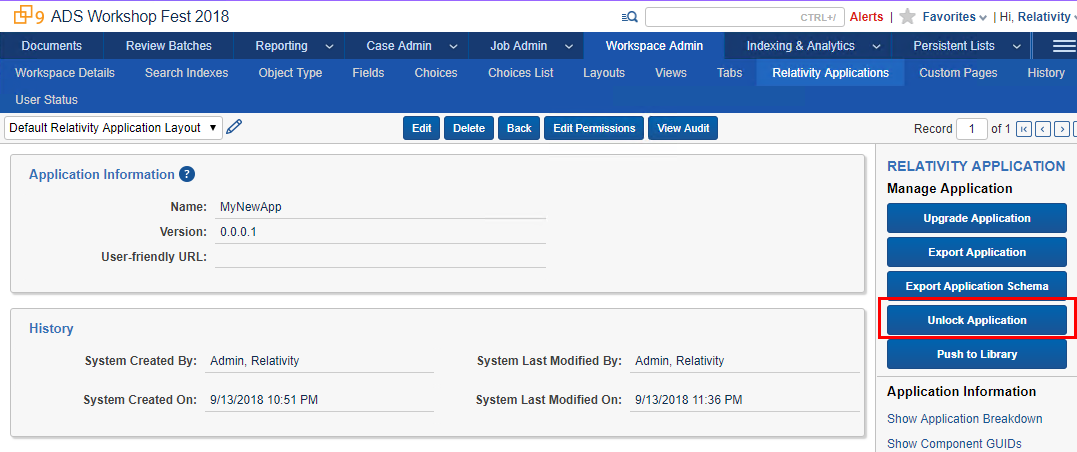
1. Enter the **Relativity Password** (Test1234!).
2. Click on **Test Connections** to validate the connections.
3. Replace the **Application GUID** with your applications GUID.
   1. To look up your applications GUID, navigate back to Relativity Application you created in the workspace. Then select **Show Component GUIDs**.
   2. **Developer Mode** must be active.



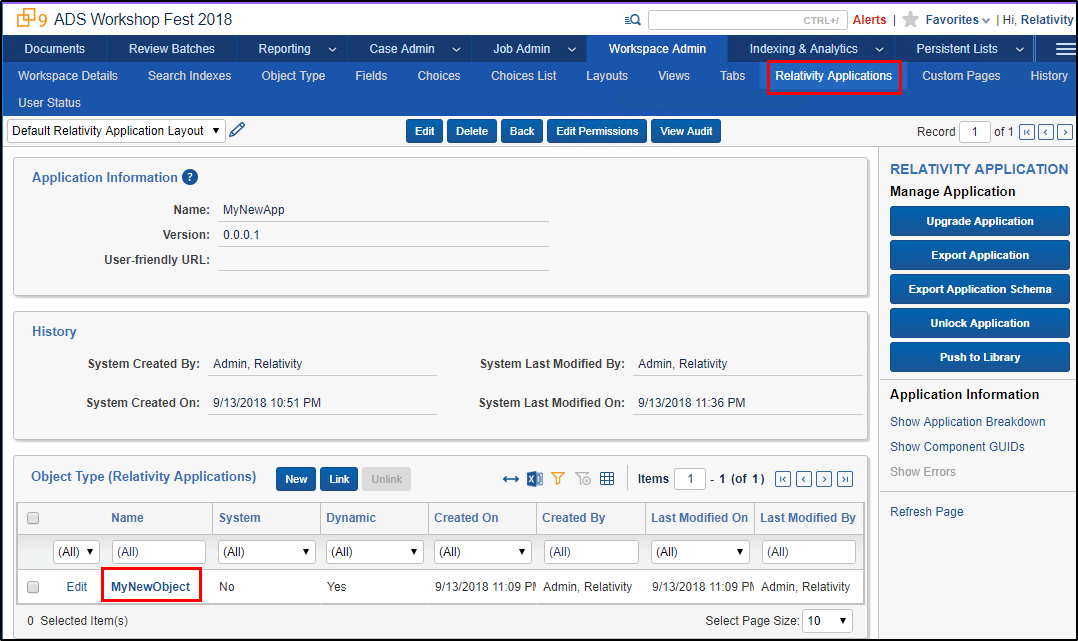
1. In the **Resource Files** Section click Add.
2. Navigate to **E:\Github\ads-workshop-fest-2018\Projects\Starting\1\_Templates\Project\EventHandlers\bin\Debug** and select **EventHandlers.dll** file.
3. Click **Publish**.



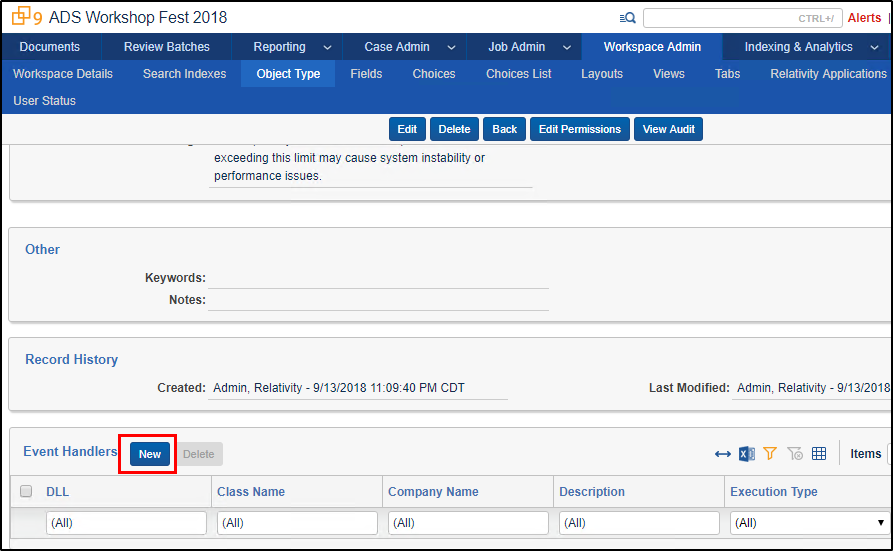
1. Open Relativity and navigate to your application created in section 3.1.
2. **Unlock** the application.



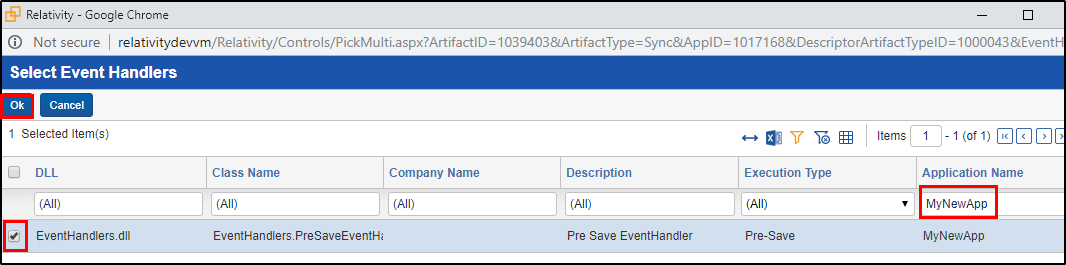
1. Navigate to the view page of the object type created in section 3.1.



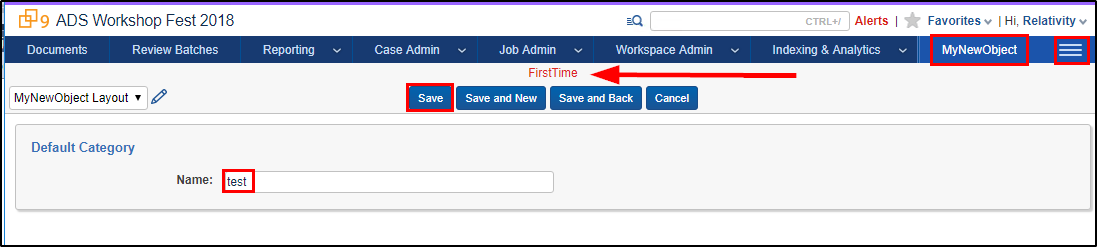
1. Click **New** on the Event handler associated item list.



1. Find the newly created Event Handler associated to your application.
   1. Filter by Application Name.
   2. Select it from the list and click **Ok**.



1. Test your event handler by creating a new instance of your object type.



1. Sample Application Overview

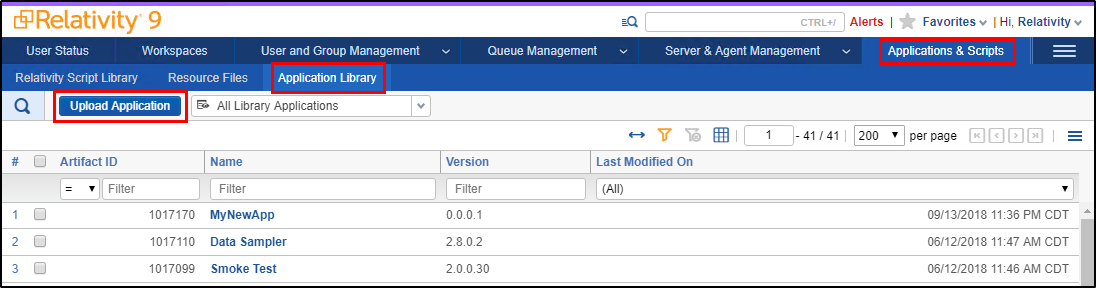
The functionality of the Sample Relativity application (ADS Workshop Fest 2018) is to calculate the selected Instance metrics and write them to the Instance Metrics Job.

The ADS Workshop Fest 2018 Relativity application consists of the following:

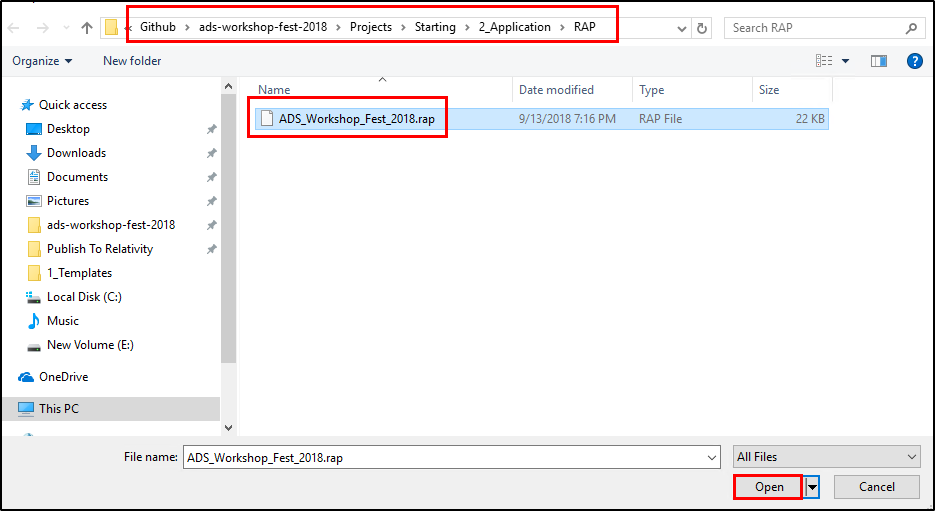
* **Custom Object**
  + Instance Metrics Job
    - Fields
      * Metrics
      * Status
      * Workspaces Count
      * Users Count
      * Groups Count
      * Errors
* **Event Handler**
  + PreSaveStatusUpdate
* **Agent**
  + Instance Metrics Calculator Agent

4.1 Install Sample Application

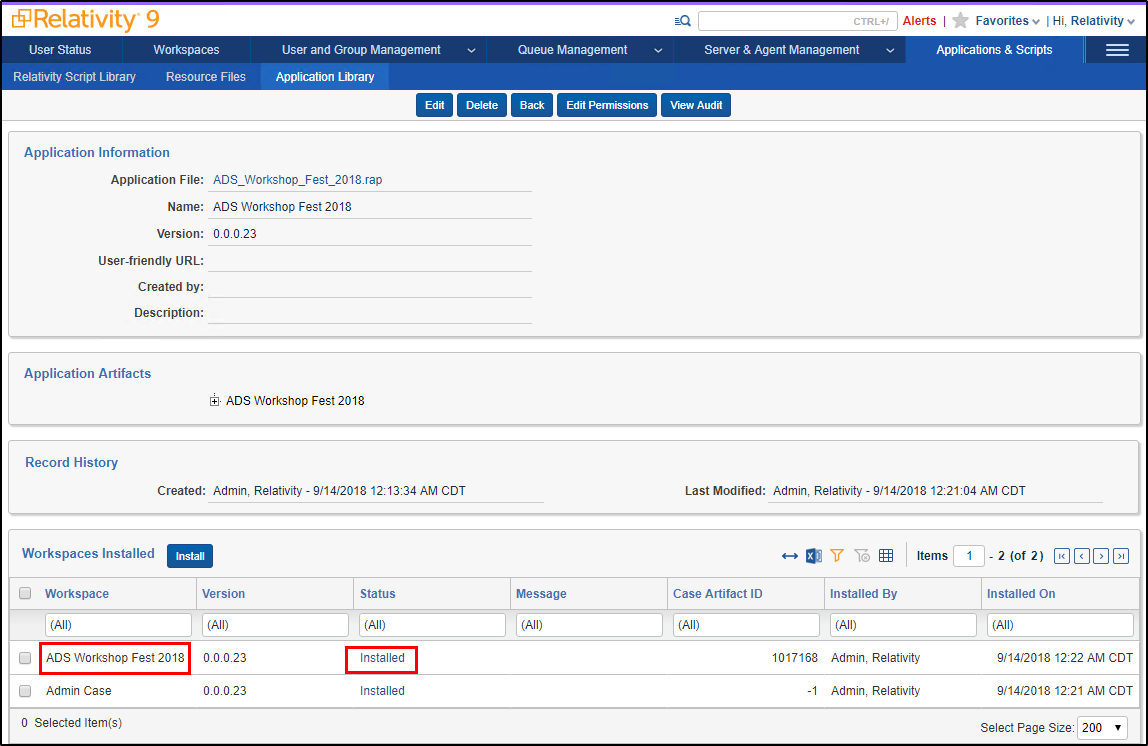
1. Go to the **Application Library** tab.
2. Click on the **Upload Application** button



1. Click on **Choose File** button.
2. Navigate to **E:\Github\ads-workshop-fest-2018\Projects\Starting\2\_Application\RAP** folder and select the **ADS\_Workshop\_Fest\_2018.rap** file.

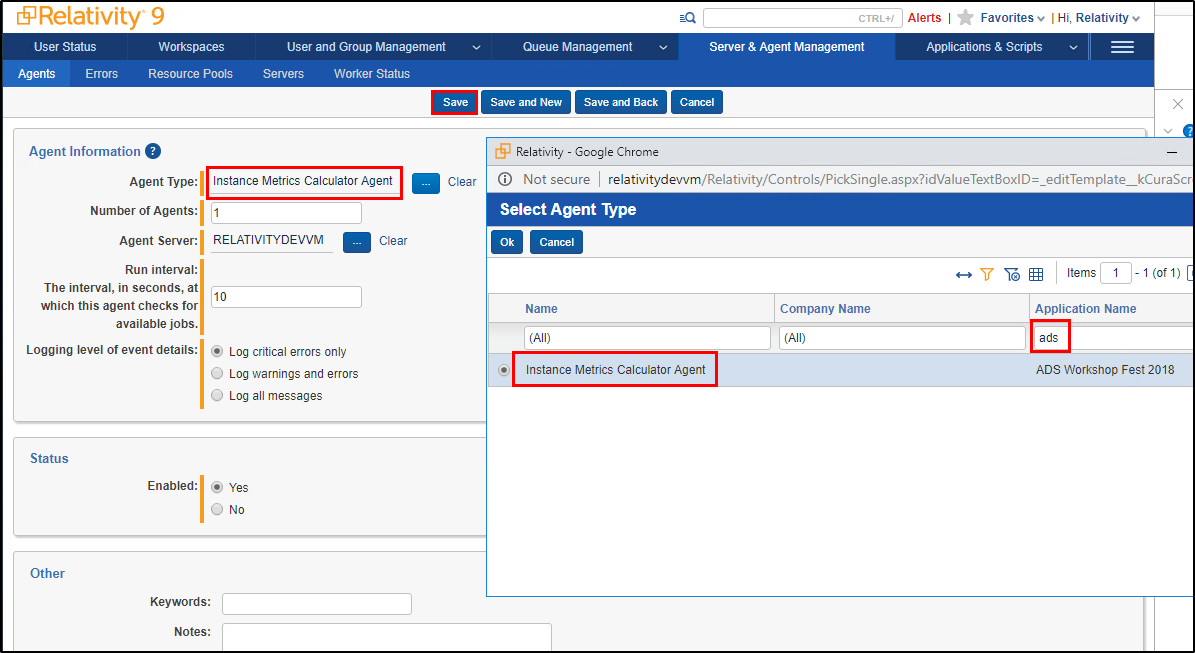


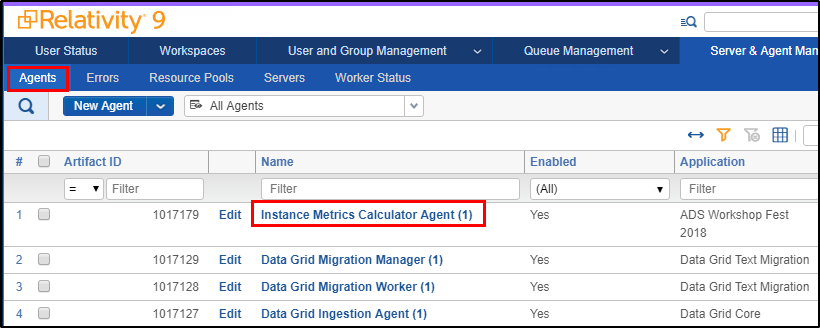
1. Click the **Save** button.
2. Under the **Workspaces Installed** section, click the **Install** button.
3. Select the **ADS Workshop Fest 2018** workspace and click the **Save** button.



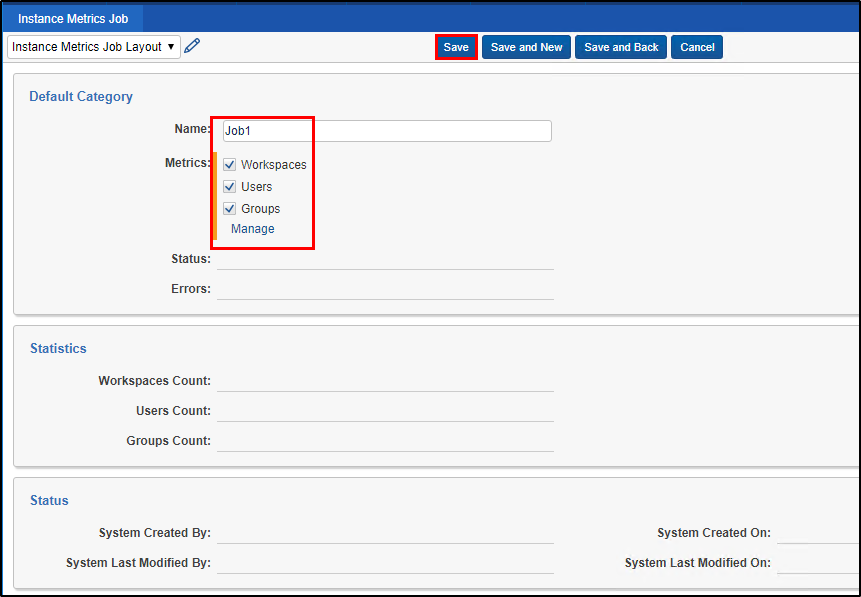
4.2 Run Sample Application

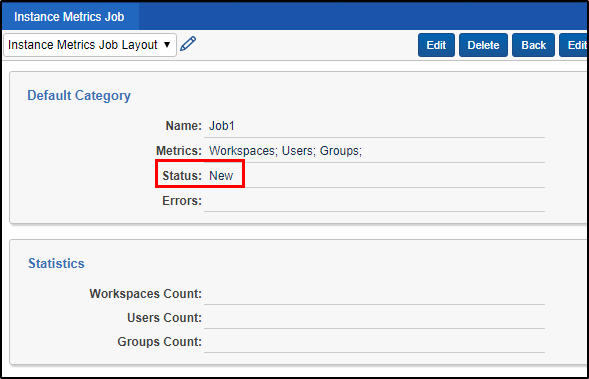
1. Once the application is installed, navigate to the **Agents** tab and create an **Instance Metrics Calculator Agent** agent.



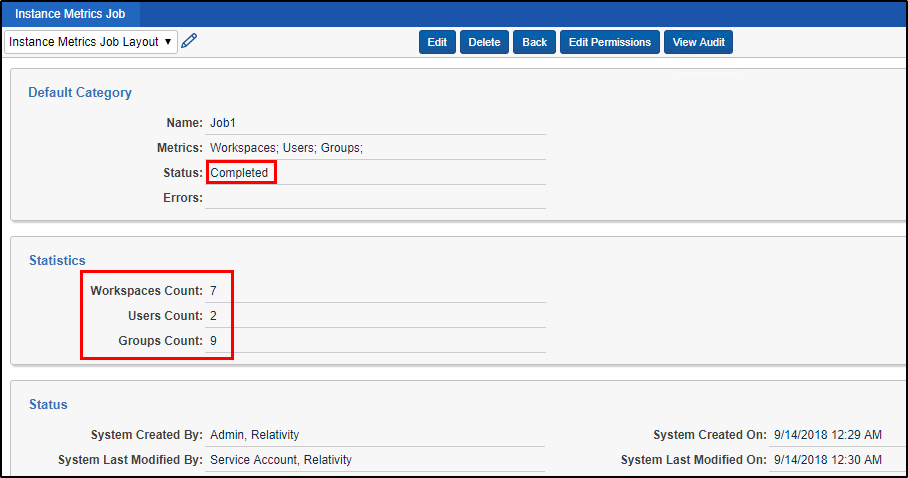


1. Now navigate the **Instance Metrics Job** tab in **ADS Workshop Fest 2018** workspace.
2. Create a new **Instance Metrics Job** and click on the **Save** button.





1. The **Instance Metrics Calculator Agent** which is running every 10 seconds will pick up the job and calculate the metrics and save it to the job.
2. Keeping refreshing the windows every few seconds until the job status is set to **Completed**.



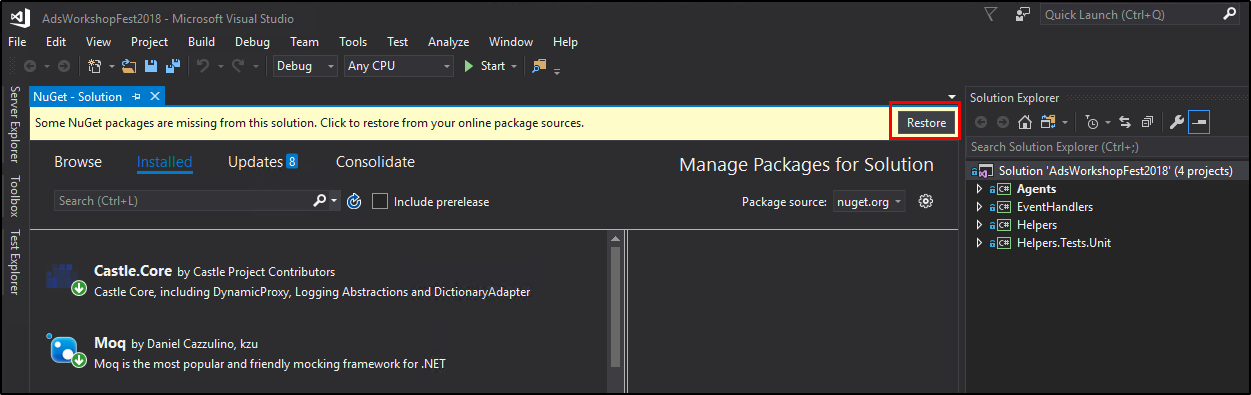
1. Unit Tests

5.1 Writing Unit Tests

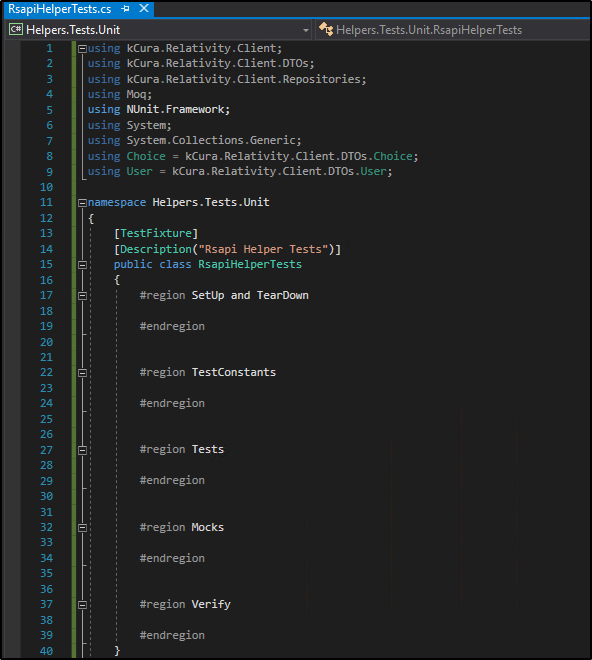
1. Navigate to the **E:\Github\ads-workshop-fest-2018\Projects\Starting\3\_UnitTests\Project** folder and open **AdsWorkshopFest2018.sln** file.
2. In this solution, you will find 4 projects.
   1. Agents
   2. EventHandlers
   3. Helpers
   4. Helpers.Tests.Unit
3. In this section we will be working in the **Helpers.Tests.Unit** project.
4. In **Helpers** project, there is an **RsapiHelper.cs** file which contains RSAPI calls used in the sample application.
5. Our goal for this section is to write couple unit tests for the **RetrieveJobsInWorkspaceWithStatus** in **RsapiHelper** class.
6. Right click on the solution and select **Manage NuGet Packages for Solution** option.



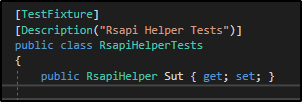
1. Next click the **Restore** button.



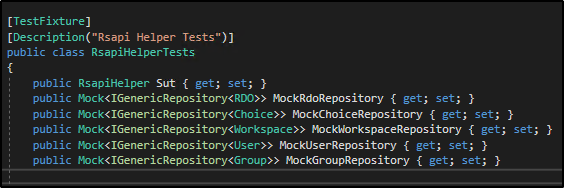
1. Now all the NuGet packages in the solution should be restored.
2. Right click on the **Helpers.Tests.Unit** project and add a new class named **RsapiHelperTests.cs**.
3. Overwrite the default class structure with the Unit Test structure we will be using.
4. Copy and paste code from **"E:\Github\ads-workshop-fest-2018\Projects\Starting\3\_UnitTests\TextFiles\1\_Structure.txt"** file replacing the entire **RsapiHelperTests.cs** class.



1. Add the RsapiHelper class reference which we will be our System under Test (SUT) for which we are writing unit tests.
2. Copy and paste code from **"E:\Github\ads-workshop-fest-2018\Projects\Starting\3\_UnitTests\TextFiles\2\_System\_Under\_Test.txt"** file at the beginning of the **RsapiHelperTests.cs** class.



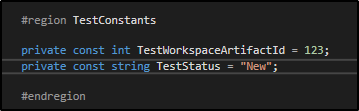
1. Next add the Mocks for various RDO repositories which are used in the Relativity application.
2. Copy and paste code from **"E:\Github\ads-workshop-fest-2018\Projects\Starting\3\_UnitTests\TextFiles\3\_Mocks.txt"** file at the beginning of the **RsapiHelperTests.cs** class.



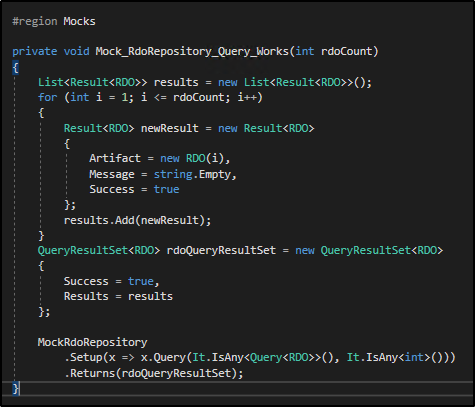
1. Next add the SetUp and TearDown methods which run once before and after all the tests execution.
2. The SetUp method is utilized to inititalize any data needed by tests.
3. The TearDown method is utilized to clean up the data used by the tests.
4. Copy and paste code from **"E:\Github\ads-workshop-fest-2018\Projects\Starting\3\_UnitTests\TextFiles\4\_SetUp\_And\_TearDown.txt"** file under the **SetUp and TearDown** region in the **RsapiHelperTests.cs** class.



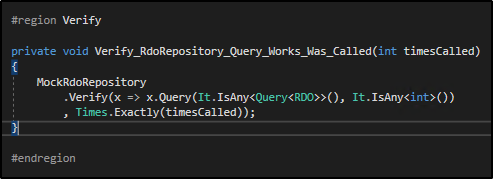
1. Next add the Test constants used in the Tests.
2. Copy and paste code from **"E:\Github\ads-workshop-fest-2018\Projects\Starting\3\_UnitTests\TextFiles\5\_Test\_Constants.txt"** file under the **TestConstants** region in the **RsapiHelperTests.cs** class.



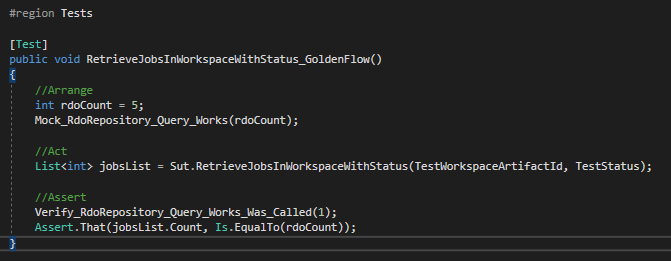
1. Next add the Mock Stubs for an RSAPI call used in the Tests. Here we will be stubbing out the Query method on the RDO repository in RSAPI.
2. Copy and paste code from **"E:\Github\ads-workshop-fest-2018\Projects\Starting\3\_UnitTests\TextFiles\6\_Mock\_Stub\_Golden\_Flow.txt"** file under the **Mocks** region in the **RsapiHelperTests.cs** class.



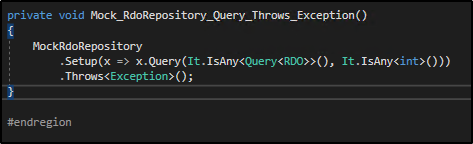
1. Next add the Verify method for the previously added Mock Stubs. Use this method we can verify if our stub was called during our test execution.
2. Copy and paste code from **"E:\Github\ads-workshop-fest-2018\Projects\Starting\3\_UnitTests\TextFiles\7\_Mock\_Stub\_Golden\_Flow\_Verify.txt"** file under the **Verify** region in the **RsapiHelperTests.cs** class.



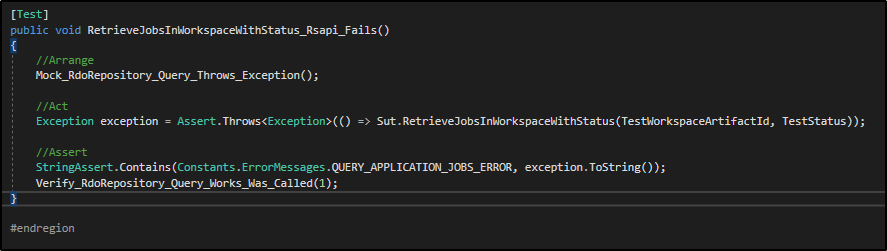
1. Next add the Golden flow test.
2. Copy and paste code from **"E:\Github\ads-workshop-fest-2018\Projects\Starting\3\_UnitTests\TextFiles\8\_UnitTest\_Golden\_Flow.txt"** file under the **Tests** region in the **RsapiHelperTests.cs** class.



1. Next add the Mock Stubs for an RSAPI call to simulate an exception in the RSAPI call.
2. Copy and paste code from **"E:\Github\ads-workshop-fest-2018\Projects\Starting\3\_UnitTests\TextFiles\9\_Mock\_Stub\_Rsapi\_Fails.txt"** file under the **Mocks** region in the **RsapiHelperTests.cs** class.

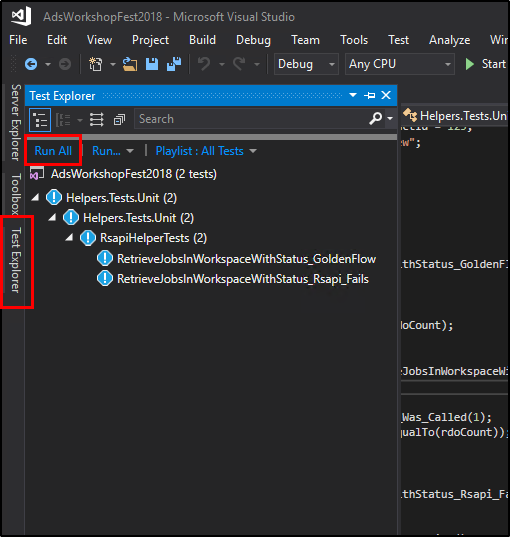


1. Next add the RSAPI Failure test.
2. Copy and paste code from **"E:\Github\ads-workshop-fest-2018\Projects\Starting\3\_UnitTests\TextFiles\10\_UnitTest\_Rsapi\_Fails.txt"** file under the **Tests** region in the **RsapiHelperTests.cs** class.

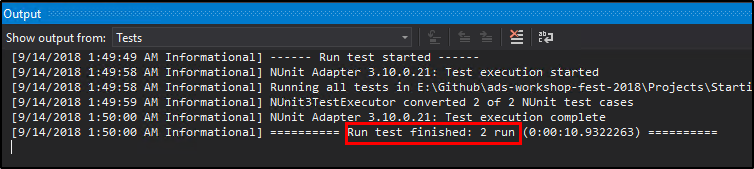


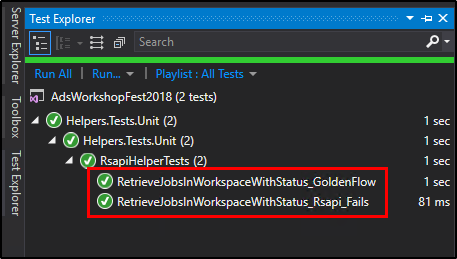
5.2 Running Unit Tests

1. Build the solution to successfully build all the projects.
2. Open the **Test Explorer** by click on the **Test Explorer** tab on the left side bar of Visual Studio.
3. Click on the **Run All** link to run the 2 unit tests we just created.



1. On successful run, click on the **Test Explorer** to see the status of the tests. Both the tests should be green.





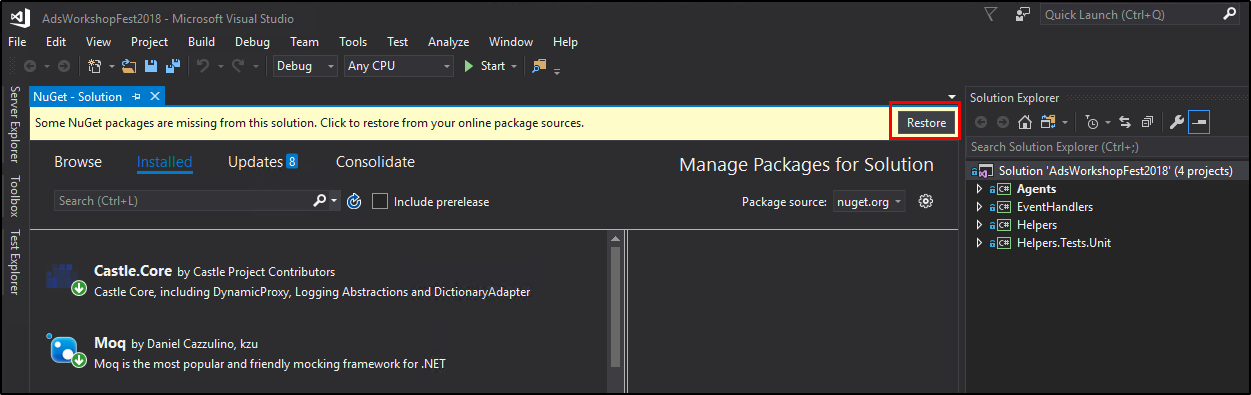
1. Integration Tests

6.1 Writing Integration Tests

1. Navigate to the **E:\Github\ads-workshop-fest-2018\Projects\Starting\4\_IntegrationTests\Project** folder and open **AdsWorkshopFest2018.sln** file.
2. In this solution, you will find 4 projects.
   1. Agents
   2. EventHandlers
   3. Helpers
   4. Helpers.Tests.Unit
3. In this section we will be working in the **Helpers.Tests.Unit** project.
4. In **Helpers** project, there is an **RsapiHelper.cs** file which contains RSAPI calls used in the sample application.
5. Our goal for this section is to write couple unit tests for the **RetrieveJobsInWorkspaceWithStatus** in **RsapiHelper** class.
6. Right click on the solution and select **Manage NuGet Packages for Solution** option.



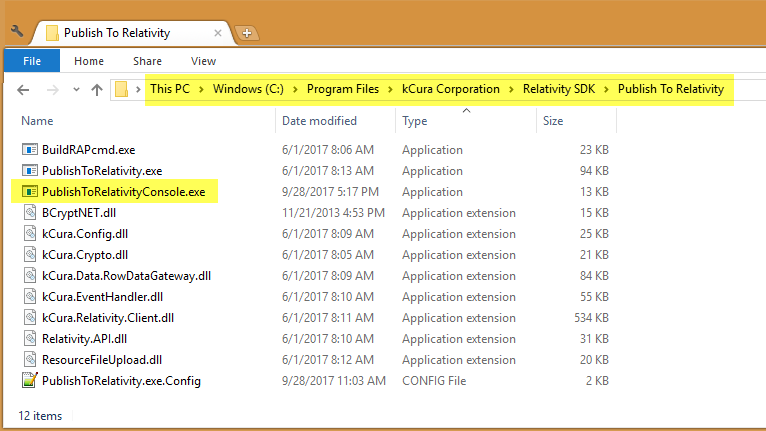
1. Next click the **Restore** button.



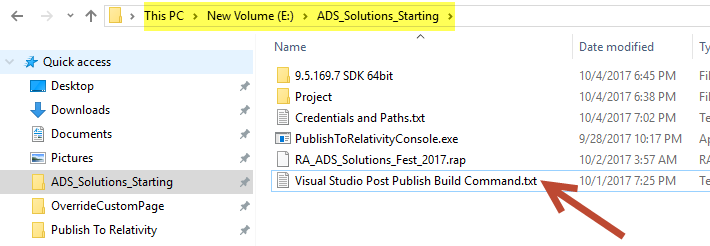
1. Now all the NuGet packages in the solution should be restored.
2. Right click on the **Helpers.Tests.Unit** project and add a new class named **RsapiHelperTests.cs**.

5.2 Visual Studio Post-Build Event & Publish to Relativity Console

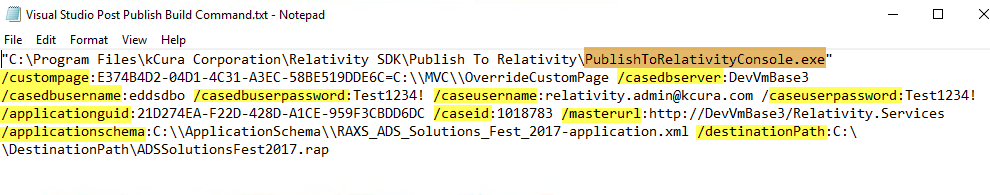
1. Instead of using the Publish to Relativity GUI to publish the custom page, we will use a command line tool called **PublishToRelativityConsole** I have created to publish the custom page. The **PublishToRelativityConsole** tool is already copied to the **Publish To Relativity** folder on your workshop machine. Go the **C:\Program Files\kCura Corporation\Relativity SDK\Publish To Relativity** folder and verify the tool exists.



1. **PublishToRelativityConsole** is an open source project you can find on Github at the link - [**https://github.com/relativitydev/publish-to-relativity-console**](https://github.com/relativitydev/publish-to-relativity-console)
2. We will also be using **Visual Studio Post-Build event** to call the **PublishToRelativityConsole** tool, which in turn publishes the custom page to Relativity application. You can learn more about Visual Studio Post-Build event at this link: [**https://docs.microsoft.com/en-us/visualstudio/ide/specifying-custom-build-events-in-visual-studio**](https://docs.microsoft.com/en-us/visualstudio/ide/specifying-custom-build-events-in-visual-studio)
3. Go to the **E:\ADS\_Solutions\_Starting** folder and make sure you see the **Visual Studio Post Publish Build Command** text file.



1. Double click on the **Visual Studio Post Publish Build Command** text file to see its contents.

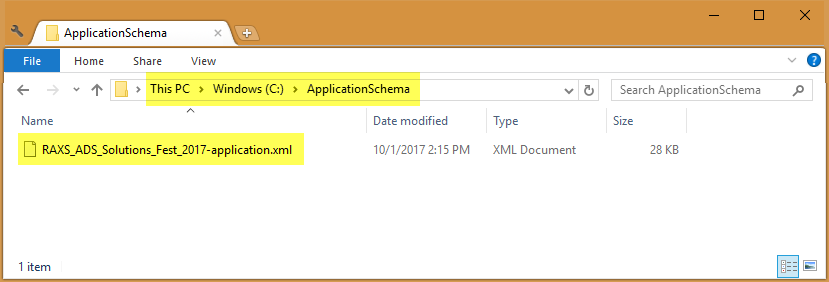


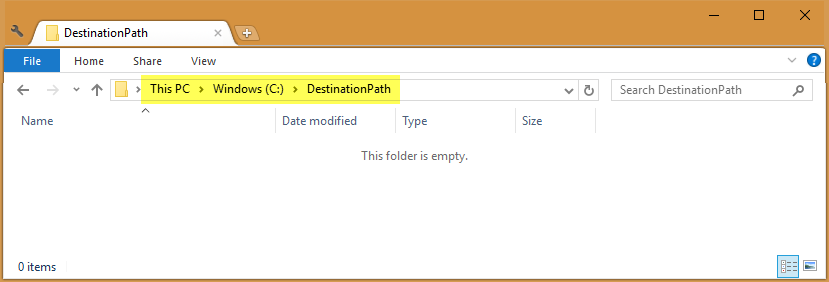
1. The post publish build command contains various **arguments** for the **PublishToRelativityConsole** tool and you can find the list below.

**Console Arguments:**

|  |  |
| --- | --- |
| **Argument** | **Description** |
| **"C:\Program Files\kCura Corporation\Relativity SDK\Publish To Relativity\PublishToRelativityConsole.exe"** | Path to the PublishToRelativityConsole tool |
| **/custompage:** **E374B4D2-04D1-4C31-A3EC-58BE519DDE6C=C:\\MVC\\OverrideCustomPage** | Custom page GUID and path |
| **/casedbserver:DevVmBase3.mshome.net** | SQL Server name |
| **/casedbusername:eddsdbo** | SQL Server login |
| **/casedbuserpassword:Test1234!** | SQL Server password |
| **/caseusername:relativity.admin@kcura.com** | Relativity admin username |
| **/caseuserpassword:Test1234!** | Relativity admin password |
| **/applicationguid:** **21D274EA-F22D-428D-A1CE-959F3CBDD6DC** | Application GUID |
| **/caseid:1024846** | Workspace Artifact ID |
| **/masterurl:http://DevVmBase3.mshome.net/Relativity.Services** | RSAPI URL |
| **/applicationschema:C:\\ApplicationSchema\\RAXS\_ADS\_Solutions\_Fest\_2017-application.xml** | Application Schema Path |
| **/destinationPath:C:\\DestinationPath\\ADSSolutionsFest2017.rap** | Generated RAP file path |

1. The **ApplicationSchema** and **DestinationPath** folders are already created on your workshop machine. The Application schema for the Relativity application is exported from the front-end and copied to the ApplicationSchema folder.

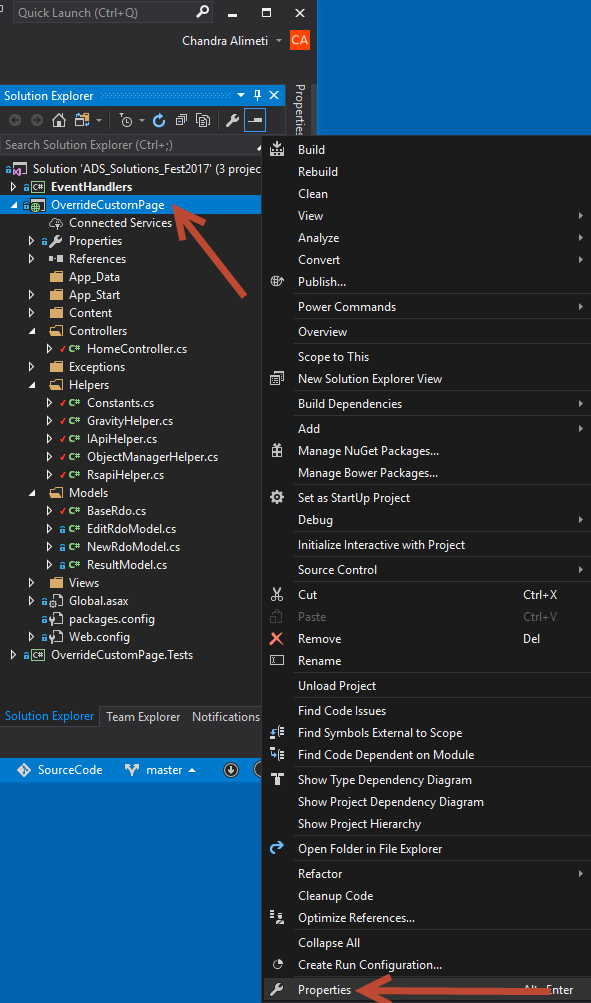




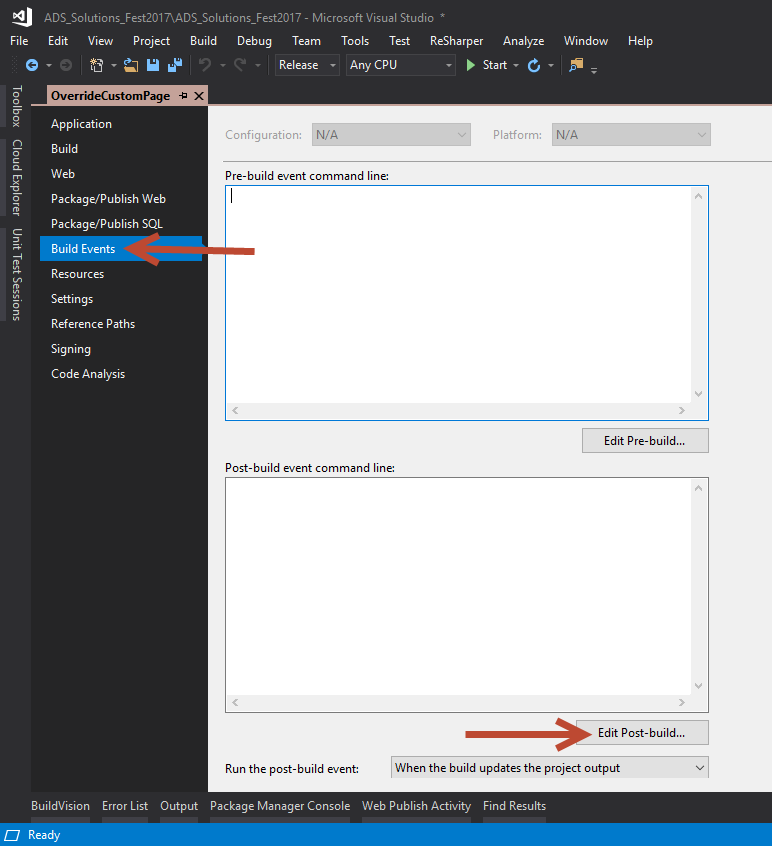
1. Copy the text for the post build publish event.



1. Right click on the **OverrideCustomPage** project and select **Properties** option.



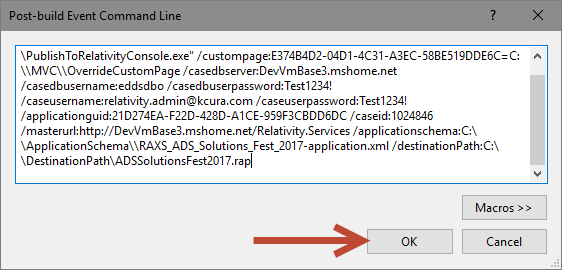
1. Select the **Build Events** option and click on the **Edit Post-build…** button.



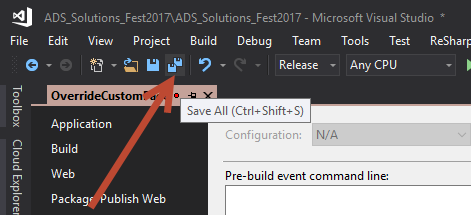
1. Now paste the copied text in the pop up window.



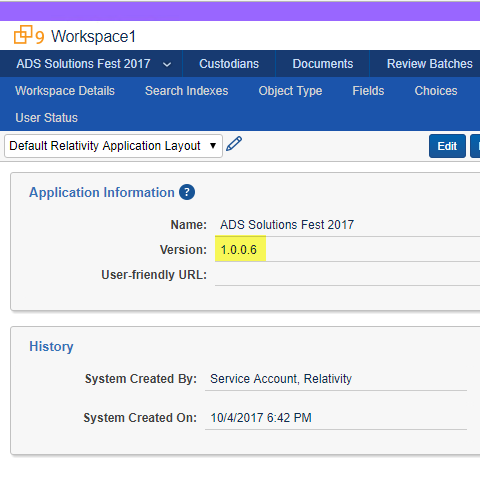
1. Click the **OK** button.



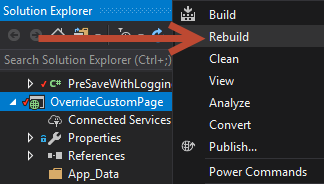
1. Next click the **Save All** button in Visual Studio.



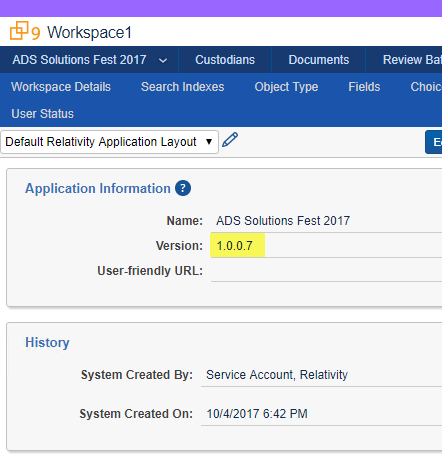
1. Open the **ADS Solutions Fest 2017** Relativity application in the **ADS Solutions Fest 2017 workspace**. Note down the current version of the application.



1. Now right click on the **OverrideCustomPage** project in Visual Studio and select the **Rebuild** option.

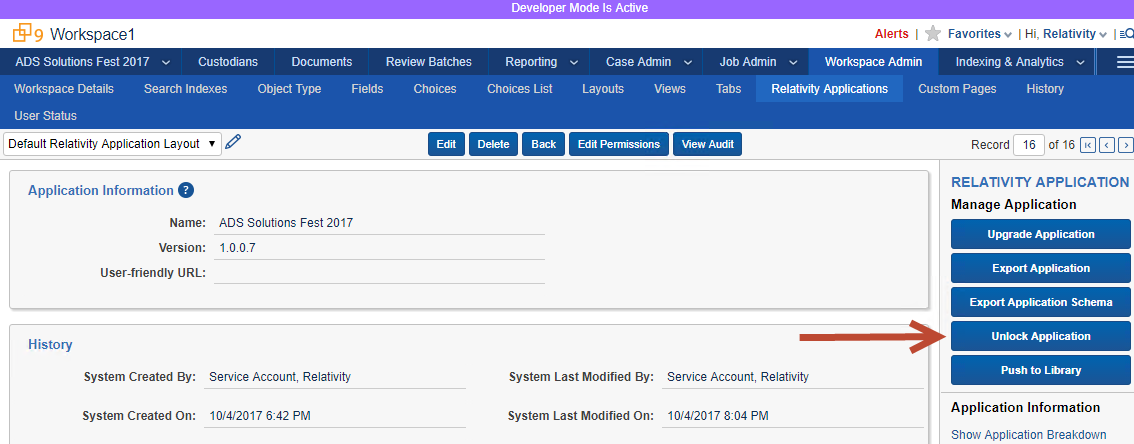


1. Now again open the **ADS Solutions Fest 2017** Relativity application in the **ADS Solutions Fest 2017 workspace** or **refresh** the page if it’s already opened. Verify that the application version is increased.

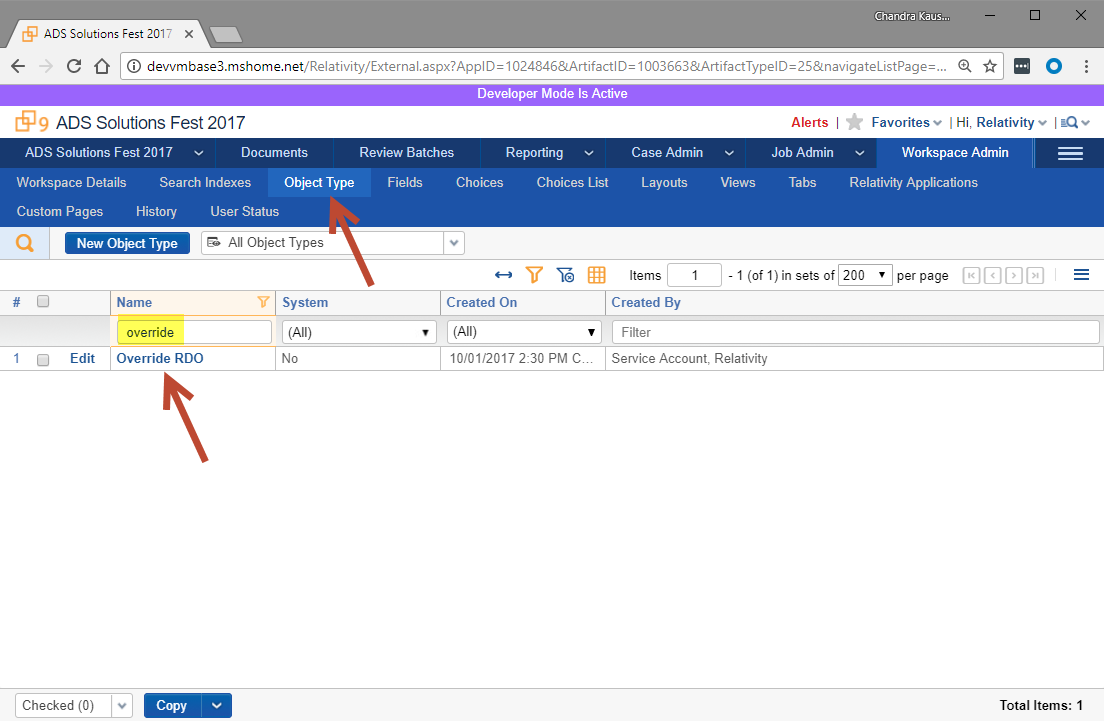


5.3 Object Rules

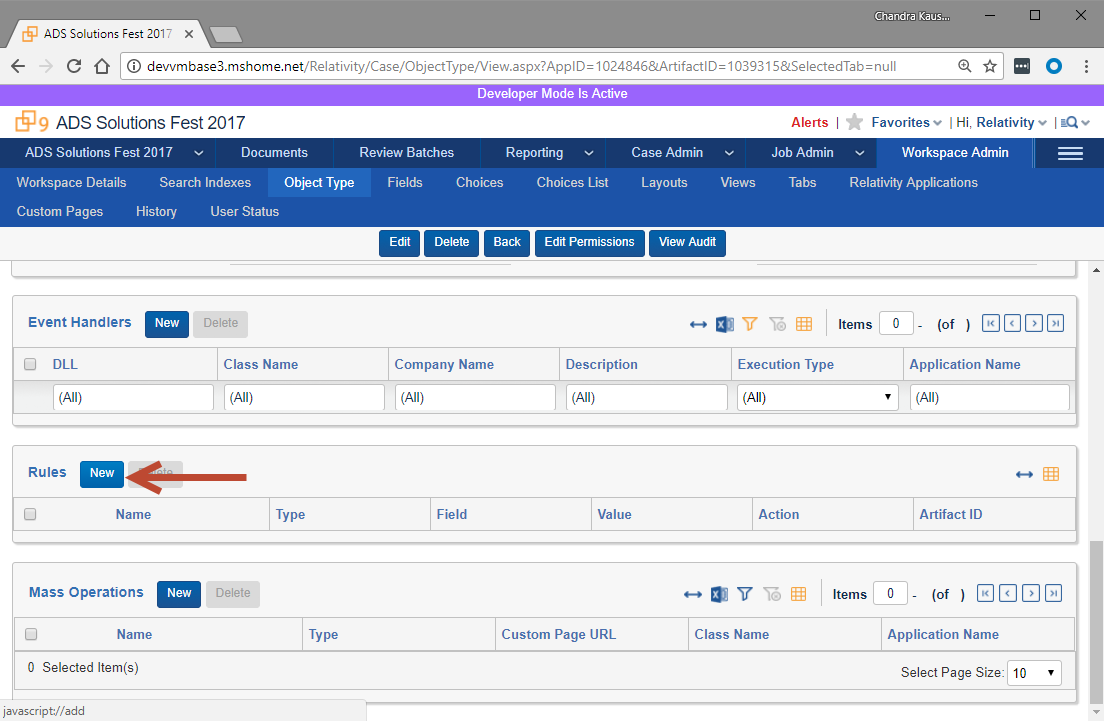
1. Unlock the Relativity application by clicking the **Unlock application** button.



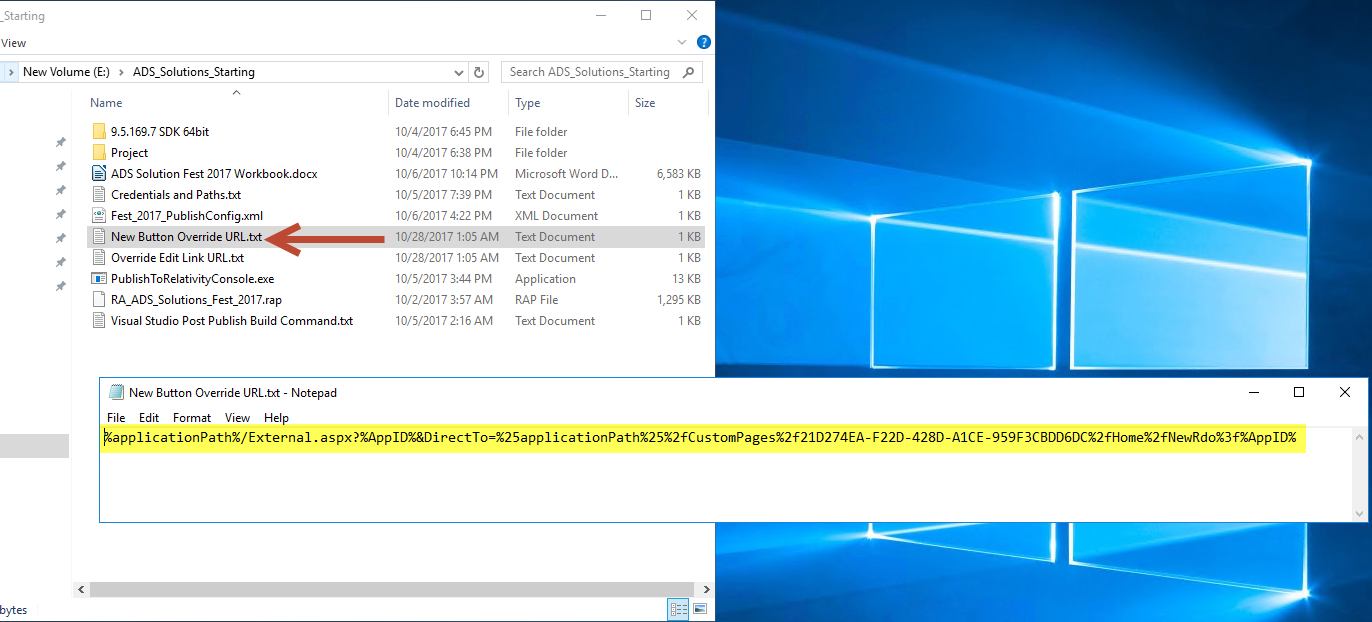
1. Next click the **Object Type** tab, search for the **override** word and click the **Override RDO** link.

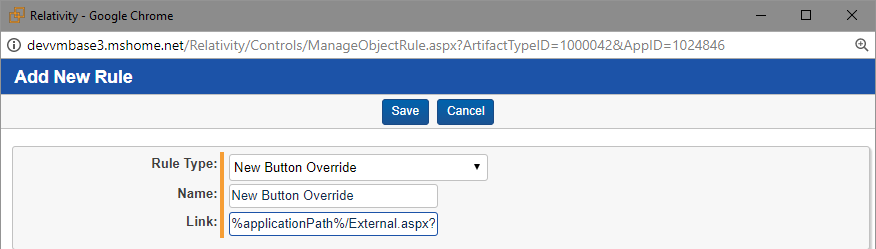


1. Scroll down to the **Rules** section and click on the **New** button.

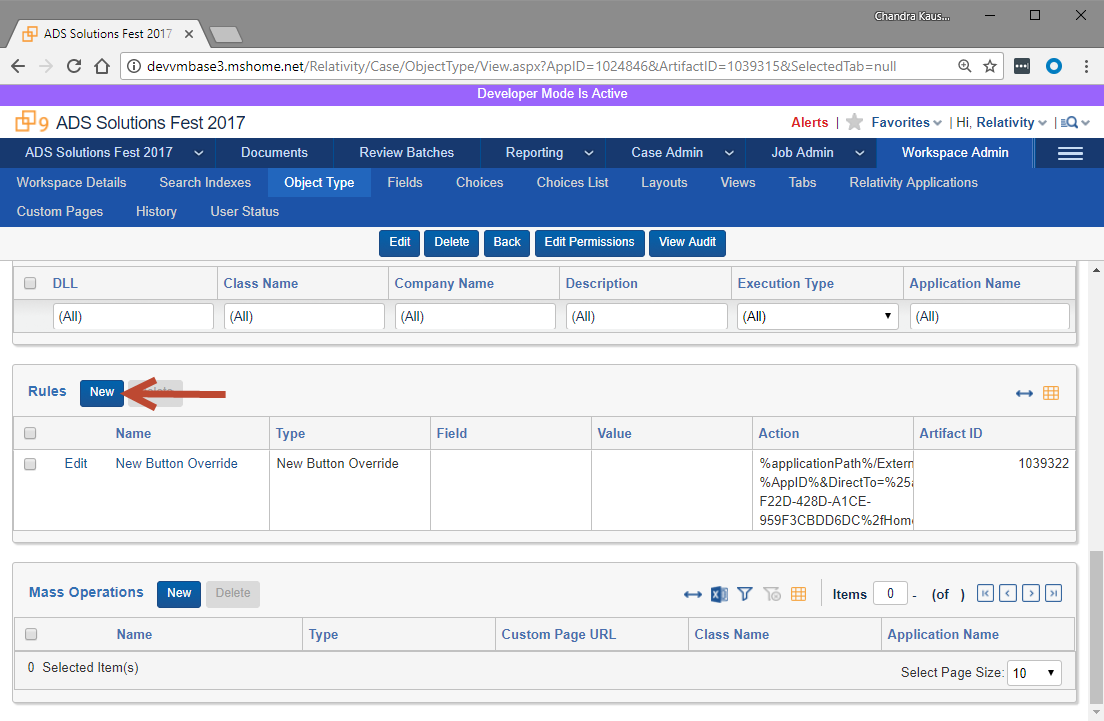


1. In the popup window, select **New Button Override** for **Rule Type**.
   1. Enter **New Button Override** for the **Name** field.
   2. For the **Link** field enter the text from the **New Button Override URL.txt** text file located in **E:\ADS\_Solutions\_Starting** folder as shown in the below screenshot.

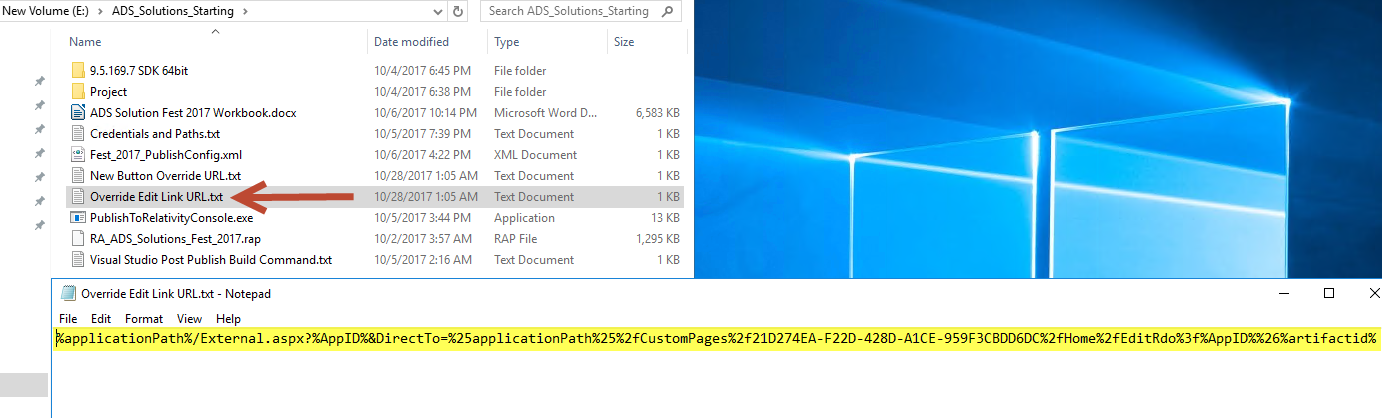


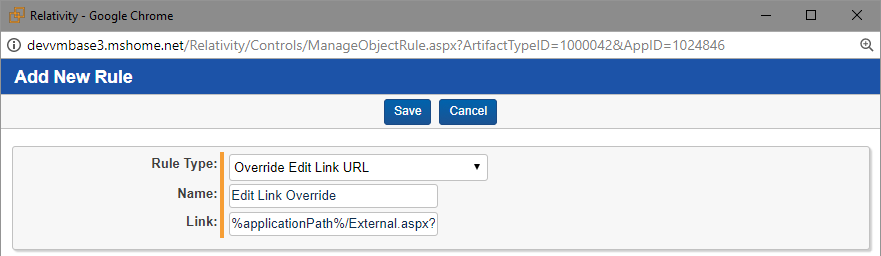


1. Click the **Save** button.
2. You can learn more about **New Button Override** at this link: [**https://platform.relativity.com/9.5/Content/Managing\_Relativity\_dynamic\_objects/RDO\_9.5/Editing\_Relativity\_Objects.htm?Highlight=edit%20link%20override#NewButtonOverride**](https://platform.relativity.com/9.5/Content/Managing_Relativity_dynamic_objects/RDO_9.5/Editing_Relativity_Objects.htm?Highlight=edit%20link%20override)
3. Click the **New** button again to add a second rule.

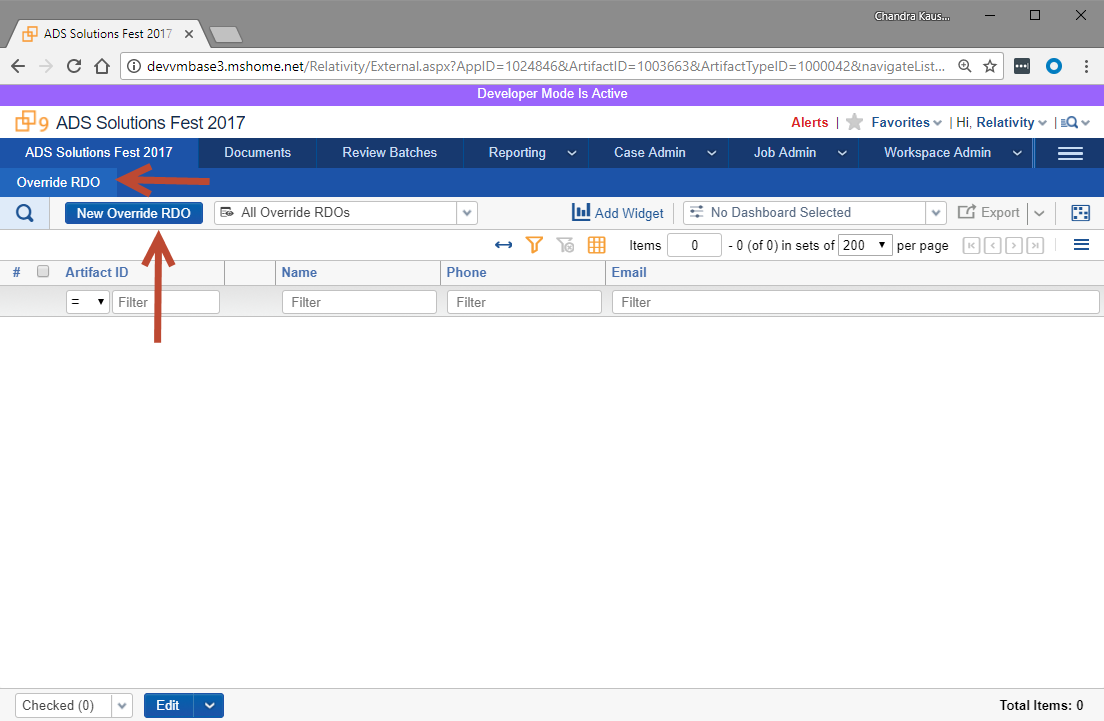


1. In the popup window, select **Override Edit Link URL** for **Rule Type**.
   1. Enter **Edit Link Override** for the **Name** field.
   2. For the **Link** field enter the text from the **Override Edit Link URL.txt** text file located in **E:\ADS\_Solutions\_Starting** folder as shown in the below screenshot.





1. Next click the **Save** button.
2. You can learn more about **Override Edit Link URL** at this link: [**https://platform.relativity.com/9.5/Content/Managing\_Relativity\_dynamic\_objects/RDO\_9.5/Editing\_Relativity\_Objects.htm?Highlight=edit%20link%20override#Override\_Edit\_Link\_URL**](https://platform.relativity.com/9.5/Content/Managing_Relativity_dynamic_objects/RDO_9.5/Editing_Relativity_Objects.htm?Highlight=edit%20link%20override)
3. Next go to **Override RDO** tab and click the **New Override RDO** button.

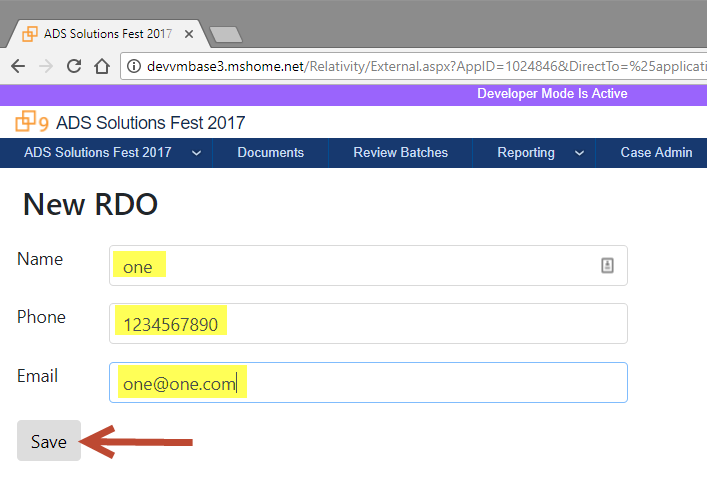


1. For the form, enter the following values

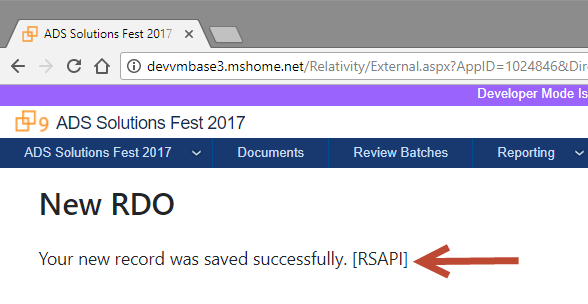
Name: one

Phone: 1234567890

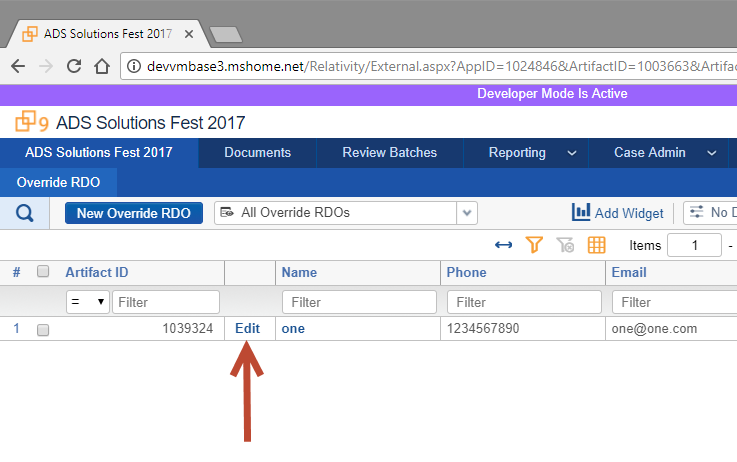
Email: [one@one.com](mailto:one@one.com)



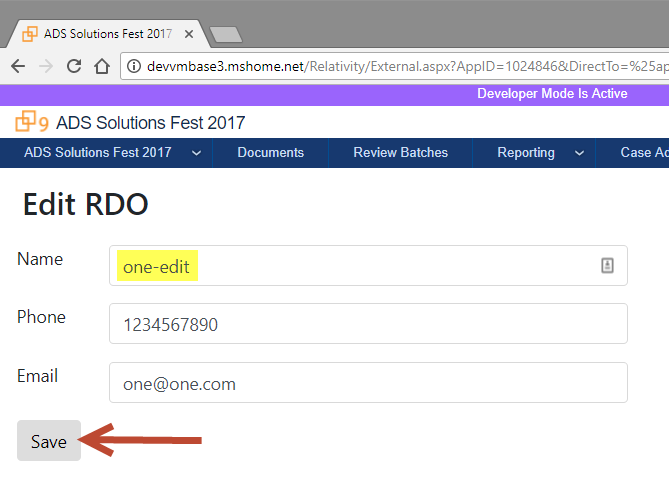
1. Click the **Save** button.
2. Once the RDO record is saved you will be shown a confirmation page as shown in the following screenshot.



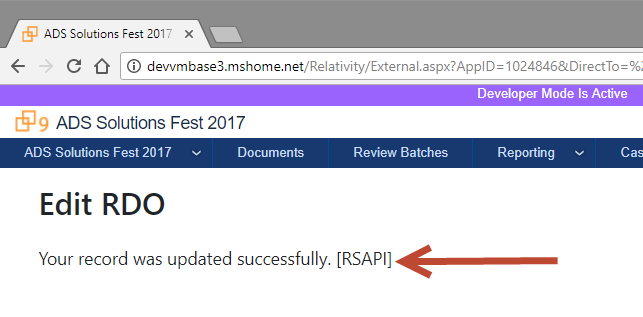
1. Next go to **Override RDO** tab and click the **Edit** link for the RDO record created above.



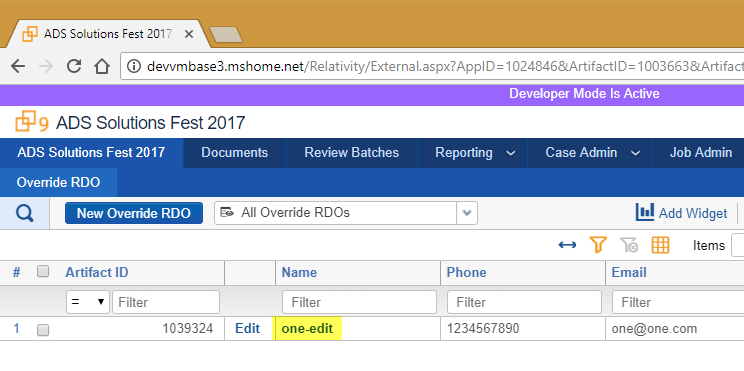
1. In the form, update the value of **Name** to **one-edit** and click on the **Save** button.



1. Once the RDO record is updated you will be shown a confirmation page as shown in the following screenshot.

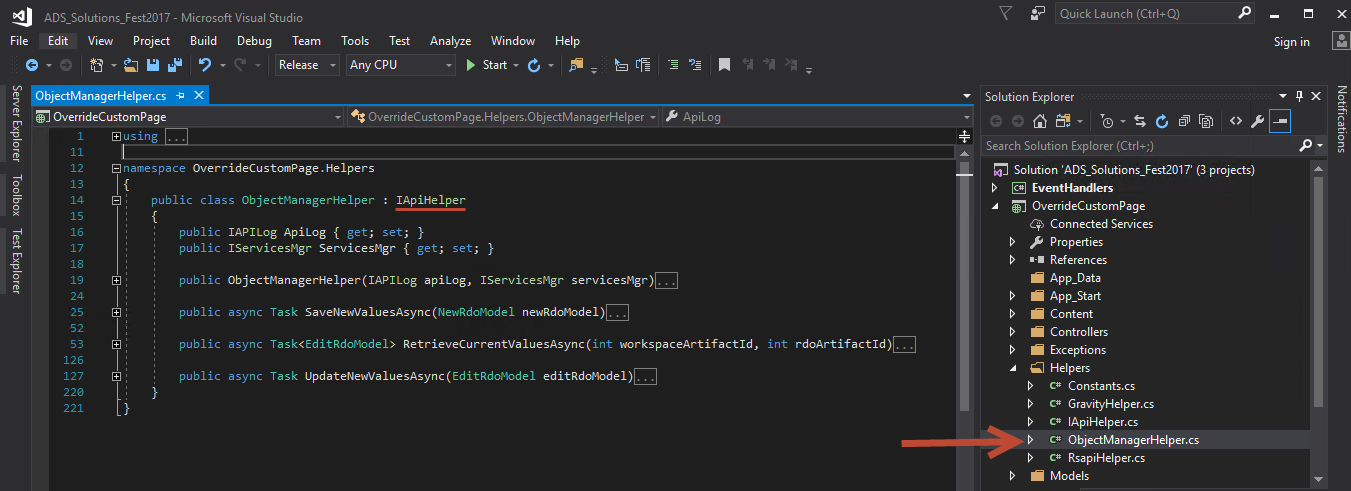


1. Go to **Override RDO** tab to verify the record was successfully updated.



5.4 ObjectManager API

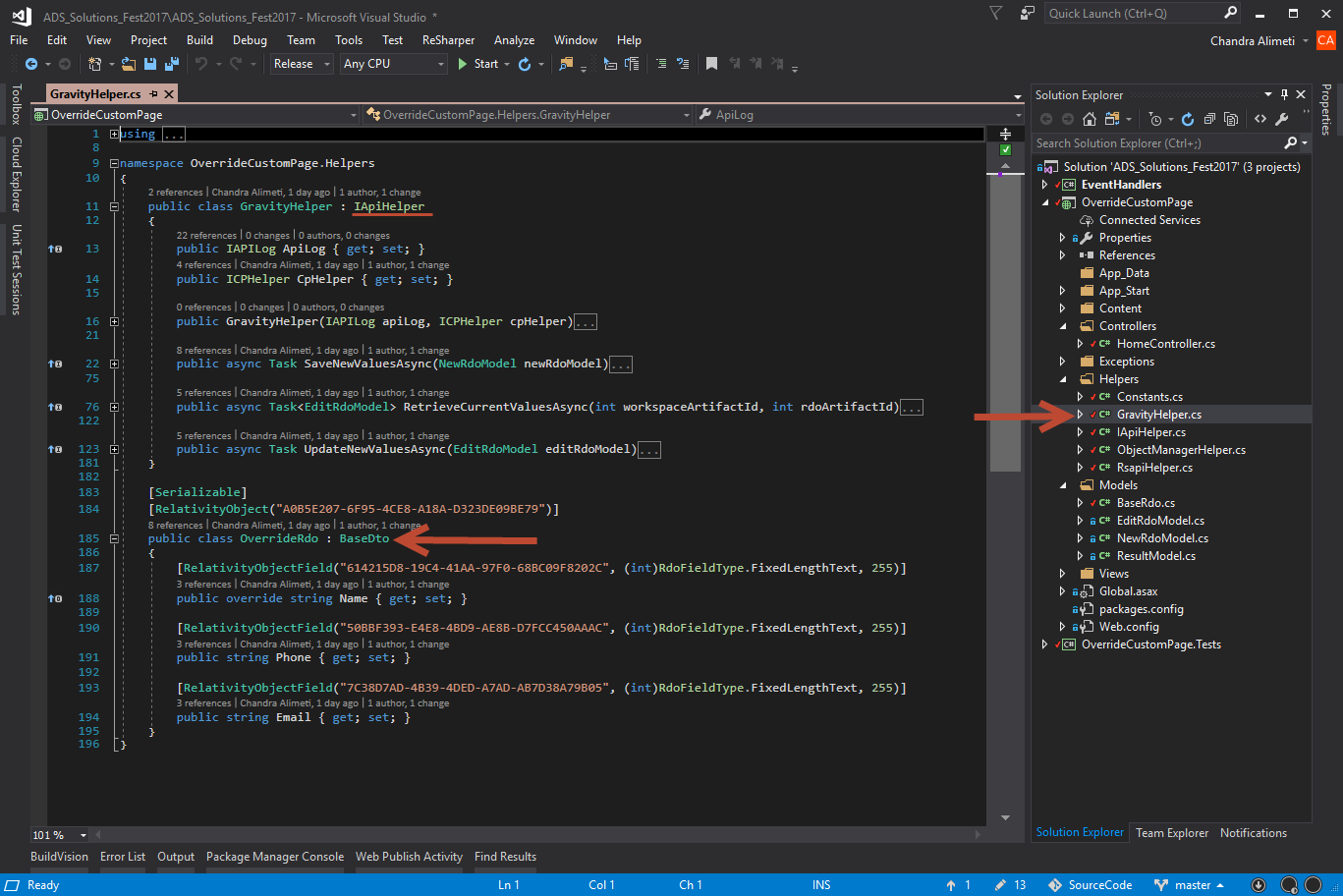
1. Go to **Visual Studio** and open the **ObjectManagerHelper.cs** file under **Helpers** folder in **OverrideCustomPage** project. This class is the implementation for the **IApiHelper** interface using the ObjectManager API.



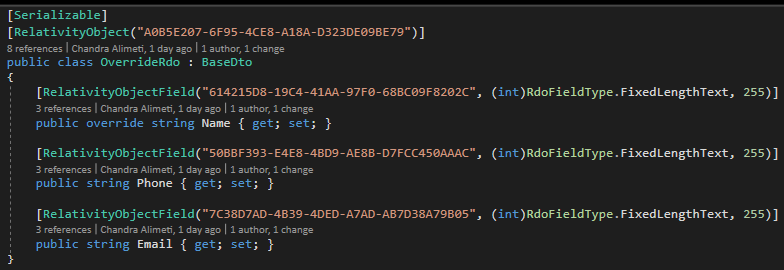
1. Just take a look at how you write code for the ObjectManager API. We will not be doing any workshop exercise on this.

5.5 Gravity API

1. In this section, we will be using the Gravity open source API instead of RSAPI to read and update RDO values. You can read more about the Gravity API at this link: [**https://github.com/tsdservices/Gravity**](https://github.com/tsdservices/Gravity)
2. Go to **Visual Studio** and open the **GravityHelper.cs** file under **Helpers** folder in **OverrideCustomPage** project.



1. To use the **Gravity** API, first we must create a class for each RDO or Document with its fields as properties as shown in below screenshot.



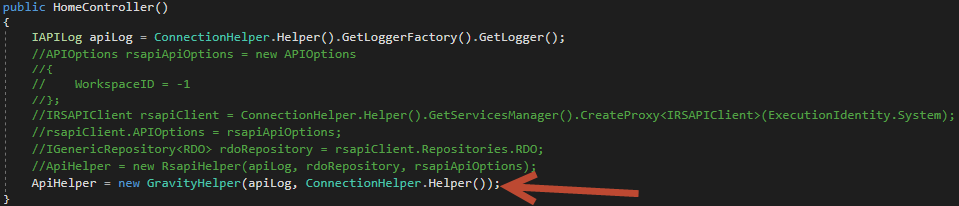
1. Next, we create an instance of **RsapiDao**.



1. Then using the RsapiDao we can call various CRUDQ methods for a Document or RDO.



1. To use Gravity API in the custom page, open the HomeController.cs file in Visual Studio. Comment the RSAPI code and replace it with the Gravity API code as shown in the below screenshot.



**Code to comment:**

//APIOptions rsapiApiOptions = new APIOptions

//{

// WorkspaceID = -1

//};

//IRSAPIClient rsapiClient = ConnectionHelper.Helper().GetServicesManager().CreateProxy<IRSAPIClient>(ExecutionIdentity.System);

//rsapiClient.APIOptions = rsapiApiOptions;

//IGenericRepository<RDO> rdoRepository = rsapiClient.Repositories.RDO;

//ApiHelper = new RsapiHelper(apiLog, rdoRepository, rsapiApiOptions);

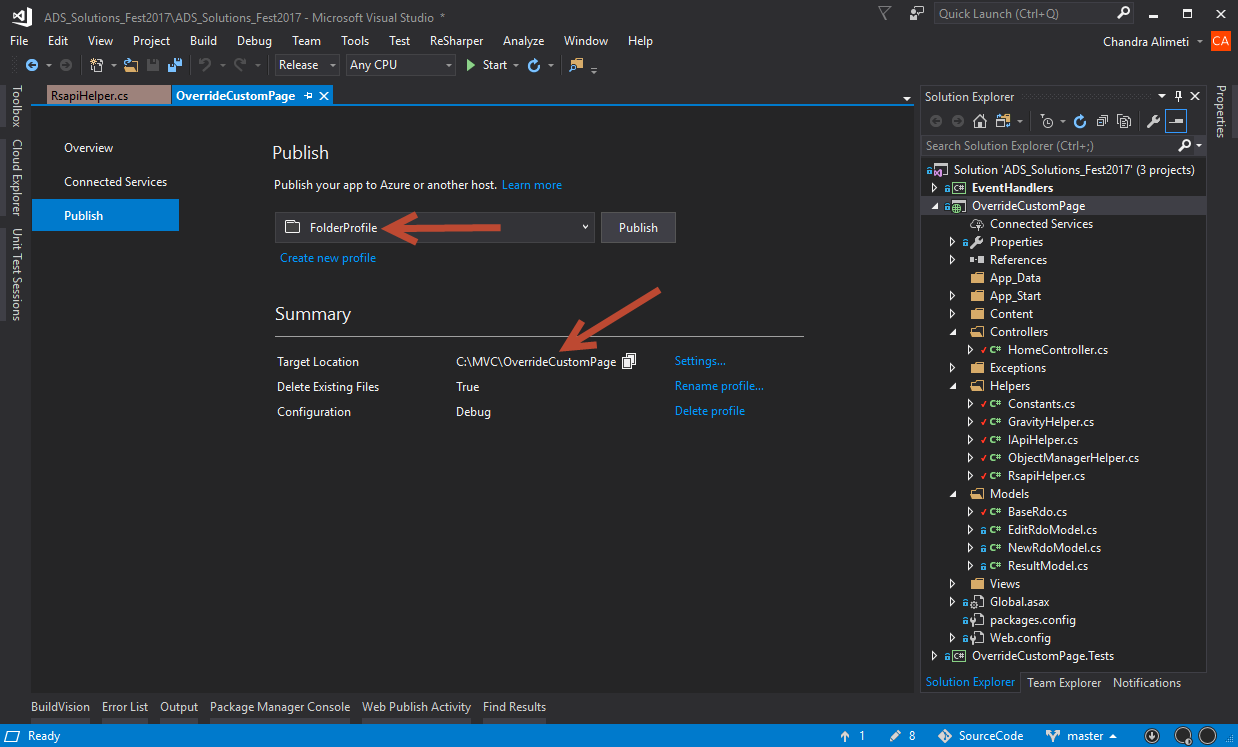
**Code to add:**

ApiHelper = new GravityHelper(apiLog, ConnectionHelper.Helper());

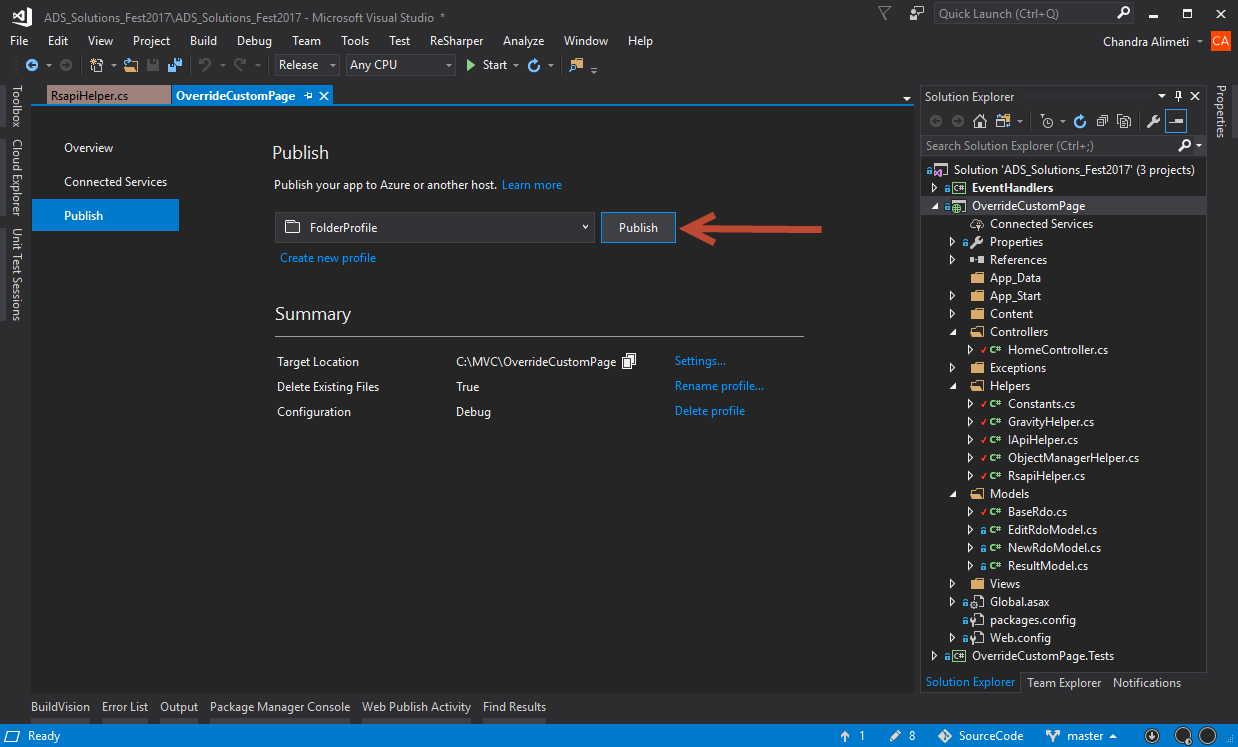
1. Right click on the **OverrideCustomPage** project and click on the **Publish** option.



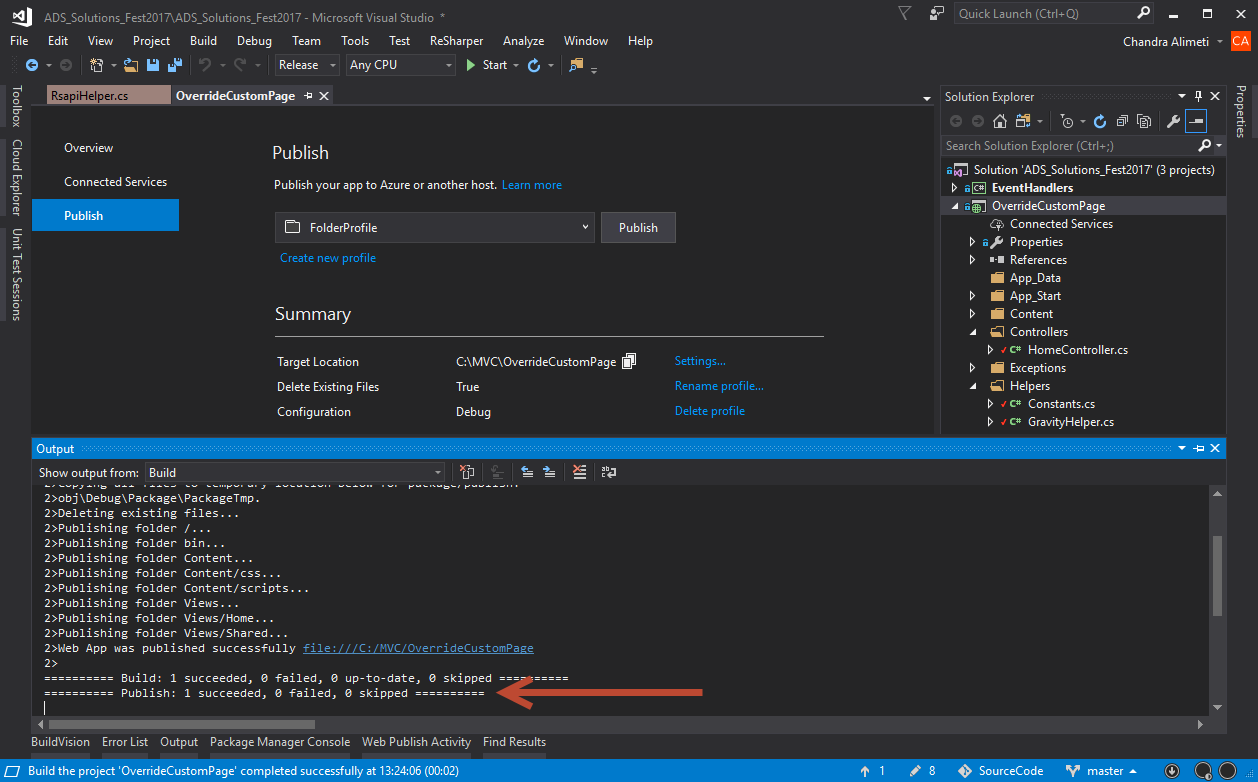
1. You should see a **Publish** settings tab opened. It already contains a **FolderProfile** which is set to publish the custom page to **C:\MVC\OverrideCustomPage** folder.



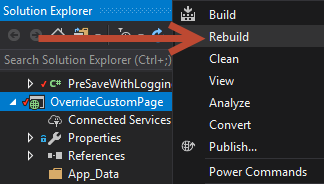
1. Click on the **Publish** button.



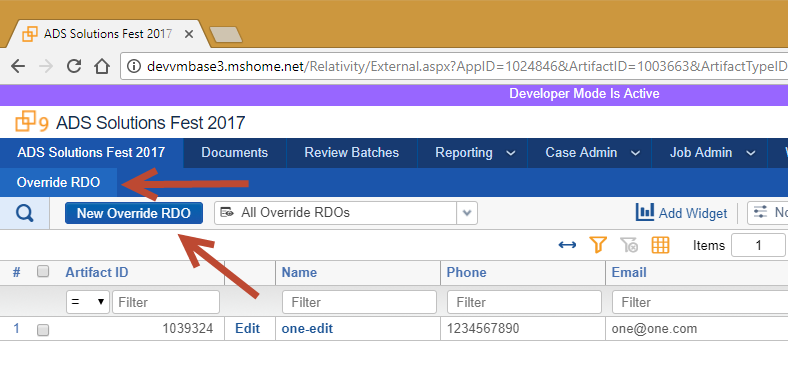
1. You will see a confirmation in the **Output** windows saying the custom page was successfully published.



1. Now right click on the **OverrideCustomPage** project in Visual Studio and select the **Rebuild** option.



1. Next go to **Override RDO** tab in Relativity and click the **New Override RDO** button.

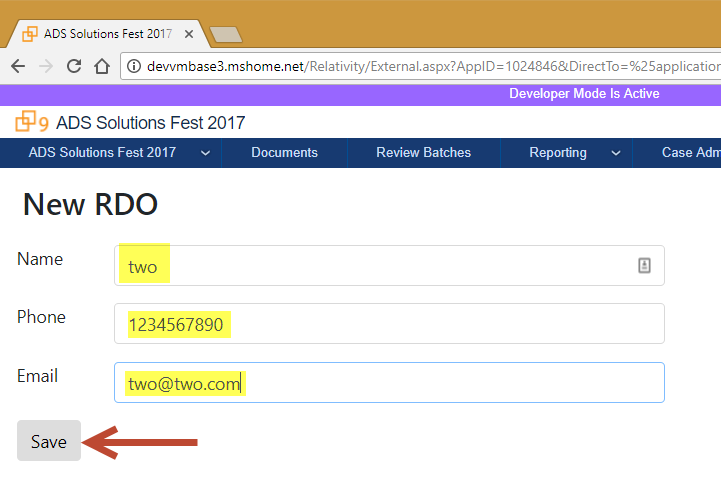


1. For the form, enter the following values

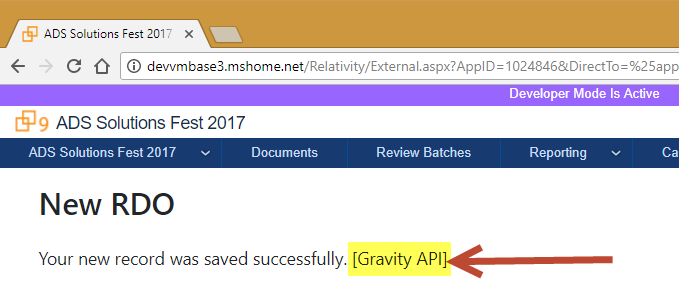
Name: two

Phone: 1234567890

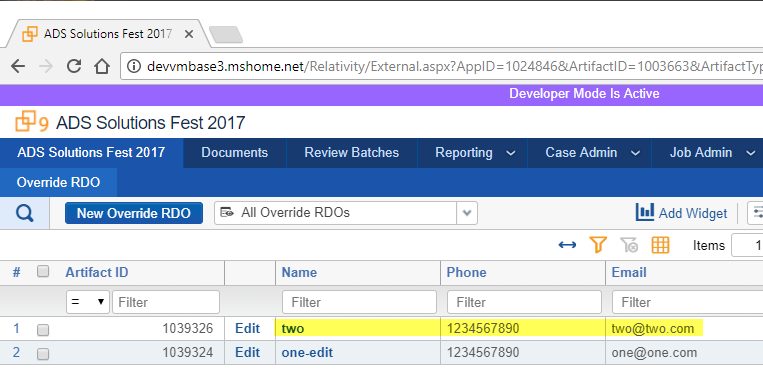
Email: [two@two.com](mailto:two@two.com)



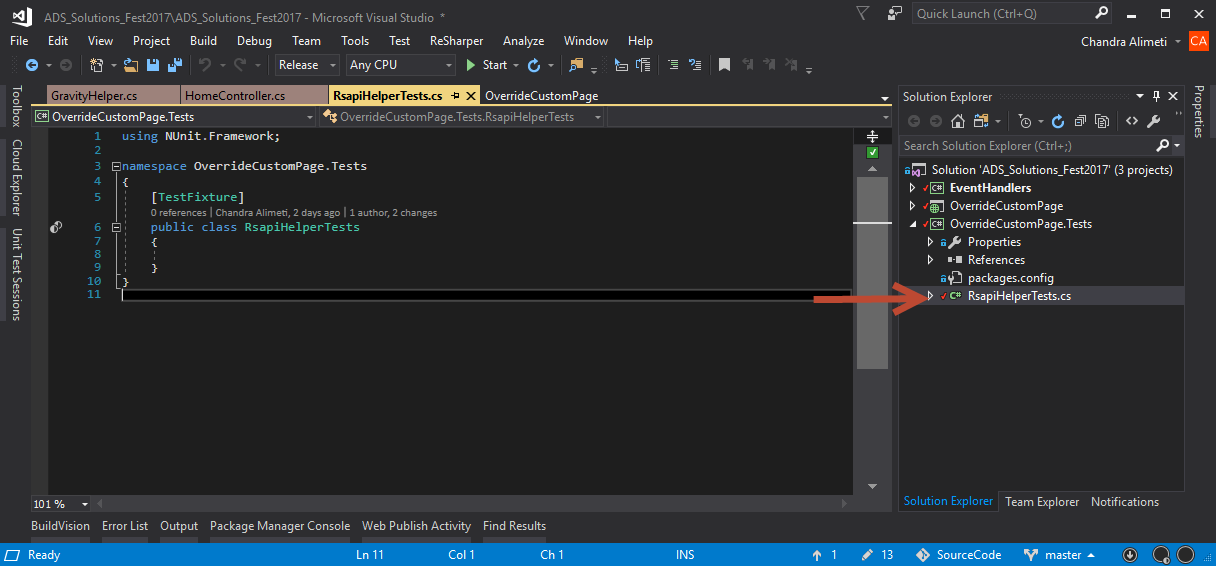
1. Click the **Save** button.
2. Once the RDO record is saved you will be shown a confirmation page as shown in the following screenshot.



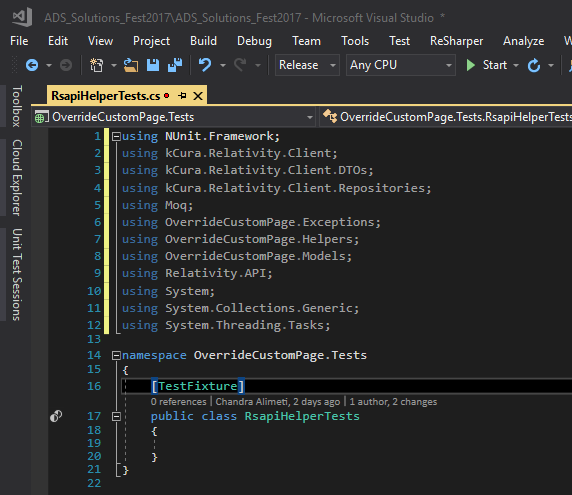
1. Go to **Override RDO** tab to verify the record was successfully saved.



1. Unit Tests
2. In this section, we will write unit tests for the **RsapiHelper** class which contains all the RSAPI calls for the custom page.
3. Go to Visual Studio, expand the **OverrideCustomPage.Tests** project.
4. Double click on the **RsapiHelperTests.cs** file. This is an empty class where we will be writing unit tests.



1. Add the following **using** statements to the class if they are not already present.



**Code:**

using kCura.Relativity.Client;

using kCura.Relativity.Client.DTOs;

using kCura.Relativity.Client.Repositories;

using Moq;

using OverrideCustomPage.Exceptions;

using OverrideCustomPage.Helpers;

using OverrideCustomPage.Models;

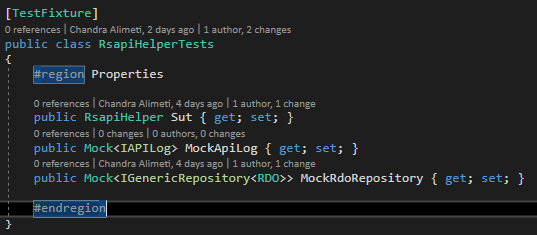
using Relativity.API;

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

1. Next add the following **properties** to the class.



**Code:**

#region Properties

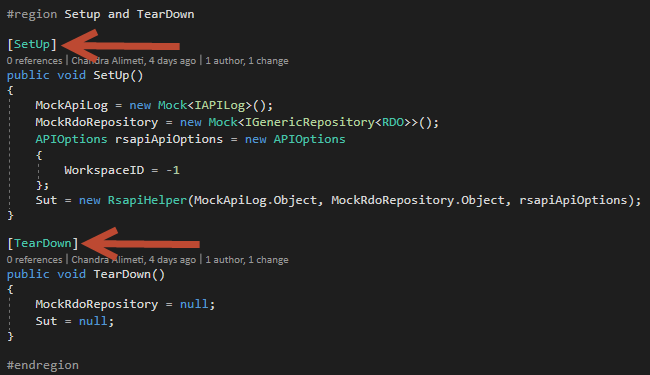
public RsapiHelper Sut { get; set; }

public Mock<IAPILog> MockApiLog { get; set; }

public Mock<IGenericRepository<RDO>> MockRdoRepository { get; set; }

#endregion

1. Next add the following **SetUp** and **TearDown** methods to the class.



**Code:**

#region Setup and TearDown

[SetUp]

public void SetUp()

{

MockApiLog = new Mock<IAPILog>();

MockRdoRepository = new Mock<IGenericRepository<RDO>>();

APIOptions rsapiApiOptions = new APIOptions

{

WorkspaceID = -1

};

Sut = new RsapiHelper(MockApiLog.Object, MockRdoRepository.Object, rsapiApiOptions);

}

[TearDown]

public void TearDown()

{

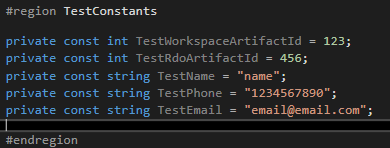
MockRdoRepository = null;

Sut = null;

}

#endregion

1. Next add the following **Test constants** to the class. We will be using them in our tests.



**Code:**

#region TestConstants

private const int TestWorkspaceArtifactId = 123;

private const int TestRdoArtifactId = 456;

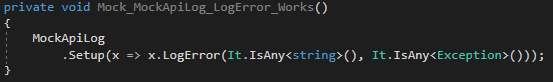
private const string TestName = "name";

private const string TestPhone = "1234567890";

private const string TestEmail = "email@email.com";

#endregion

1. Next, add the following method which mocks the **LogError** method on the **IApiLog** interface.



**Code:**

private void Mock\_MockApiLog\_LogError\_Works()

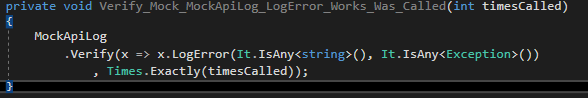
{

MockApiLog

.Setup(x => x.LogError(It.IsAny<string>(), It.IsAny<Exception>()));

}

1. Next, add the following method which verifies that the **LogError** method on the **IApiLog** interface was called. This will be part of the Assert section of the unit test.



**Code:**

private void Verify\_Mock\_MockApiLog\_LogError\_Works\_Was\_Called(int timesCalled)

{

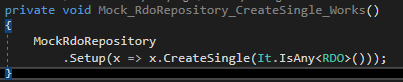
MockApiLog

.Verify(x => x.LogError(It.IsAny<string>(), It.IsAny<Exception>())

, Times.Exactly(timesCalled));

}

1. Next add the following method which mocks the **CreateSingle** method on the **IRSAPIClient** interface.



**Code:**

private void Mock\_RdoRepository\_CreateSingle\_Works()

{

MockRdoRepository

.Setup(x => x.CreateSingle(It.IsAny<RDO>()));

}

1. Next add the following method which verifies that the the **CreateSingle** method on the **IRSAPIClient** interface was called. This will be part of the Assert section of the unit test.



**Code:**

private void Verify\_Mock\_RdoRepository\_CreateSingle\_Works\_Was\_Called(int timesCalled)

{

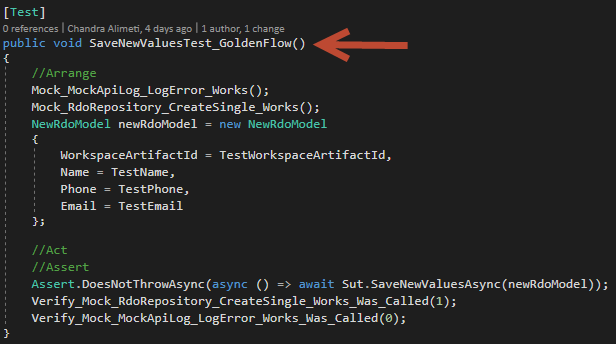
MockRdoRepository

.Verify(x => x.CreateSingle(It.IsAny<RDO>())

, Times.Exactly(timesCalled));

}

1. Next, add the following unit test for the **SaveNewValues()** method in the **RsapiHelper** class. This unit test tests the scenario where the RSAPI CreateSingle API call works without any errors.



**Code:**

[Test]

public void SaveNewValuesTest\_GoldenFlow()

{

//Arrange

Mock\_MockApiLog\_LogError\_Works();

Mock\_RdoRepository\_CreateSingle\_Works();

NewRdoModel newRdoModel = new NewRdoModel

{

WorkspaceArtifactId = TestWorkspaceArtifactId,

Name = TestName,

Phone = TestPhone,

Email = TestEmail

};

//Act

//Assert

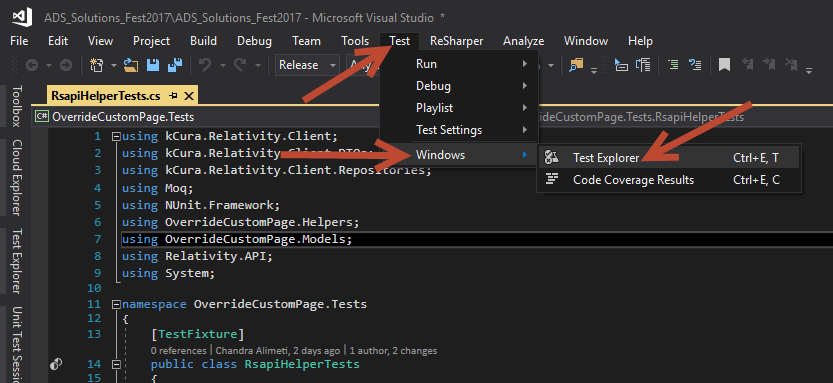
Assert.DoesNotThrowAsync(async () => await Sut.SaveNewValuesAsync(newRdoModel));

Verify\_Mock\_RdoRepository\_CreateSingle\_Works\_Was\_Called(1);

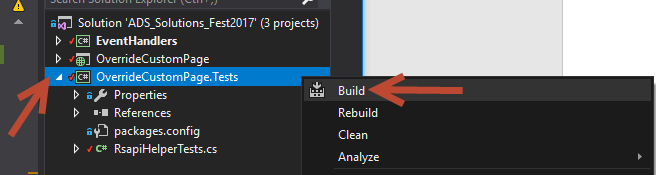
Verify\_Mock\_MockApiLog\_LogError\_Works\_Was\_Called(0);

}

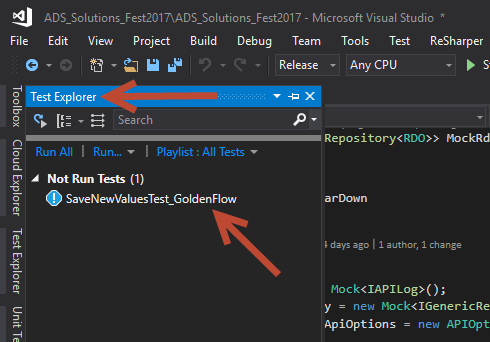
1. Go to Visual Studio, click on **Test** menu option, select **Windows** option, then click on the **Test Explorer** option.



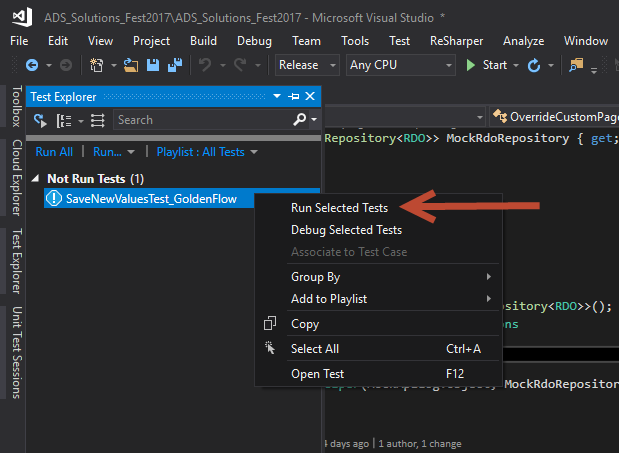
1. Build the project to see the new test in the **Test Explorer** window. Right click on the **OverrideCustomPage.Tests** project and select the **Build** option.



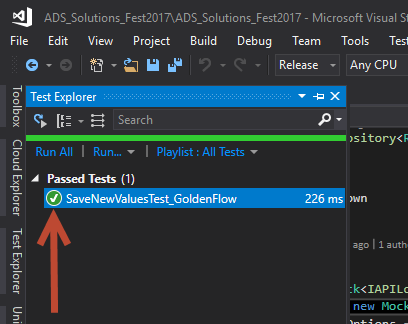
1. In Test Explorer, you should the **SaveNewValuesTest\_GlodenFlow** test we have just written.



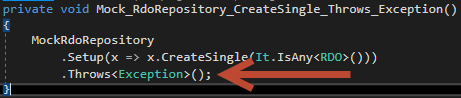
1. Right click on the **SaveNewValuesTest\_GlodenFlow** test and select **Run Selected Tests** option.



1. Once the test is finished running, you should see a green check mark next to the test indicating the test has passed.



1. Next add the following method which mocks the **CreateSingle** method on the **IRSAPIClient** interface to throw an Exception.



**Code:**

private void Mock\_RdoRepository\_CreateSingle\_Throws\_Exception()

{

MockRdoRepository

.Setup(x => x.CreateSingle(It.IsAny<RDO>()))

.Throws<Exception>();

}

1. Next add the following unit test for the **SaveNewValues()** method in the **RsapiHelper** class. This unit test tests the scenario where the RSAPI CreateSingle API call throws an exception.



**Code:**

[Test]

public void SaveNewValuesTest\_Rsapi\_Fails()

{

//Arrange

Mock\_MockApiLog\_LogError\_Works();

Mock\_RdoRepository\_CreateSingle\_Throws\_Exception();

NewRdoModel newRdoModel = new NewRdoModel

{

WorkspaceArtifactId = TestWorkspaceArtifactId,

Name = TestName,

Phone = TestPhone,

Email = TestEmail

};

//Act

OverrideCustomPageException overrideCustomPageException = Assert.ThrowsAsync<OverrideCustomPageException>(async () => await Sut.SaveNewValuesAsync(newRdoModel));

//Assert

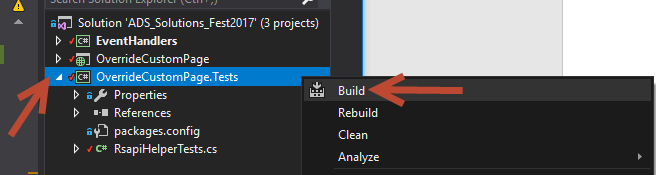
StringAssert.Contains(Helpers.Constants.ErrorMessages.NewRdoCreateError, overrideCustomPageException.ToString());

Verify\_Mock\_RdoRepository\_CreateSingle\_Works\_Was\_Called(1);

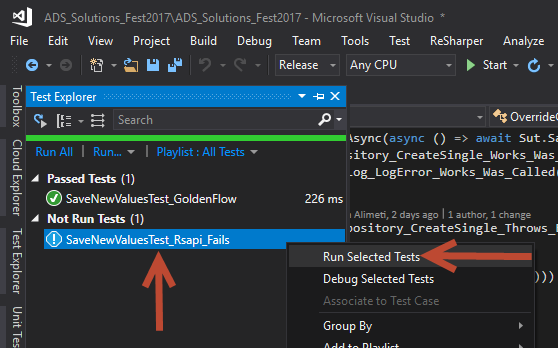
Verify\_Mock\_MockApiLog\_LogError\_Works\_Was\_Called(2);

}

1. Build the project to see the new test in the **Test Explorer** window. Right click on the **OverrideCustomPage.Tests** project and select the **Build** option.



1. Right click on the **SaveNewValuesTest\_Rsapi\_Fails** test and select **Run Selected Tests** option.



1. Once the test is finished running, you should see a green check mark next to the test indicating the test has passed.

r