How to Create Solutions for Relativity & ADS Solutions Workshop

September 12, 2018 - Version 1.0

Contents

1 Course Overview

ADS is the standard way to build applications on the Relativity Platform. As a developer, you want a good process for building apps so you can focus on delivering business value quickly. In this session, we'll review common patterns for setting up an effective ADS development solution. Topics will include: setting up a Visual Studio build workflow; versioning; which assemblies to include in your application; source code management; Relativity developer tools; and more. We'll also highlight recent improvements to ADS that'll make your life easier.

1.1 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

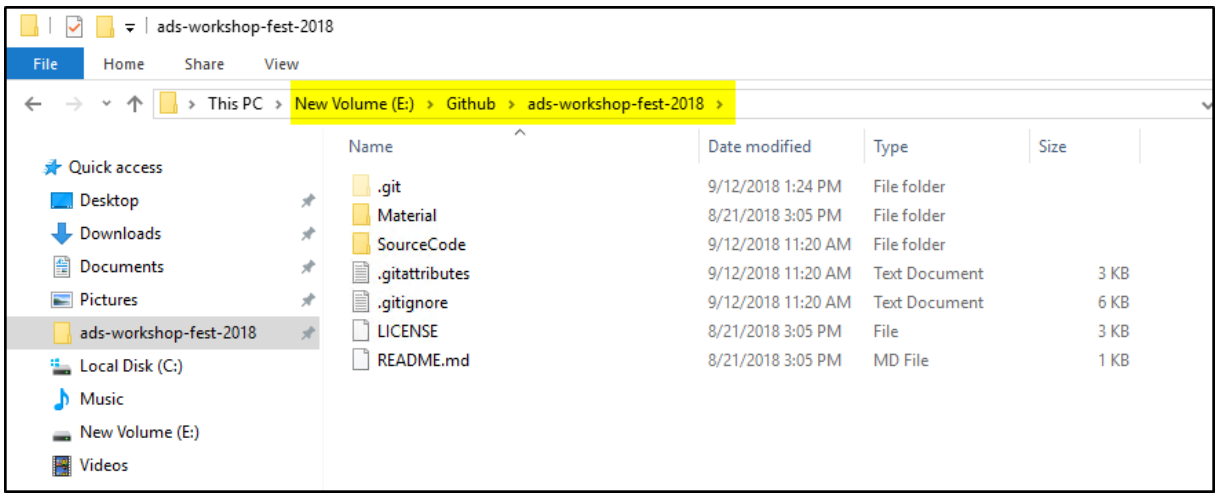
We have provided the required software for you for this workshop.

2 Getting Started

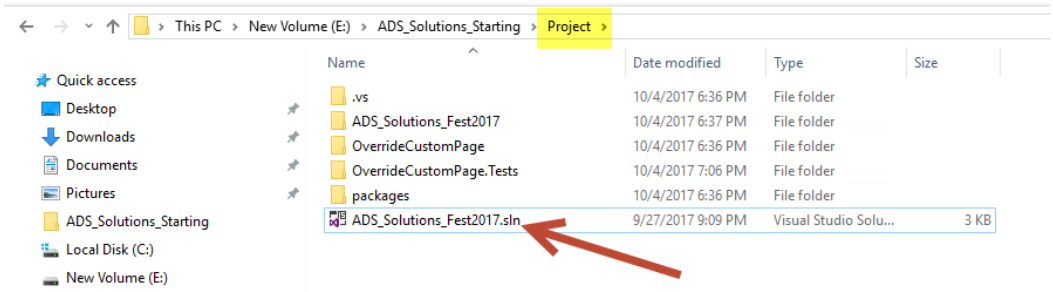
2.1 Visual Studio Project

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

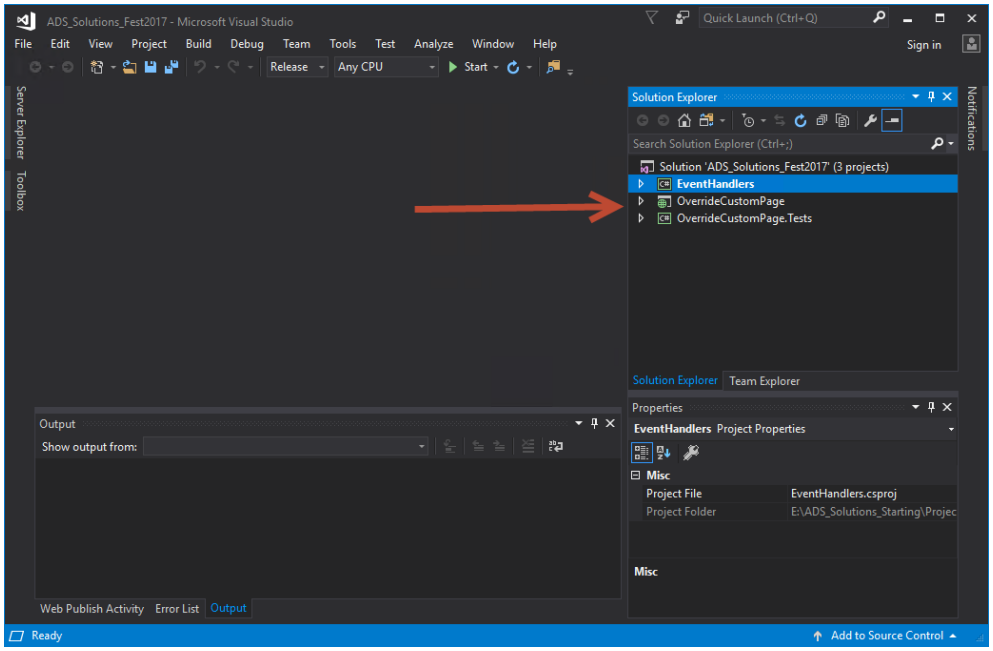
2. Double click on the **ads-workshop-fest-2018** folder to open it. Make sure you see the files as shown in the below screenshot.



3. Go to the **Project** folder and double click the **ADS_Solutions_Fest2017.sln** file to open the Visual Studio solution.



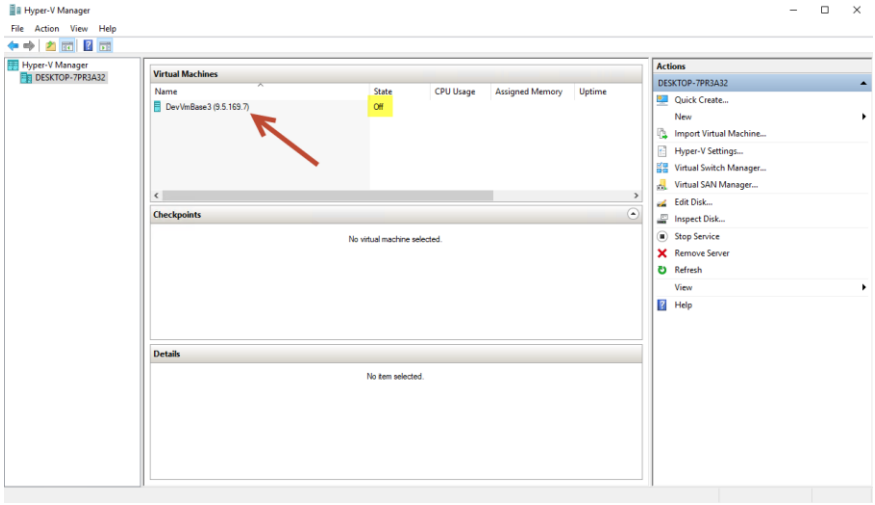
4. **Visual Studio 2017** will be opened and it contains three projects listed in the **Solution Explorer** as shown in the below screenshot.



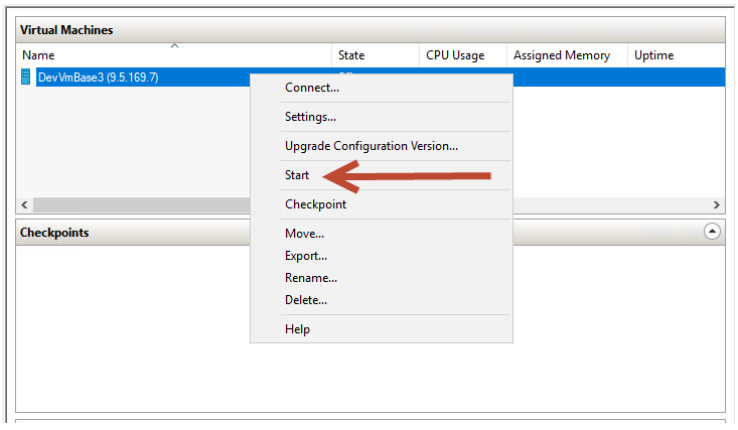
2.2 DevVM Setup

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

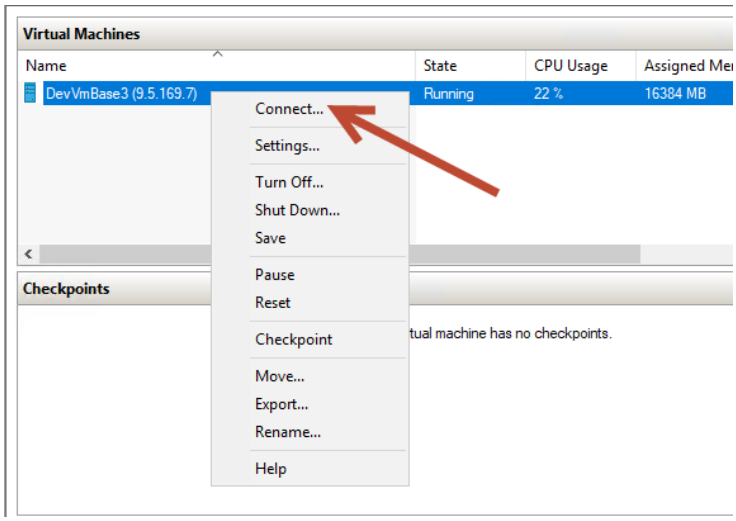
2. You will see a **Relativity 9.5 DevVM** listed in the Hyper-V Manager Virtual Machine section.



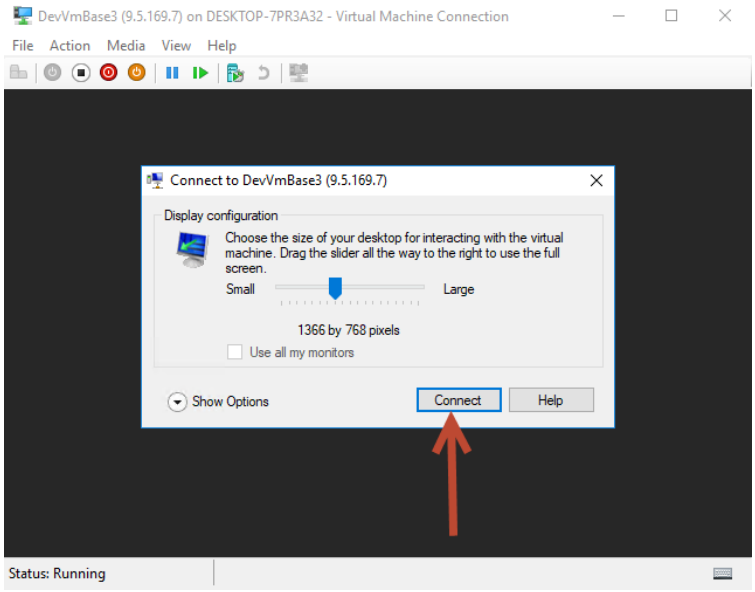
3. Right click on the **DevVM** and select the **Start** option.



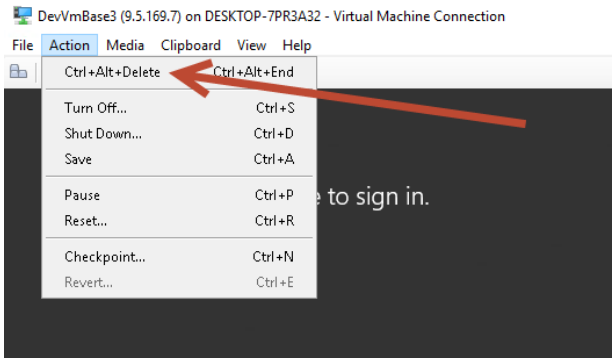
4. Right click on the **DevVM** and select the **Connect** option.

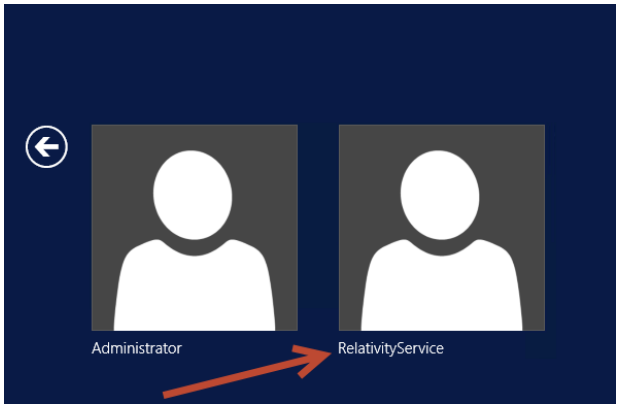


5. If you get a Display configuration pop-up, click the **Connect** button.

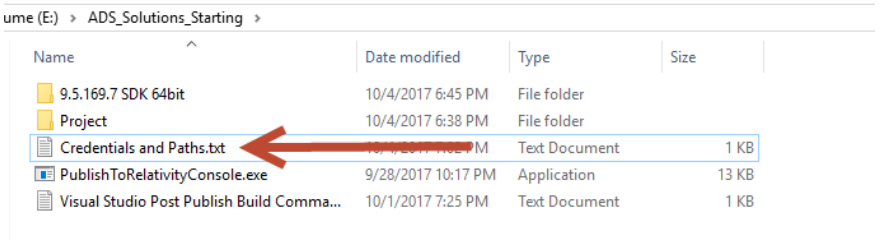


6. Select the **Action** Menu item and then select **Ctrl+Alt+Delete** option.

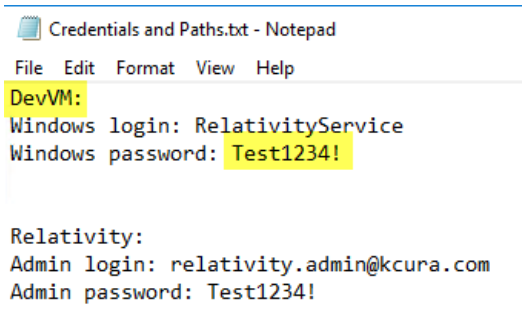




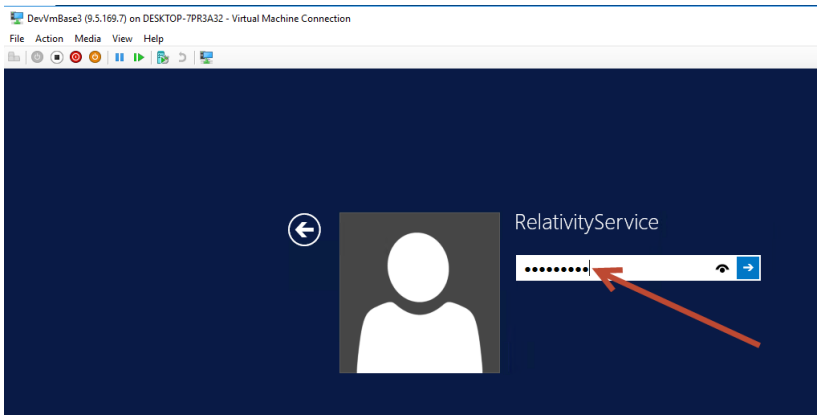
7. Go to the **ADS_Solutions_Starting** folder shortcut on Desktop and open the **Credentials and Paths.txt** file.



8. **Credentials and Paths** file has the credentials for the DevVM, Relativity and SQL.

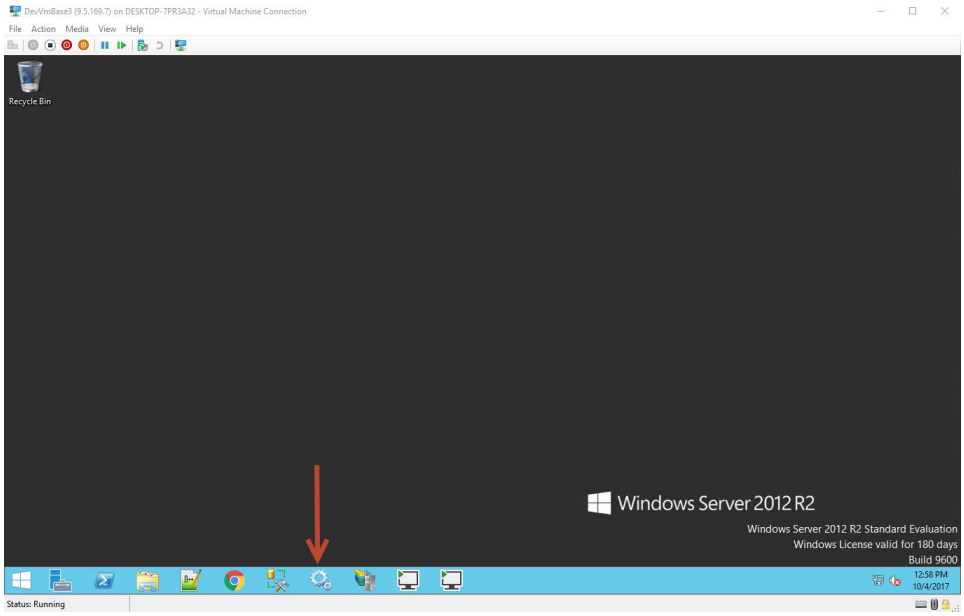


9. Use the DevVM credentials to login to DevVM

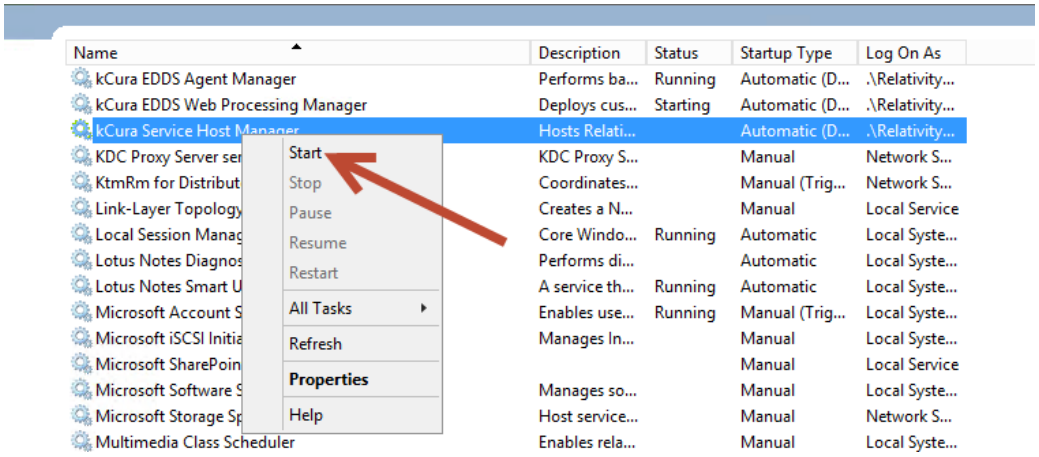


10. Once you log in to the DevVM, you should see the Desktop with few applications pinned to the taskbar, which you will be using throughout the workshop.

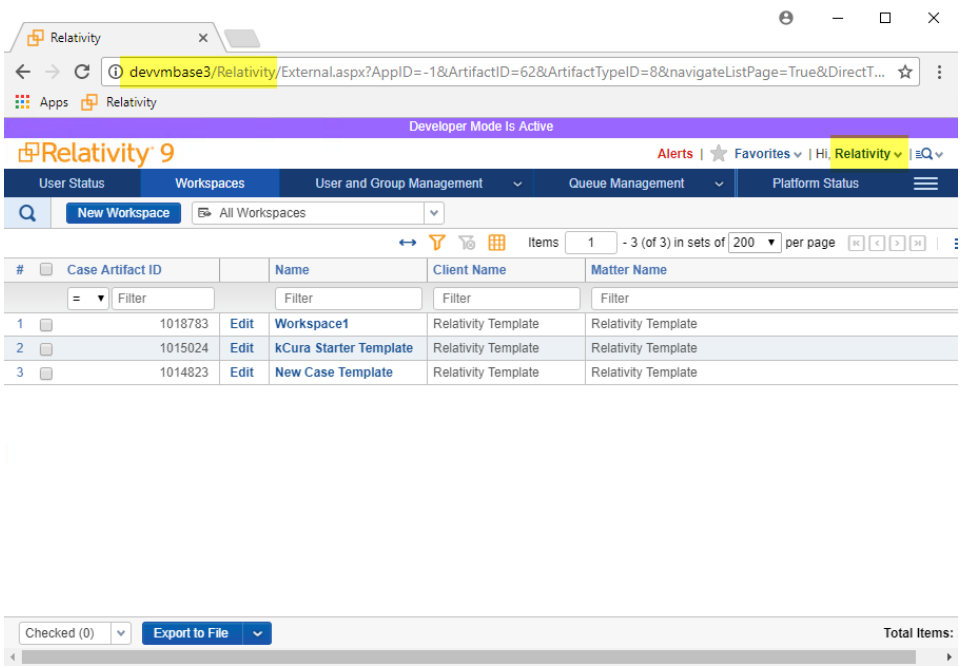
11. Click on the **Services** program icon  in the taskbar as shown in the below screenshot.



12. 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,
    - kCura Service Host Manager
    - kCura EDDS Web Processing Manager
    - kCura EDDS Agent Manager
    - Service Bus VSS
    - Service Bus Resource Provider
    - Service Bus Message Broker
    - Service Bus Gateway



13. Open **Chrome** on your Workshop machine (Not the DevVM) and go to **http://devvmbase3/Relativity**. Use the Relativity admin credentials from the **Credentials and Paths.txt** file to login. Verify that you can login to Relativity.

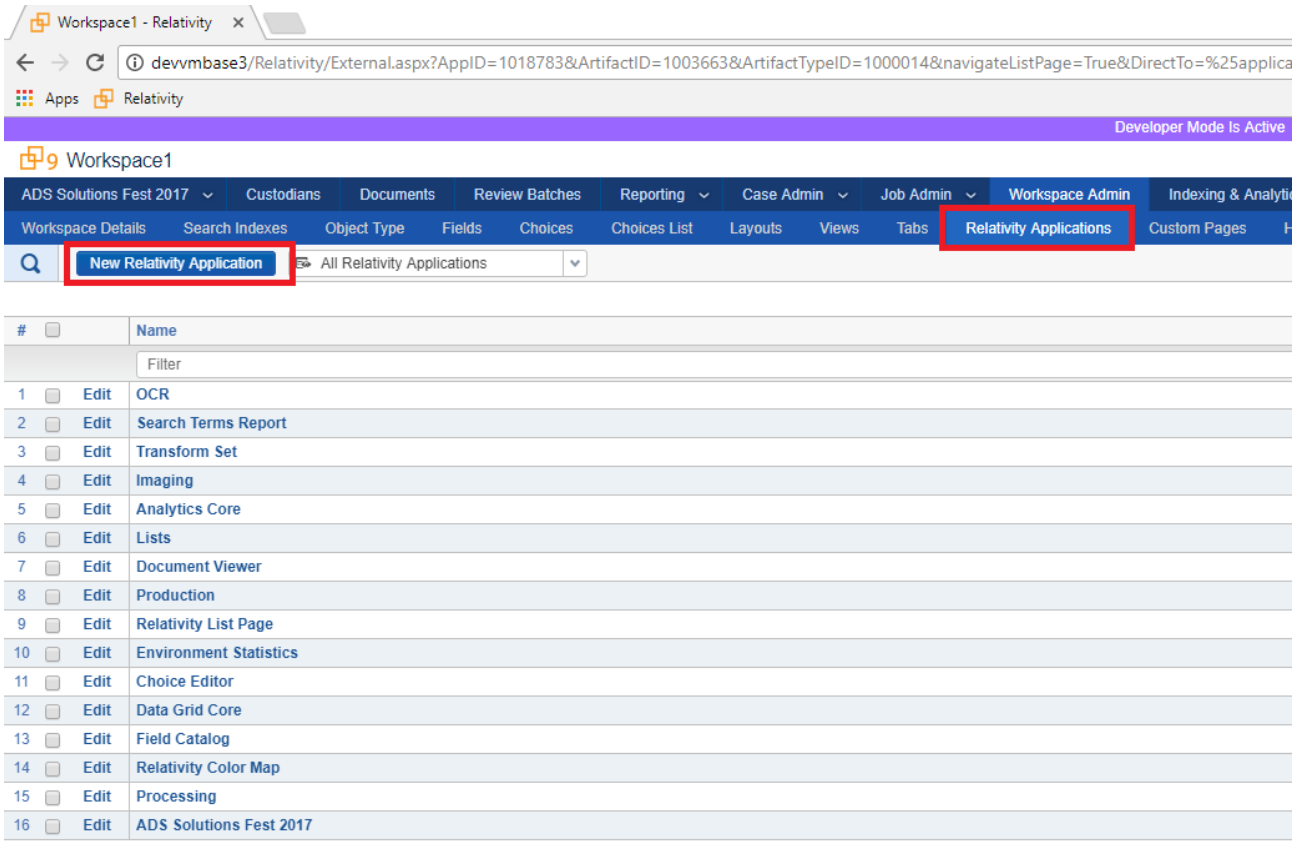


3 Starting Development

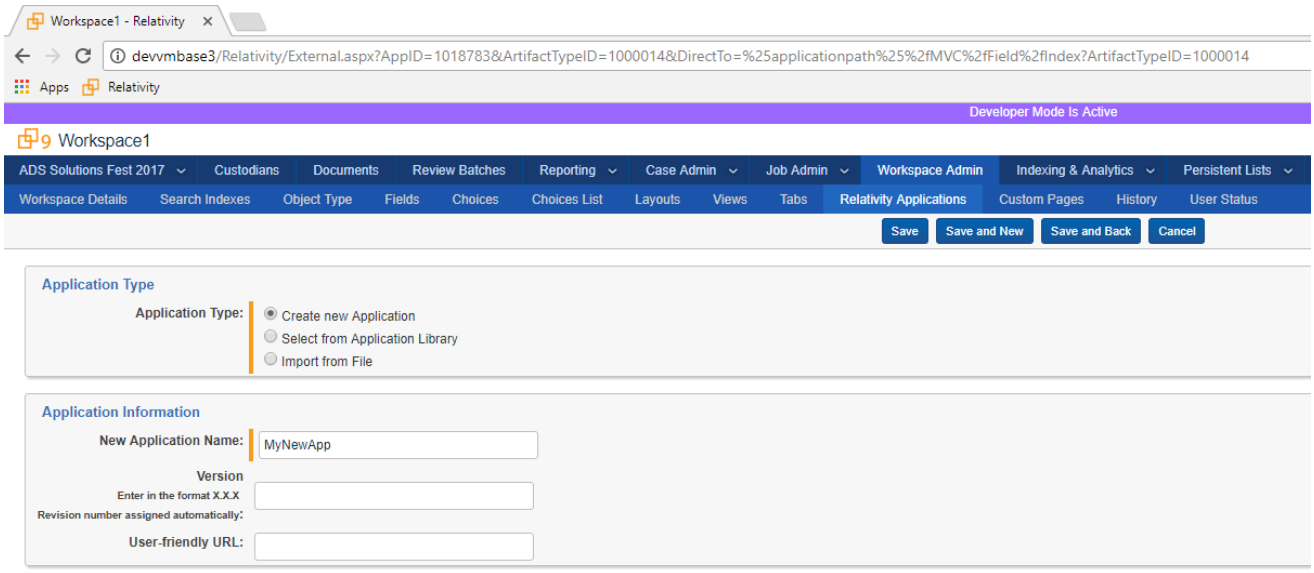
Relativity is an extensible platform that allows an administrator/developer to create customizations. Development on the Relativity Platform is made easier with the use of some tools such as Visual Studio Templates and the Publish to Relativity GUI. This section demonstrates utilizing the visual studio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

3.1 Application Set Up

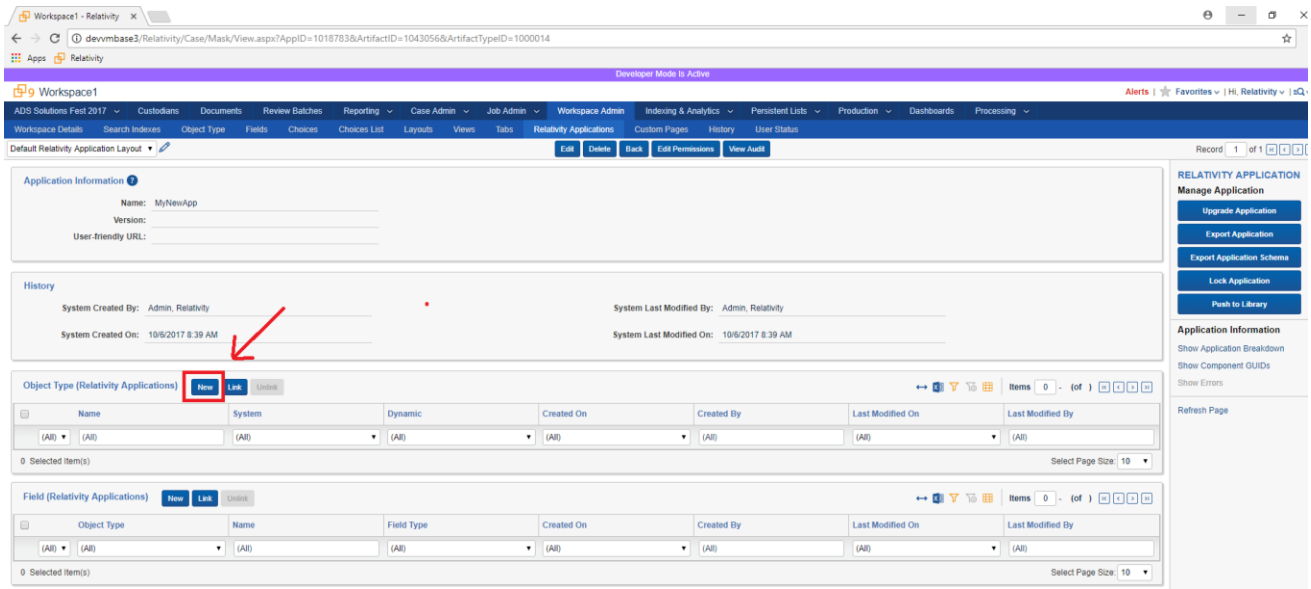
1. Open **Chrome** on your Workshop machine (Not the DevVM) and go to **http://devvmbase3/Relativity**. Use the Relativity admin credentials from the Credentials and Paths.txt file to login.
2. Enter **Workspace 1** and navigate to the **Relativity Applications** Tab.

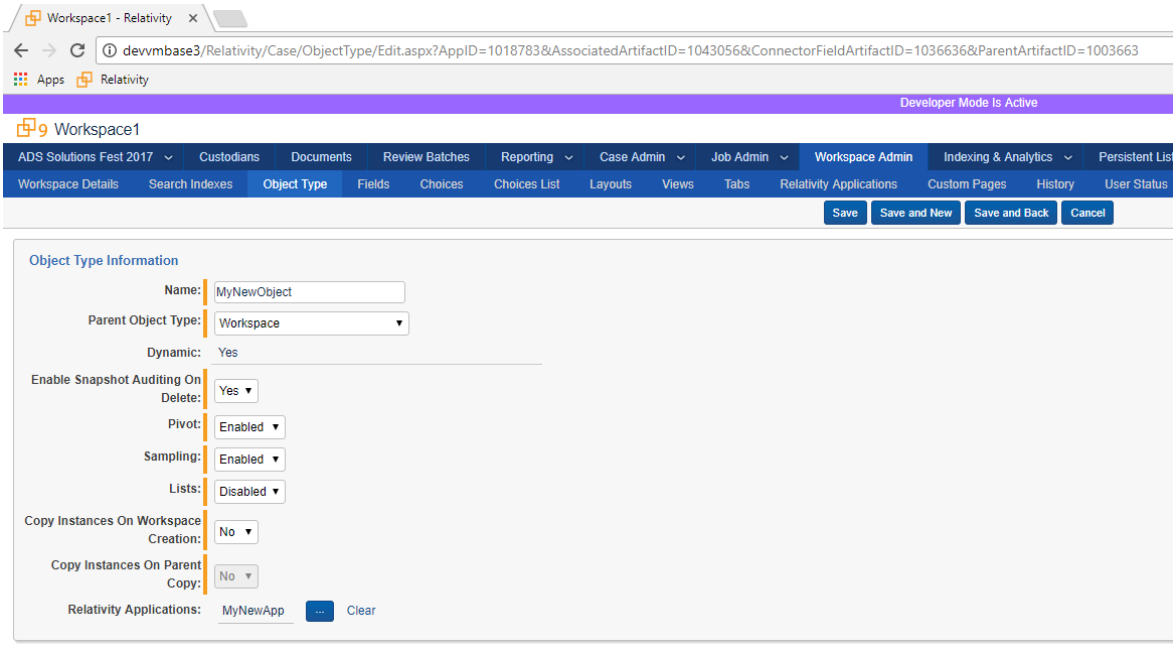


3. Create a new **Relativity Application** with the name of your choice.



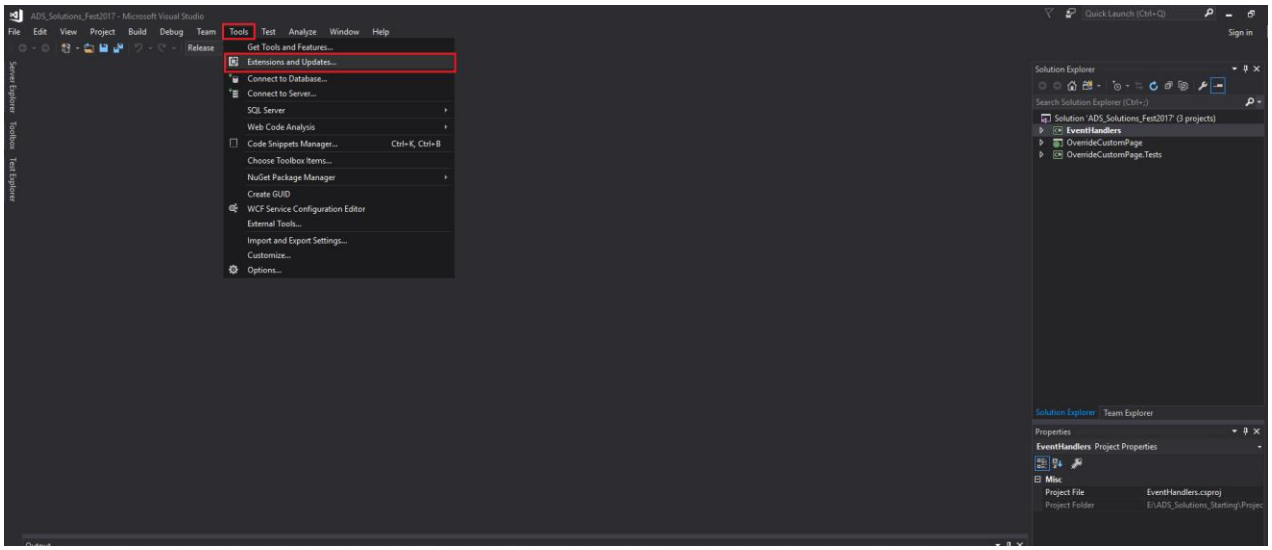
4. Create a new **Object Type** to the application from the view page. This auto-associates the objects initial views, fields, and layouts.

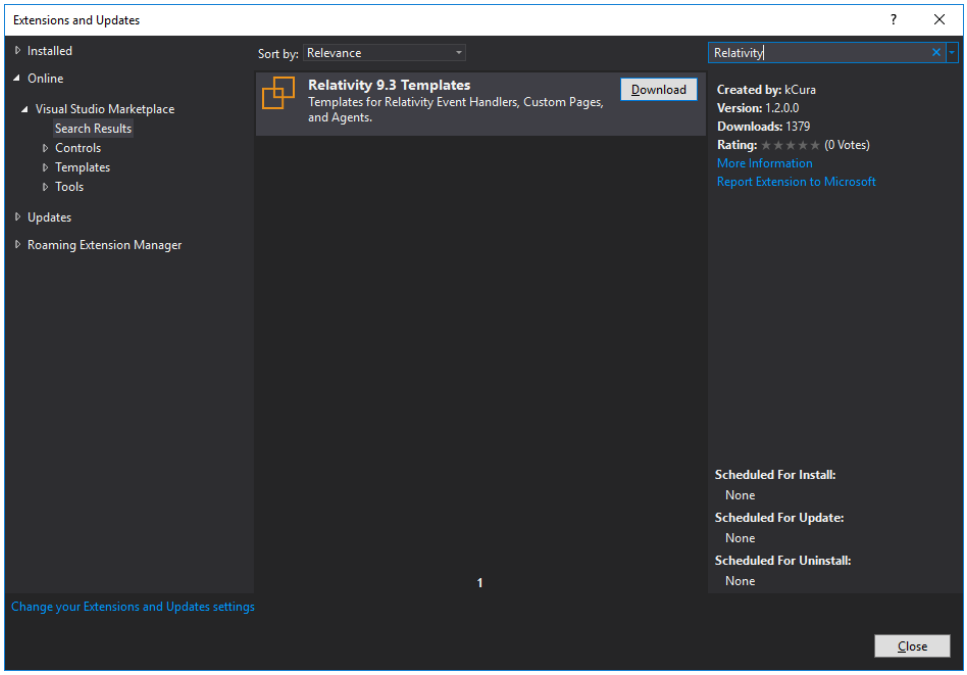




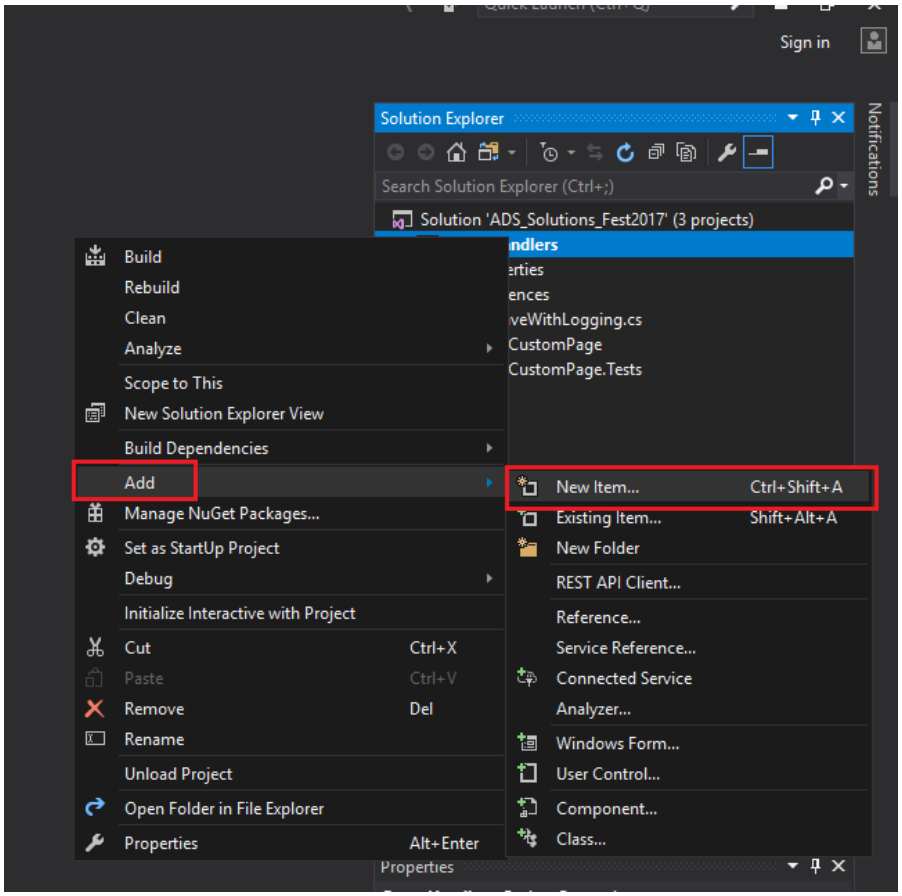
3.2 New Event Handler

1. Open the Visual Studio solution **ADS_Solutions_Fest2017.sln** in the **ADS_Solutions_Starting** folder.
2. Verify the **Relativity Templates** are installed. You may need to **restart** visual studio.
3. Tools — **Extensions and Updates** — Search "**Relativity**"

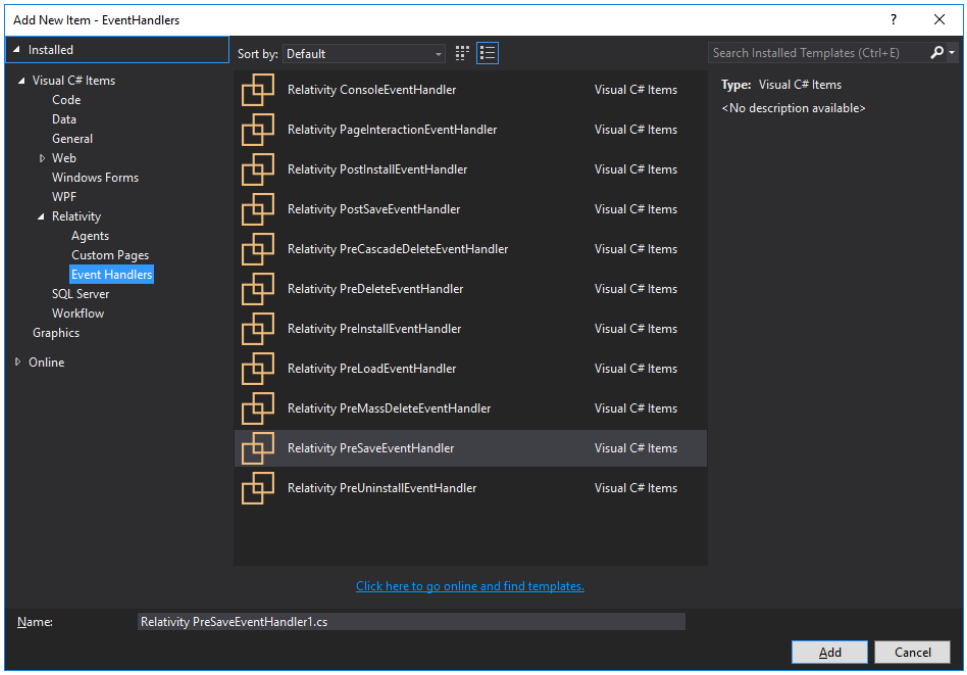




4. Right click the **EventHandlers** Project in the solution explorer. Select Add — **New Item**.



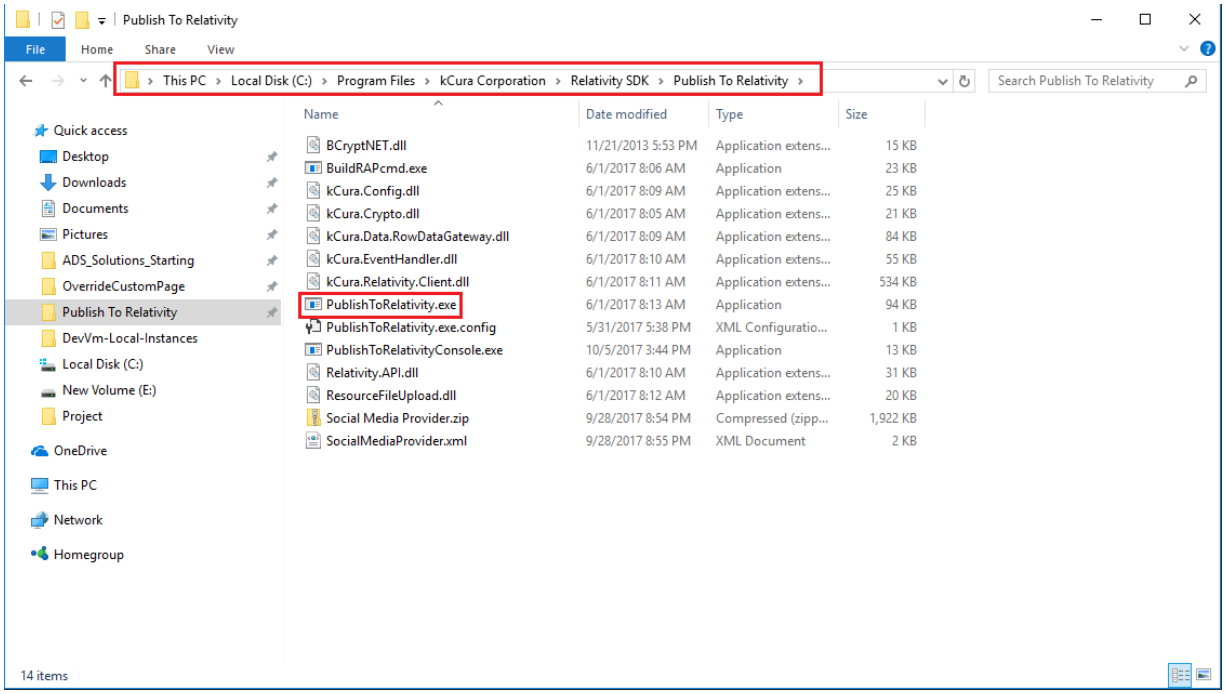
5. Under the Relativity Section, select the **pre-save event handler template.**



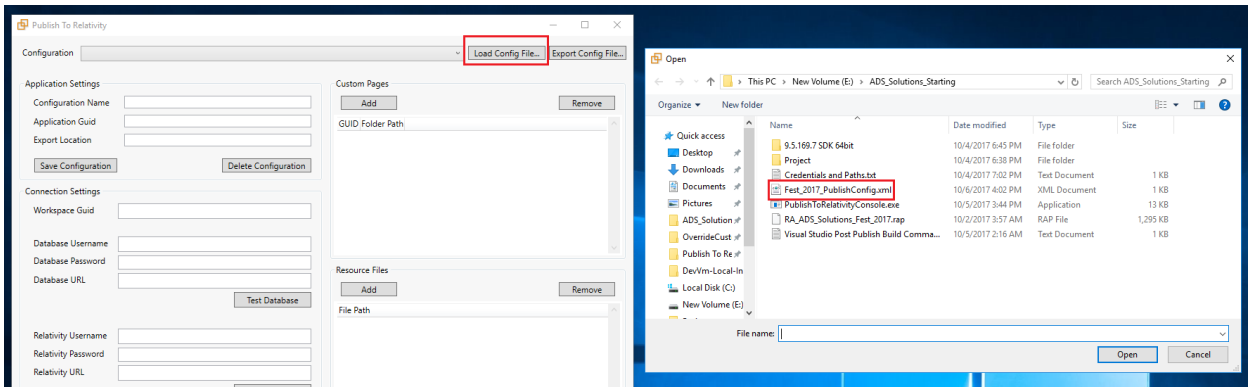
6. Click **Add**.
7. Update the event handler response.
8. Success – false
9. Message – Message of your choice

3.3 Publish

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

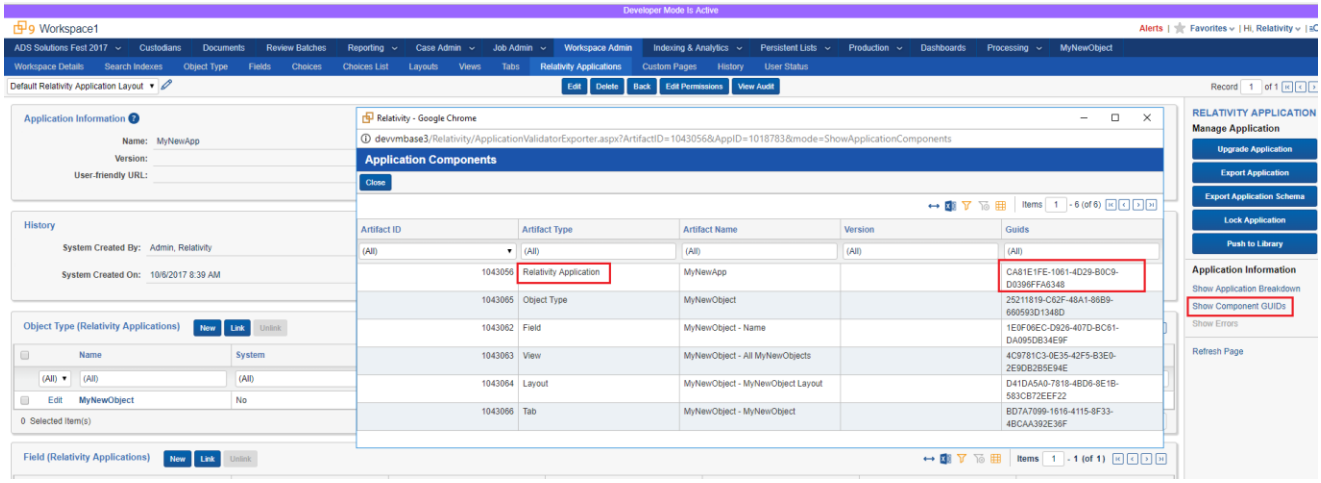


4. Click the **Load Config File** button and select the **Fest_2017_PublishConfig.xml** in the **ADS_Solutions_Starting** folder.

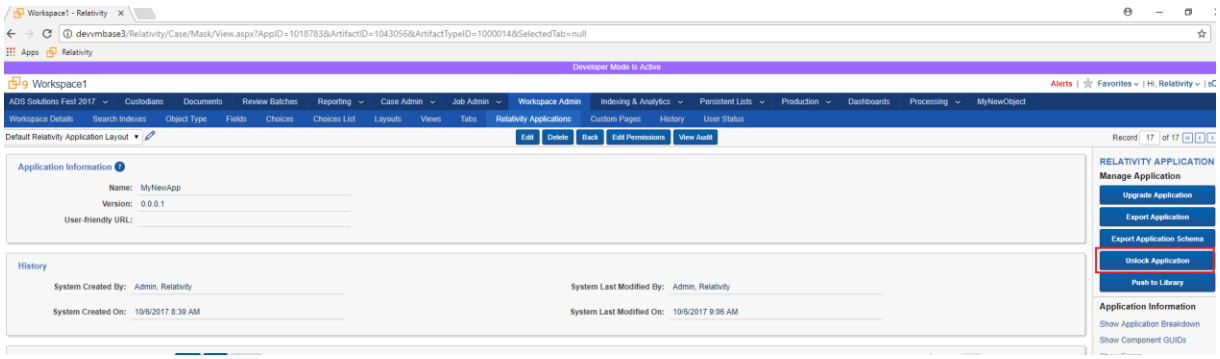


5. Replace the **Application GUID** with your applications GUID.

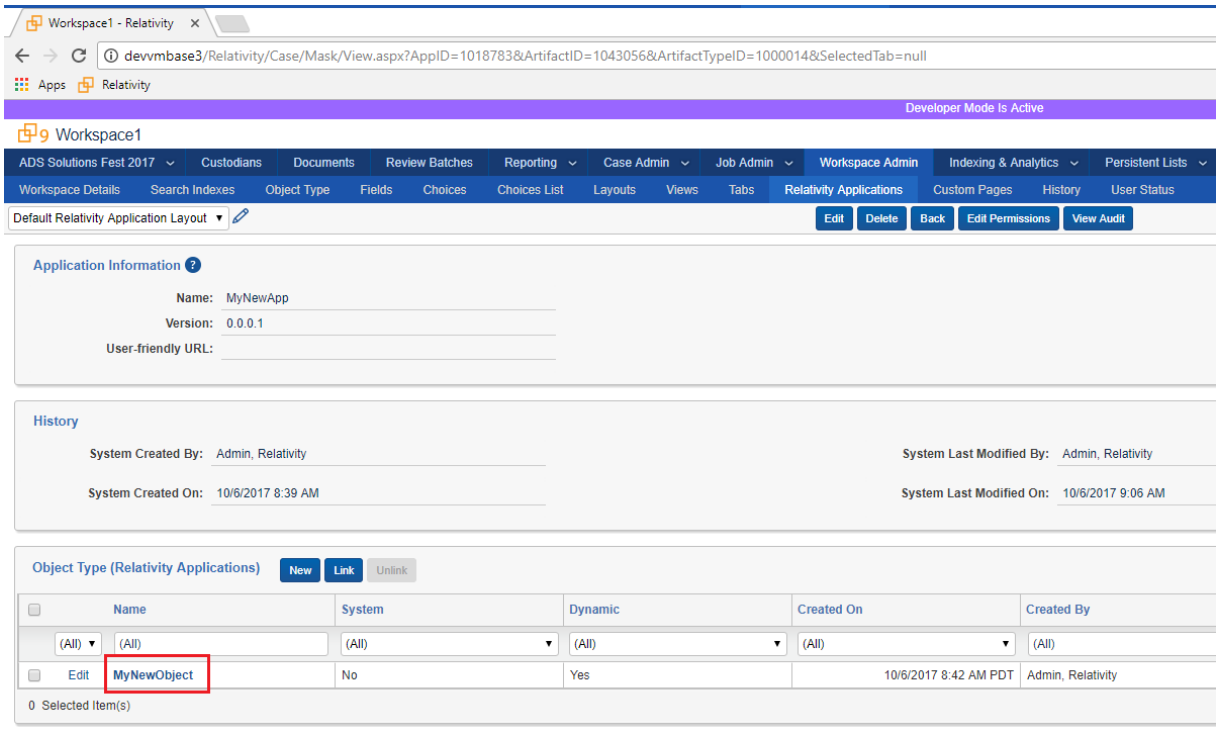
6. To look up your applications GUID, navigate back to Relativity Application you created in the workspace. Then select **Show Component GUIDs**. **Developer Mode** must be active.



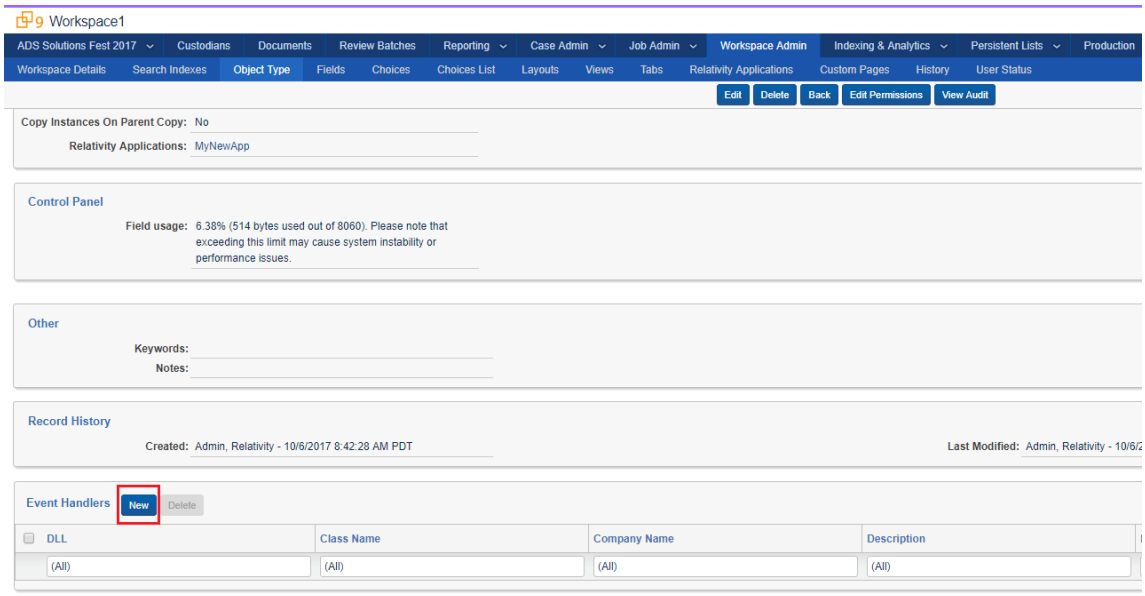
7. In the **Resource File** Section click Add.
8. Navigate to **E:\ADS_Solutions_Starting\Project\ADS_Solutions_Fest2017\bin\Release** and select **ADS_Solutions_Fest2017.dll**
9. Click **Publish**.
10. Open Relativity and navigate to your application created in section 3.1.
11. **Unlock** the application.



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

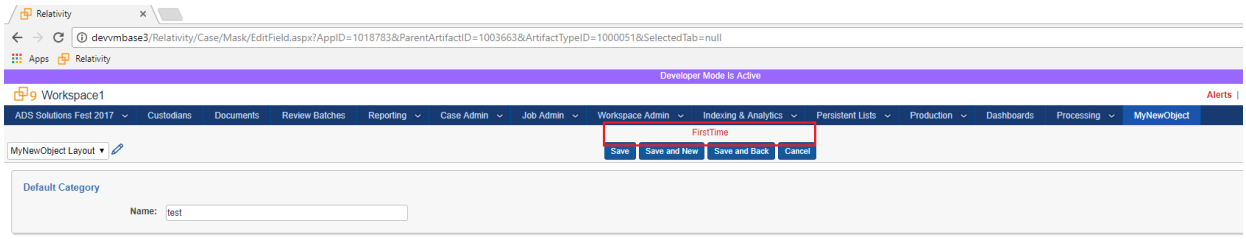


13. Click **New** on the Event handler associated item list.



14. Find the newly created Event Handler associated to your application. Filter by Application Name.

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



4 Logging

Relativity logging enables developers to gather runtime diagnostic information of their application. It can give detailed insight into the system to help troubleshooting application problems. This section covers adding a logging statement to a Relativity extensibility point.

4.1 Adding Logging

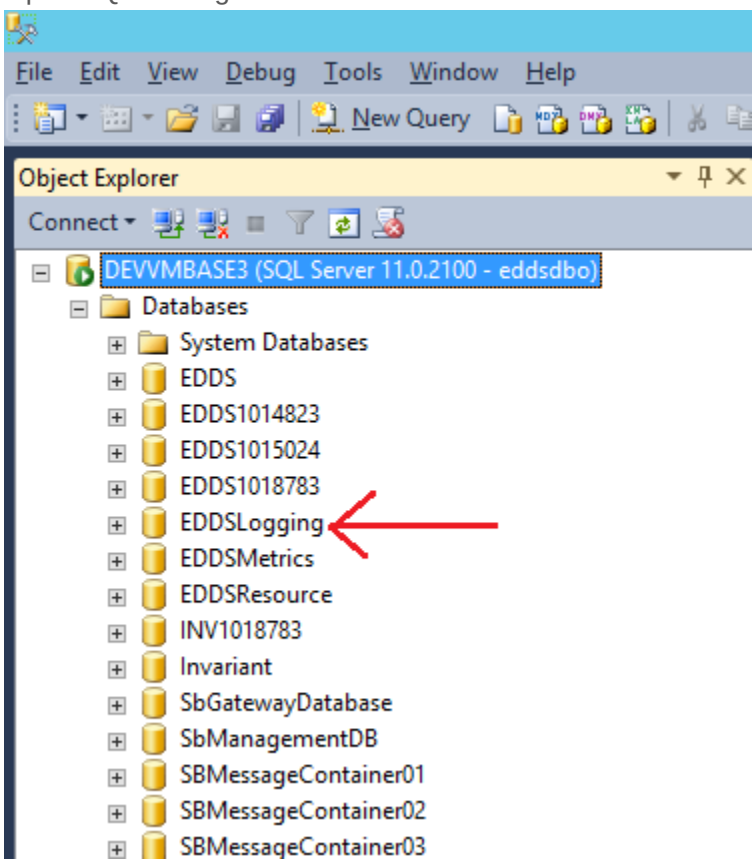
1. Open the visual studio solution ADS_Solutions_Fest2017 and open the EventHandlers project.
2. Create a new pre-save event handler using a Visual Studio Template, or the event handler created in section 3.
3. Update the Execute function to do the following (see screenshot for sample):
4. Throws a new exception with a message.
5. Handles the thrown exception with a try/catch.
6. Add a log statement during exception handling. Log the exception and a helpful message.

```
public override kCura.EventHandler.Response Execute()
{
    Response retVal = new kCura.EventHandler.Response();
    try
    {
        Helper.GetLoggerFactory().GetLogger().LogDebug("About to try something!");
        throw new System.Exception("Boom");
    }
    catch (System.Exception ex)
    {
        Helper.GetLoggerFactory().GetLogger().LogError("That definitly did not work", ex);
        retVal.Success = false;
        retVal.Message = ex.ToString();
    }

    return retVal;
}
```

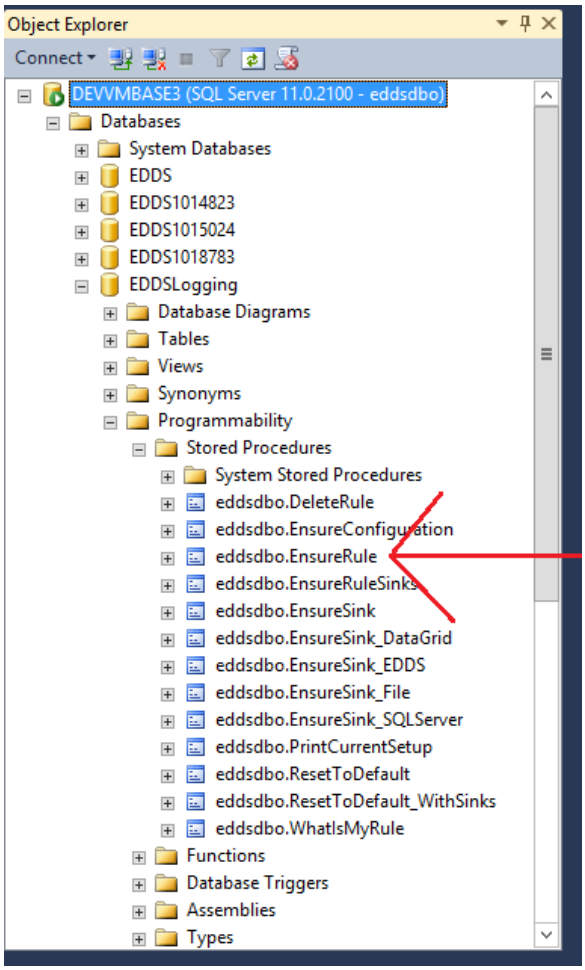
7. Build the project.
8. Click Publish in the Publish to Relativity Tool.

4.2 Viewing Logs

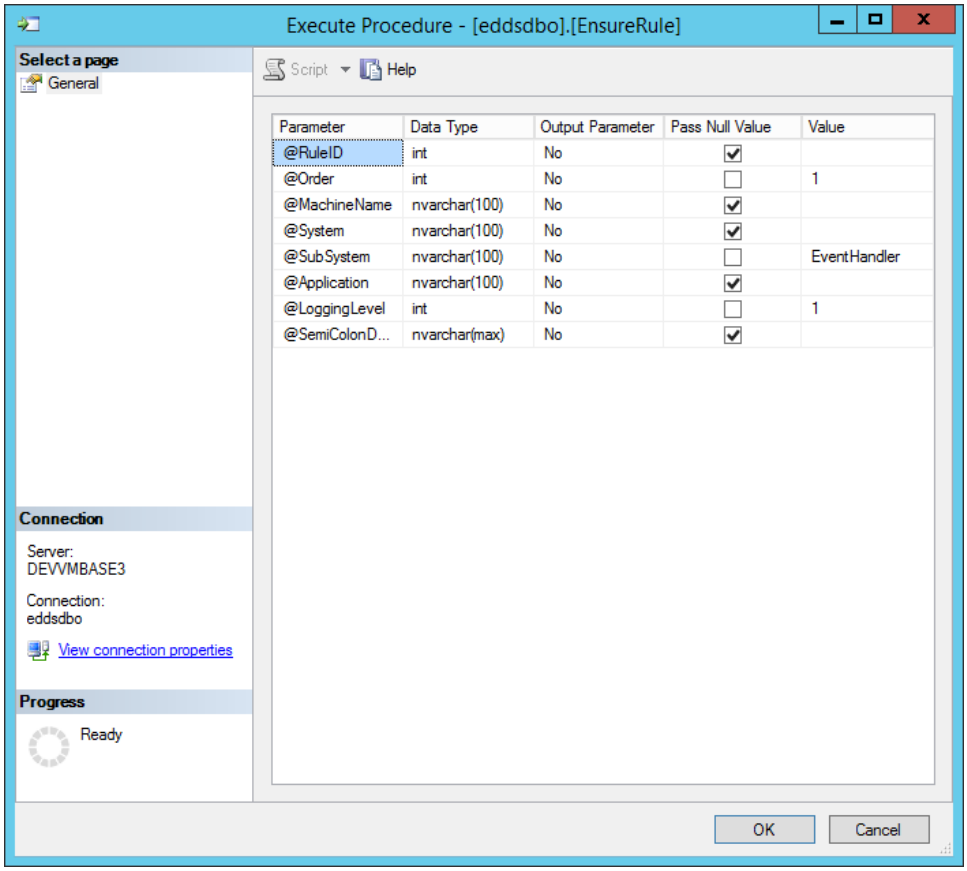
1. Log into the Relativity 9.5 DevVM.
2. Open SQL Management Studio.



3. Right click the EnsureRule stored procedure and execute.



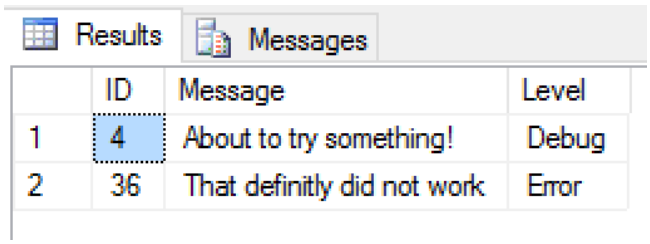
4. Enter the following values. This sets a debug log level for all event handlers.
    a. Order – 1
    b. SubSystem – EventHandler
    c. LoggingLevel – 1



5. Click OK.
6. Execute your event handler with logging.
7. Run the following command:

    SELECT TOP 10 * FROM [EDDSLogging].[eddsdbo].[RelativityLogs] WHERE Properties.value('(/properties/property[@key="SubSystem"])[1]','varchar(max)') = 'EventHandler'

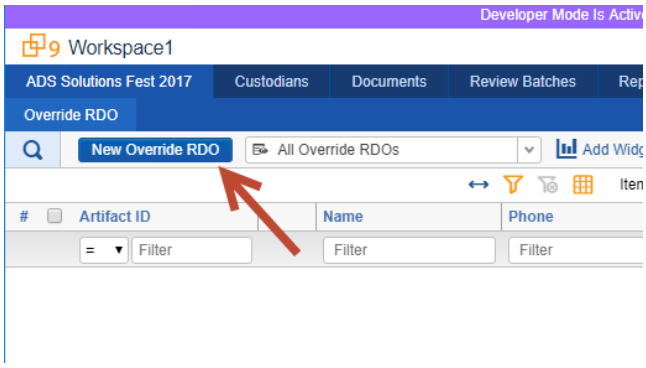
8. If there are a lot of logs in the system, this command can take some time to execute. It can be faster to query against Message Template if that is the case.
9. Verify the logs from your event handler appear.



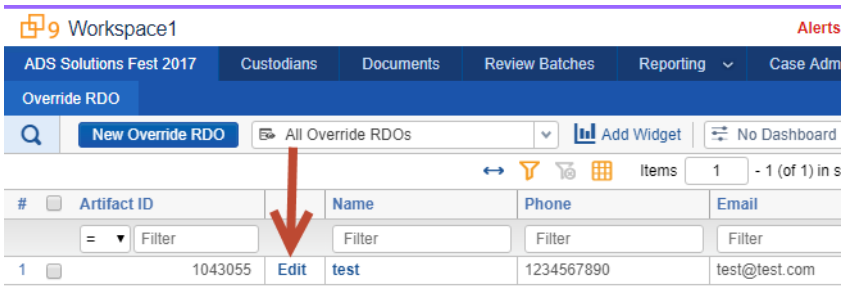
5 Customize New Button Override & Override Edit Link URL Object Rules

5.1 Custom page

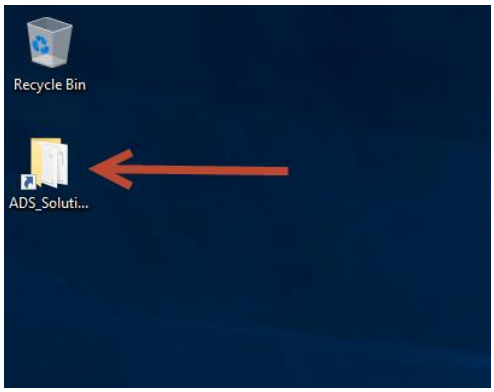
1. In this section we will be working with a Relativity application (**ADS Solutions Fest 2017**) which consists of a custom RDO named **Override RDO**. Our goal for this session is to override the **New Button Link** and **Edit Link** for the custom RDO with a custom page implementation.
2. New Button Link

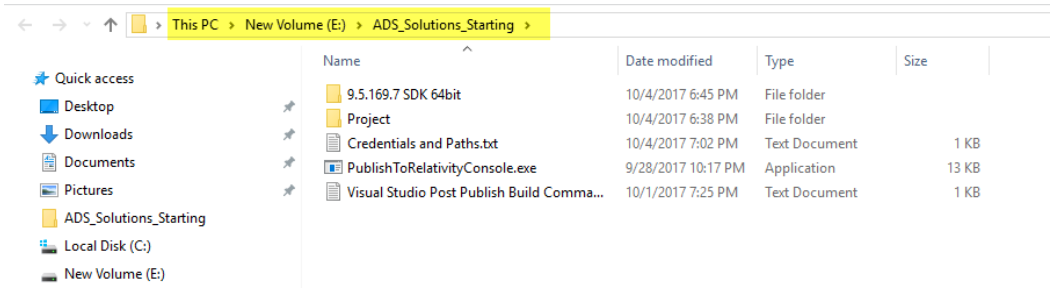


3. Edit Link

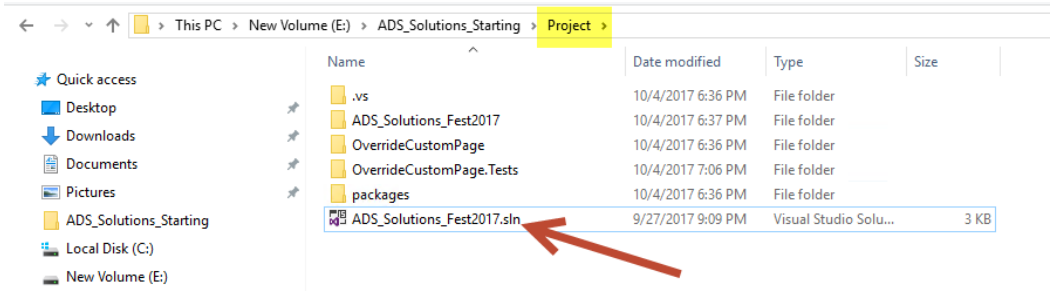


4. Go to Desktop and double click on the **ADS_Solutions_Starting** folder to open it.

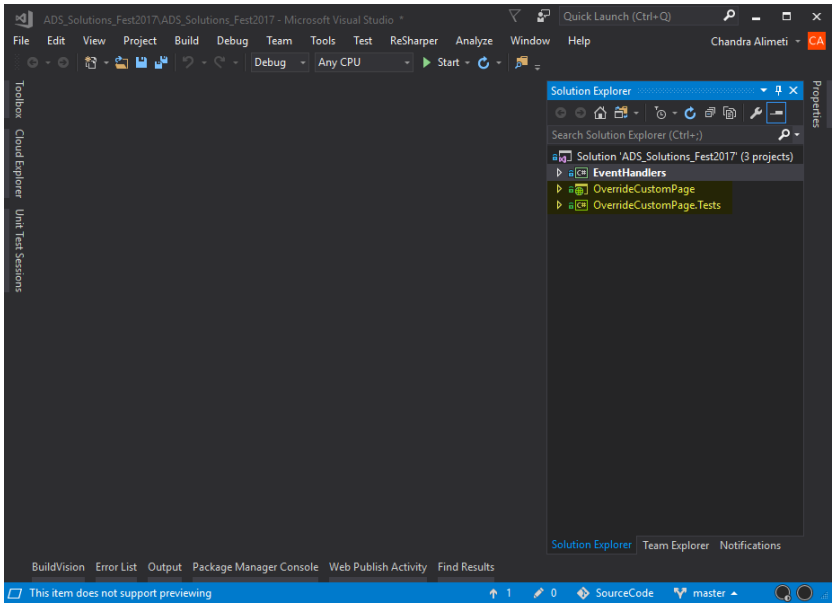




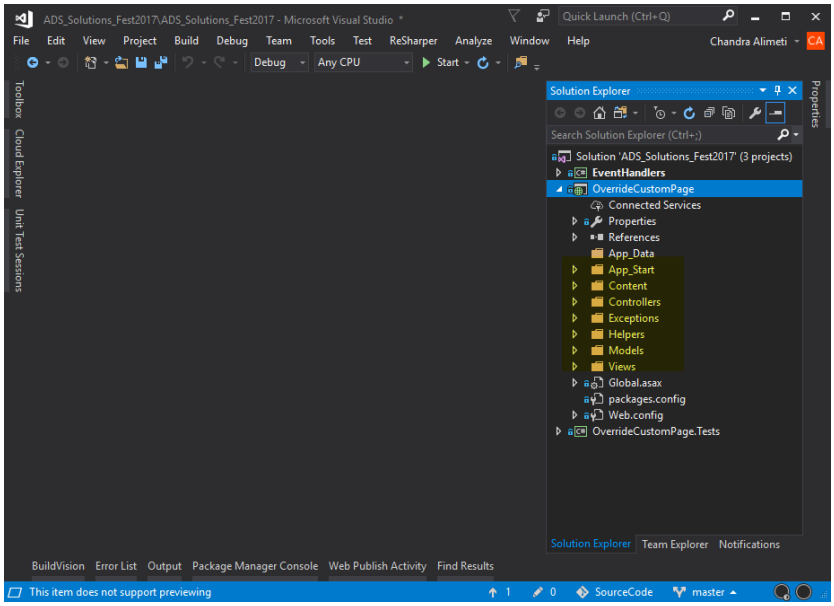
5. Go to the **Project** folder and double click the **ADS_Solutions_Fest2017.sln** file to open the Visual Studio solution.



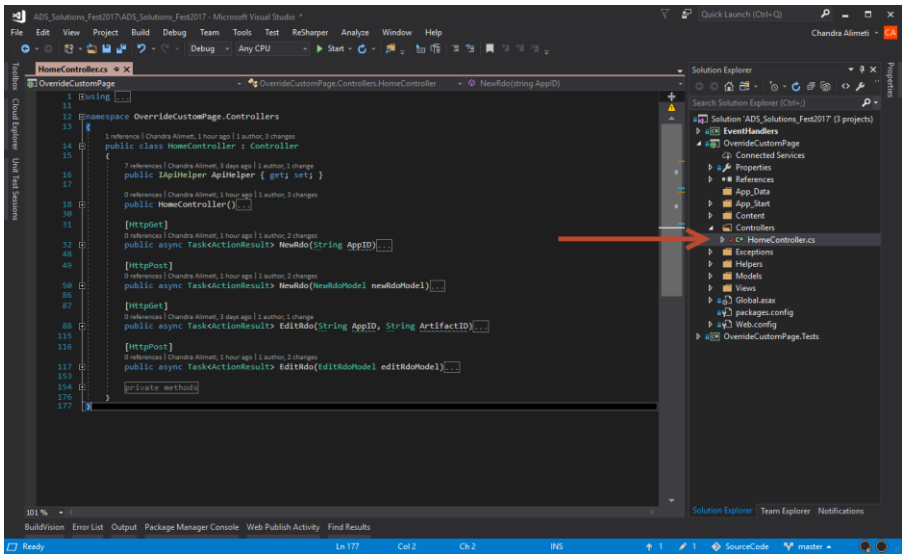
6. **Visual Studio 2017** will be opened and shows you the list of the projects contained in the solution. Make sure you see the following two projects.
    - **OverrideCustomPage**
    - **OverrideCustomPage.Tests**



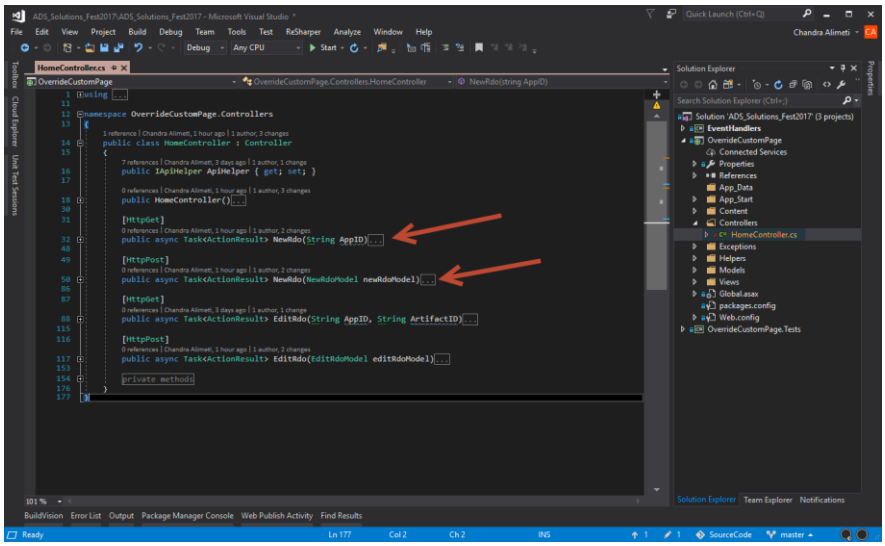
7. **OverrideCustomPage** is an ASP.NET MVC 5 project. Double click the project to expand it.



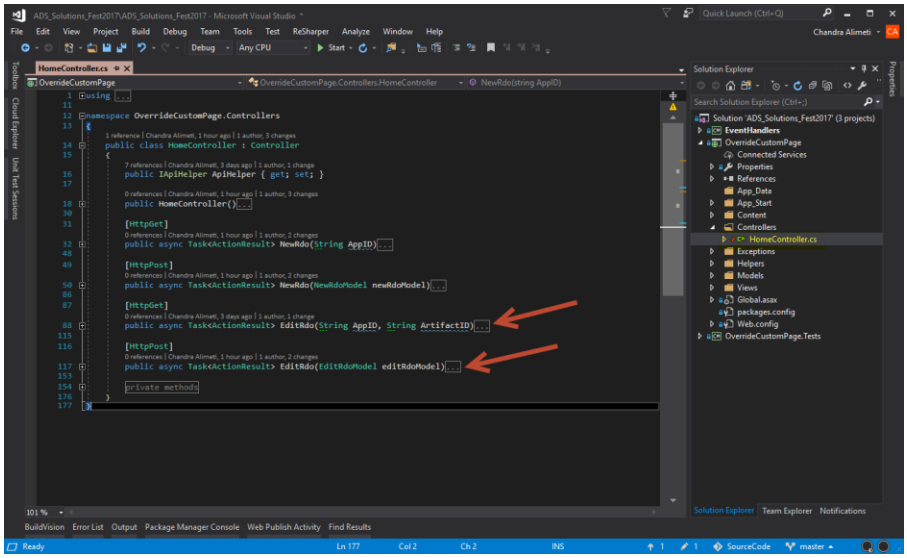
8. Expand the **Controllers** folder and double click on the **HomeController.cs** file.



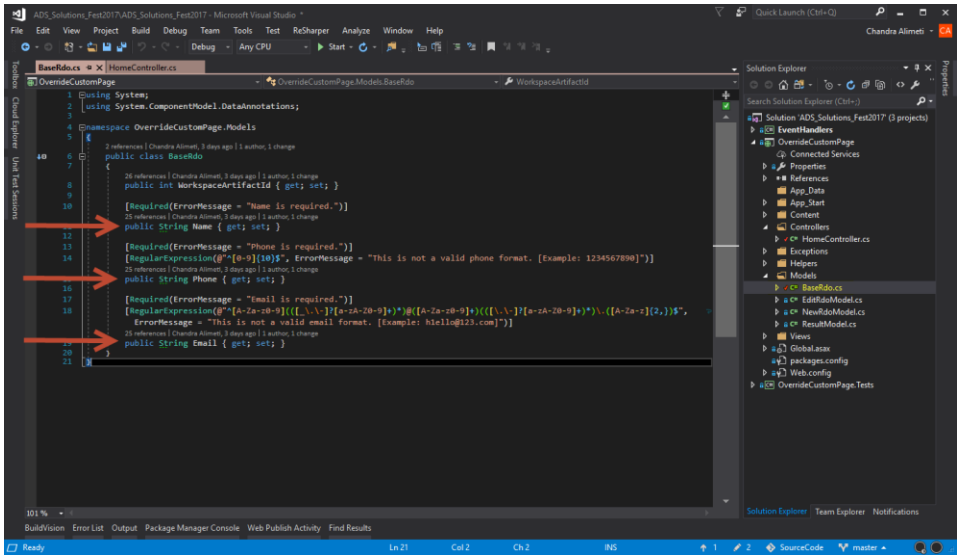
9. The first two methods with the name **NewRdo** are web pages which contain a form to create a new RDO record.



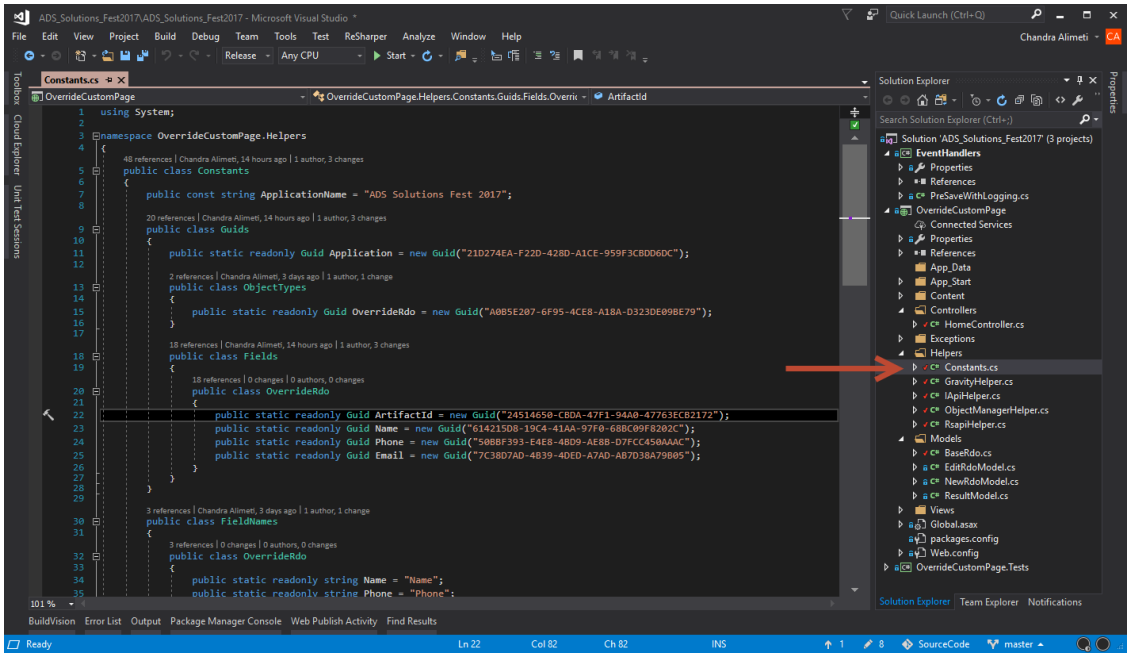
10. The next two methods with the name **EditRdo** are web pages which contains a form to edit a RDO record.



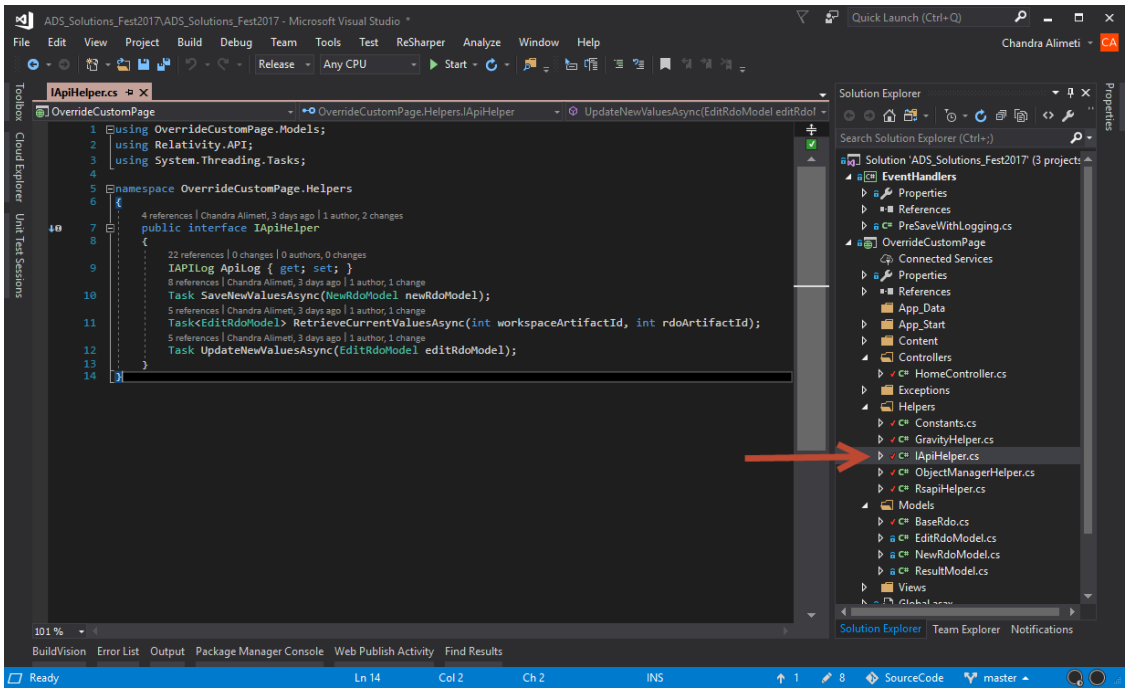
11. Next expand the **Models** folder and double click on the **BaseRdo.cs** file. This file contains three properties **Name**, **Phone** and **Email**.



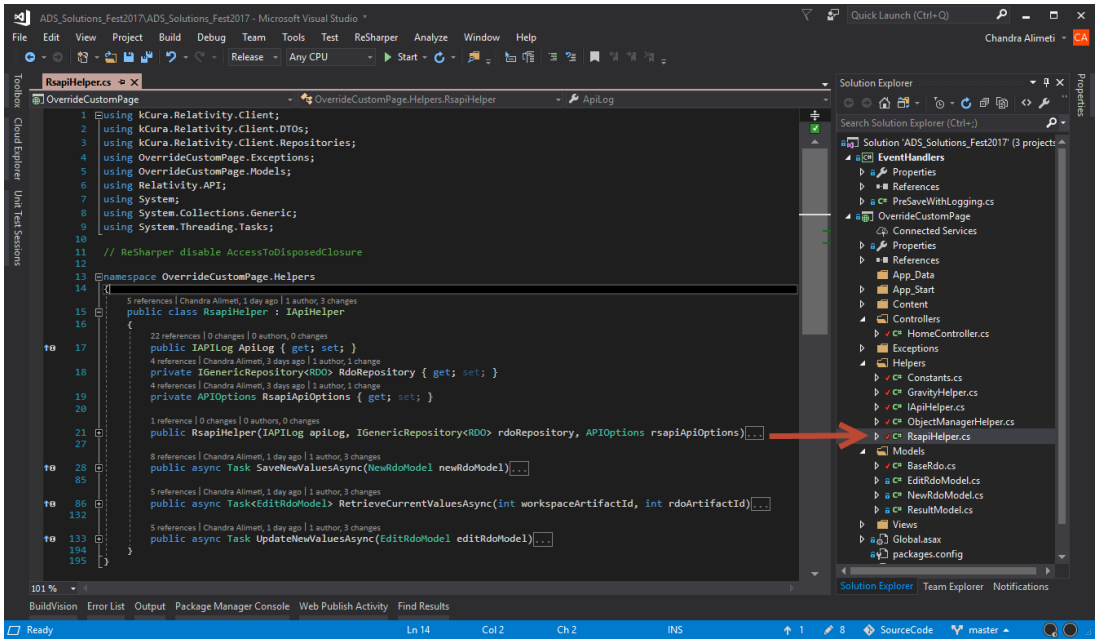
12. Next expand the **Helpers** folder and double click on the **Constants.cs** file. This class contains the GUID's of your application and any other static shared data for your application.



13. Next double click on the **IApiHelper.cs** file to open it. This interface contains the methods to be implemented by the API to read and make any updates to the Override RDO.



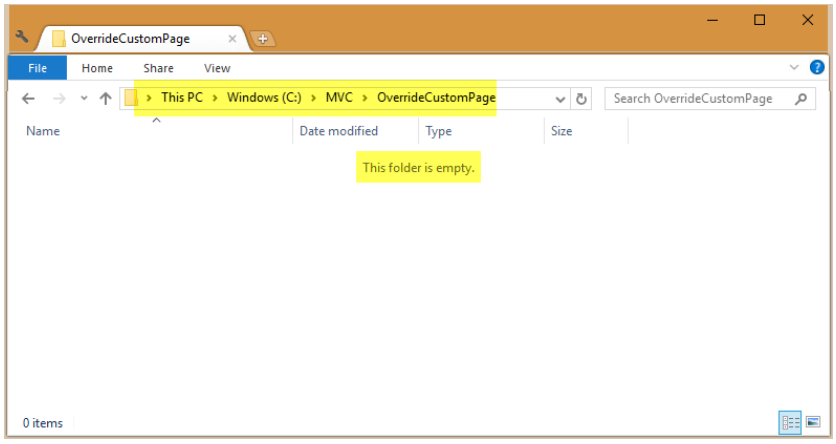
14. Next, double click on the **RsapiHelper.cs** file to open it. This class is the implementation for the IApiHelper interface using the RSAPI.



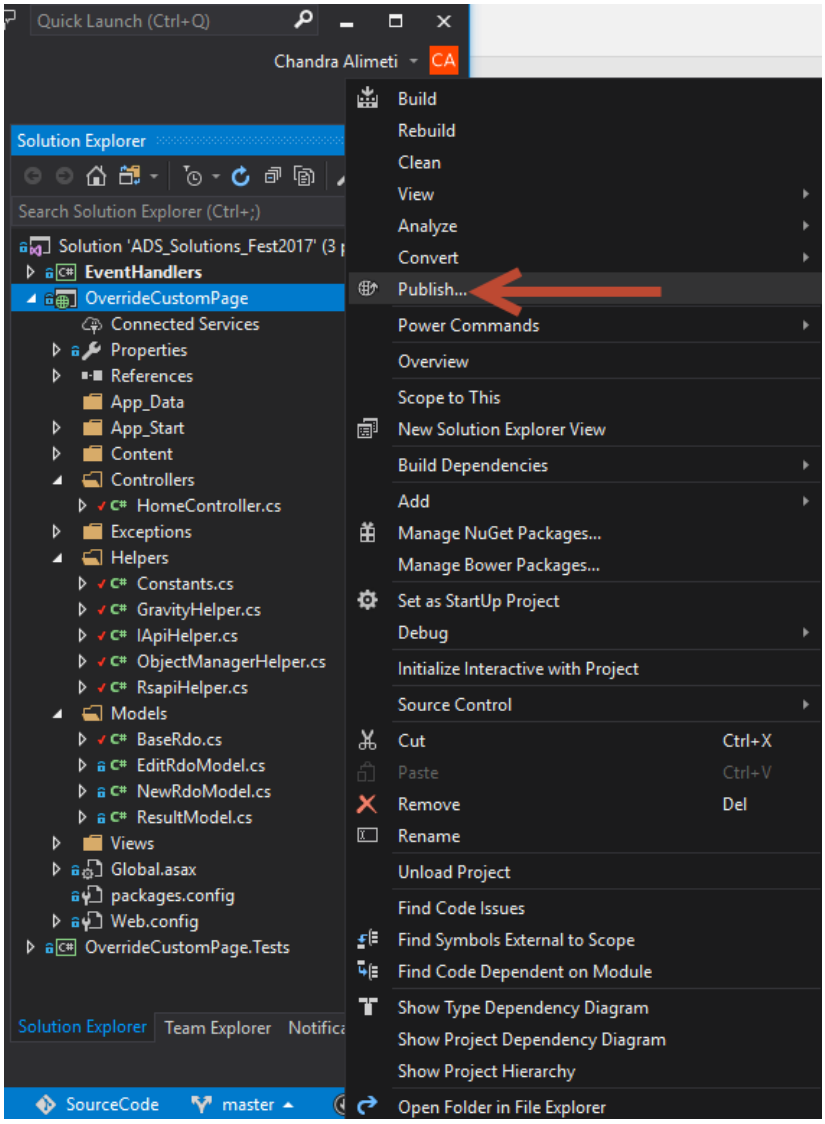
15. In the HomeController.cs file we create a new instance of the RsapiHelper class.

```
public HomeController()
{
    IAPILog apiLog = ConnectionHelper.Helper().GetLoggerFactory().GetLogger();
    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);
}
```

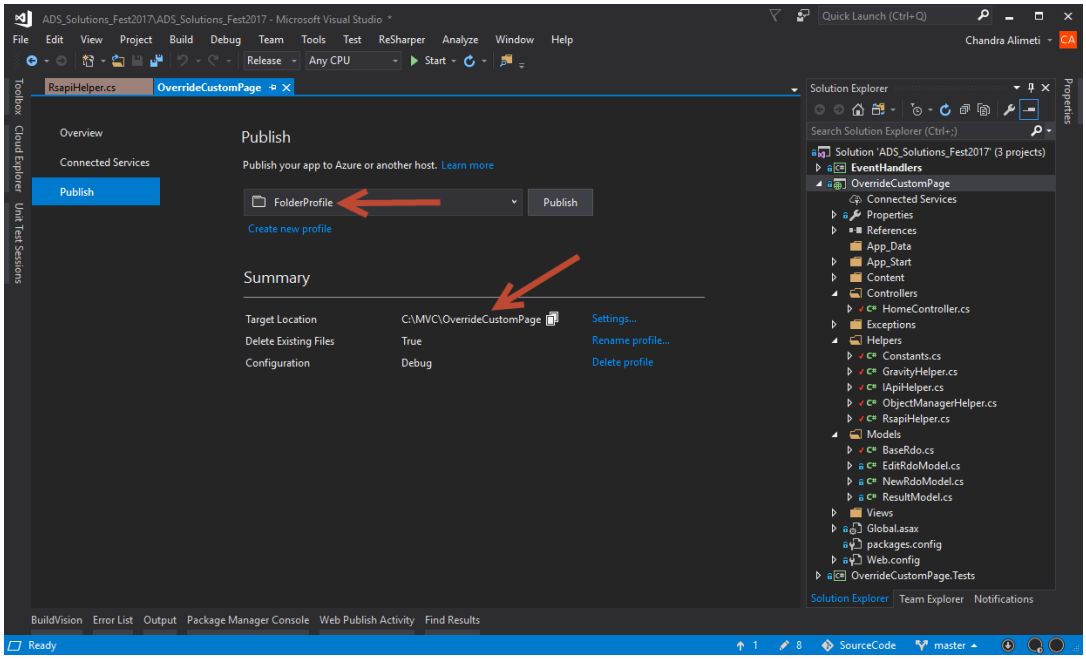
16. Next, we will publish the custom page to a **OverrideCustomPage** folder so that we could zip it up and attach it to our Relativity application. Open the **File Explorer**; make sure **C:\MVC\OverrideCustomPage** folder exists and it's empty.



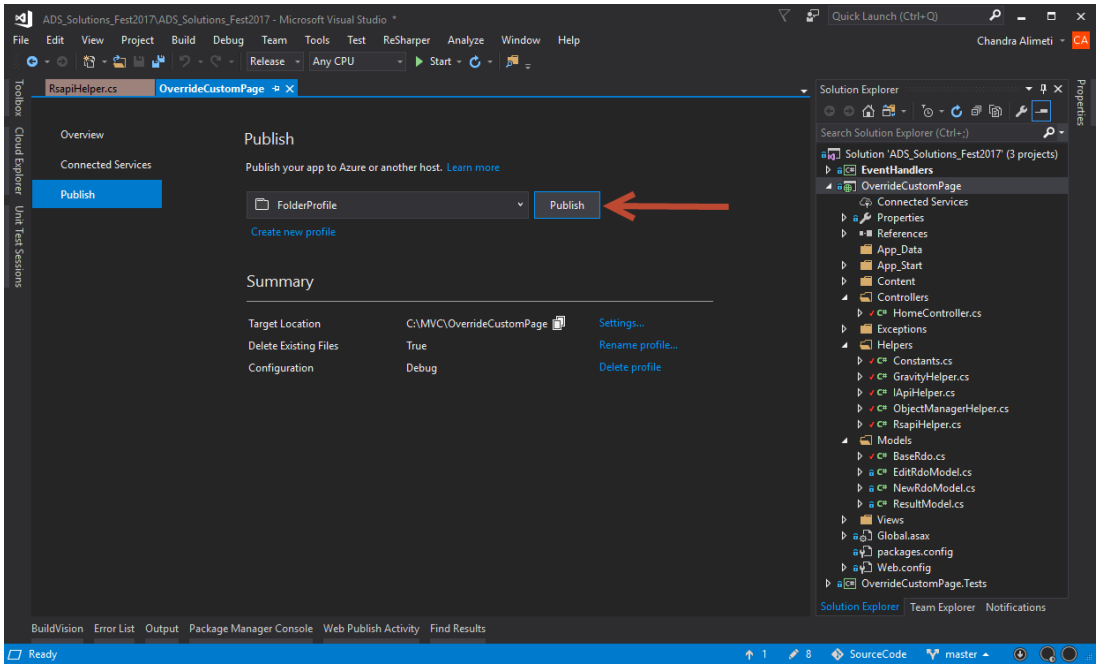
17. Right click on the **OverrideCustomPage** project and click on the **Publish** option.



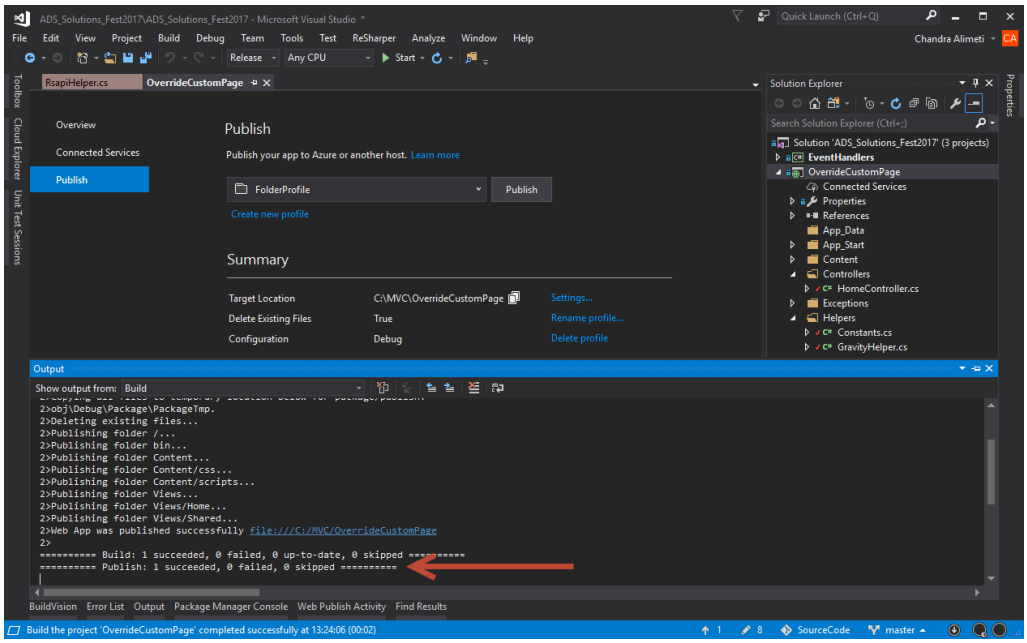
18. You should see a **Publish** settings tab opened. It already contains a **FolderProfile** which is set to publish the custom page to **C:\MVC\OverrideCustomPage** folder.



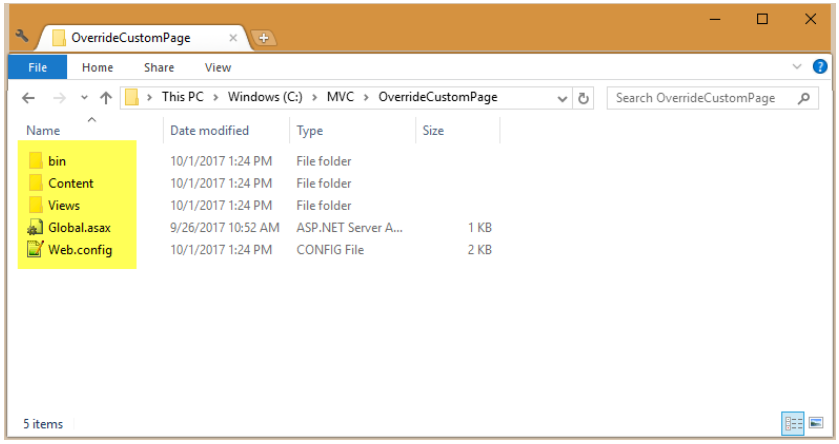
19. Click on the **Publish** button.



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



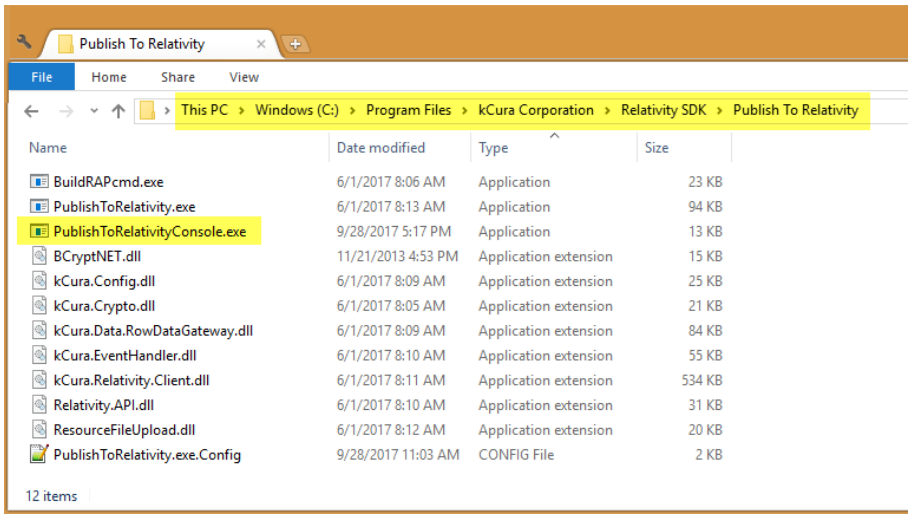
21. Now go to the **C:\MVC\OverrideCustomPage** folder to verify the files are copied as shown in the following screenshot.



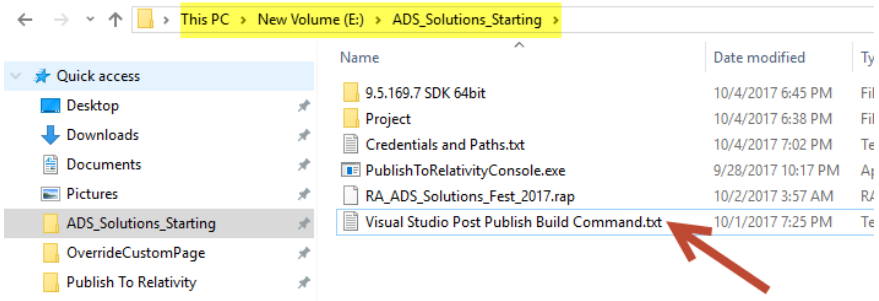
Now we have a custom page published and ready to be attached to our Relativity application.

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

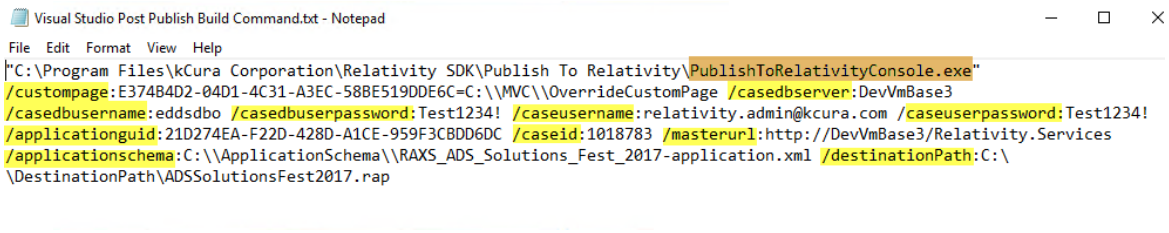
1. Instead of using the Publish to Relativity GUI to publish the custom page, we will use a command line tool called **PublishToRelativityConsole** I have created to publish the custom page. The **PublishToRelativityConsole** tool is already copied to the **Publish To Relativity** folder on your workshop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



2. **PublishToRelativityConsole** is an open source project you can find on Github at the link - **https://github.com/relativitydev/publish-to-relativity-console**
3. 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**
4. Go to the **E:\ADS_Solutions_Starting** folder and make sure you see the **Visual Studio Post Publish Build Command** text file.



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

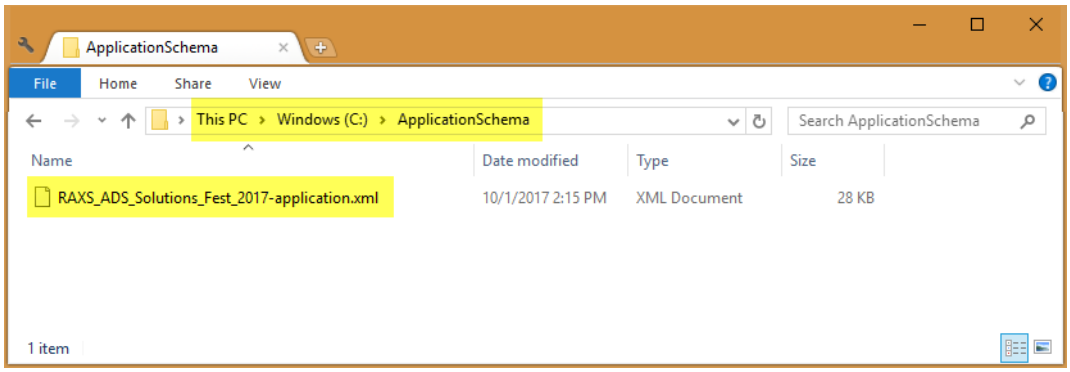


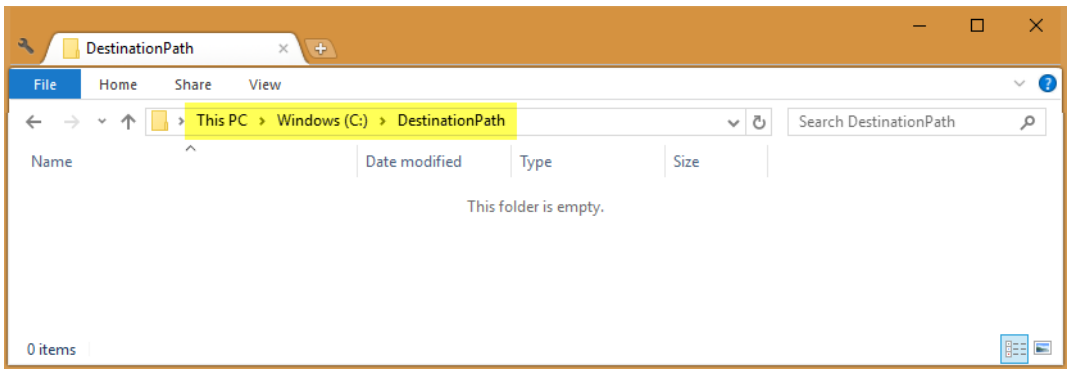
6. The post publish build command contains various **arguments** for the **PublishToRelativityConsole** tool and you can find the list below.

Console Arguments:

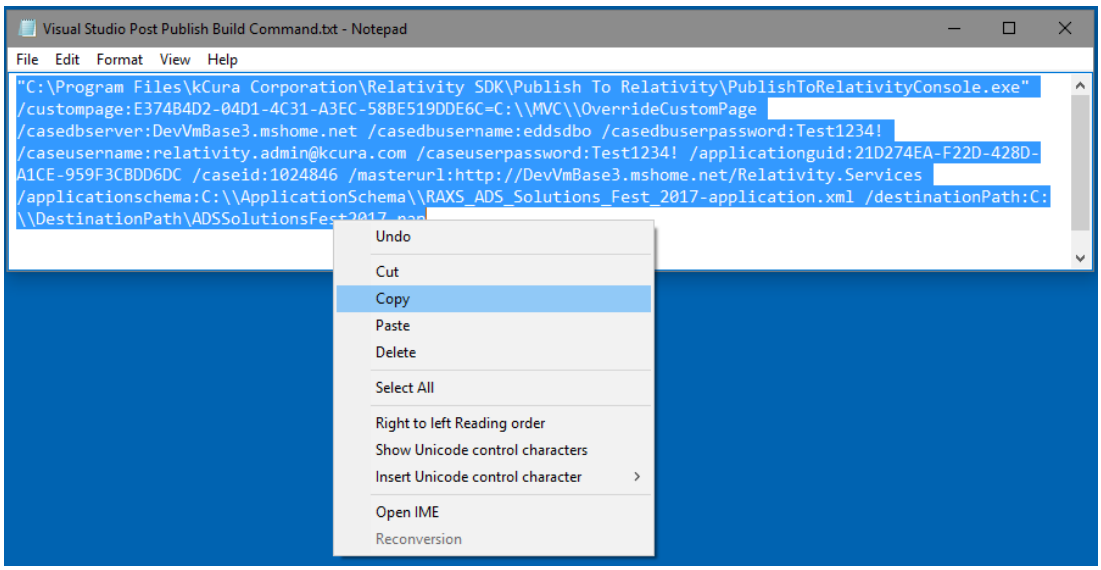
Argument	Description
"C:\Program Files\kCura Corporation\Relativity SDK\Publish To Relativity\PublishToRelativityConsole.exe"	Path to the PublishToRelativityConsole tool
/custompage: E374B4D2-04D1-4C31-A3EC-58BE519DDE6C=C:\\MVC\\OverrideCustomPage	Custom page GUID and path
/casedbserver:DevVmBase3.mshome.net	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

7. 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.

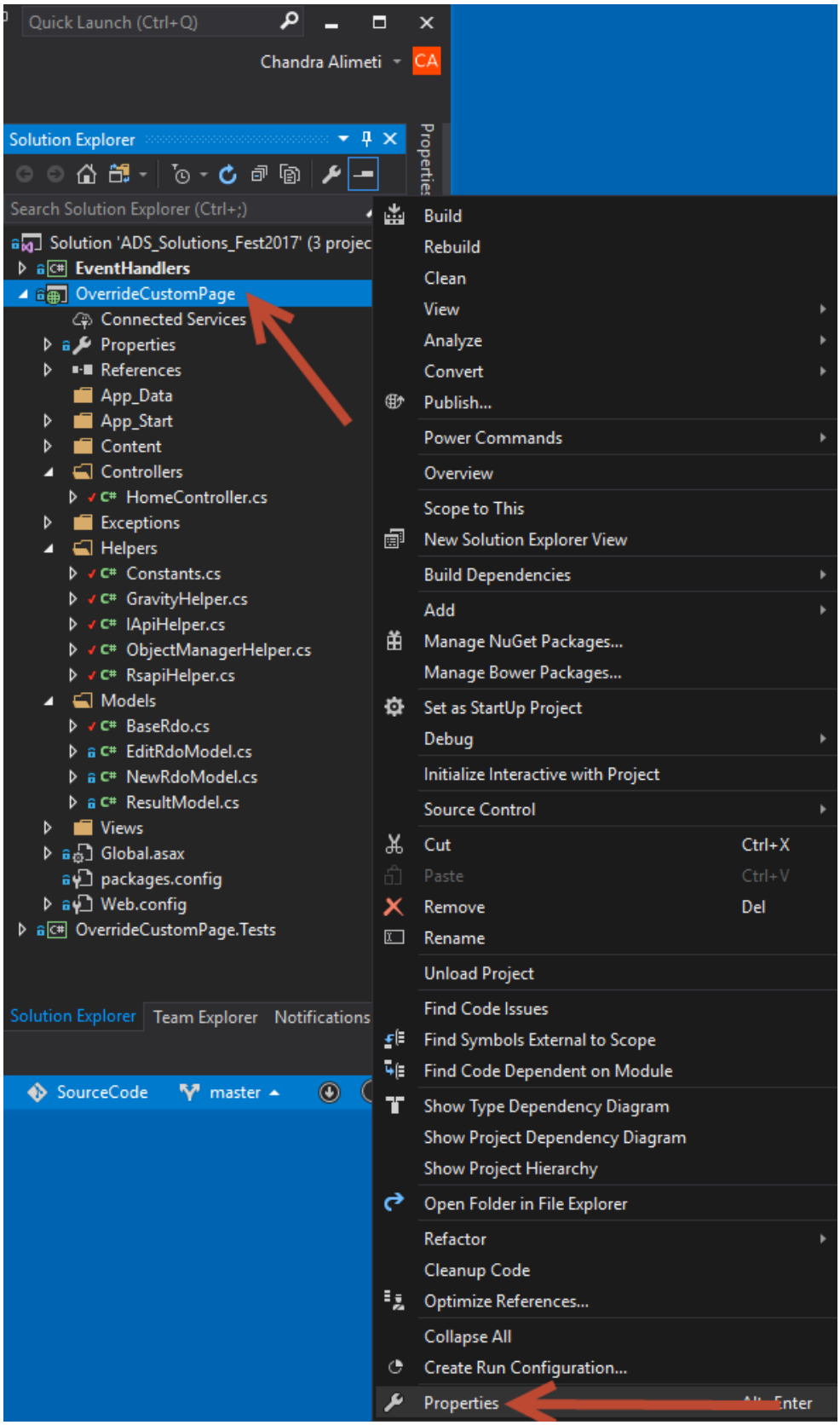




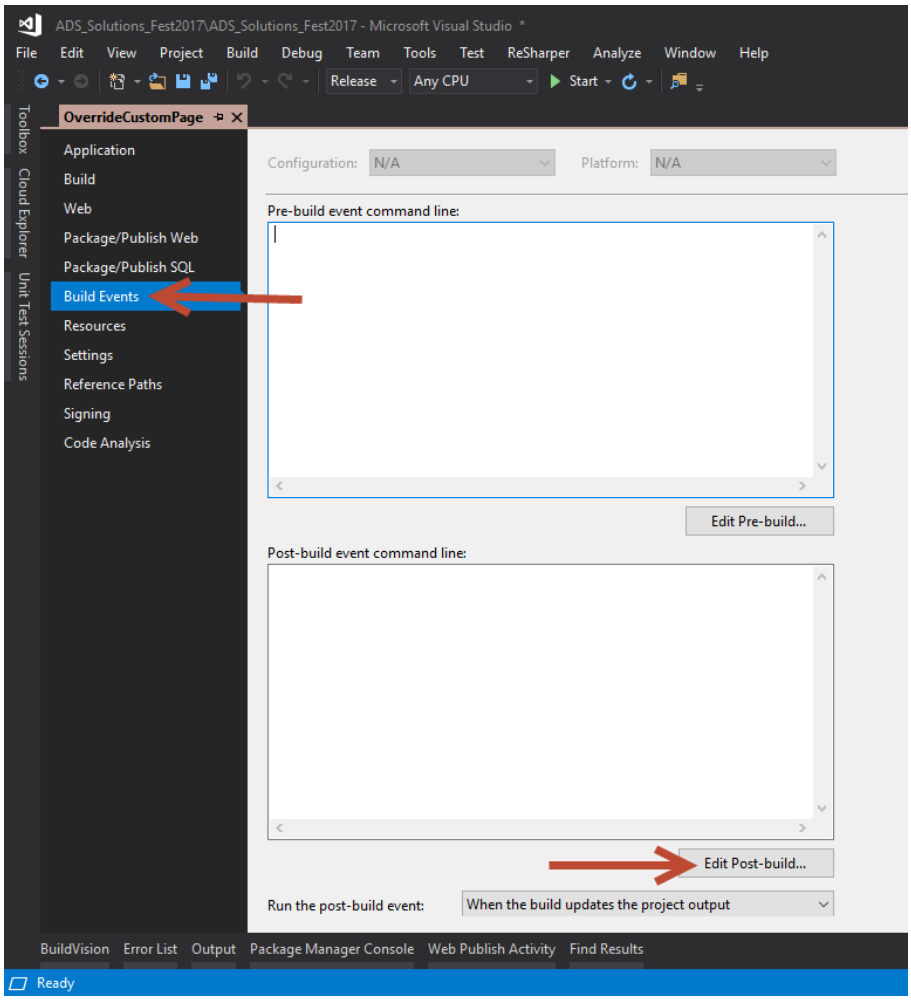
8. Copy the text for the post build publish event.



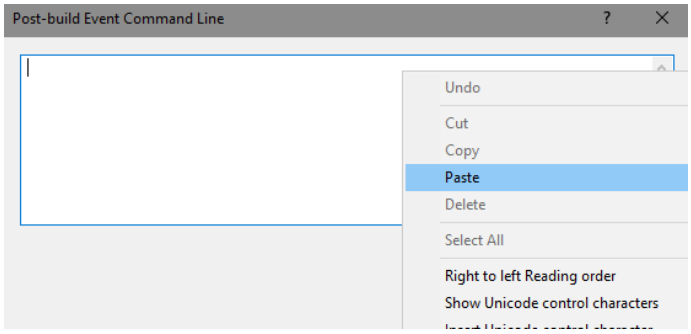
9. Right click on the **OverrideCustomPage** project and select **Properties** option.



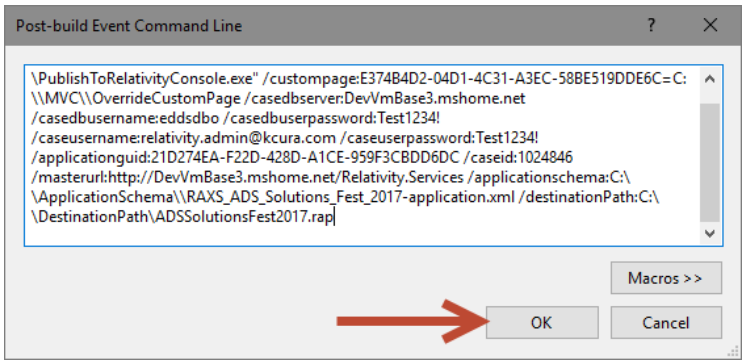
10. Select the **Build Events** option and click on the **Edit Post-build…** button.



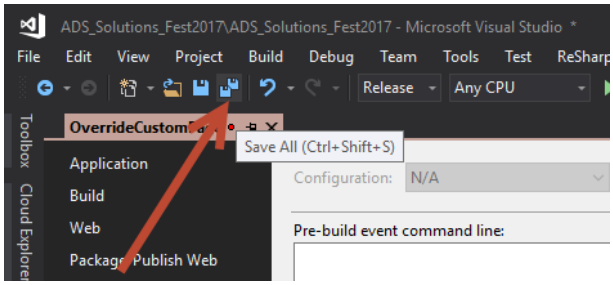
11. Now paste the copied text in the pop up window.



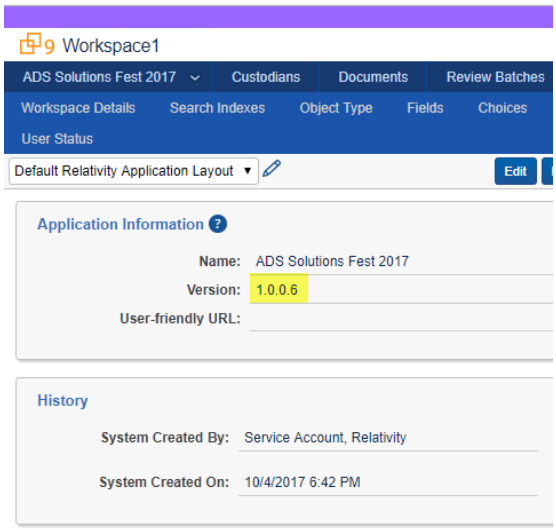
12. Click the **OK** button.



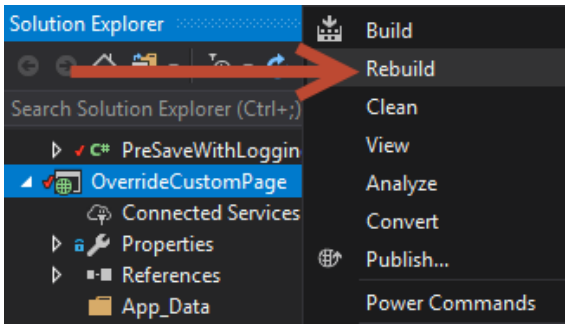
13. Next click the **Save All** button in Visual Studio.



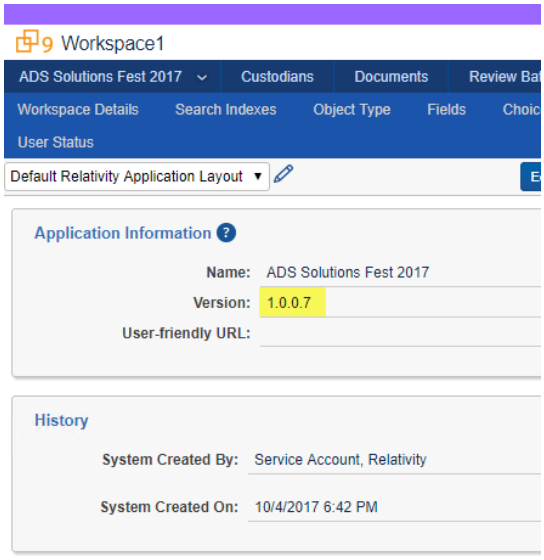
14. Open the **ADS Solutions Fest 2017** Relativity application in the **ADS Solutions Fest 2017 workspace**. Note down the current version of the application.



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

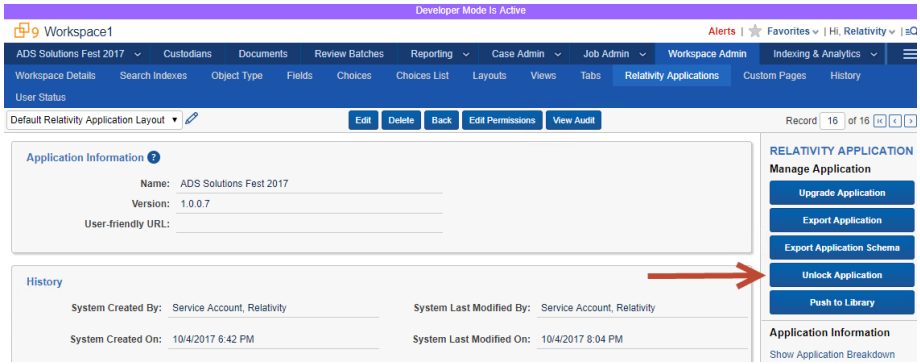


16. Now again open the **ADS Solutions Fest 2017** Relativity application in the **ADS Solutions Fest 2017 workspace** or **refresh** the page if it's already opened. Verify that the application version is increased.

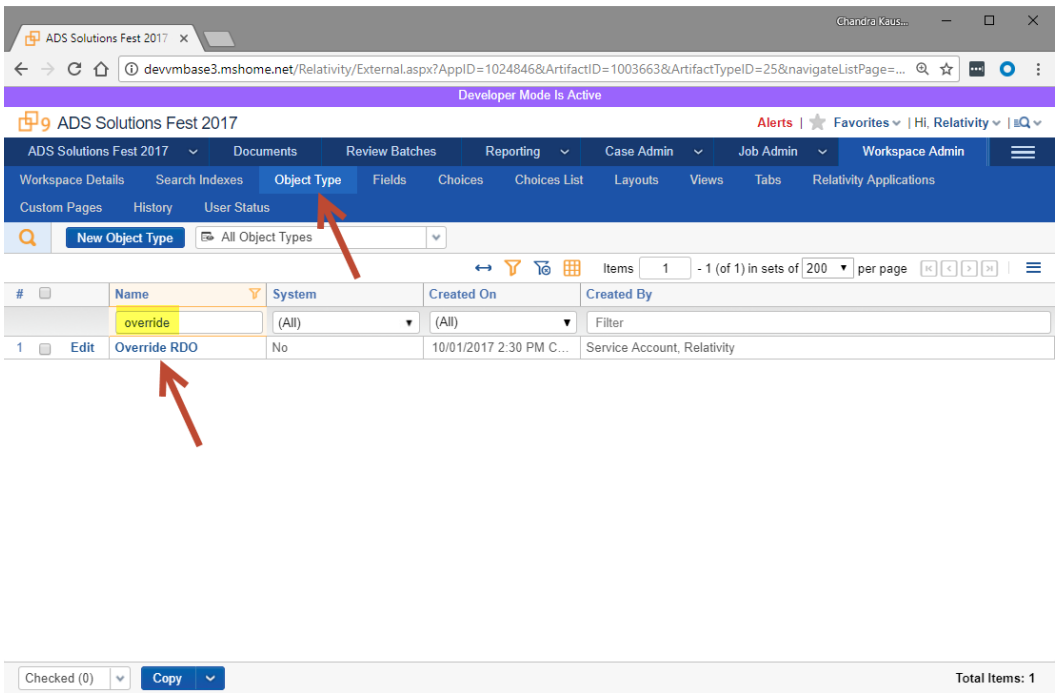


5.3 Object Rules

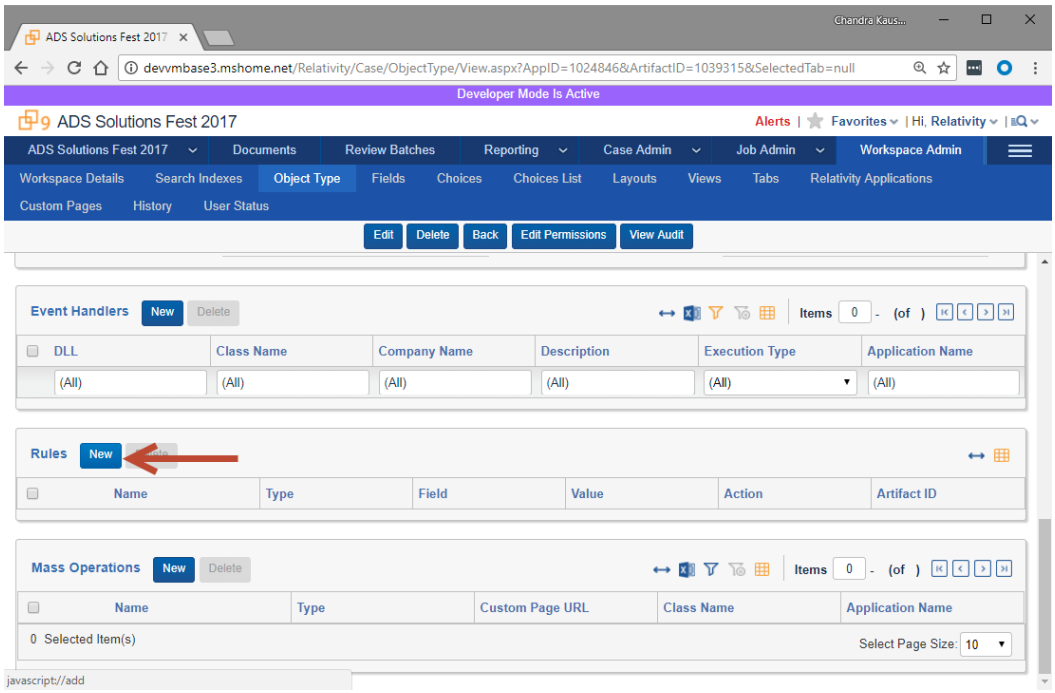
1. Unlock the Relativity application by clicking the **Unlock application** button.



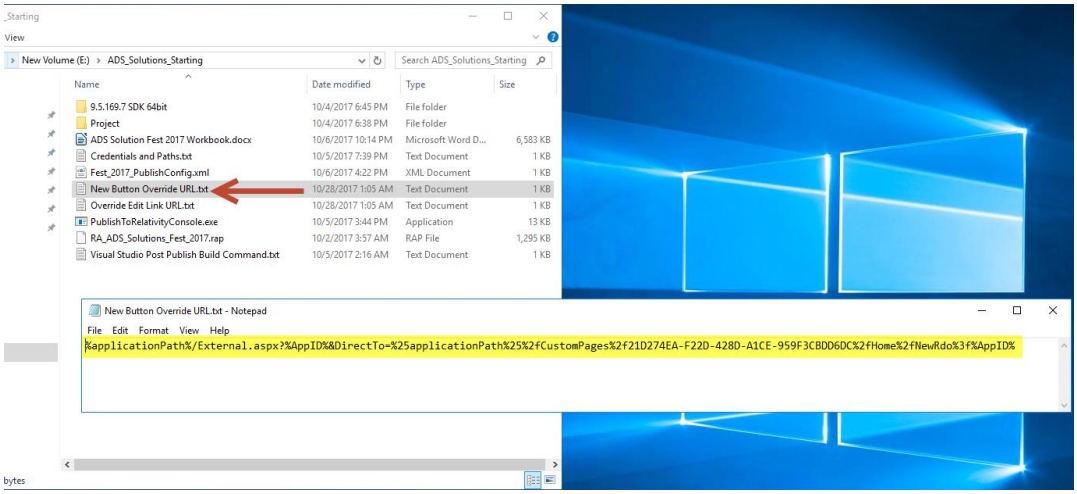
2. Next click the **Object Type** tab, search for the **override** word and click the **Override RDO** link.

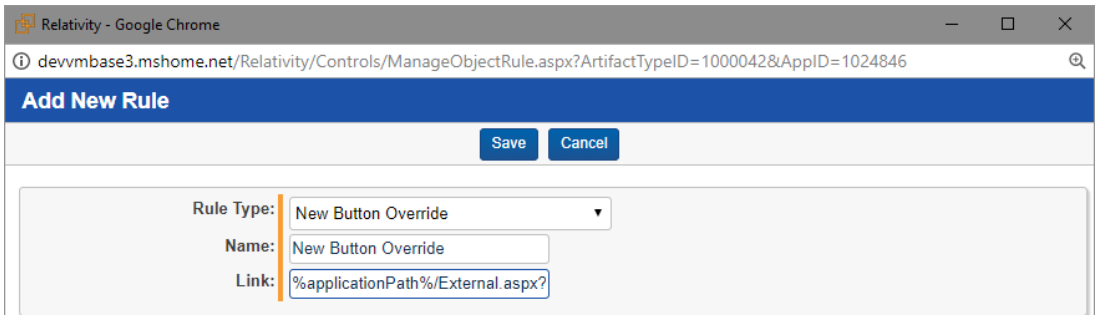


3. Scroll down to the **Rules** section and click on the **New** button.

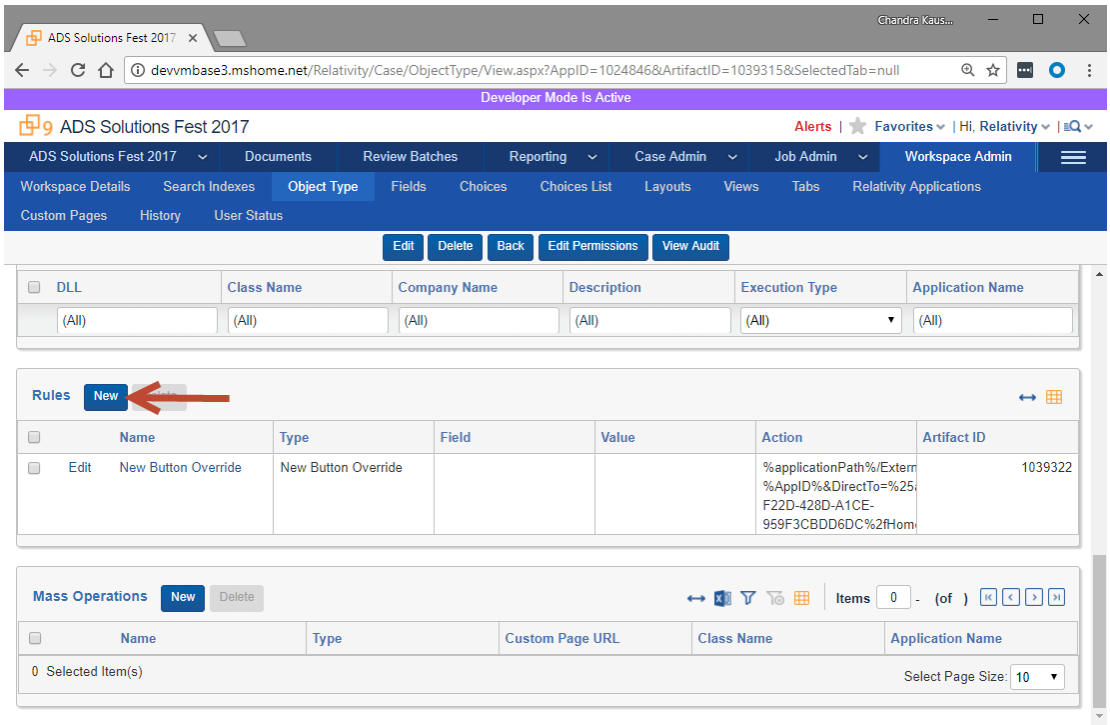


4. In the popup window, select **New Button Override** for **Rule Type**.
    a. Enter **New Button Override** for the **Name** field.
    b. For the **Link** field enter the text from the **New Button Override URL.txt** text file located in **E:\ADS_Solutions_Starting** folder as shown in the below screenshot.

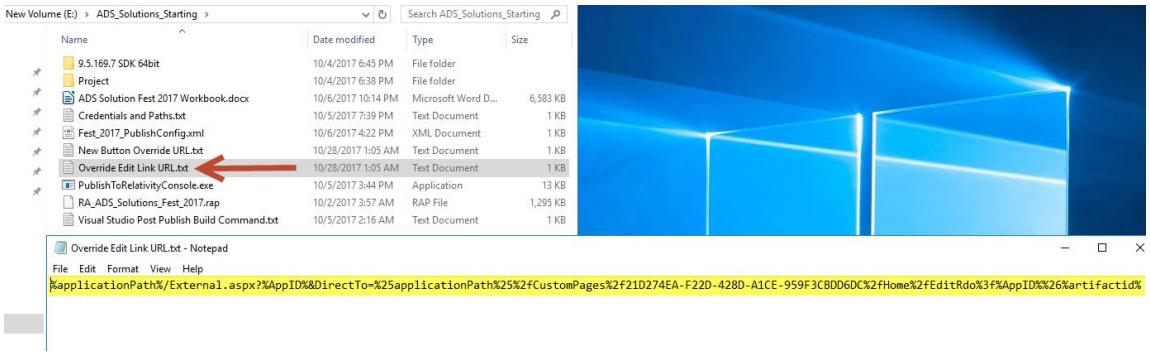


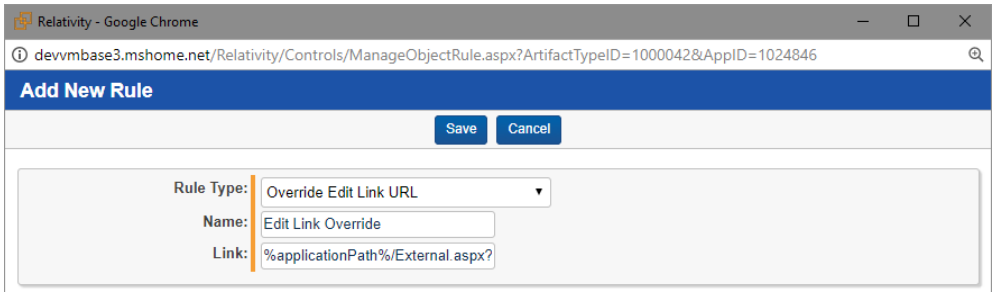


5. Click the **Save** button.
6. You can learn more about **New Button Override** at this link: **https://platform.relativity.com/9.5/Content/Managing_Relativity_dynamic_objects/RDO_9.5/Editing_Relativity_Objects.htm?Highlight=edit%20link%20override#NewButtonOverride**
7. Click the **New** button again to add a second rule.



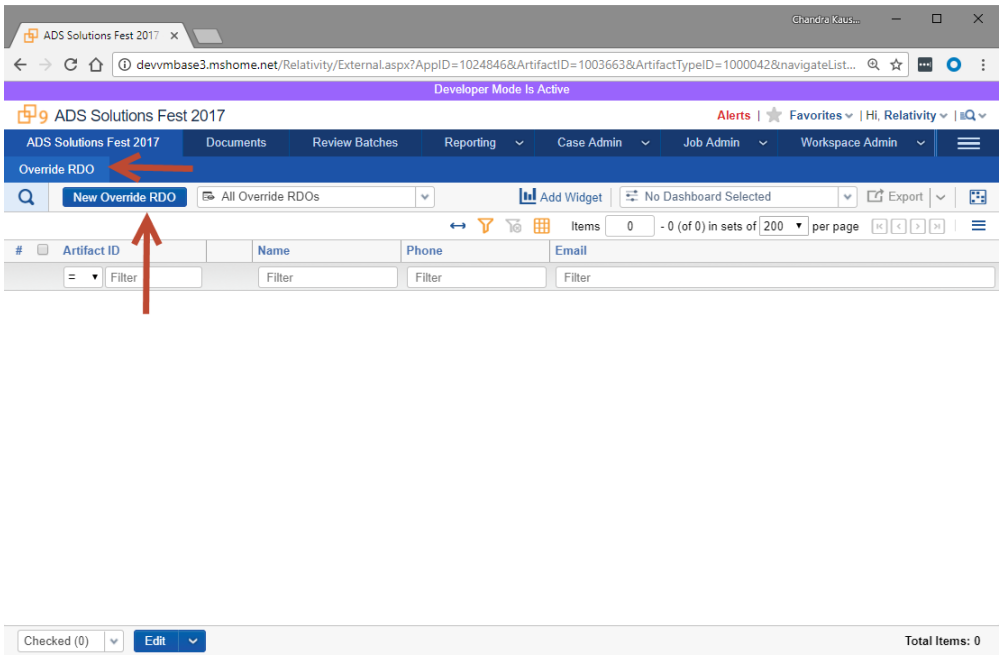
8. In the popup window, select **Override Edit Link URL** for **Rule Type**.
    a. Enter **Edit Link Override** for the **Name** field.
    b. For the **Link** field enter the text from the **Override Edit Link URL.txt** text file located in **E:\ADS_Solutions_Starting** folder as shown in the below screenshot.
    c.



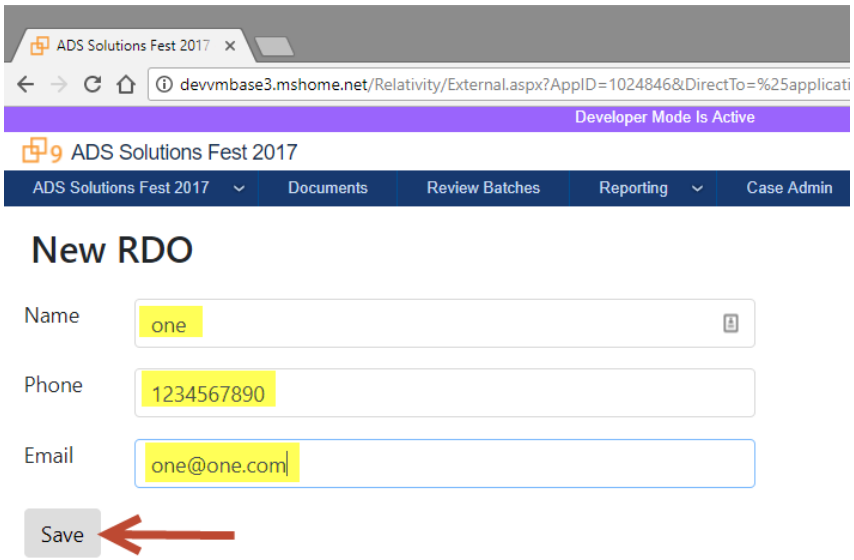


9. Next click the **Save** button.
10. You can learn more about **Override Edit Link URL** at this link: **https://platform.relativity.com/9.5/Content/Managing_Relativity_dynamic_objects/RDO_9.5/Editing_Relativity_Objects.htm?Highlight=edit%20link%20override#Override_Edit_Link_URL**

11. Next go to **Override RDO** tab and click the **New Override RDO** button.

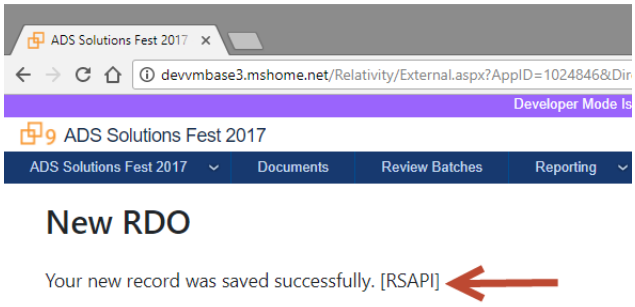


12. For the form, enter the following values
    Name: one
    Phone: 1234567890
    Email: one@one.com

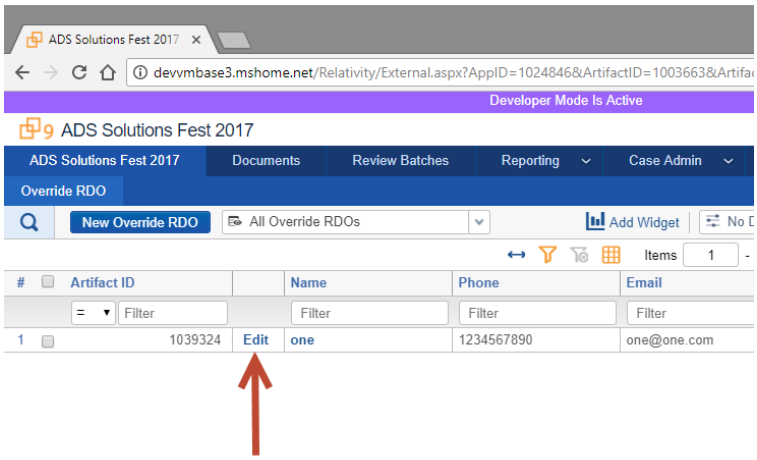


13. Click the **Save** button.

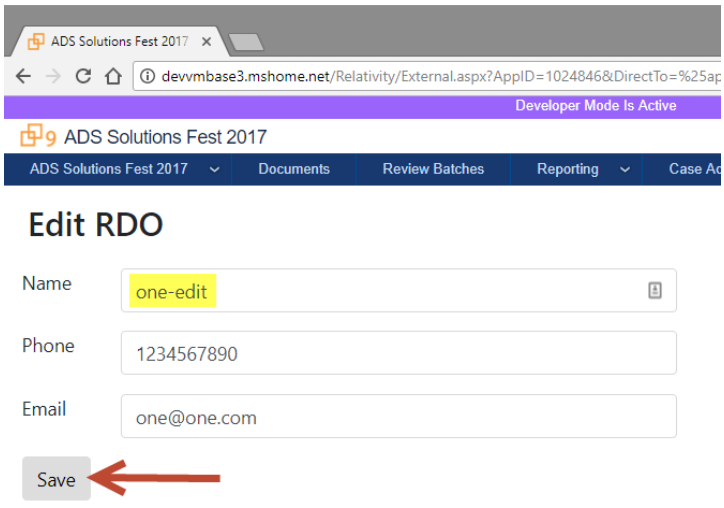
14. Once the RDO record is saved you will be shown a confirmation page as shown in the following screenshot.



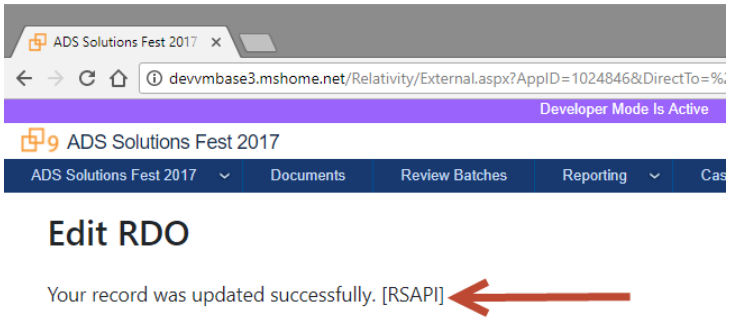
15. Next go to **Override RDO** tab and click the **Edit** link for the RDO record created above.



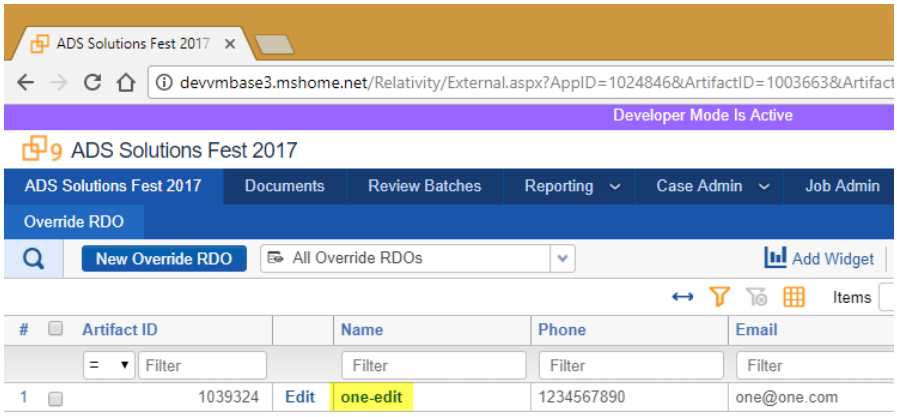
16. In the form, update the value of **Name** to **one-edit** and click on the **Save** button.



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

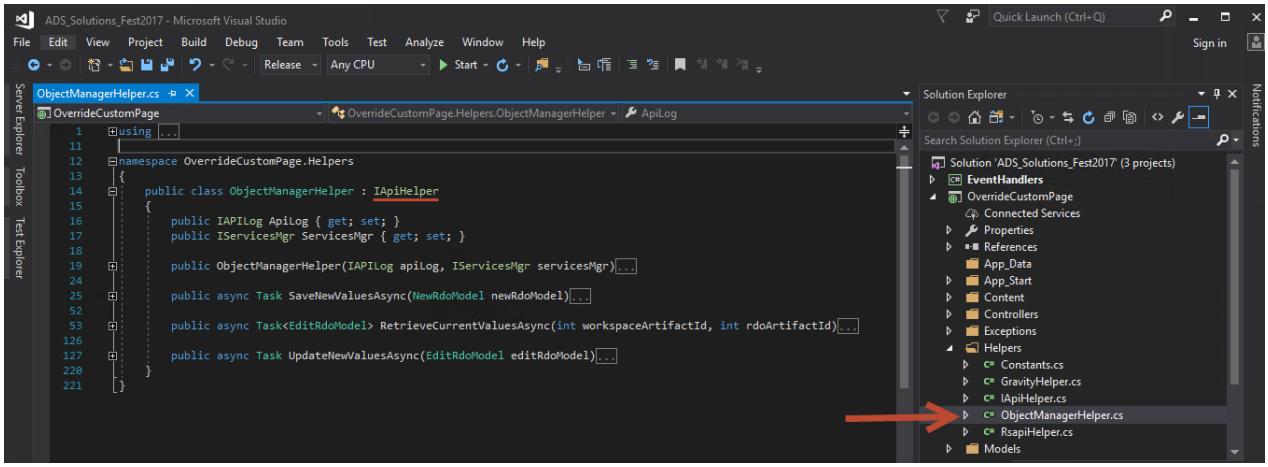


18. Go to **Override RDO** tab to verify the record was successfully updated.



5.4 ObjectManager API

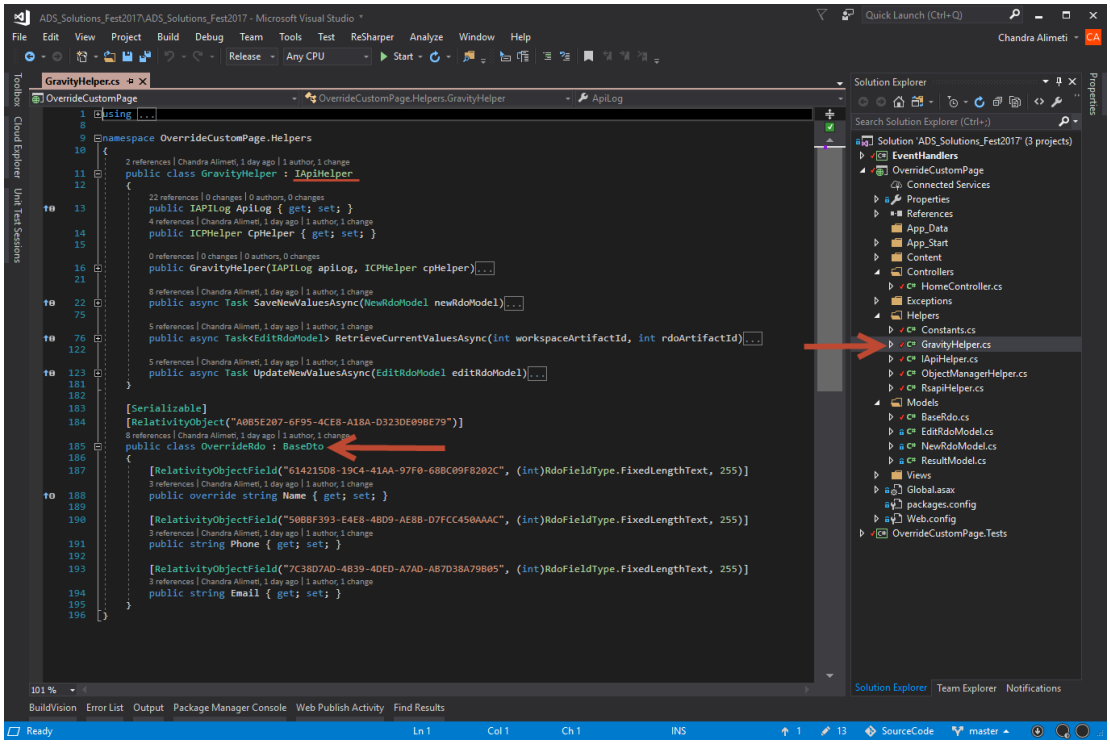
1. Go to **Visual Studio** and open the **ObjectManagerHelper.cs** file under **Helpers** folder in **OverrideCustomPage** project. This class is the implementation for the **IApiHelper** interface using the ObjectManager API.



2. 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**
2. Go to **Visual Studio** and open the **GravityHelper.cs** file under **Helpers** folder in **OverrideCustomPage** project.



3. 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.

```
[Serializable]
[RelativityObject("A0B5E207-6F95-4CE8-A18A-D323DE09BE79")]
8 references | Chandra Alimeti, 1 day ago | 1 author, 1 change
public class OverrideRdo : BaseDto
{
    [RelativityObjectField("614215D8-19C4-41AA-97F0-68BC09F8202C", (int)RdoFieldType.FixedLengthText, 255)]
    3 references | Chandra Alimeti, 1 day ago | 1 author, 1 change
    public override string Name { get; set; }

    [RelativityObjectField("50BBF393-E4E8-4BD9-AE8B-D7FCC450AAAC", (int)RdoFieldType.FixedLengthText, 255)]
    3 references | Chandra Alimeti, 1 day ago | 1 author, 1 change
    public string Phone { get; set; }

    [RelativityObjectField("7C38D7AD-4B39-4DED-A7AD-AB7D38A79B05", (int)RdoFieldType.FixedLengthText, 255)]
    3 references | Chandra Alimeti, 1 day ago | 1 author, 1 change
    public string Email { get; set; }
}
```

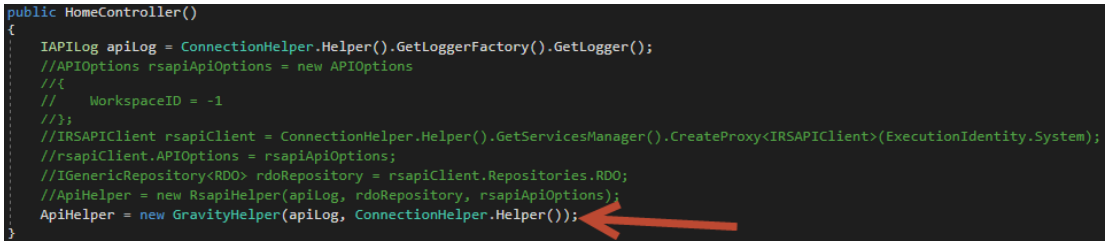
4. Next, we create an instance of **RsapiDao**.

```
RsapiDao gravityRsapiDao = new RsapiDao(CpHelper, newRdoModel.WorkspaceArtifactId);
```

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

```
gravityRsapiDao.InsertRelativityObject<OverrideRdo>(newOverrideRdo);
```

6. 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;

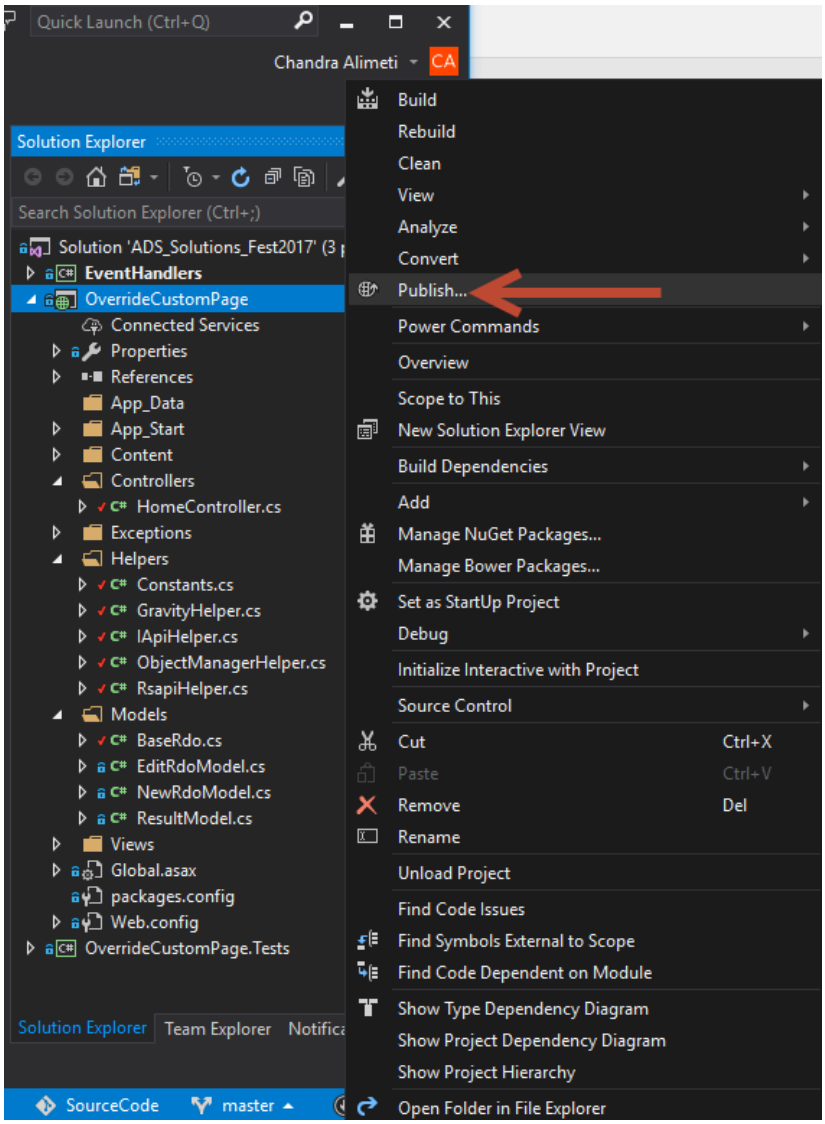
//IGenericRepository<RDO> rdoRepository = rsapiClient.Repositories.RDO;

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

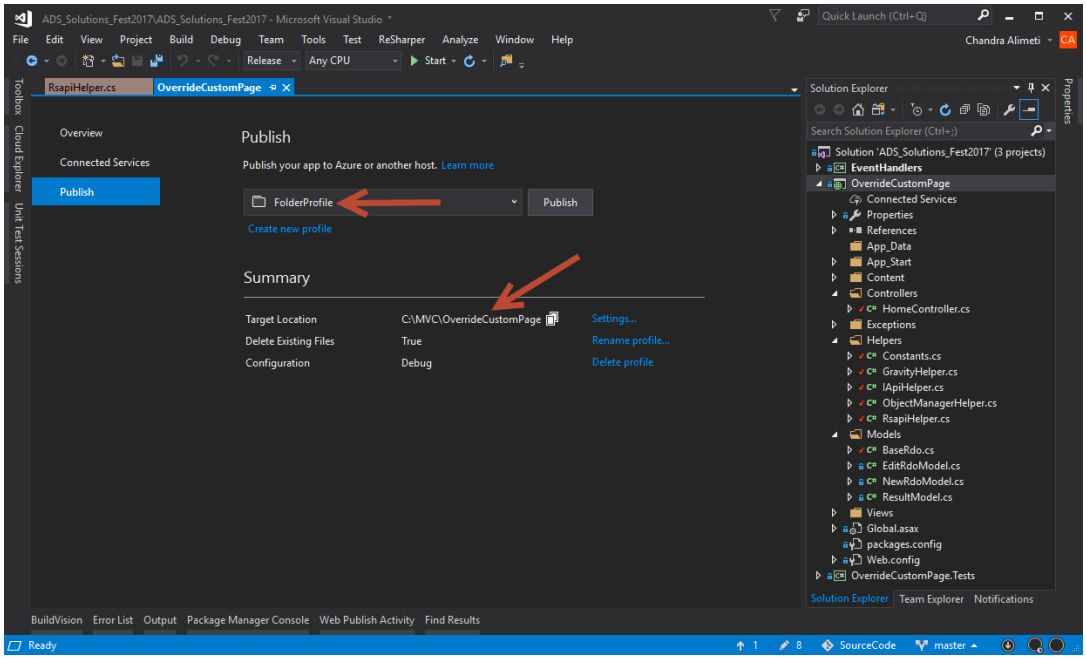
Code to add:

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

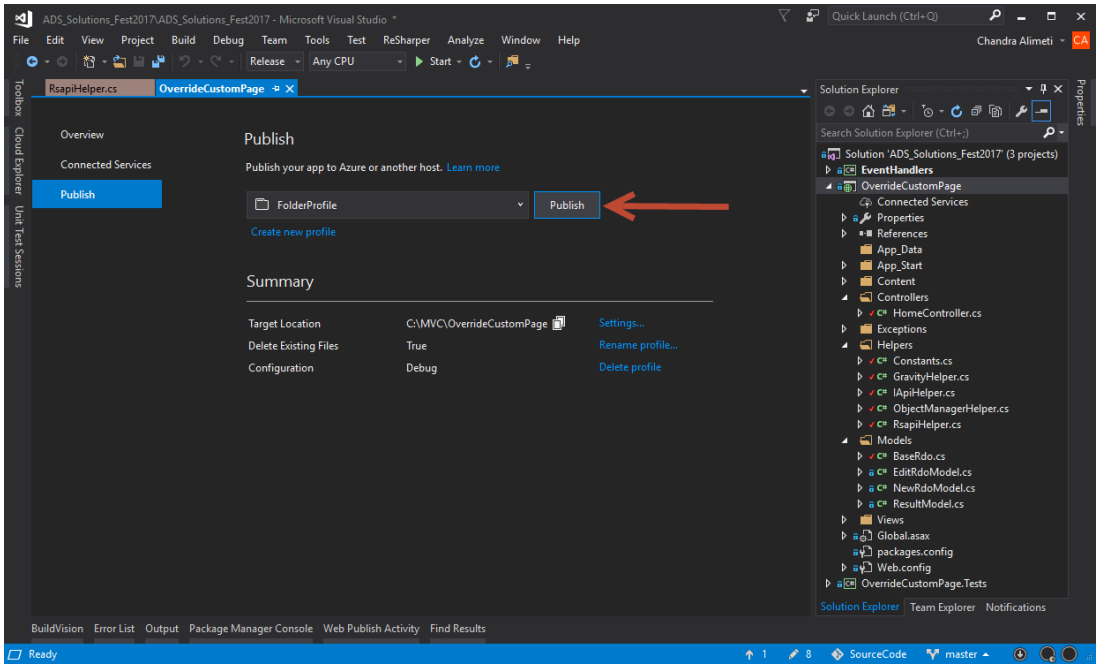
7. Right click on the **OverrideCustomPage** project and click on the **Publish** option.



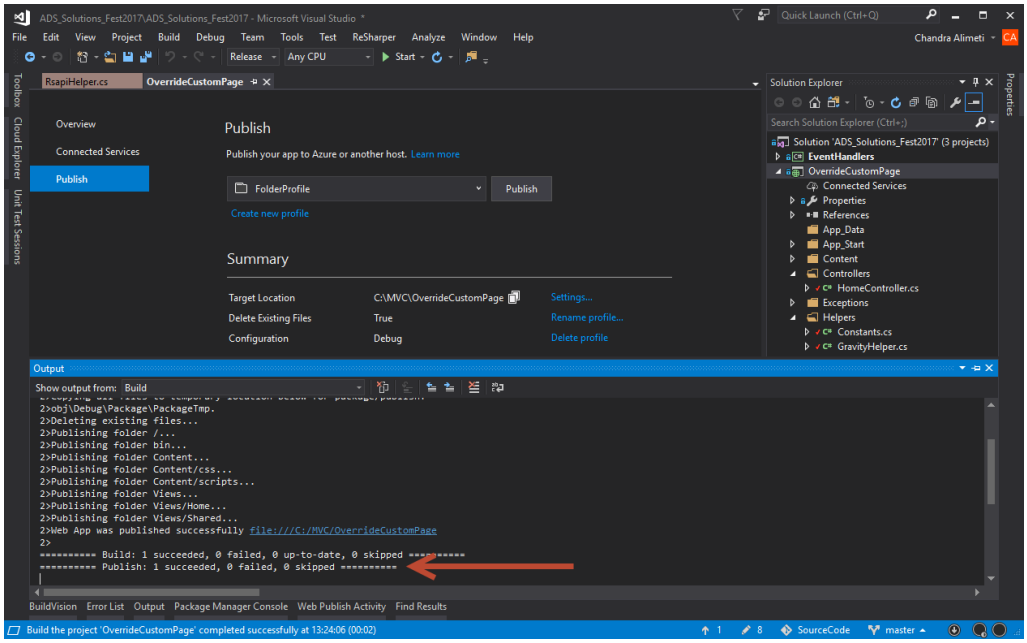
8. You should see a **Publish** settings tab opened. It already contains a **FolderProfile** which is set to publish the custom page to **C:\MVC\OverrideCustomPage** folder.



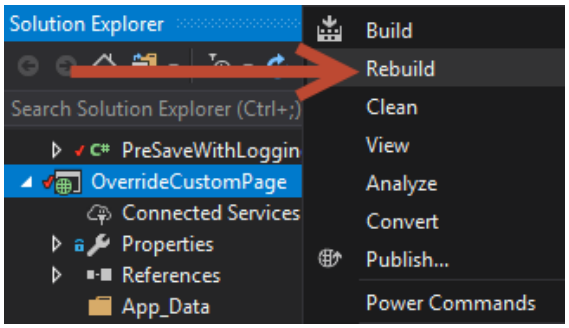
9. Click on the **Publish** button.



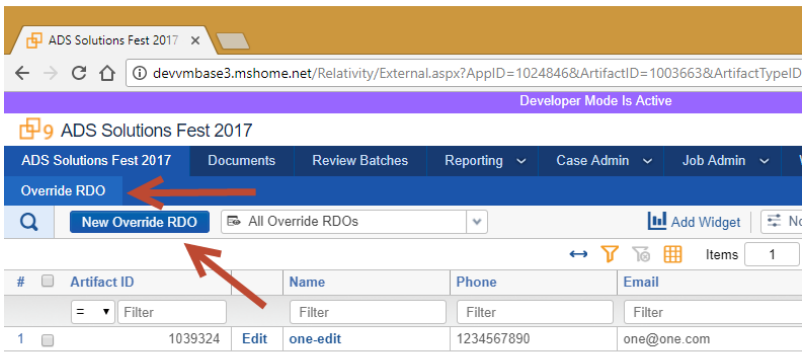
10. You will see a confirmation in the **Output** windows saying the custom page was successfully published.



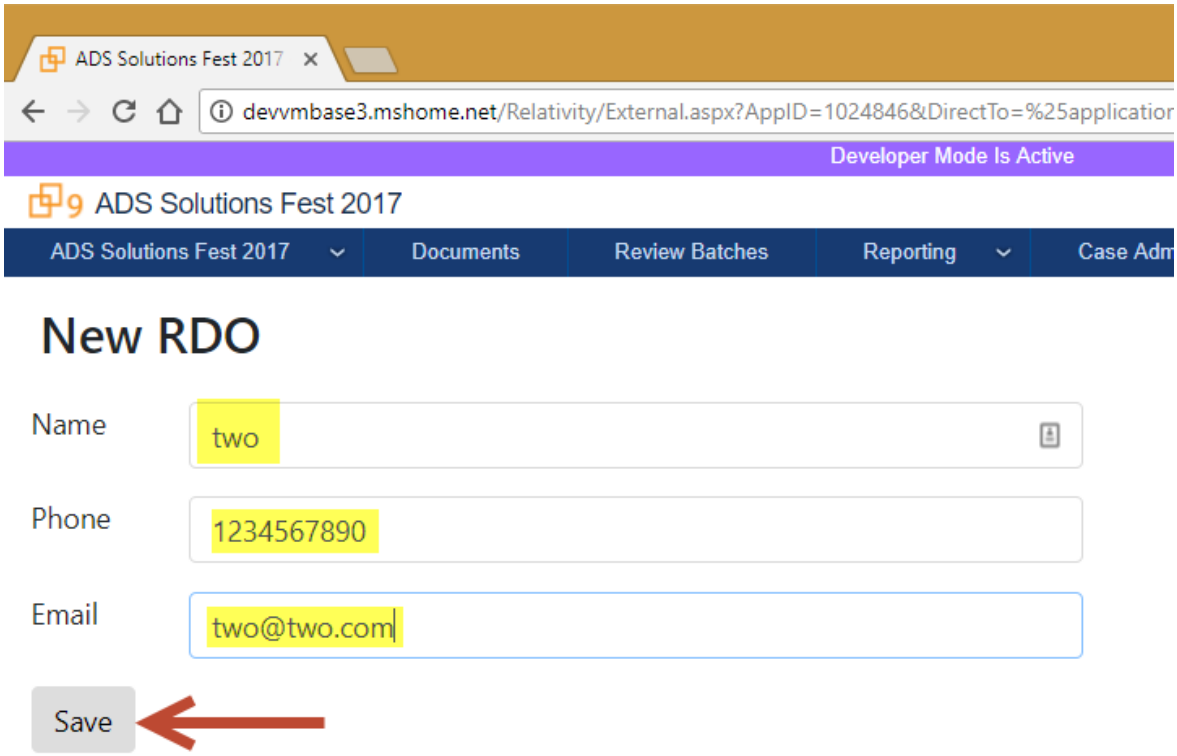
11. Now right click on the **OverrideCustomPage** project in Visual Studio and select the **Rebuild** option.



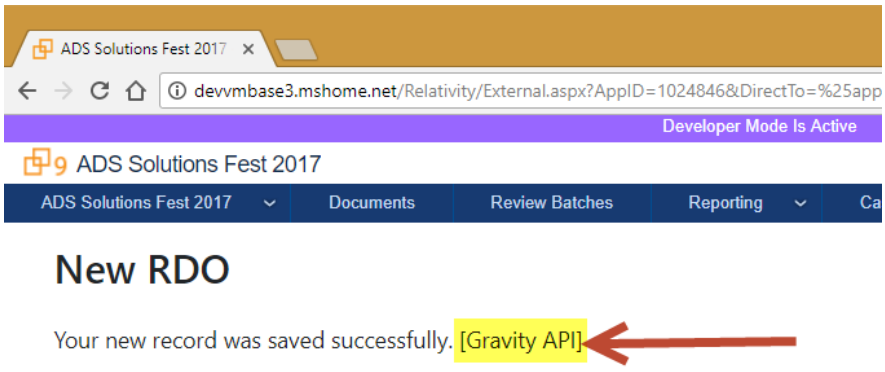
12. Next go to **Override RDO** tab in Relativity and click the **New Override RDO** button.



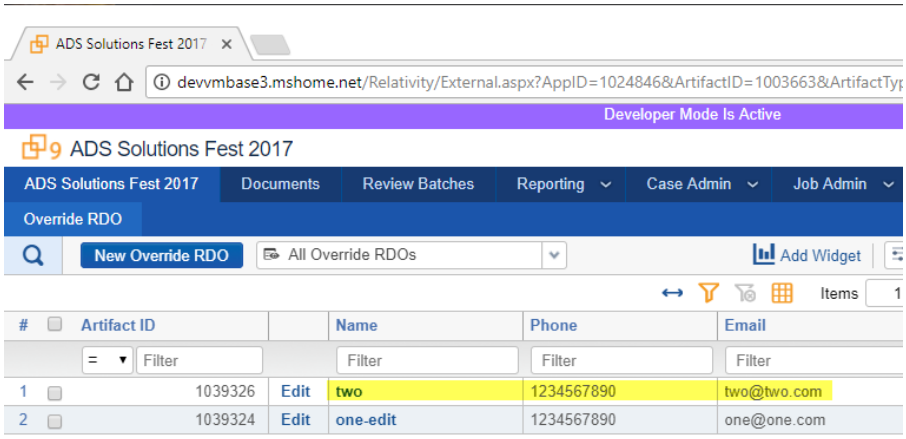
13. For the form, enter the following values
    Name: two
    Phone: 1234567890
    Email: two@two.com



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

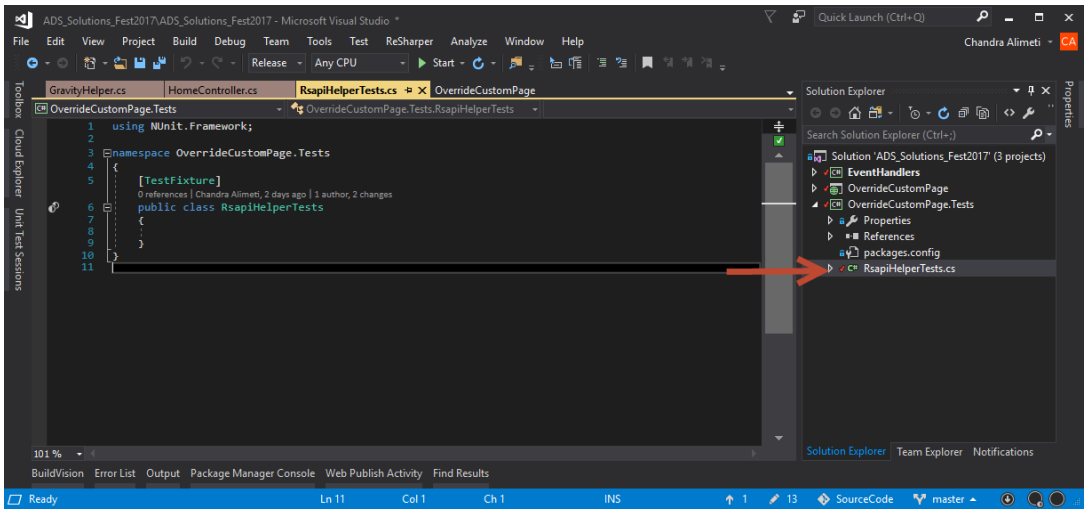


16. Go to **Override RDO** tab to verify the record was successfully saved.

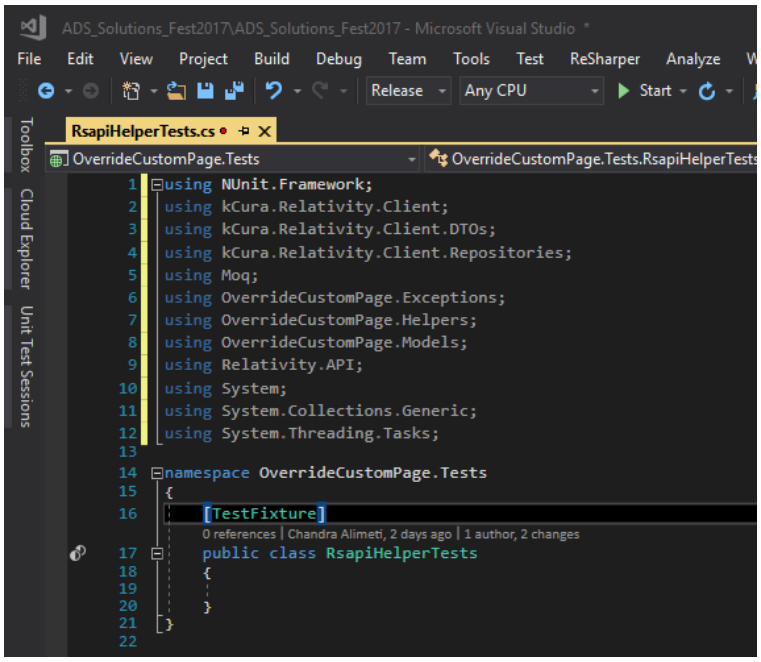


6 Unit Tests

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



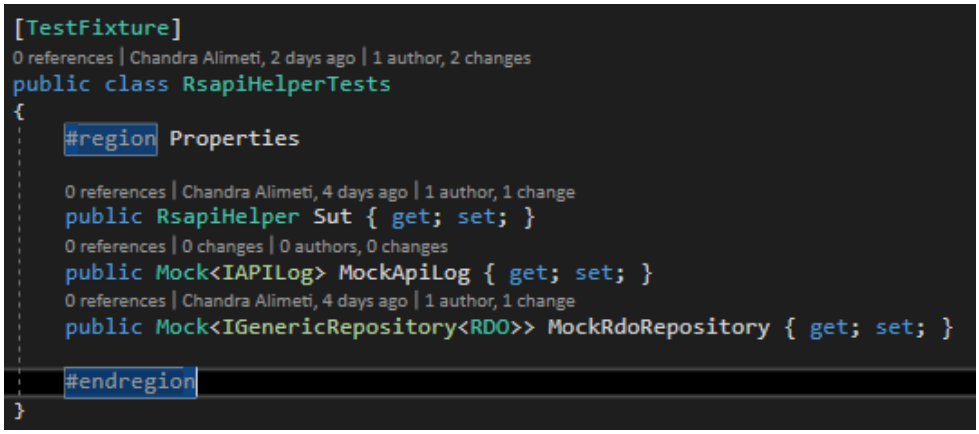
4. Add the following **using** statements to the class if they are not already present.



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;
```

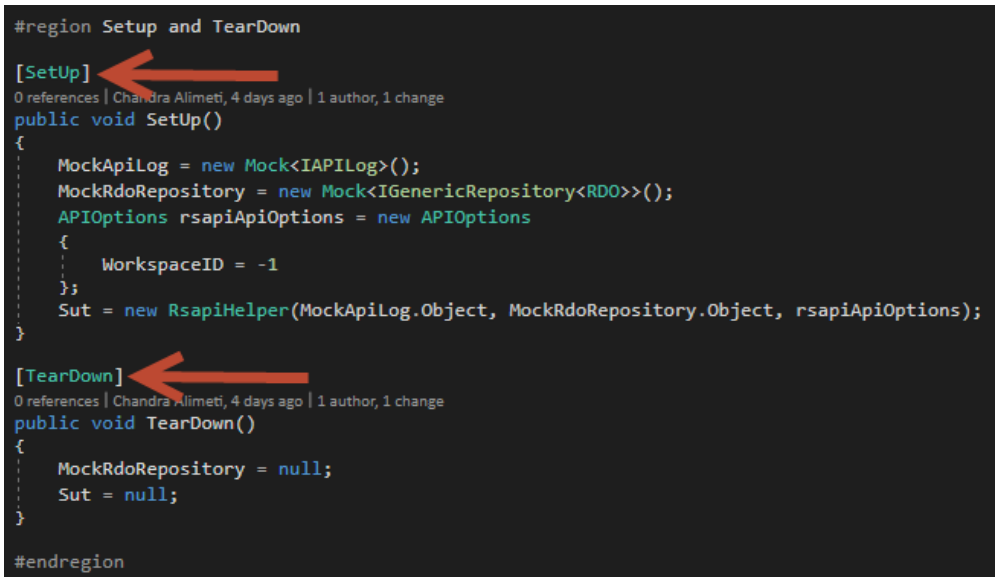
5. Next add the following **properties** to the class.



Code:

```
#region Properties
        public RsapiHelper Sut { get; set; }
        public Mock<IAPILog> MockApiLog { get; set; }
        public Mock<IGenericRepository<RDO>> MockRdoRepository { get; set; }

#endregion
```

6. 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
```

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

```
#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
```

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
```

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

```
private void Mock_MockApiLog_LogError_Works()
{
    MockApiLog
        .Setup(x => x.LogError(It.IsAny<string>(), It.IsAny<Exception>()));
}
```

Code:

```
private void Mock_MockApiLog_LogError_Works()
{
   MockApiLog
      .Setup(x => x.LogError(It.IsAny<string>(), It.IsAny<Exception>()));
}
```

9. 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.

```
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));
}
```

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));
}
```

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

```
private void Mock_RdoRepository_CreateSingle_Works()
{
    MockRdoRepository
        .Setup(x => x.CreateSingle(It.IsAny<RDO>()));
}
```

Code:

```
private void Mock_RdoRepository_CreateSingle_Works()
{
    MockRdoRepository
        .Setup(x => x.CreateSingle(It.IsAny<RDO>()));
}
```

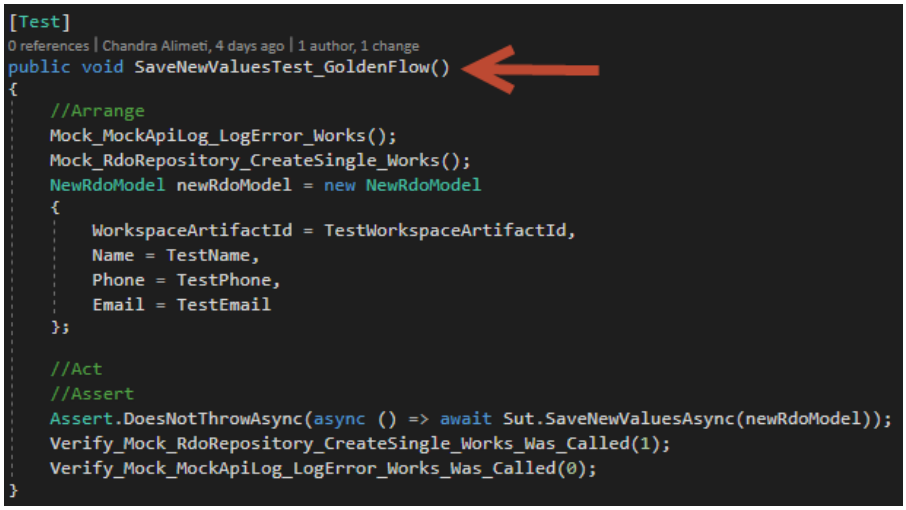
11. Next add the following method which verifies that the the **CreateSingle** method on the **IRSAPIClient** interface was called. This will be part of the Assert section of the unit test.

```
private void Verify_Mock_RdoRepository_CreateSingle_Works_Was_Called(int timesCalled)
{
    MockRdoRepository
        .Verify(x => x.CreateSingle(It.IsAny<RDO>())
            , Times.Exactly(timesCalled));
}
```

Code:

```
private void Verify_Mock_RdoRepository_CreateSingle_Works_Was_Called(int timesCalled)
{
    MockRdoRepository
        .Verify(x => x.CreateSingle(It.IsAny<RDO>())
            , Times.Exactly(timesCalled));
}
```

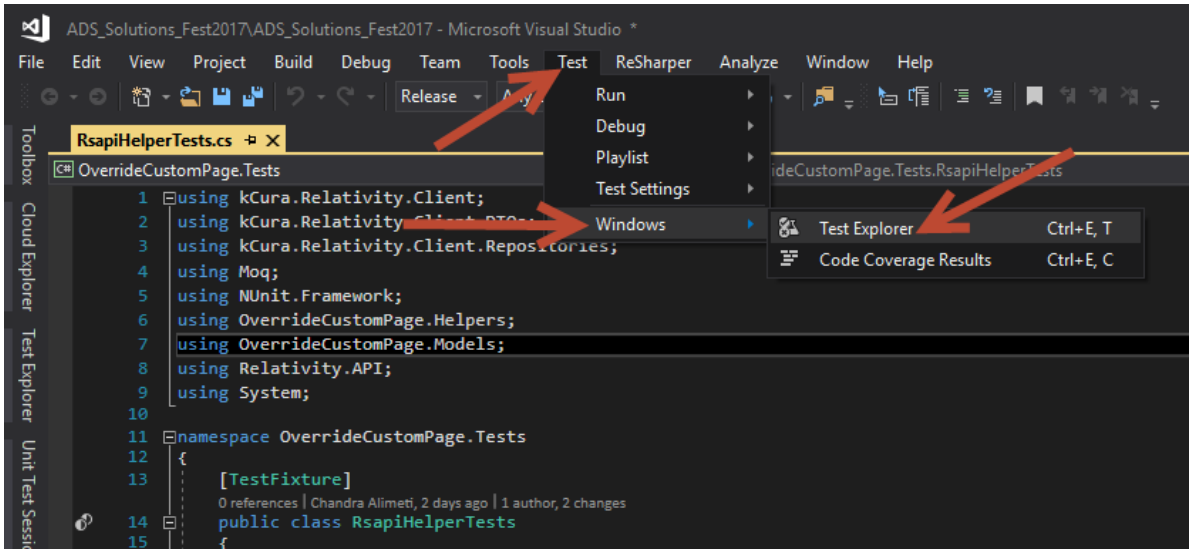
12. Next, add the following unit test for the **SaveNewValues()** method in the **RsapiHelper** class. This unit test tests the scenario where the RSAPI CreateSingle API call works without any errors.



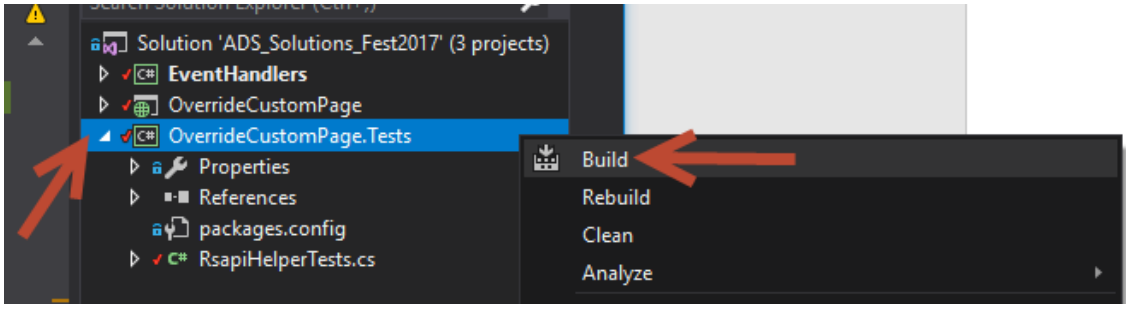
Code:

```
[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);
}
```

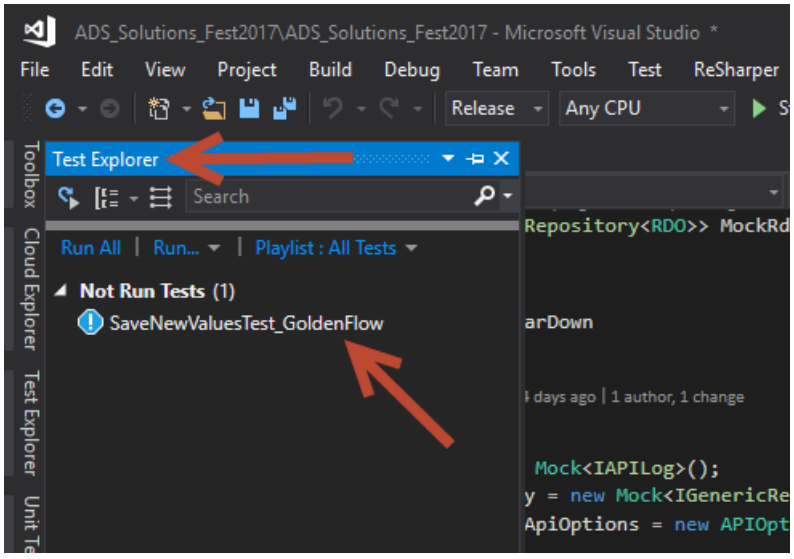
13. Go to Visual Studio, click on **Test** menu option, select **Windows** option, then click on the **Test Explorer** option.



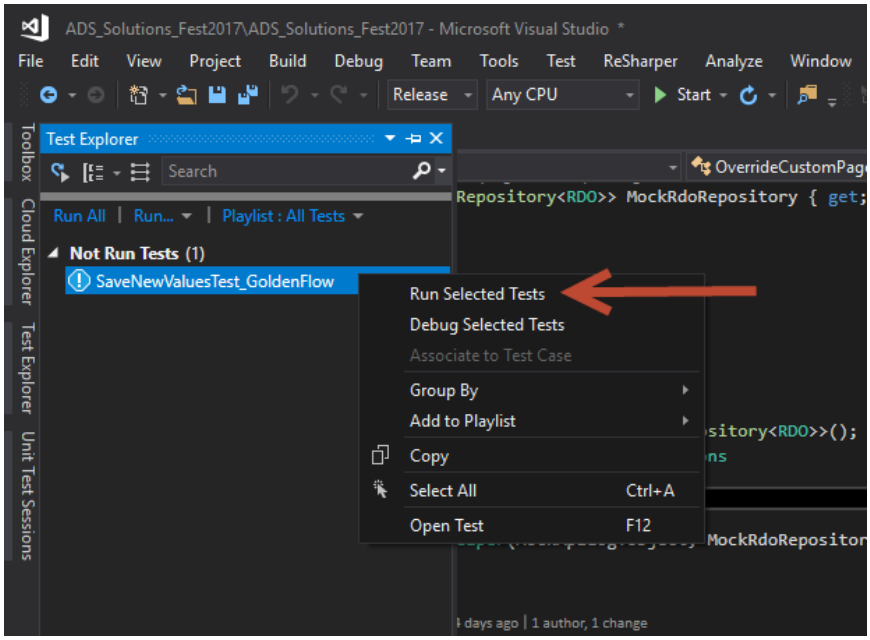
14. Build the project to see the new test in the **Test Explorer** window. Right click on the **OverrideCustomPage.Tests** project and select the **Build** option.



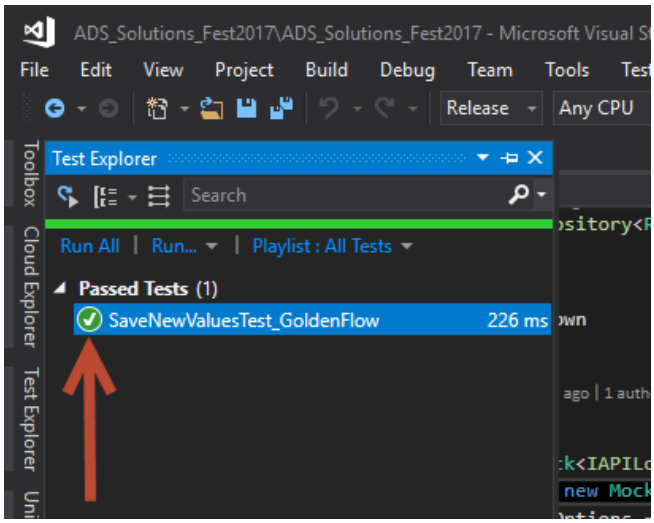
15. In Test Explorer, you should the **SaveNewValuesTest_GlodenFlow** test we have just written.



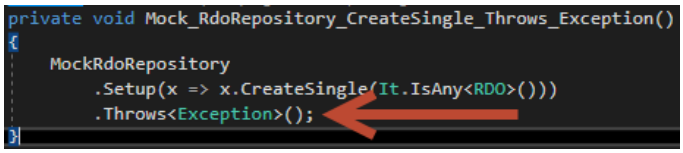
16. Right click on the **SaveNewValuesTest_GlodenFlow** test and select **Run Selected Tests** option.



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



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

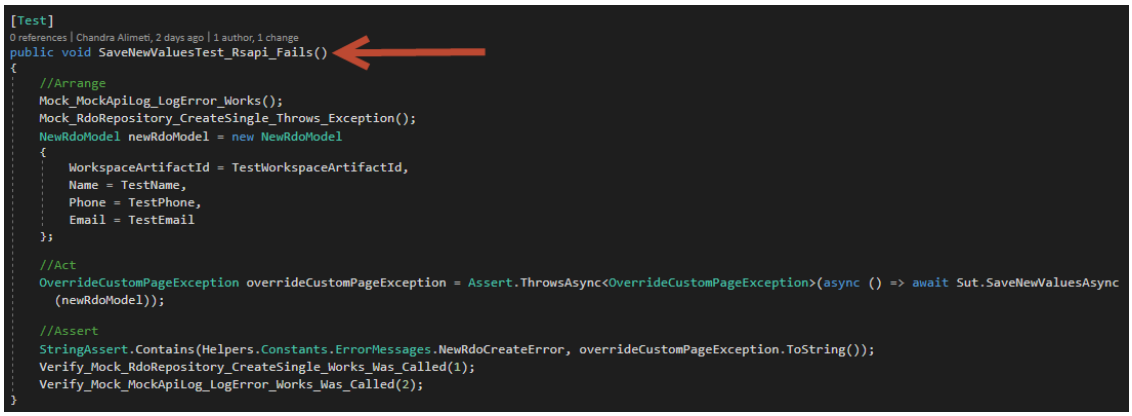


```
private void Mock_RdoRepository_CreateSingle_Throws_Exception()
{
    MockRdoRepository
        .Setup(x => x.CreateSingle(It.IsAny<RDO>()))
        .Throws<Exception>();
}
```

Code:

```
private void Mock_RdoRepository_CreateSingle_Throws_Exception()
{
    MockRdoRepository
        .Setup(x => x.CreateSingle(It.IsAny<RDO>()))
        .Throws<Exception>();
}
```

19. 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.

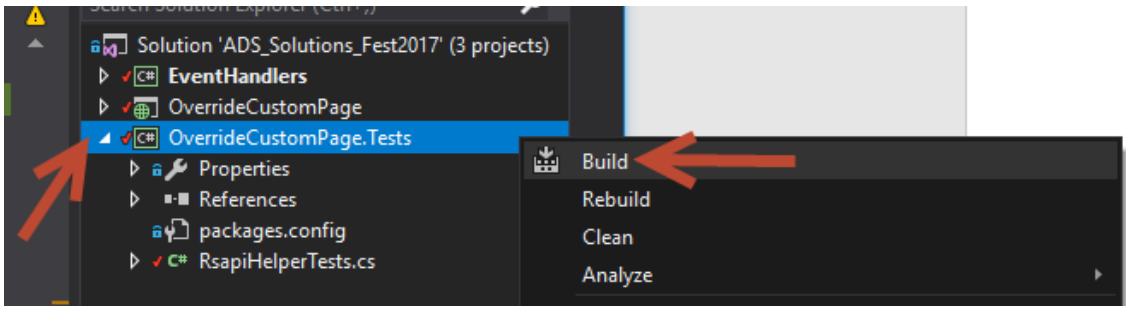


Code:

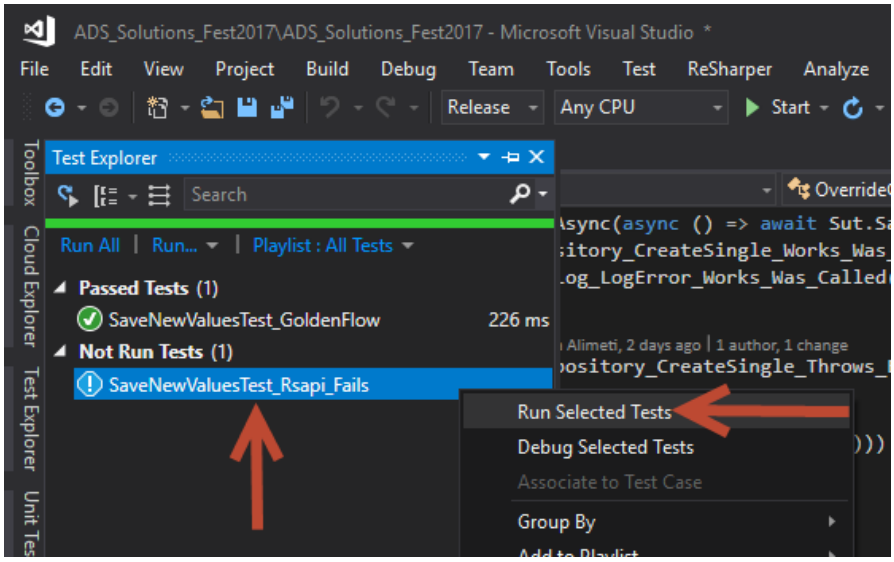
```
[Test]
public void SaveNewValuesTest_Rsapi_Fails()
{
    //Arrange
    Mock_MockApiLog_LogError_Works();
    Mock_RdoRepository_CreateSingle_Throws_Exception();
    NewRdoModel newRdoModel = new NewRdoModel
    {
        WorkspaceArtifactId = TestWorkspaceArtifactId,
        Name = TestName,
        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);
}
```

20. 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.



21. Right click on the **SaveNewValuesTest_Rsapi_Fails** test and select **Run Selected Tests** option.



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