Integration Points WikiLoader Workshop

Version 1.0

Contents

1 Course Overview

The objective of this course is to introduce you to the Relativity Integration Points (RIP) Framework. This course provides detailed instructions about how to create the WikiLoader Provider which is a redistributable Relativity application comprised of an Integration Points Provider, Relativity Event Handlers and a Relativity Custom Page.

Relativity Integration Points is a framework built on Relativity Agents and the Import API to allow your application to optimally scale data import. RIP is a benefit to you, as the developer, because it allows you to focus on connecting your data source to Relativity instead of implementing job progress reports, job statistics, and data batching in an efficiently optimized parallel fashion. Your application has the full benefits of the RIP framework after implementing only a provider to connect your data source, a Relativity Custom Page to allow your users to configure the job, and two event handlers to register and uninstall your RIP Provider.

The WikiLoader Provider includes an Installation Event Handler, an Uninstallation Event Handler, and a Custom Page, which are standard requirements for any RIP Provider. The provider will load a Wikipedia XML file into Relativity.

Although we will use a simpler example, if you want an advanced example download the open-source GitHub repository Social Media Provider. Feel welcome to make contributions to it at the following URL: https://github.com/relativitydev/rip_social_media_provider

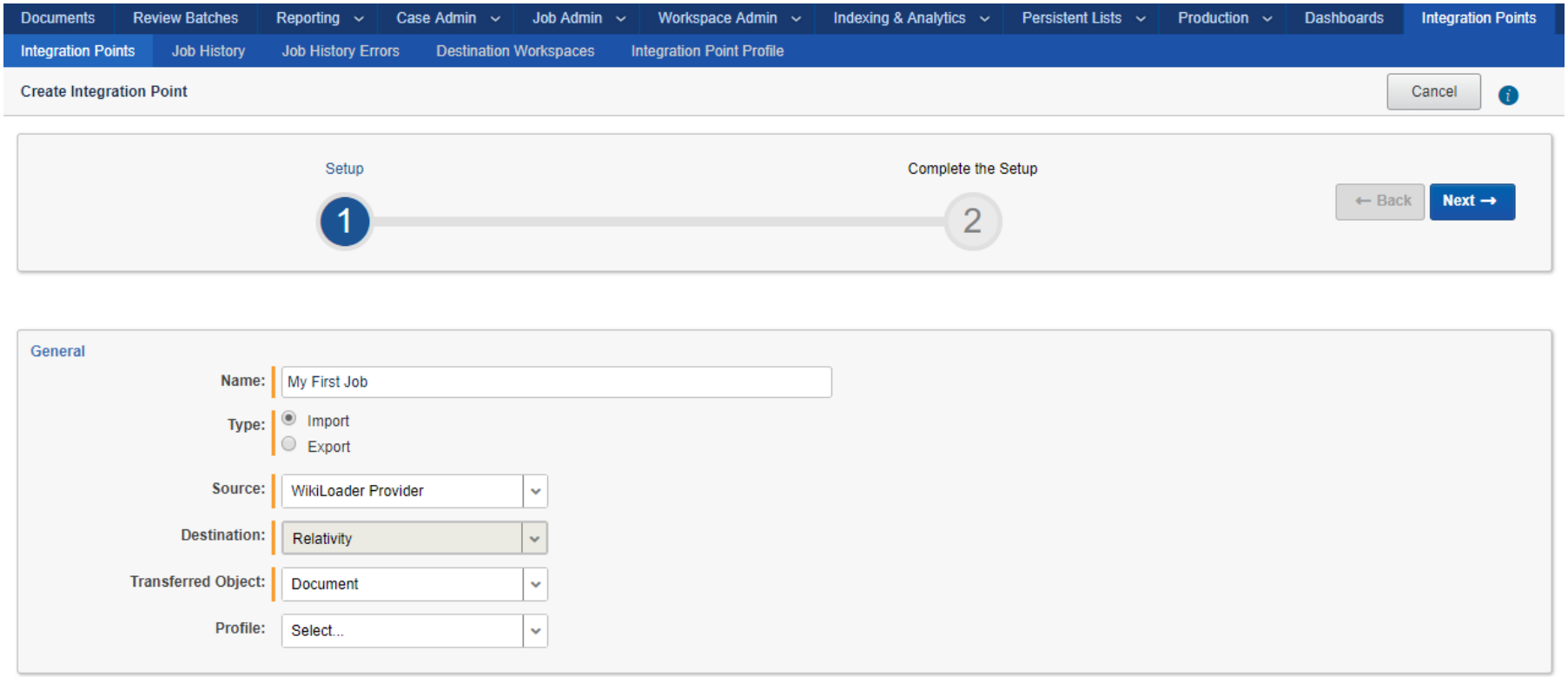
2 Required Development Software

This workbook requires the following development environment:

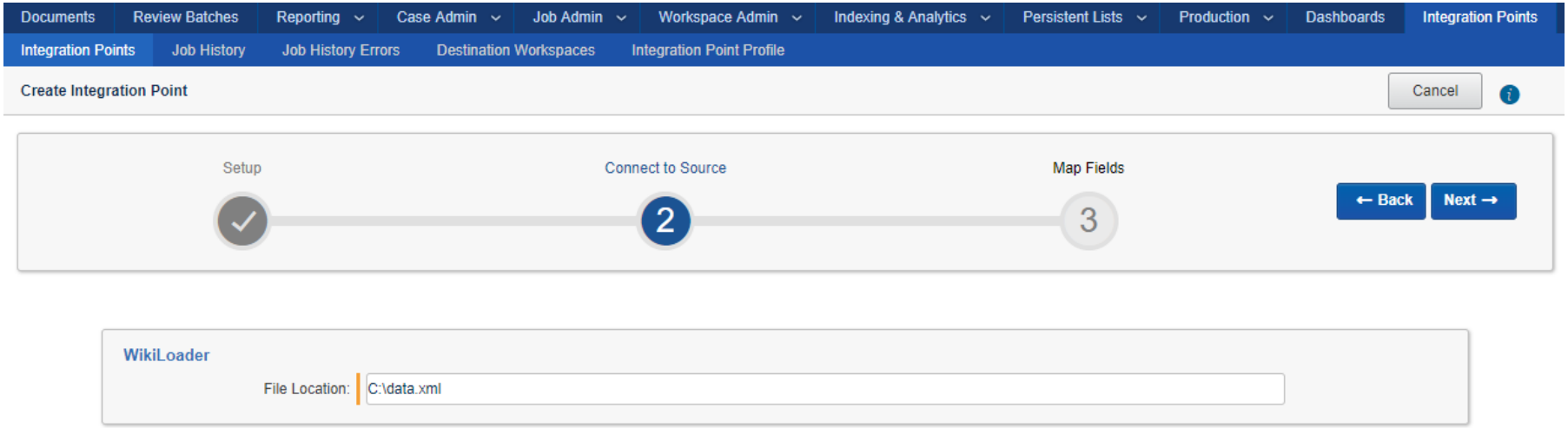
- Relativity 9.6 or greater
- Integration Points Application 9.6.64.2
- Visual Studio 2017
- .NET Framework 4.6.2
- Relativity SDK 9.6.174.30
- Relativity Integration Points SDK 9.6.33.7
    - We recommend installing this at the same path as the Relativity SDK.

3 General Overview - Relativity Integration Points

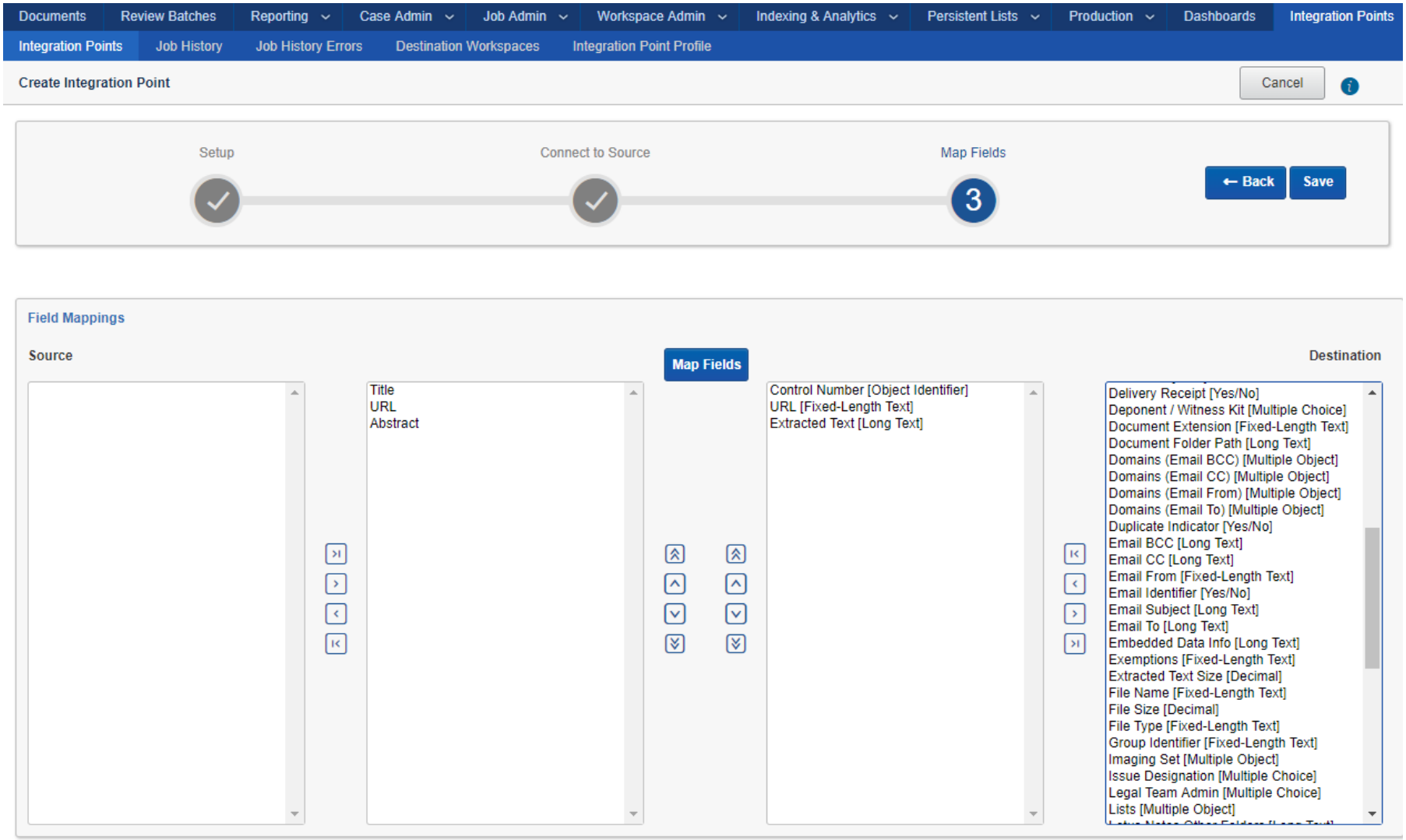
Relativity Integration Points (RIP) differs from traditional extensibility points like an event handler because it is a fully redistributable Relativity Application that must be installed at a workspace level to use. Installing you custom RIP Provider registers it with the Integration Points application and allows your users to access it in the Integration Points Source Drop-down list. On the same page, the user selects the type of object in which they would like to import data.



After a user names the RIP job, selects your provider and the object to import data into, and clicks **Next ->**, your custom page is loaded in a nested frame below the main Integration Points Navigation. Your custom page allows the user to configure their job in a user-friendly manner. Users are sent to this page when they create a new job and when they change settings of an existing job, therefore your custom page must implement methods to load and save job settings. We will go into more detail later in this workbook.



After the user has configured the RIP job and clicks **Next ->**, the user is presented with a list of fields to map. The fields are retrieved by your implementation of the **GetFields()** method. The user maps the provider fields to the fields of the import object (in this example, the Document object).



After the job is saved, the user is sent to the final RIP page where they can start a job and view information about a job in progress. RIP records job statistics and errors automatically for the convenience of the developer and the user.

Documents | Review Batches | Reporting | Case Admin | Job Admin | Workspace Admin | Indexing & Analytics | Persistent Lists | Production | Dashboards | Integration Points

Integration Points | Job History | Job History Errors | Destination Workspaces | Integration Point Profile

Integration Point Details ▾ | Edit | Delete | Back | Edit Permissions | View Audit | Record 1 of 1

General | Scheduling

Name: My First Job

Log Errors: Yes

Has Errors: No

Overwrite: Append Only

Email Notification Recipients:

Source Provider: WikiLoader Provider

Promote Eligible: No

Destination Configuration: {"artifactTypeID":10,"destinationProviderType":"74A863B9-00EC-4BB7-9B3E-1E22323010C6","CaseArtifactId":1017294,"ImportOverwriteMode":"AppendOnly","importNativeFile":"false","importNativeFileCopyMode":"DoNotImportNativeFiles","UseFolderPathInformation":"false","UseDynamicFolderPath":"false","ImageImport":"false","ImagePrecedence":[],"ProductionPrecedence":0,"IncludeOriginalImages":false,"MoveExistingDocuments":"false","ExtractedTextFieldContainsFilePath":"false","ExtractedTextFileEncoding":"utf-16","CustodianManagerFieldContainsLink":"true","FieldOverlayBehavior":"Use Field Settings","ArtifactTypeName":"Document"}

Source Configuration: C:\data.xml

TRANSFER OPTIONS

Stop

Save as a Profile

Status

Items 1 - 1 (of 1)

Job ID	Start Time (U...	Artifact ID	Name	Integration P...	Job Type	Job Status	Destination ...	Destination I...	Items Transf...	Total Items	Items with Er...	Sys
(All)	(All)	(All)	(All)	(All)	(All)	(All)	(All)	(All)	(All)	(All)	(All)	(A
11	8/24/2018 4:46 PI	1039688	My First Job	My First Job	Run	Pending	WikiLoader Case - 1017294	This Instance	0		0	Adr

After a job is started, the RIP Agent(s) take over. First, the **GetBatchableIDs()** method from your custom provider is called. It is called once per job from the Agent Server and returns a list of all identifiers of the data you want to import. Next, your job settings and a batch of IDs are passed to your provider's **GetData()** method. It is also called from the Agent Server and executed multiple times to incrementally access the data by ID and import it in parallel between agents until the job is complete.

4 Application Overview

The WikiLoader Provider is an application that leverages the Relativity Integration Points (RIP) Framework to import XML Wiki abstract data into Relativity. Users of the application create a new job, select a data source Wiki XML file and map the fields into the workspace. Each time the job is run, the WikiLoader Provider will import unique Wiki entries contained in the XML file.

4.1 Application Components

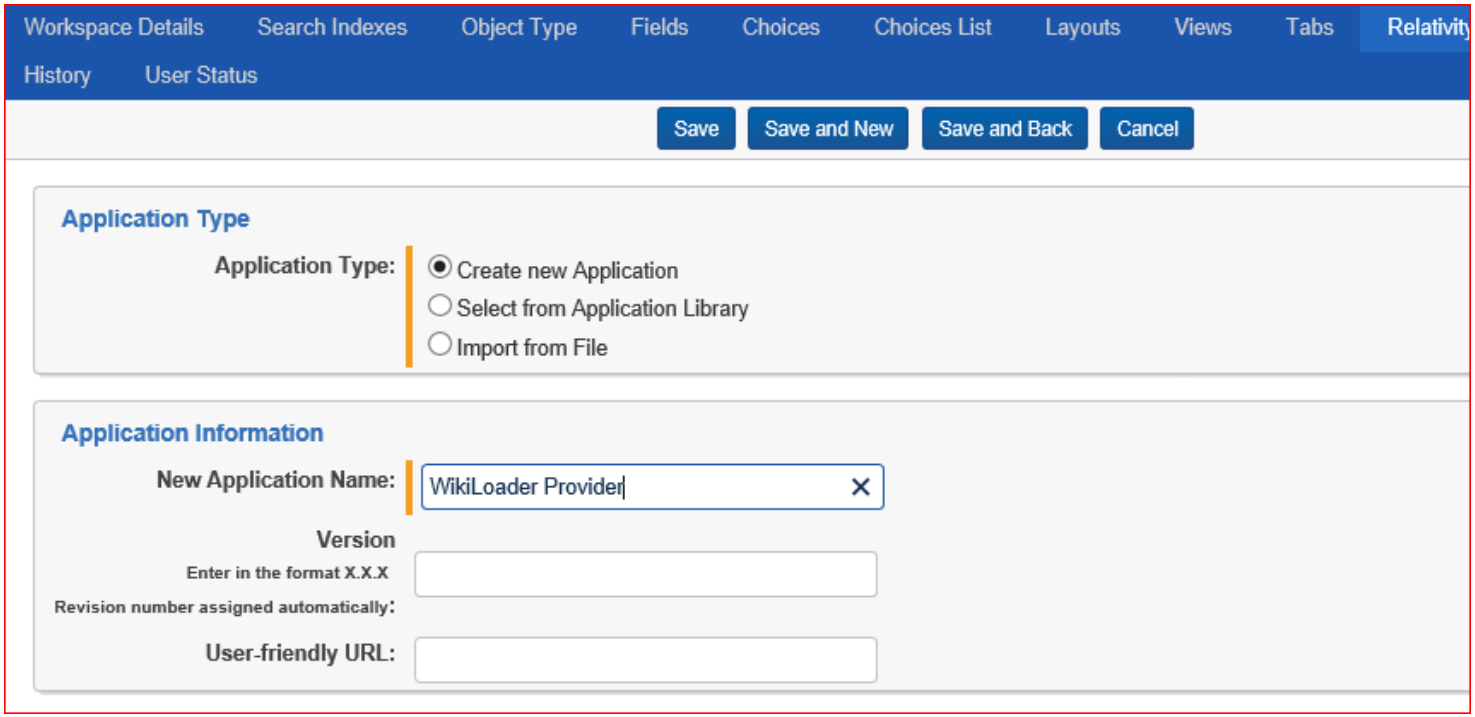
- Relativity Application
- Relativity Custom Page
- Integration Points Provider
- Integration Points Registration Event Handler
- Integration Points Un-Install Event Handler

5 Create Relativity Application

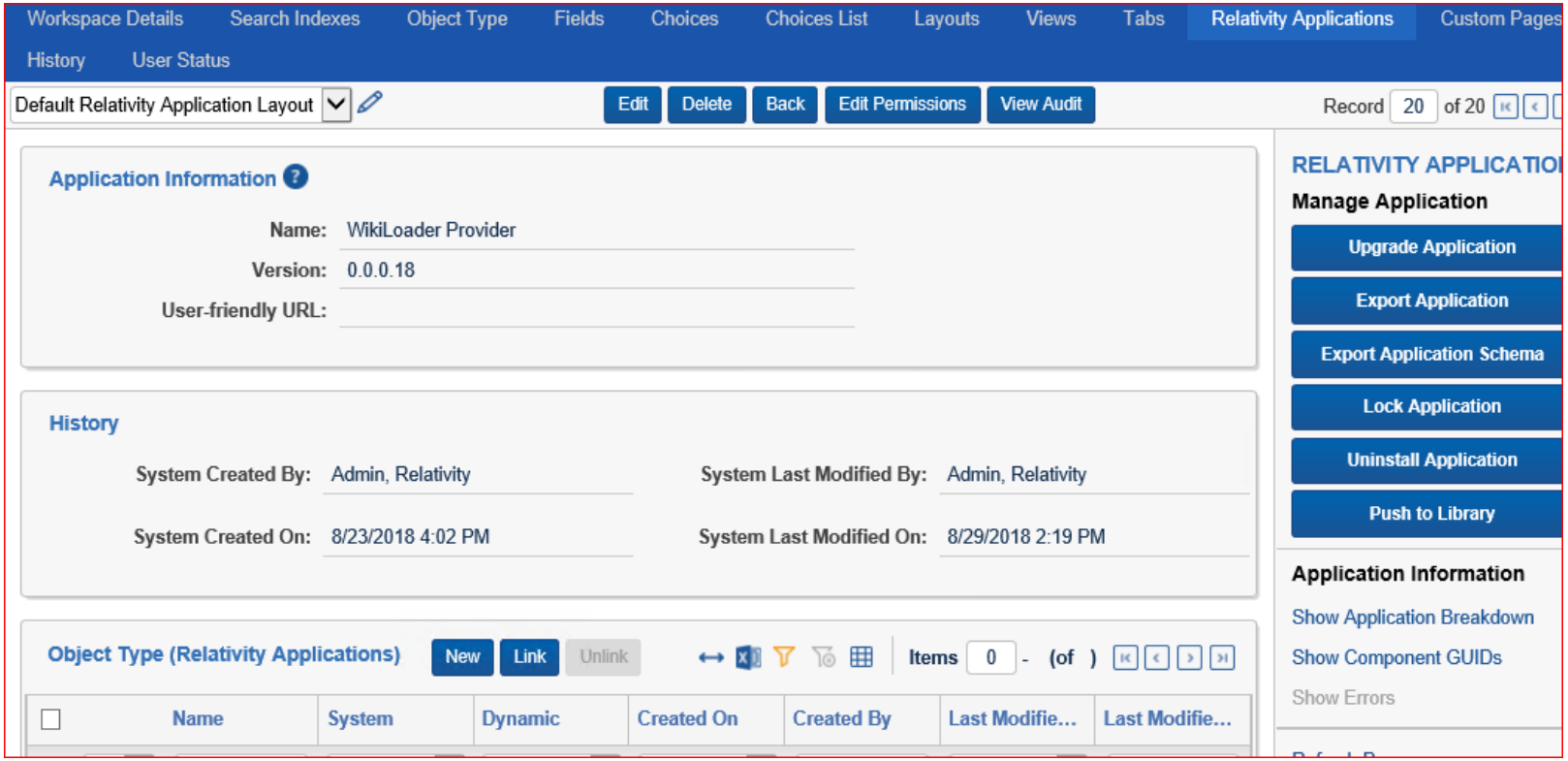
Before you begin writing the code for your integration point, you must create your Relativity application. You'll need the unique identifier (GUID) of the application and your custom page to implement the provider code.

5.1 Create Relativity Application

1. Log into Relativity.
2. Navigate to a workspace.
3. Click the Relativity Applications tab.
4. Click the New Relativity Application button.
5. Create a new application Named **WikiLoader Provider**



6. Click **Save** and you'll be taken to the Application Layout that lists all the Application's associated components.



5.2 Push Application to Application Library

Follow the steps below to push the application to the Application Library. This step is necessary because it allows resource files to be associated with the application.

1. Return to the **Application Layout** page.
2. Click the **Push to Library** button.

5.3 Copy Application GUID

To retrieve the unique identifier (GUID) of the application for your provider code, click the **Show Component GUIDs** link on the console under the **Application Information** section. Note: This link is only displayed while your Relativity environment has developer mode on.

6 Prepare Development Environment

The Relativity Integration Points (RIP) Framework requires a provider, two event handlers and a custom page. A Visual Studio Template is freely available on the Visual Studio Gallery to ease the creation of a RIP Provider. However, for the sake of this workbook we will build the provider from scratch.

In sections 7, 8, and 9 we will explain in detail the code we are now only pasting into our new projects.

6.1 Create Visual Studio Solution

1. Open Visual Studio 2017
2. Click **File** > **New** > **Project > Web > ASP.NET Web Application (.NET Framework)**.
3. Enter **WikiLoaderPage** in the **Name** box.
4. Change the Solution name to **WikiLoader**.
    - Note: you can leave the location at its default value.
    - Note: If it is not already, set the target framework to .NET Framework 4.6.2
5. Click **OK**.

New Project

Recent

Installed

Visual C#
- Windows Universal
- Windows Classic Desktop
- Web
- .NET Core
- .NET Standard
- Cloud
- Relativity
- Test
- WCF

Azure Data Lake

Stream Analytics

Other Languages

Other Project Types

Online

Not finding what you are looking for?

Open Visual Studio Installer

.NET Framework 4.6.2 | Sort by: Default | Search (Ctrl+E)

ASP.NET Core Web Application	Visual C#
ASP.NET Web Application (.NET Framework)	Visual C#

Type: Visual C#

Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Name: WikiLoaderPage

Location: D:\Source

Solution name: WikiLoader

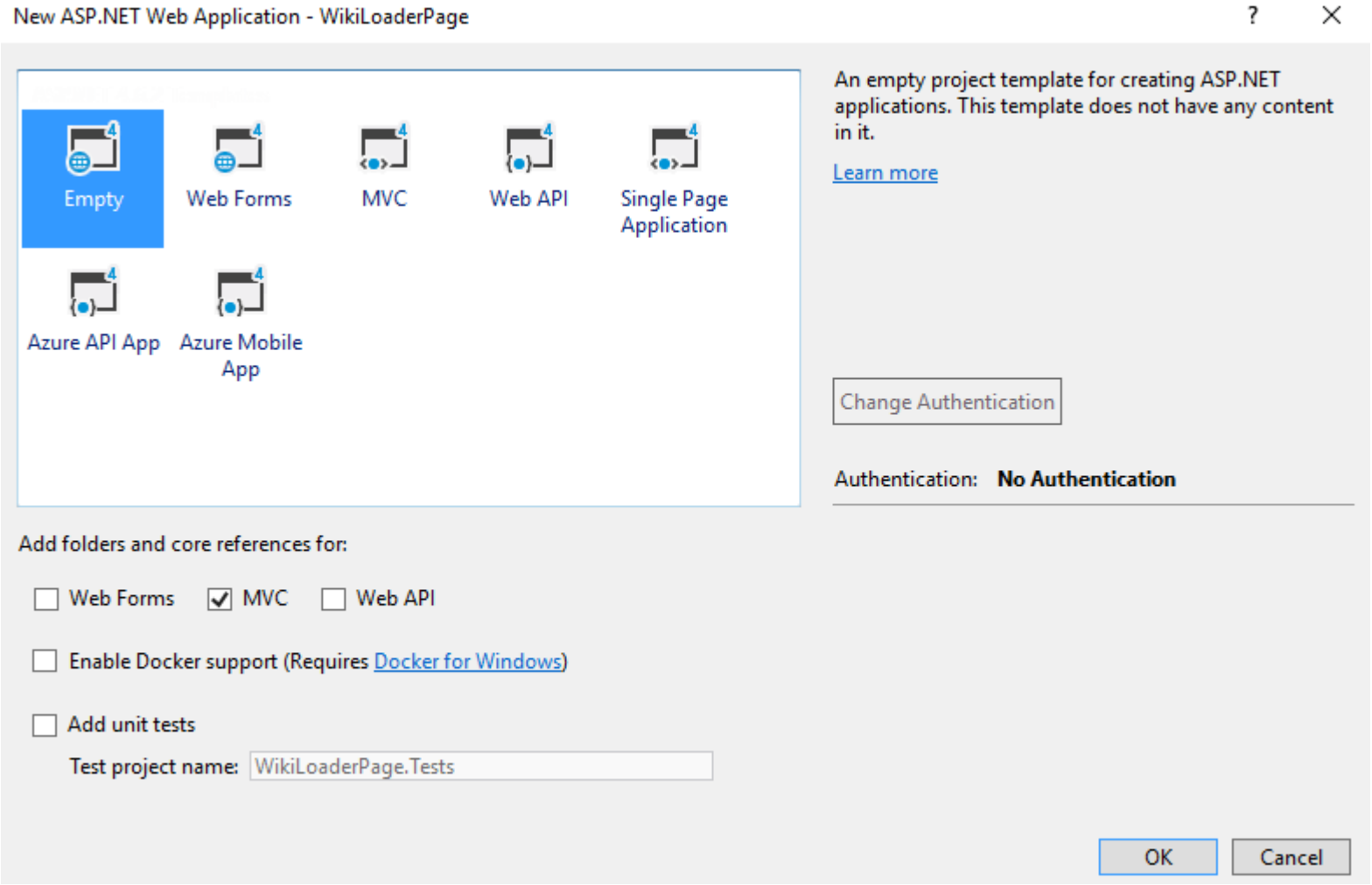
Browse...

☑ Create directory for solution

☐ Create new Git repository

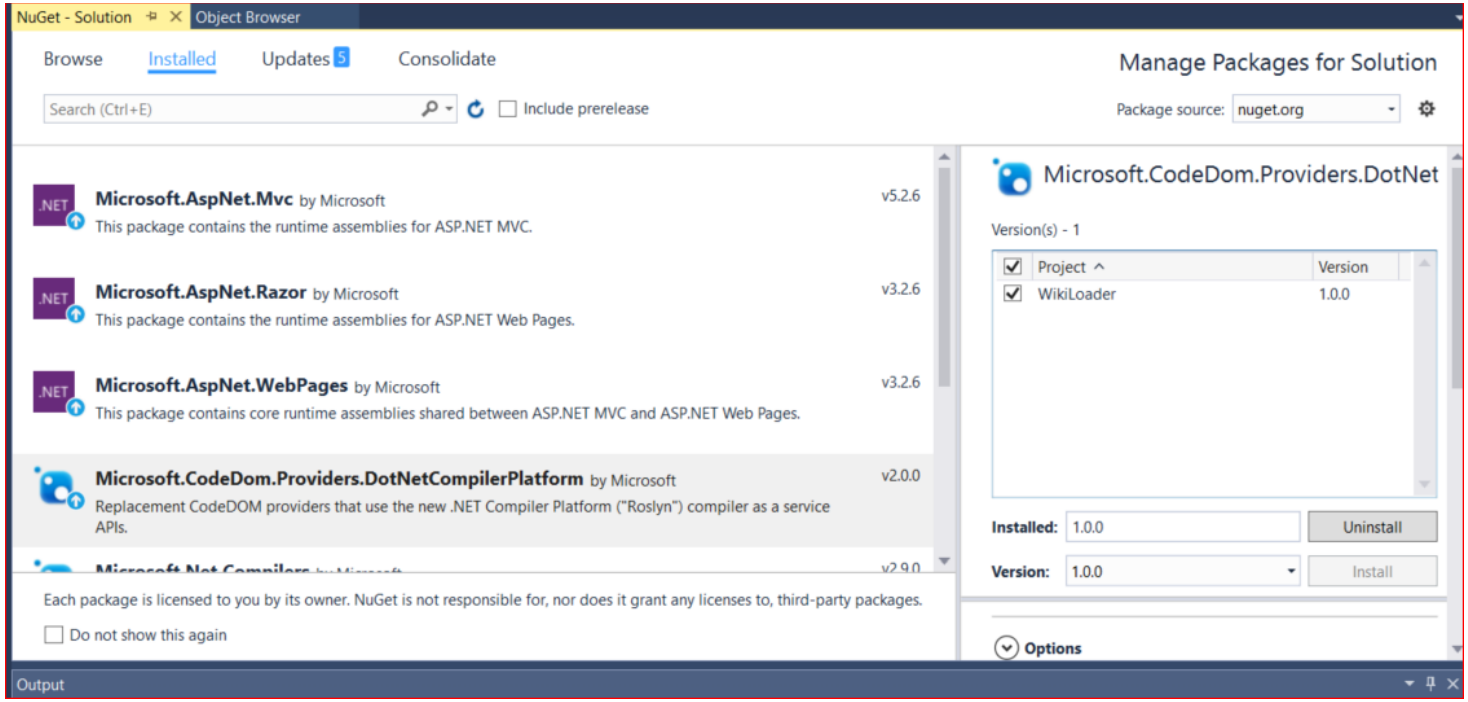
OK | Cancel

6. On the next screen, select an **Empty** template for your project
7. Check the **MVC** type.
8. Click **OK**.



9. Use these steps to uninstall the NuGet packages used for the .NET compilers:

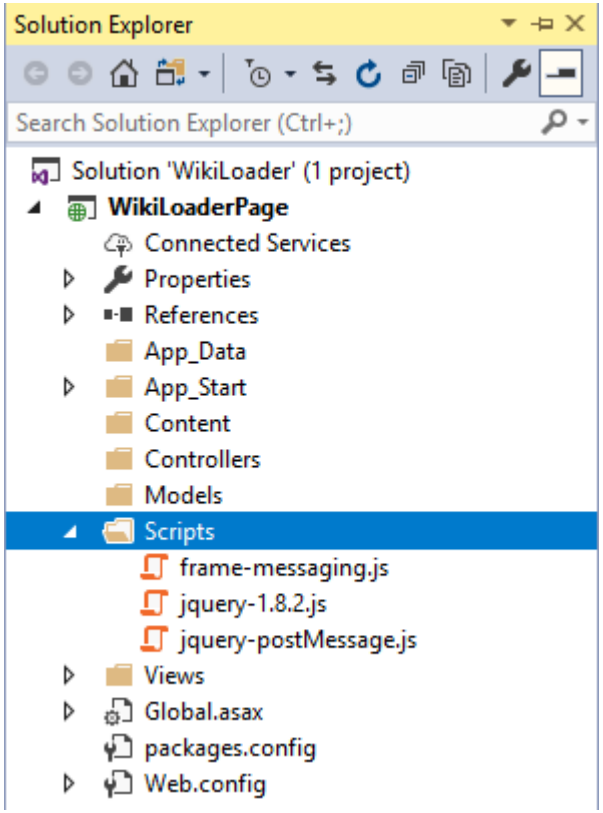
- Right-click on the **Solution 'WikiLoader'**, and click **Manage NuGet Packages for Solution**.
- Highlight the **Microsoft.CodeDom.Providers.DotNetCompilerPlatform**.
- Select the **WikiLoaderPage** project in the Versions table.
- Click **Uninstall**, and then click **OK**.
- Repeat these steps to remove the **Microsoft.Net.Compilers** package.



10. Right-click on the **WikiLoaderPage** project in the Solution Explorer. Click **Add** > **New Folder**, and add a folder called **Scripts**.

11. Follow the same steps to create a **Content** folder.

12. To add the required JavaScript files, right-click on the **Scripts** folder and click **Add** > **Existing Item**. Navigate to the folder where you extracted the Integration Points SDK files, and select the following files:
    - **frame-messaging.js**
    - **jquery-1.8.2.js**
    - **jquery-postMessage.js**



13. Create a new JavaScript file in the same folder by right-clicking **Scripts** and selecting Add -> New Item…
14. Select JavaScript file and name it **wikiloader-provider.js**
15. Paste the following code into the new JavaScript file and save it:

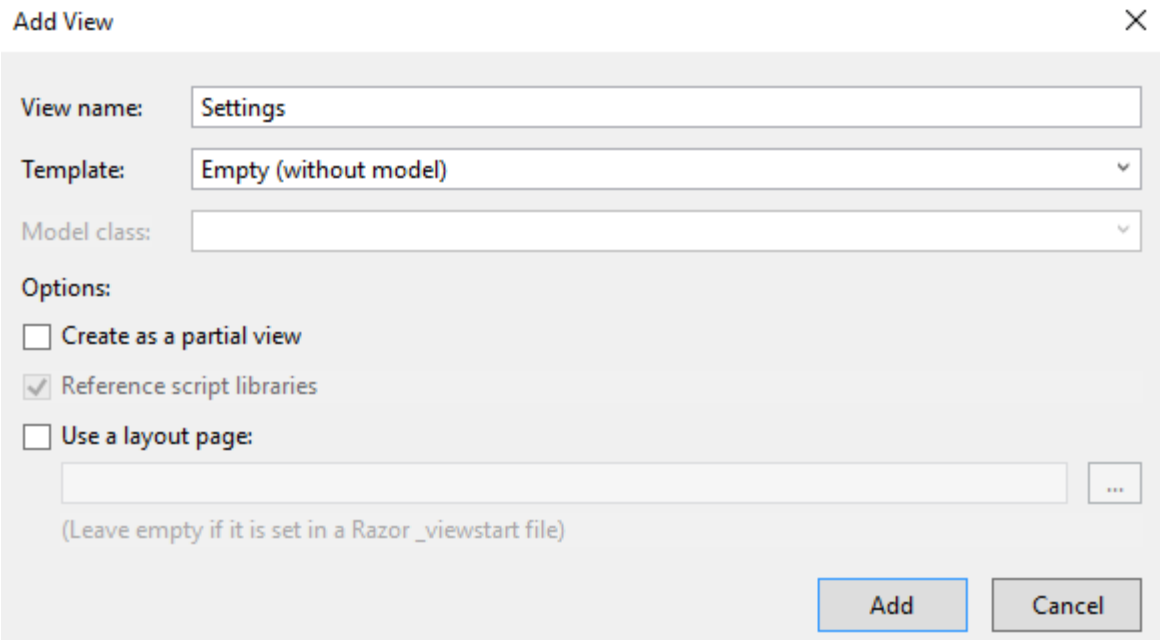
```
$(function () {
    //Create a new communication object that talks to the host page.
    var message = IP.frameMessaging();

    var _getModel = function () {
        return $('#fileLocation').val();
    };

    //An event raised when the user has clicked the Next or Save button.
    message.subscribe('submit', function () {
        //Execute save logic that persists the state.
        var localModel = _getModel();
        this.publish("saveState", localModel);
        //Communicate to the host page that it to continue.
        this.publish('saveComplete', localModel);
    });

    //An event raised when a user clicks the Back button.
    message.subscribe('back', function () {
        //Execute save logic that persists the state.
        this.publish('saveState', _getModel());
    });

    //An event raised when the host page has loaded the current settings page.
    message.subscribe('load', function (model) {
        $('#fileLocation').val(model);
    });
   });
```

16. To add the RIP dependencies for custom pages, right-click **References** under the **WikiLoaderPage** project, and then click **Add Reference** on the menu. Navigate to the folder where you installed the Relativity SDK and select the **Relativity.CustomPages.dll** file.
17. Create a new stylesheet file under the Content folder called **Site.css**. Paste the following code into it and save it:

```
.input-box
{
	height: 26px;
	border: 1px solid #c1c1c1;
	color: #102d4f;
	border-radius: 3px;
	width: 750px;
}

*{
	box-sizing: border-box;
}

html
{
	background-color: white;
	font-family: "Open Sans", sans-serif;
	font-weight: 400;
	font-size: 1em;
	color: #666666;
}

.card
{
	background-color: #f7f7f7;
	border: 1px solid #c1c1c1;
	border-radius: 3px;
	box-shadow: 1px 1px 3px #b7b7b7;
	padding: 10px;
	margin: 15px 12px 0 12px;
}

	.card .label
	{
		color: #3471b7;
		font-size: 13px;
		font-weight: 600;
	}

	.card .field-label
	{
		color: #404040;
		text-align: right;
		padding-right: 5px;
		min-width: 200px;
```

```
                        display: table-cell;
                        border-right: 3px solid transparent;
                        font-size: .85em;
                  }

                        .card .field-label.required
                        {
                              border-right: 3px solid #f59d1f;
                        }

                  .card .field-value
                  {
                        display: table-cell;
                        padding-left: 5px;
                        font-size: 0.85em;
                  }

                  .card .field-row
                  {
                        padding: 5px 0;
      }
```

18. Next, create an MVC Controller by right-clicking the **Controllers** folder and selecting Add -> Controller…
19. Select **MVC 5 Controller – Empty** and call it **ProviderController**.
20. Rename the public method in your new Controller from **Index** to **Settings**
21. Right-click on the new folder under **Views -> Provider** and select Add -> View…
22. Name the new View **Settings** and uncheck the box **Use a layout page**



23. Paste the following code into the new View:

```
@{
    ViewBag.Title = "Settings";
}

<link href="~/Content/site.css" rel="stylesheet" />
<div class="card">

    <label class="section label">WikiLoader</label>
    <div class="field-row">
        <div class="field-label required">
            File Location:
        </div>
        <div class="field-value ">
            <input type="text" name="field" id="fileLocation" class="input-box" placeholder="Please enter a file
location" />
        </div>
    </div>
</div>

   <script src="~/Scripts/jquery-1.8.2.js"></script>
   <script src="~/Scripts/jquery-postMessage.js"></script>
   <script src="~/Scripts/frame-messaging.js"></script>
   <script src="~/Scripts/wikiloader-provider.js"></script>
```

24. Add a new folder under Controller called **API**

25. Create a new Controller under the API folder just like the **ProviderController** and call it **ProviderAPIController**

26. Copy the following code into the file and save it:

```
using System.Collections.Generic;
using System.Net;
using System.Net.Http;
using System.Web.Http;

namespace Web.Controllers.API
{
    public class ProviderAPIController : ApiController
    {
        [HttpPost]
        public HttpResponseMessage GetViewFields([FromBody] object data)
        {
            string fileLocation = data.ToString();
            var model = new List<KeyValuePair<string, string>>()
            {
                new KeyValuePair<string, string>("File Location", fileLocation)
            };

            return Request.CreateResponse(HttpStatusCode.OK, model);
        }
    }
   }
```

27. Remove the ProviderAPI folder created under View.
28. Create a new route config by right-clicking the **App_Start** folder and selecting Add -> New Item…
29. Select **Class** as the item type and call it **WebAPIConfig**
30. Copy the following code into the file and save it:

31.

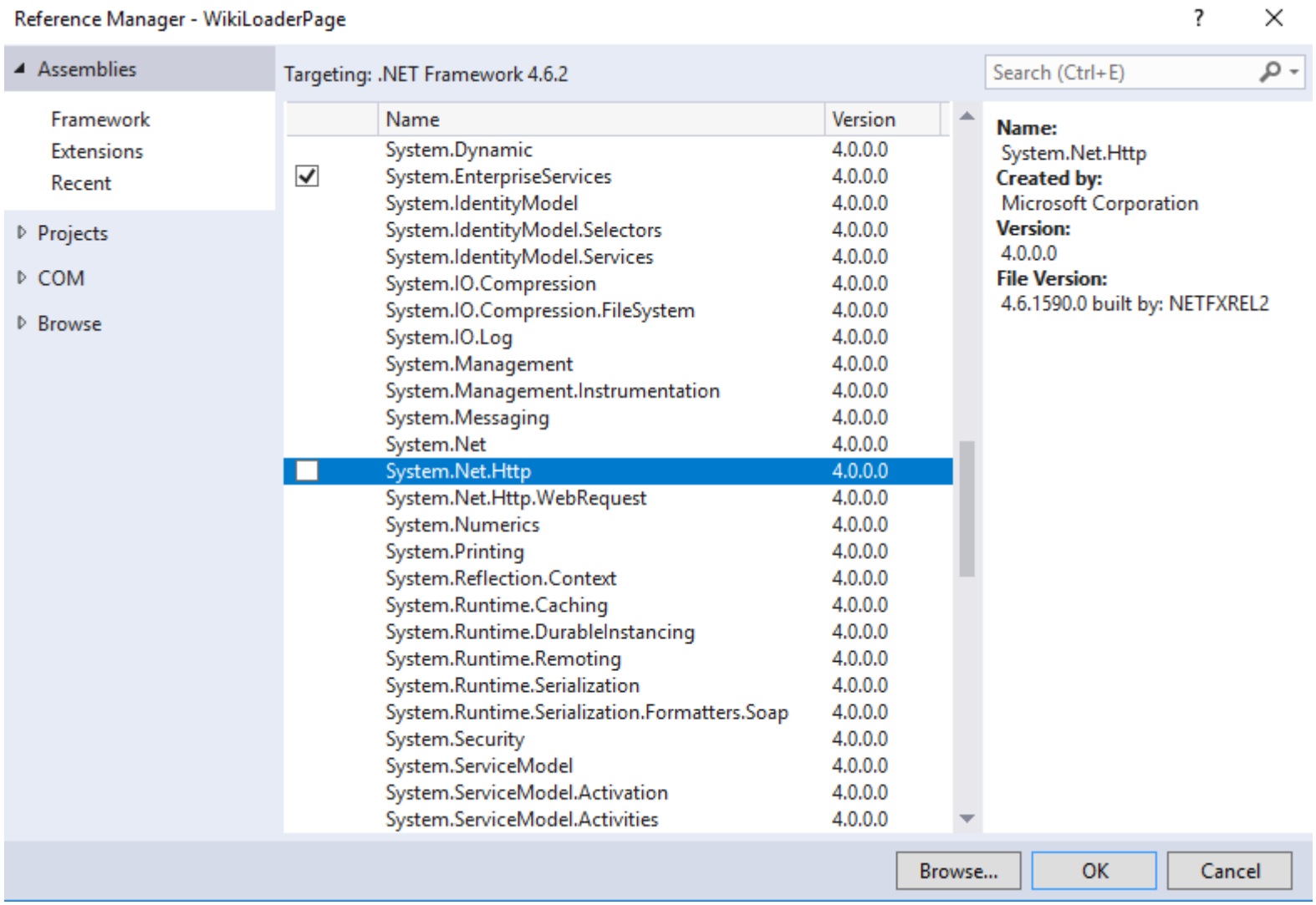
```
using System.Web.Http;

namespace Web
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            config.Routes.MapHttpRoute(
                        name: "ProviderViewSettings",
                        routeTemplate: "{workspaceID}/api/ProviderAPI/GetViewFields",
                        defaults: new { controller = "ProviderAPI", action = "GetViewFields" }
                    );
        }
    }
}
```

32. Add the required references from Nuget by the following steps:

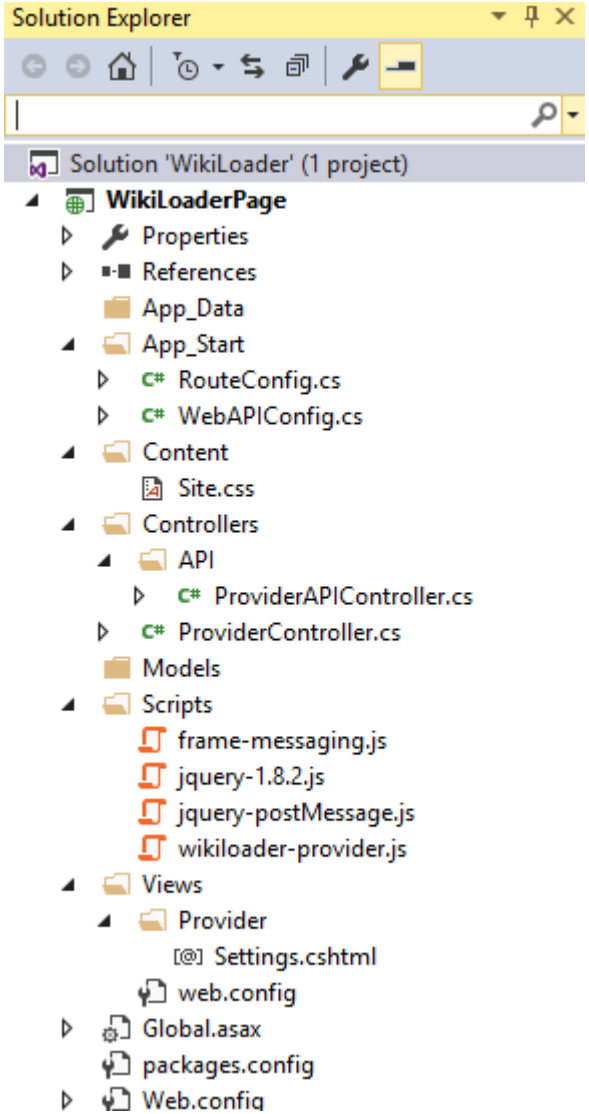
- Right-click the solution and select **Manage Nuget Packages for Solution…**
- Select **Browse** and search for **ASP.NET WebAPI Core**
- Select **Microsoft.AspNet.WebApi.Core** and install it into your project

33. Also add the **System.Net.Http** reference by the following steps:

- Right-click on the **References** folder
- Select the **Assemblies** dropdown on the left column
- Scroll down and check the box next to System.Net.Http



34. You should now have the following solution structure:



6.2 Create WikiLoaderProvider Project

Now that you've created the entire Custom Page as a single project, we'll create a new project for the Provider and Event Handlers. This will give us a separate assembly which we'll need because custom pages have a different installation process than single assemblies.

1. In Solution Explorer, right-click your solution **'WikiLoader'**, and then click **Add** > **New Project**.
2. Select the **Class Library (.NET Framework)** project type and enter **WikiLoaderProvider** in the Name field.

Add New Project

- Recent
- Installed
  - Visual C#
    - Windows Universal
    - Windows Classic Desktop
    - Web
      - Web Site
    - .NET Core
    - .NET Standard
    - Cloud
    - Relativity
    - Test
    - WCF
  - Azure Data Lake
  - Stream Analytics
  - Other Languages
- Online

Not finding what you are looking for?
Open Visual Studio Installer

.NET Framework 4.6.2 | Sort by: Default | Search (Ctrl+E)

WPF App (.NET Framework)	Visual C#
Windows Forms App (.NET Framework)	Visual C#
Console App (.NET Framework)	Visual C#
Class Library (.NET Framework)	Visual C#
Shared Project	Visual C#
Windows Service (.NET Framework)	Visual C#
Empty Project (.NET Framework)	Visual C#
WPF Browser App (.NET Framework)	Visual C#
WPF Custom Control Library (.NET Framework)	Visual C#
WPF User Control Library (.NET Framework)	Visual C#
Windows Forms Control Library (.NET Framework)	Visual C#

Type: Visual C#

A project for creating a C# class library (.dll)

Name: WikiLoaderProvider

Location: D:\Source

Browse...

OK Cancel

3. Add references to the following required DLLs:
    - **KCura.IntegrationPoints.SourceProviderInstaller.dll**
    - **KCura.IntegrationPoints.Contracts.dll**
    - **kCura.EventHandler.dll**

Note: The Event Handler DLL is in the Relativity SDK and the two Integration Points DLLs are in the Integration Points SDK.

6.3 Rename Default Class Name

1. Rename the **Class1.cs** file to **WikiLoaderProvider.**cs in the **WikiLoaderProvider** project.
2. Paste the following code into the class (yes, it's a lot. We'll explain more later):

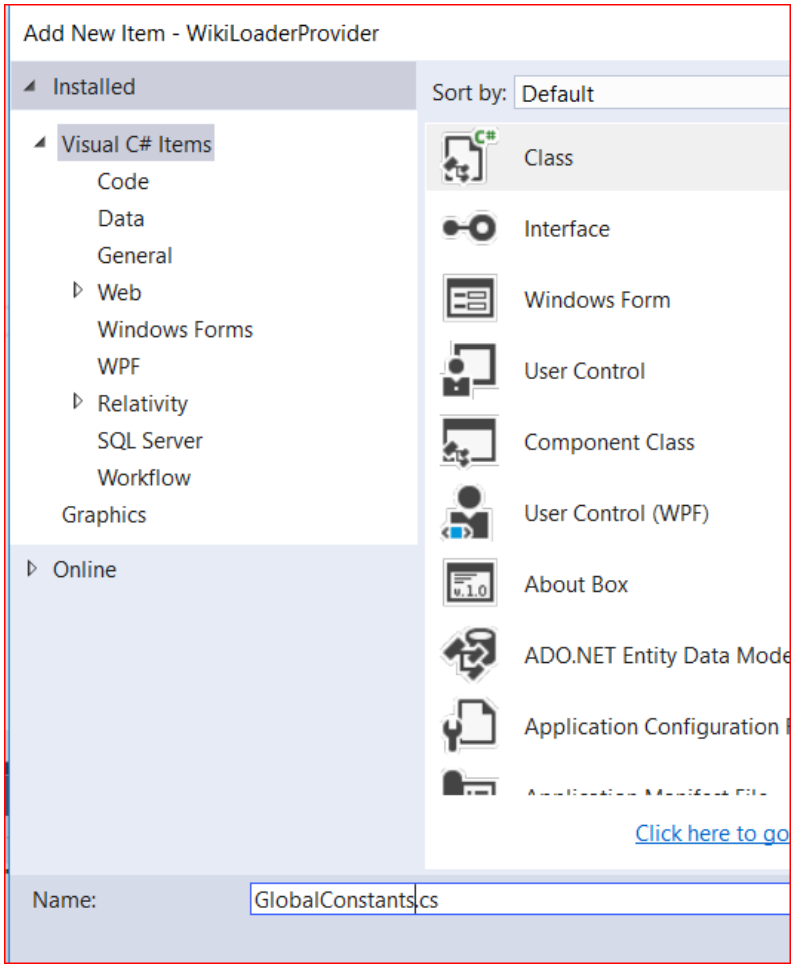
```
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Xml;
using kCura.IntegrationPoints.Contracts.Models;
using WikiLoaderProvider;

namespace WikiLoaderProvider
{
    /// <summary>
    /// This code is a sample fully operational Integration Point Provider
    /// for demonstration purposes only
    /// </summary>
    [kCura.IntegrationPoints.Contracts.DataSourceProvider(GlobalConstants.WIKILOADER_GUID)]
    public class WikiLoaderProvider : kCura.IntegrationPoints.Contracts.Provider.IDataSourceProvider
    {
        public IEnumerable<FieldEntry> GetFields(DataSourceProviderConfiguration providerConfiguration)
        {
            string fileLocation = providerConfiguration.Configuration;

            // Because the wikipedia XML doesn't have a column key, I've hard-coded the field names here.
            var fieldEntries = new List<FieldEntry>()
            {
                new FieldEntry { DisplayName = "Title", FieldIdentifier = "title", IsIdentifier = true },
                new FieldEntry { DisplayName = "URL", FieldIdentifier = "url", IsIdentifier = false },
                new FieldEntry { DisplayName = "Abstract", FieldIdentifier = "abstract", IsIdentifier = false }
            };

            return fieldEntries;
        }

        public IDataReader GetBatchableIds(FieldEntry identifier, DataSourceProviderConfiguration
providerConfiguration)
        {
            string fileLocation = providerConfiguration.Configuration;

            DataTable dt = new DataTable();
            dt.Columns.Add(identifier.FieldIdentifier);

            XmlDocument doc = new XmlDocument();
            doc.Load(fileLocation);
            XmlNodeList nodes = doc.DocumentElement.SelectNodes(string.Format("/feed/doc/{0}",
identifier.FieldIdentifier));

            foreach (XmlNode node in nodes)
```

```
                {
                    var row = dt.NewRow();
                    row[identifier.FieldIdentifier] = node.InnerText;
                    dt.Rows.Add(row);
                }
                return dt.CreateDataReader();
            }

            public IDataReader GetData(IEnumerable<FieldEntry> fields, IEnumerable<string> entryIds,
    DataSourceProviderConfiguration providerConfiguration)
            {
                string fileLocation = providerConfiguration.Configuration;
                List<string> fieldList = fields.Select(f => f.FieldIdentifier).ToList();
                string keyFieldName = fields.FirstOrDefault(f => f.IsIdentifier).FieldIdentifier;
                return new XMLDataReader(entryIds, fieldList, keyFieldName, fileLocation);
            }
        }
       }
```

6.4 Define global constants

In your Visual Studio solution, you need to define global constants for your provider class, and to store the application GUID that Relativity generated when you created your application.

1. To add a new class for global constants, right-click the **WikiLoaderProvider** project, and then click **Add** > **Class**. Enter **GlobalConstants** in the **Name** box and click **Add.**



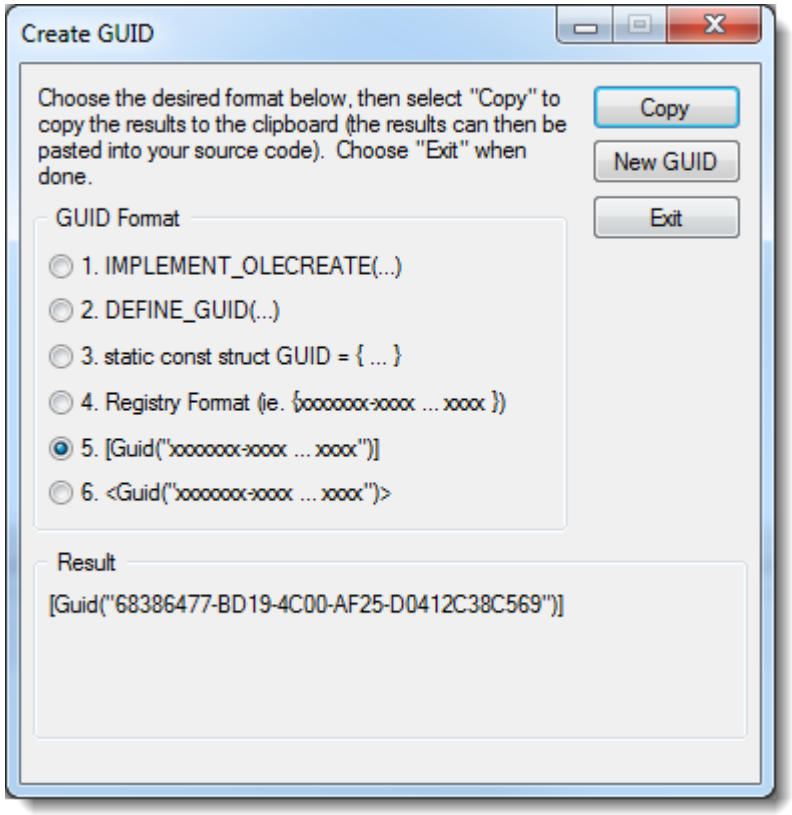
2. For the sake of this example, we will paste the following code into the **GlobalConstants** class. However, you will want to generate your own GUID in a new provider project.

```
namespace WikiLoaderProvider
{
    public class GlobalConstants
    {
        public const string WIKILOADER_GUID = "1763A7C5-9A9E-4FF0-B224-C4F6D031E662";
        public const string APPLICATION_GUID = "8A2B6B81-534D-4B8F-B1B5-AF81DD611883";
    }
}
```

For future reference, these are the steps to generate your own GUID:

a. To create a GUID for your provider, click **Tools**, and select **Create GUID** from the menu.

b. Select the **5. [Guid("xxxxxxx-xxxx...xxxx")]**, and then click **Copy**.



c. Open your GlobalConstants class and then paste the GUID in the assignment statement for the WIKILOADER_GUID constant.

d. Paste your application GUID in the assignment statement for APPLICATION_GUID constant. To review how to retrieve the application GUID, go back to workbook step 5.3.

3. Add a new folder called **EventHandlers**
4. Add a new class under the EventHandlers folder and call it **RegisterWikiLoaderProvider**
5. Paste the following code into the class:

```
        using WikiLoaderProvider;
        using System;
        using System.Collections.Generic;
        using System.Runtime.InteropServices;

        namespace WikiLoaderProvider.EventHandlers
        {
            [kCura.EventHandler.CustomAttributes.Description("Update WikiLoader Provider - On Every Install")]
            [kCura.EventHandler.CustomAttributes.RunOnce(false)]
            [Guid("0CD3CCEC-2133-4E9F-821B-7FE4E0480880")]
            public class RegisterWikiLoaderProvider :
        kCura.IntegrationPoints.SourceProviderInstaller.IntegrationPointSourceProviderInstaller
            {
                public override IDictionary<Guid, kCura.IntegrationPoints.SourceProviderInstaller.SourceProvider>
        GetSourceProviders()
                {
                    Dictionary<Guid, kCura.IntegrationPoints.SourceProviderInstaller.SourceProvider> sourceProviders = new
        Dictionary<Guid, kCura.IntegrationPoints.SourceProviderInstaller.SourceProvider>();
                    var WikiLoaderProviderEntry = new kCura.IntegrationPoints.SourceProviderInstaller.SourceProvider();
                    WikiLoaderProviderEntry.Name = "WikiLoader Provider";
                    WikiLoaderProviderEntry.Url = string.Format("/%applicationpath%/CustomPages/{0}/Provider/Settings",
        GlobalConstants.APPLICATION_GUID);
                    WikiLoaderProviderEntry.ViewDataUrl =
        string.Format("/%applicationpath%/CustomPages/{0}/%appId%/api/ProviderAPI/GetViewFields",
        GlobalConstants.APPLICATION_GUID);
                    sourceProviders.Add(Guid.Parse(GlobalConstants.WIKILOADER_GUID), WikiLoaderProviderEntry);

                    return sourceProviders;
                }
            }
}
```

6. Add another new class under the EventHandlers folder and call it **RemoveWikiLoaderProvider**
7. Paste the following code into the class:

8.

```
using System.Runtime.InteropServices;

namespace WikiLoaderProvider.EventHandlers
{
    [kCura.EventHandler.CustomAttributes.Description("Update WikiLoader provider - Uninstall")]
    [kCura.EventHandler.CustomAttributes.RunOnce(false)]
    [Guid("5BAE6D1B-9F47-4C65-949B-1B57DF535B57")]
    public class RemoveWikiLoaderProvider :
kCura.IntegrationPoints.SourceProviderInstaller.IntegrationPointSourceProviderUninstaller
    {
    }
}
```

9. For both classes in the EventHandlers folder, generate new GUIDs to replace the existing GUID attributes.
10. Finally, create a class called **XMLDataReader** and paste the following code into it (last time, I promise):

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Xml;

namespace WikiLoaderProvider
{
    public class XMLDataReader : IDataReader
    {
        private DataTable _dataTable;
        private bool _readerOpen;
        private int _position = 0;
        private IEnumerator<string> _itemsEnumerator;
        private List<string> _fields;
        private string _xmlFilePath;
        private XmlDocument _xmlDocument;
        private XmlNode _currentDataNode;
        private string _keyFieldName;

        public XMLDataReader(IEnumerable<string> itemIds, List<string> fields, string keyFieldName, string xmlFilePath)
        {
            _xmlFilePath = xmlFilePath;
            _readerOpen = true;
            _itemsEnumerator = itemIds.GetEnumerator();
```

```
            _fields = fields;

            _dataTable = new DataTable();
            _dataTable.Columns.AddRange(fields.Select(f => new DataColumn(f)).ToArray());
            _keyFieldName = keyFieldName;
        }

        public void Close()
        {
            _readerOpen = false;
            return;
        }

        public int Depth
        {
            get { return 0; }
        }

        public DataTable GetSchemaTable()
        {
            return _dataTable;
        }

        public bool IsClosed
        {
            get { return _readerOpen; }
        }

        public bool NextResult()
        {
            return false;
        }

        public bool Read()
        {
            if (_readerOpen)
            {
                _readerOpen = _itemsEnumerator.MoveNext();
                if (_readerOpen)
                {
                    if (_xmlDocument == null)
                    {
                        _xmlDocument = new XmlDocument();
                        _xmlDocument.Load(_xmlFilePath);
                    }
```

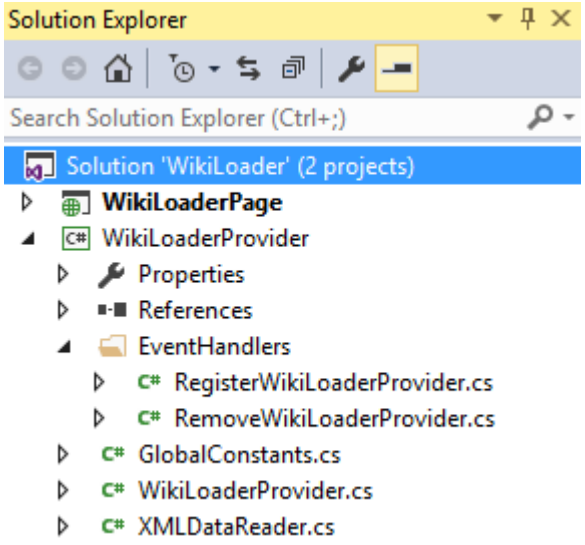
```
                    string xpath = string.Format("/feed/doc[{0}='{1}']", _keyFieldName, _itemsEnumerator.Current);
                    XmlNodeList nodes = _xmlDocument.DocumentElement.SelectNodes(xpath);
                    _currentDataNode = nodes[0];
                    _position++;
                }
            }
            return _readerOpen;
        }

        public int RecordsAffected
        {
            get { return _position; }
        }

        public void Dispose()
        {
            _readerOpen = false;
            _dataTable.Dispose();
            _itemsEnumerator.Dispose();
        }

        public int FieldCount
        {
            get { return _dataTable.Columns.Count; }
        }

        public bool GetBoolean(int i)
        {
            return Convert.ToBoolean(GetValue(i));
        }

        public byte GetByte(int i)
        {
            return Convert.ToByte(GetValue(i));
        }

        public long GetBytes(int i, long fieldOffset, byte[] buffer, int bufferoffset, int length)
        {
            throw new System.NotImplementedException();
        }

        public char GetChar(int i)
        {
            return Convert.ToChar(GetValue(i));
        }
```

```
        public long GetChars(int i, long fieldoffset, char[] buffer, int bufferoffset, int length)
        {
            throw new System.NotImplementedException();
        }

        public IDataReader GetData(int i)
        {
            throw new System.NotImplementedException();
        }

        public string GetDataTypeName(int i)
        {
            return _dataTable.Columns[i].DataType.Name;
        }

        public System.DateTime GetDateTime(int i)
        {
            return Convert.ToDateTime(GetValue(i));
        }

        public decimal GetDecimal(int i)
        {
            return Convert.ToDecimal(GetValue(i));
        }

        public double GetDouble(int i)
        {
            return Convert.ToDouble(GetValue(i));
        }

        public System.Type GetFieldType(int i)
        {
            return _dataTable.Columns[i].DataType;
        }

        public float GetFloat(int i)
        {
            return Convert.ToSingle(GetValue(i));
        }

        public System.Guid GetGuid(int i)
        {
            return Guid.Parse(GetValue(i).ToString());
        }
```

```
public short GetInt16(int i)
{
    return Convert.ToInt16(GetValue(i));
}

public int GetInt32(int i)
{
    return Convert.ToInt32(GetValue(i));
}

public long GetInt64(int i)
{
    return Convert.ToInt64(GetValue(i));
}

public string GetName(int i)
{
    return _dataTable.Columns[i].ColumnName;
}

public int GetOrdinal(string name)
{
    return _dataTable.Columns[name].Ordinal;
}

public string GetString(int i)
{
    return GetValue(i) as string;
}

public object GetValue(int i)
{
    return _currentDataNode.SelectSingleNode(_fields[i]).InnerText;
}

public int GetValues(object[] values)
{
    if (values != null)
    {
        int fieldCount = Math.Min(values.Length, _fields.Count);
        object[] Values = new object[fieldCount];
        for (int i = 0; i < fieldCount; i++)
        {
            Values[i] = GetValue(i);
```

```
                }
                Array.Copy(Values, values, this.FieldCount);
                return fieldCount;
            }
            return 0;
        }

        public bool IsDBNull(int i)
        {
            return (GetValue(i) is System.DBNull);
        }

        public object this[string name]
        {
            get { return GetValue(_fields.IndexOf(name)); }
        }

        public object this[int i]
        {
            get { return GetValue(i); }
        }
    }
}
```

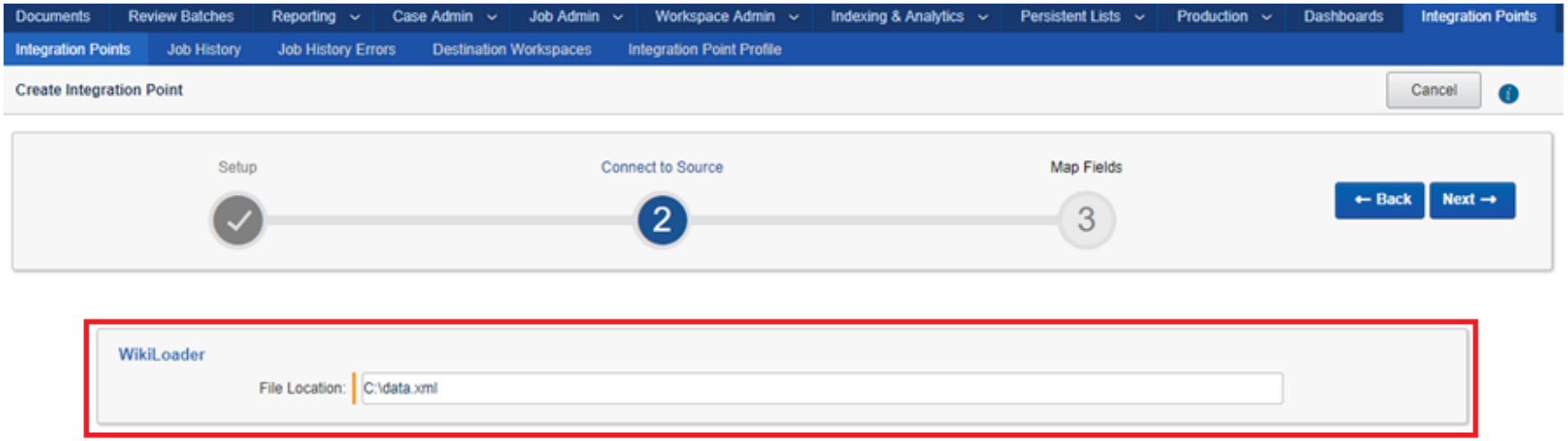
11. Your WikiLoaderProvider project should now have this layout:



7 WikiLoader Description - Custom Page

The Custom page will be used on the second step in the Relativity Integration Points (RIP) wizard for your source provider. It is loaded as a nested frame in the RIP framework navigation and serves as the user interface to enter parameters.

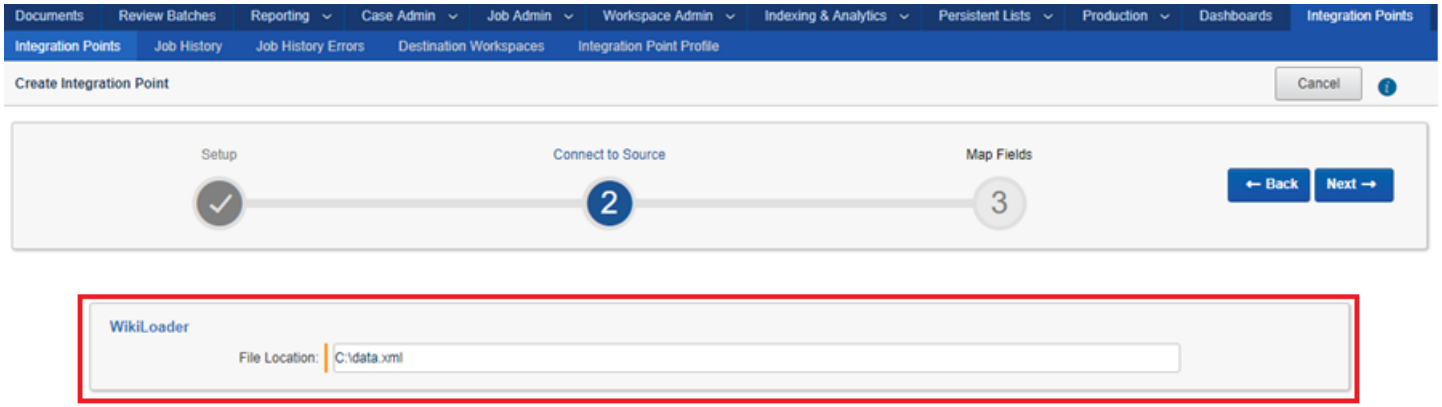
The custom page for this project is outlined in red below:



7.1 Provider Controller

7.1.1 Actions – Provider Settings

The chosen action of the Provider Controller will load the view used to configure the job (in this case, the action is called **Settings**). The action url path is specified in **Provider.EventHandlers.RegisterWikiLoaderProvider** and is set at application install. Registering the action location allows the Relativity Integration Points (RIP) Framework to direct users to your custom page.



In the screenshot above our view gives the user the ability to enter the full path to the XML file they wish to upload. Note: the XML file must be present on the same server that Relativity Integration Points runs its pages on, i.e. the Web Server.

7.1.2 Actions – Provider API

The second action that the Relativity Integration Points (RIP) framework needs on your custom page is a POST request to get the user fields (in this case, the action is called **GetViewFields**). Like the action to retrieve the view, this action's URL path is specified in **Provider.EventHandlers.RegisterWikiLoaderProvider**. This action is used to specify which settings are displayed on the Integration Points Job page.

7.1.3 CSS Layout

Implementing standard Relativity CSS files can be an effective technique to keep the look and feel of your custom page consistent with the Relativity environment where it's deployed. In our example, we've created our own style sheet with the necessary attributes to mimic Relativity, but you might also reference the buttermilk stylesheet at "../../../../SASS/buttermilk.min.css".

7.1.4 Settings View

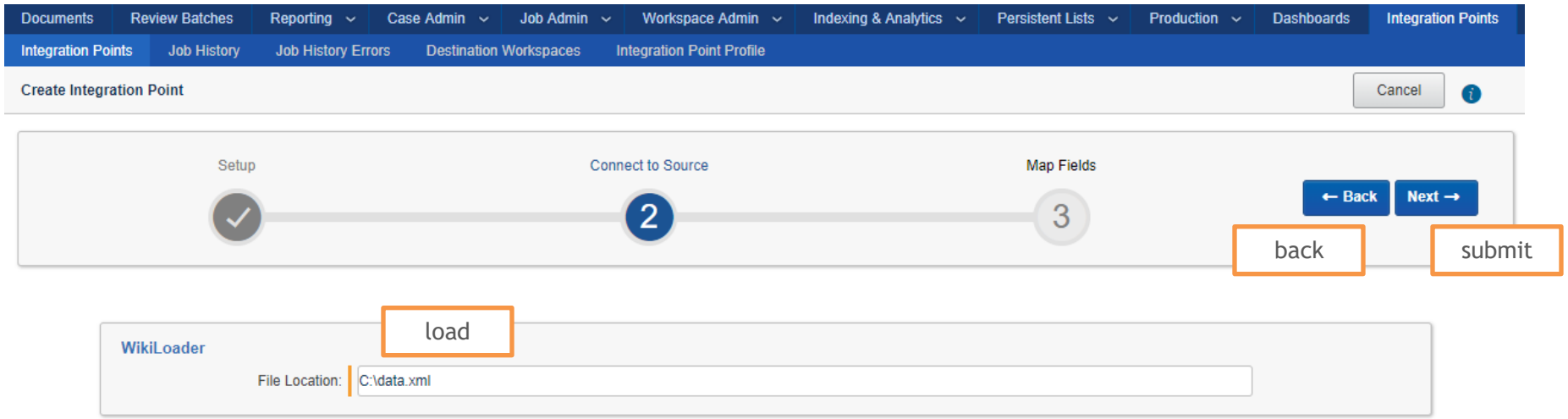
This HTML is inserted as an iFrame into the second Integration Point screen. A few lines to note:

1. We reference our **site.css** style sheet. This allows us to mimic the Relativity interface by using the same styles. These styles are seen in the class attributes of various HTML elements. Relativity uses styles over specific HTML elements to render structure to the page, hence the proliferation of DIV tags.
2. We reference all the JavaScript files in the Integration Point SDK and our provider script. The order of the scripts matter because subsequent scripts depend on jQuery.

7.1.5 Provider Javascript File

The frame-message.js script gives you access to a helper object called IP.FrameMessaging. We'll look at three events that you must subscribe to for your custom page to be wired with the Back and Next buttons on the page, and to load the user's entries when they return to your custom page.

- The **submit** event allows you to store the user's entries before moving to the next page, then initiates the frame switch.
- The **back** event allows you to store the user's entries before returning to the previous page.
- The **load** event allows you to replace the user's entries on the page when they've navigated to after moving forward or backward.



8 WikiLoader Description - Event Handlers

The Relativity Integration Points (RIP) Framework requires each custom provider to include an Install and Uninstall event handler. The Install event handler can be used to execute any pre-installation logic but is mainly used to register your custom provider with the main RIP Application. The registration process adds your provider as a selectable option in the main RIP Application. The Uninstall event handler can be used to execute logic while the provider is being uninstalled and is required regardless if it includes any custom logic.

8.1 Install Event Handler

The first event handler we will need is the installation event handler. For an Integration Points provider that has additional dependencies, this is the place you would set those dependencies up. The following are required:

- An attribute to set the description (this displays in the application event handler object list)
- An attribute to set whether the event handler fires each time the application is installed into a workspace, or only the first time.
- An attribute to uniquely identify the event handler (a GUID)
- A dictionary to enter the following values:
    - The name of the provider (this determines what the provider is called in the Source dropdown)
    - A path to the Settings action and the GetViewFields action (more on these actions under the Custom Page section later)
    - The provider's unique identifier

8.2 Uninstall Event Handler

The second event handler we will need is the uninstall event handler. It has the same attributes as the install event handler. The standard Integration Points objects – the application, custom page, event handlers, and provider – will be uninstalled by the framework, but you can add additional logic here to uninstall additional dependencies you may have added in the install event handler code.

9 WikiLoader Description - Data Source Provider

The Data Source Provider is the main class that the Relativity Integration Points (RIP) Framework uses to control how data is imported. The Data Source Provider must implement the **IDataSourceProvider** interface which requires the **GetFields()**, **GetBatchableIDs()** & **GetData()** methods. Carefully read the descriptions and expectations of each method because they run at different times and using different processes. Knowing more about when and where the methods run will help you optimize the design of your application and successfully debug (Section 16) any issues.

- **GetFields()** runs on the webserver to retrieve the fields the user will match with Relativity fields. It should be designed to return all the possible fields available to be mapped from your provider. The DataSourceProviderConfiguration object passed as an argument comes from your custom page and has already been serialized into the Configuration method as a string.

- **GetBatchableIDs()** runs on the Agent Server after a user's job is fully configured and the user starts the job. The RIP Agent expects to return every identifier in your dataset. The identifiers will be passed in smaller batches to RIP Agents simultaneously to optimize the data import and carry it out in a parallel fashion. **GetBatchableIDs()** is executed only once per job. The FieldEntry object passed as an argument and matches the FieldEntry object specified in your **GetFields()** method as the unique identifier (IsIdentifier == true).

- **GetData()** runs on the Agent Server after the identifiers are retrieved by the **GetBatchableIDs()** method. A small batch of identifiers are passed to the **GetData()** method and the method is executed multiple times between multiple agents simultaneously until the data is fully imported. The enumerable FieldEntry objects are the list of fields selected by the user on the field mapping screen and the enumerable strings are the current batch of unique identifiers to be processed by the RIP framework.

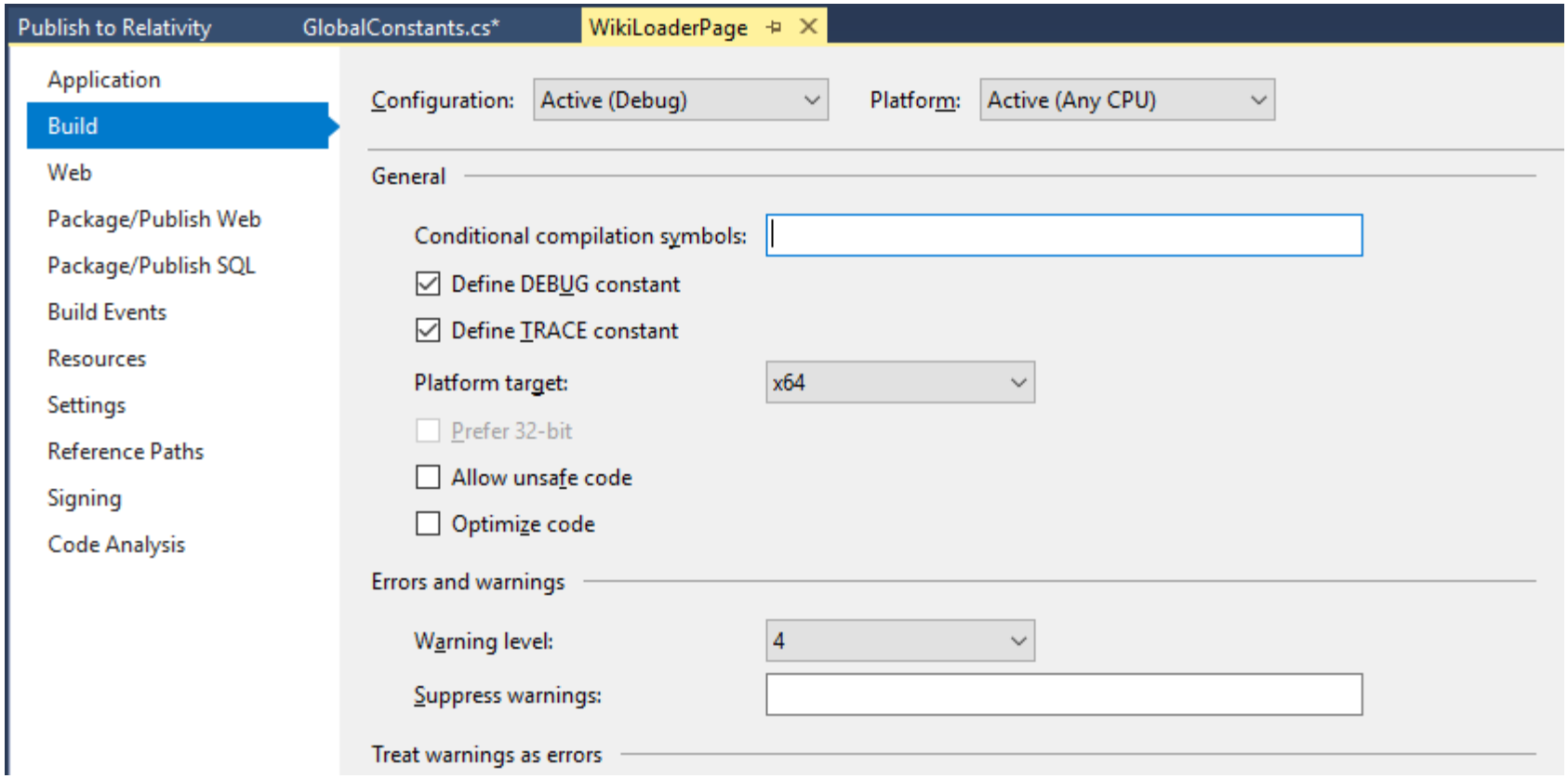
10 Pulling it Together

At this point all the requirements for a fully functional Relativity Integration Points (RIP) Provider have been fulfilled. We've completed the Provider, the Event Handlers and the Custom Page. Now follow the instruction below to update the WikiLoader Provider in Relativity, install it into a workspace and start your first job.

10.1 Build The Solution

The final step before pushing the WikiLoader DLLs to Relativity is to confirm the platform target is set to x64.

1. **Right Click** the **WikiLoaderPage** project in Solution Explorer and select **Properties**.
2. Select the **Build Menu**.
3. Change the **Target platform** to **x64.**

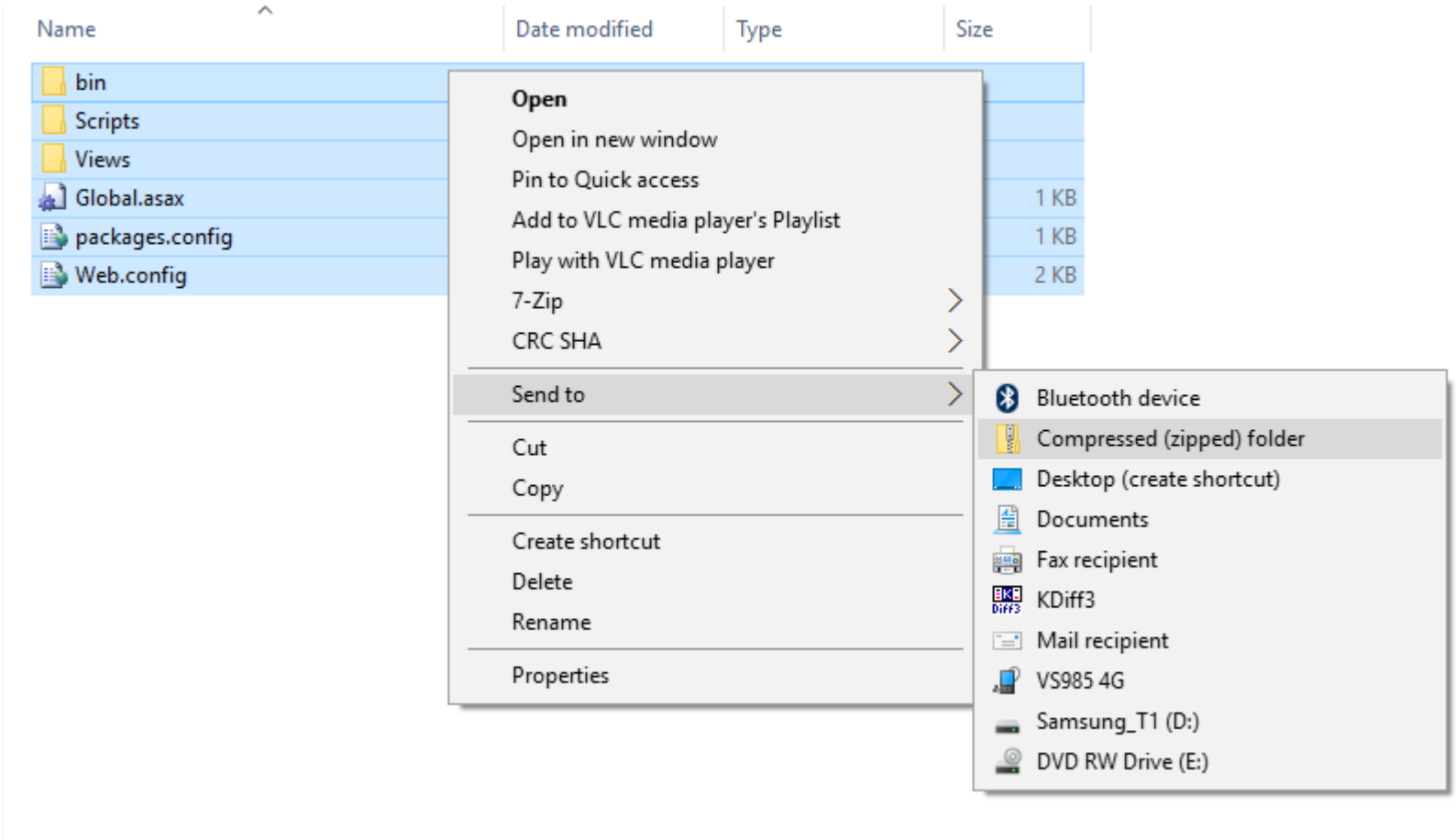


4. Repeat the previous steps to update the the platform target of the **WikiLoaderProvider** project.
5. Build the solution

10.2 **Publish Custom Page**

1. Right click on the **WikiLoaderPage** project and select **Publish.**
2. Highlight **Profile** in the sidebar, and click **Custom**.
3. In the **Profile name** box type **WikiLoaderPageProfile** Click **OK**.
4. In the Publish Method drop-down box, select **File System**.

5. Click ... to set the **Target location** for your published files (C:\WikiLoader\Publish).
6. Click **Next** and select **Debug** from the Configuration drop-down box.
7. Under File Publish Options, click **Delete all existing files prior to publish**.
8. Click **Publish**.
9. Navigate to the directory where you choose to publish your web application.
10. Add the contents of the parent directory to a .zip file named **WikiLoaderPage.zip**.
    - Don't add the parent directory to this file.
    - Note: we recommend the Default Windows Zip Application over Specialty Apps.



10.3 **Deploy your Integration Point**

Note: Before you finish deploying the Integration Point you've built, if you haven't already, add the following Instance Setting to your Relativity environment:

Instance Setting Name: WebAPIPath

Instance Setting Value: http://[hostname]/RelativityWebAPI

Instance Setting Section: kCura.IntegrationPoints

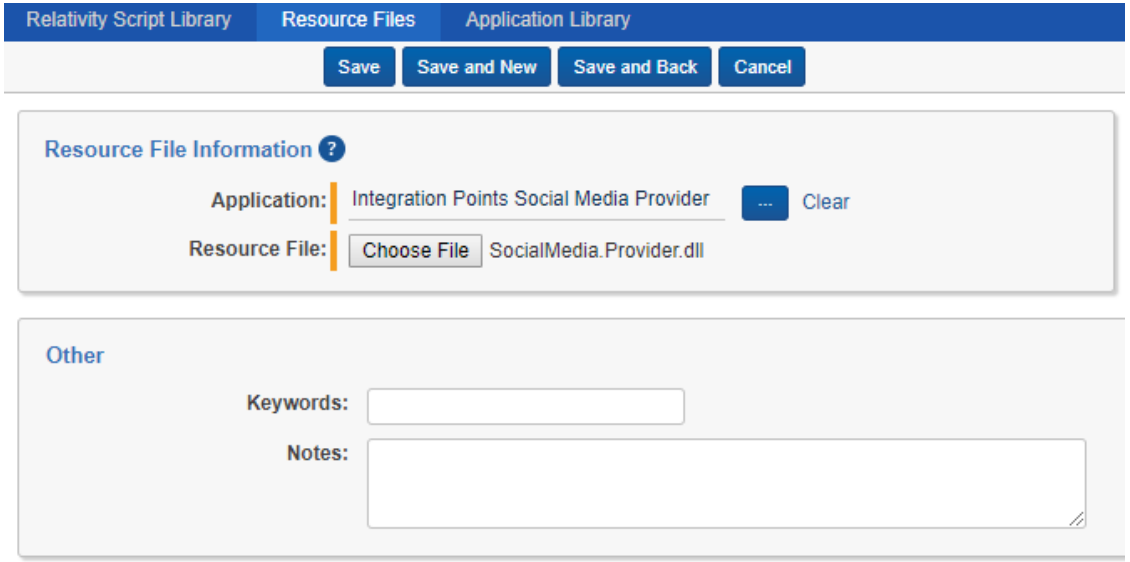
After your integration point assembly builds successfully, you can upload it to Relativity. This process includes the following steps:

- Uploading the required Integration Points DLL files and your provider DLL file to the **Resource** tab in Relativity and associating it with your custom application.
- Adding your event handlers to your custom application.
- Packaging the files that your custom page uses, adding them to your application, and then pushing the application to the Application Library, so that your custom page deploys on the web server.
- Install your new application in a workspace to trigger the event handlers to run.

10.3.1 Upload required DLL files

1. Navigate to the **Resource Files** tab in the Admin Level of Relativity.
2. Click **New Resource File**.
3. In the **Application** field, click ... to select **WikiLoader Provider** as the application. Click **OK**.
4. Add *every* assembly in the Integration Points SDK Provider folder. This must be done individually, one at a time. There are many, but make sure you don't miss any or you will see an error when you attempt to install the provider in a new workspace. You must add these in the order listed *before* you add your custom provider assembly.

5. In the **Resource File** field, click **Choose File** to select the **WikiLoaderProvider.dll** from the bin directory of your WikiLoader Provider Project.



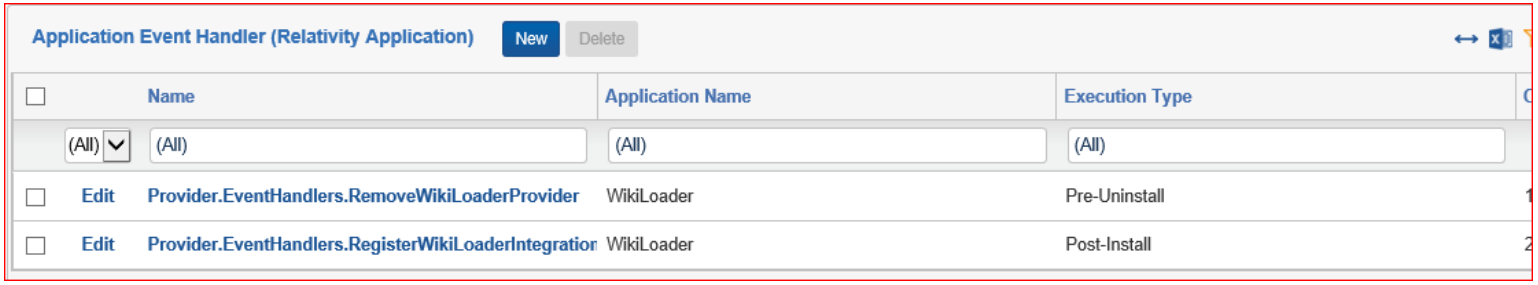
10.3.2 Link Event Handlers to the Application

To link the Install and Uninstall Event Handlers to the application, navigate to the workspace where your application exists.

1. Unlock the Application from the Application Layout Page

   Scroll down to the **Application Event Handler (Relativity Application)** Section
2. Click **New.**
3. Next to the Event Handler Input, click ... to get a list of available event handlers.
4. Select the Event Handler named: **WikiLoader.Provider.RegisterWikiLoaderProvider**
5. Click **Set.**
6. Enter “10” for the order.
7. Click **Save**

8. Repeat those steps and use an order of 20 for the Event Handler named: **WikiLoader.Provider.RemoveWikiLoaderProvider**



10.3.3 Upload your custom page

Use these steps to add the custom page published and zipped in Section 6.6 to the **WikiLoader Provider** application:

1. Return to the **WikiLoader Provider** Application Layout page.
2. Scroll down to the **Custom Page (Relativity Applications)** Section.
3. In the Custom Page associative list, click **New** to display a form for adding a new custom page.
4. Enter **WikiLoader Provider** in the **Name** box.
5. In the **File** field, click **Browse** to find the zipped archive in the directory you specified in section 6.6 step 5.
6. In the Relativity Applications field, click ... to select the **WikiLoader Provider** that you created.
7. Click **Save and Back** to display the detail view of your application. You have now added the custom page to your application.
8. Click **Push to Library** in the console on the Application Layout Page to update the application at the admin level and deploy your custom page.

After the application is pushed to the Application Library the Custom Page Deployment Manager agents detect the new entry and deploy the page to the appropriate area on the web server. You may want to confirm the Custom Page Deployment Manager is enabled in the Agents tab.

10.4 **Install the WikiLoader Provider Application**

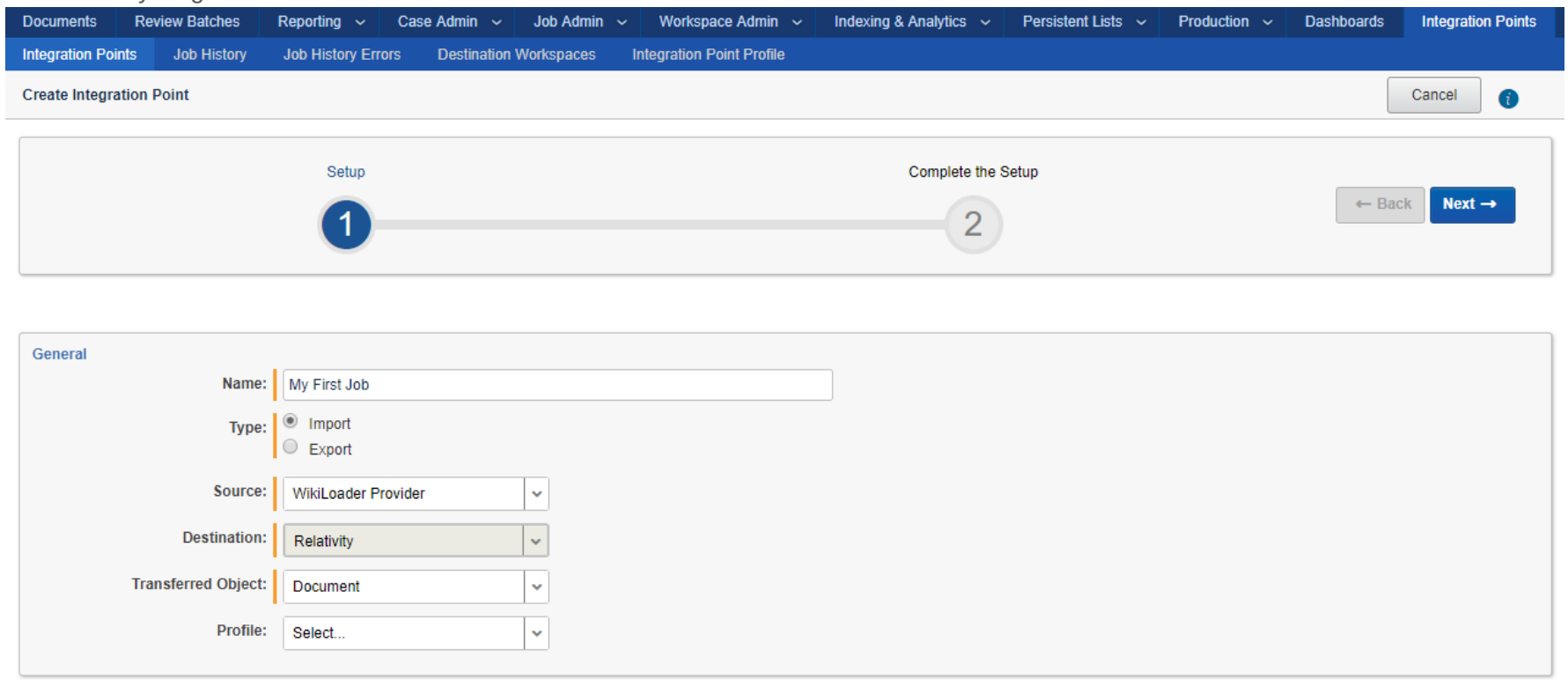
At this point, the application is updated in Relativity; we'll need to install it in a workspace other than the one originally used for development in order to make sure the installation event handlers have a chance to fire.

1. Select a different Workspace other than the one used for developing the WikiLoader Provider Application.
2. Install the Standard **Relativity Integration Points** Application from the Application Library.
3. Install the **WikiLoader Provider Application** from the Application Library.

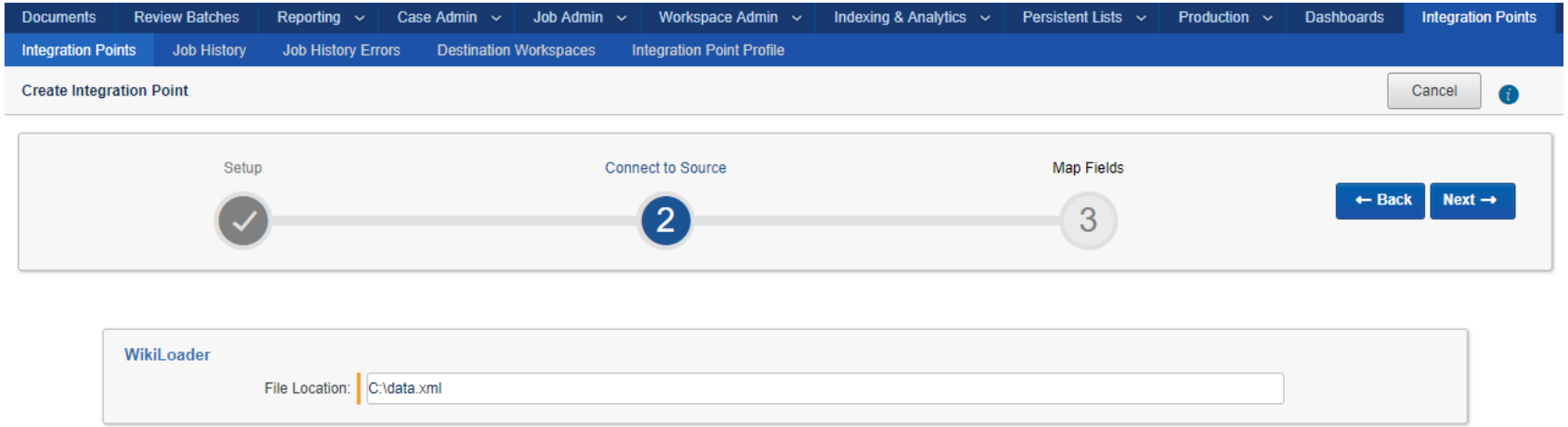
11 Using Your Provider

11.1 Create Integration Point

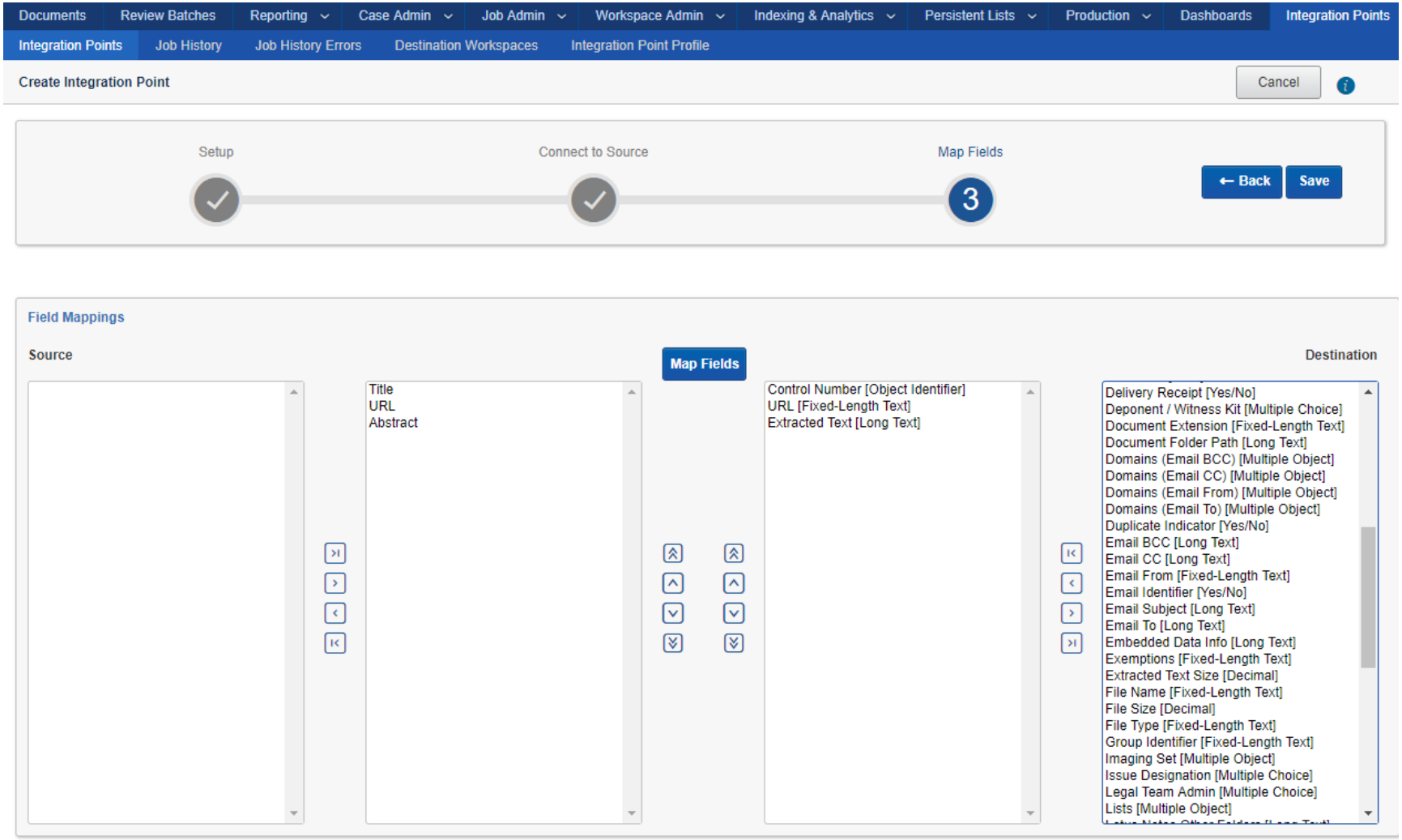
1. Navigate to the Integration Points tab.
2. Click **New Integration Point.**
3. Enter a Name, select the Import type, and select the **WikiLoader Provider** source and **Document** as the Transferred Object. Leave everything else as default.



4. Click **Next ->**
5. Your custom page is displayed. Enter the full path to the wikipedia XML file (C:\data.xml)
    a. Note: This path resides on your DevVM; the same place Relativity is hosted.



6. Click **Next ->**
7. The Field mapping pages is displayed with the fields from your provider on the left and the selected import object fields on the right. Update your field Mapping as follows:
    - **Title-> Control Number [Object Identifier] (or any field that represents the document identifier)**
    - **URL -> URL**
    - **Abstract -> Extracted Text**

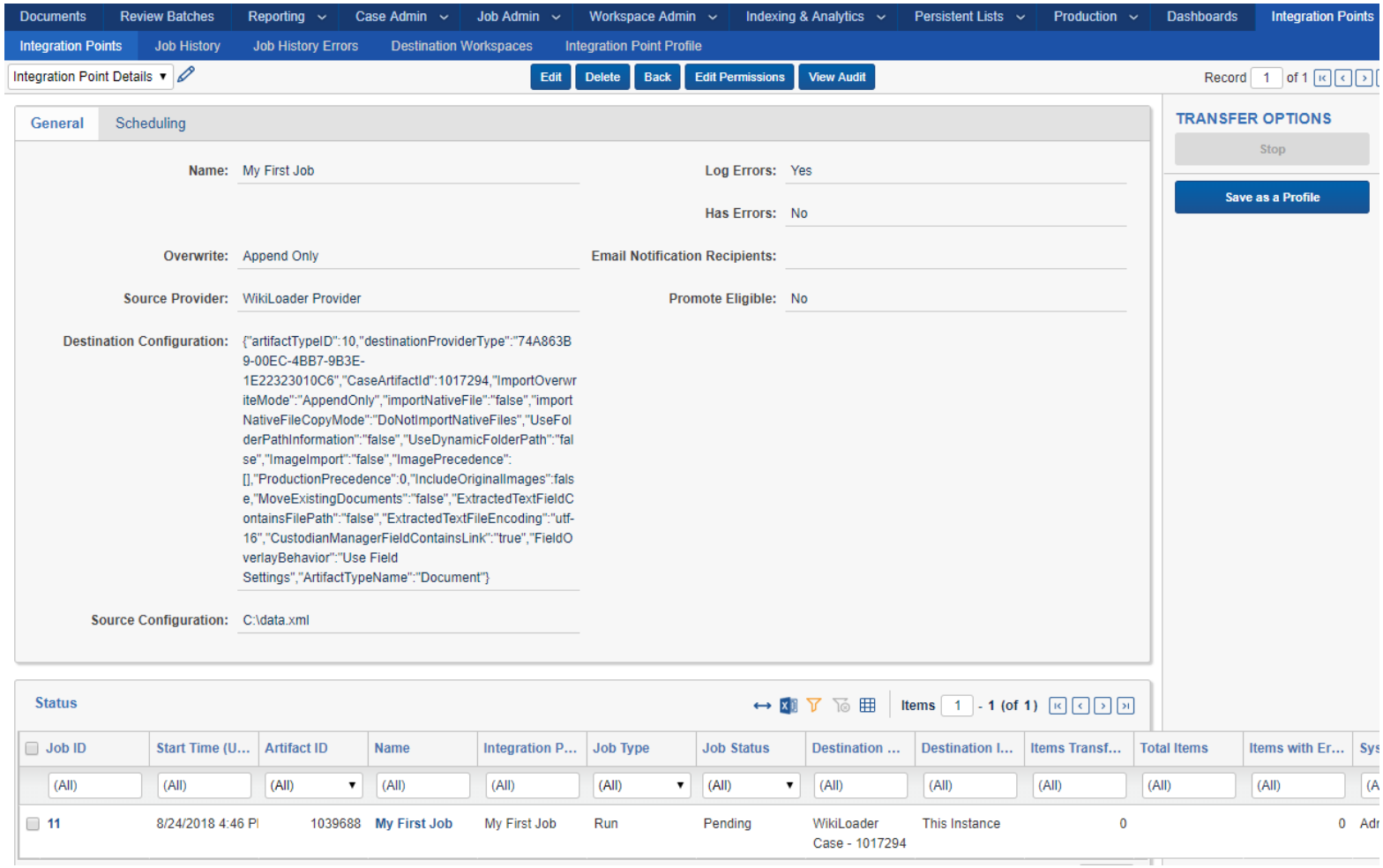


8. Leave every other setting as default.
9. Click **Save.**

11.2 **Run Integration Points Job**

After your integration point has been fully configured and saved, you'll be taken to the main job screen. You can also access the main job screen by clicking on your integration point from the main list view.

1. Click the Run button and review the results as the job is processed.



2. Navigate to the Document tab to view results.

Documents | Review Batches | Reporting | Case Admin | Job Admin | Workspace Admin | Indexing & Analytics | Persistent Lists | Production | Integration Points

New Document | WikiLoader Documents | + No Related Items | Add Widget | No Dashboard Selected | Export

Keyword Search | Enter Search Terms | Search | Clear

1 - 25 / 8,263 | 25 per page

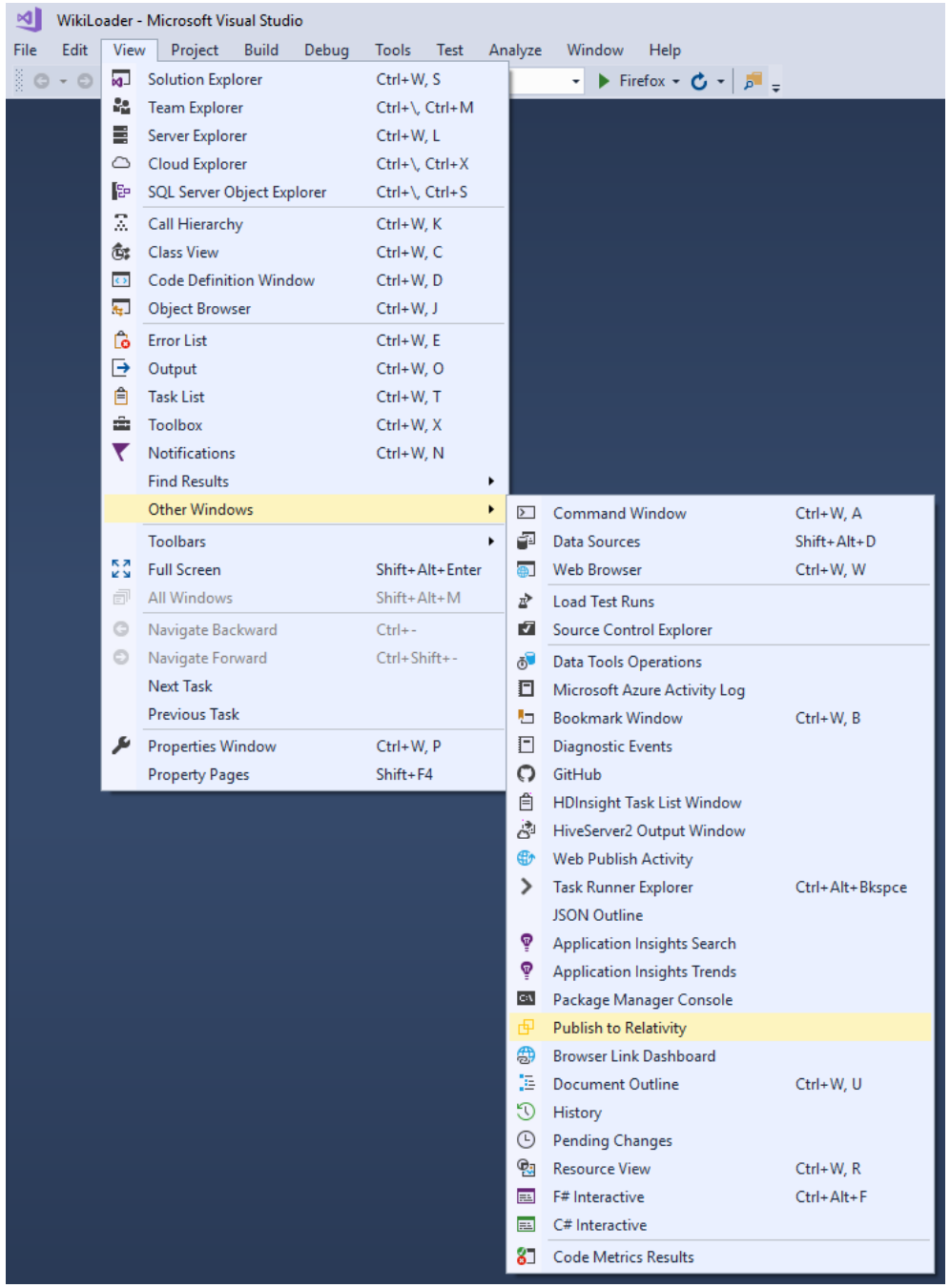
#	Control Number	URL	Extracted Text
	Filter	Filter	Filter
1	Wikipedia: Teqerideamani I	https://en.wikipedia.org/w	Teqerideamani I is believed to have been a King of Kush dating to the end of the 1st and beginning of the 2nd century AD. He was preceded by King Teritnide and succeeded by King Tamelerdeamani.
2	Wikipedia: 1972 in association football	https://en.wikipedia.org/w	2000s
3	Wikipedia: Werlte (Samtgemeinde)	https://en.wikipedia.org/w	|lat_hem = N
4	Wikipedia: 1971 in association football	https://en.wikipedia.org/w	2000s
5	Wikipedia: Electoral district of Warren-Blackwood	https://en.wikipedia.org/w Blackwood	Warren-Blackwood is an electoral district of the Legislative Assembly in the Australian state of Western Australia from 1950 to 2008, and from 2013 onwards.
6	Wikipedia: Acting Crazy	https://en.wikipedia.org/w	| last_aired =
7	Wikipedia: Margie Ackles	https://en.wikipedia.org/w	Margie Ackles (born 1939) was an American figure skater, who competed in ice dance with Charles Phillips. The pair won the gold medal at the 1960 U.
8	Wikipedia: Charles Phillips (figure skater)	https://en.wikipedia.org/w	Charles Phillips (born 1938) was an American ice dancer. Competing with Margie Ackles, he won the gold medal at the 1960 U.
9	Wikipedia: Tapanahony River	https://en.wikipedia.org/w	The Tapanahoni River (sometimes called Tapanahony) is a major river in the south eastern part of Suriname, South America. The river originates in the Southern part of the Eilerts de Haan Mountains, near the border with Brazil.

12 Appendix A: Publish to Relativity Tool

The Publish to Relativity tool speeds Relativity application development by automating the assembly and web page upload process. Imagine how long it would take to manually update each resource file every time you made slight changes to your code. Even worse, imagine re-publishing, zipping and attaching the custom page with every code change. The Publish to Relativity tool allows you to update all the resources and custom pages with a single button click. An initial setup is required but its time saving benefits make the setup process worth the effort. Configuration settings can also be saved, loaded and shared among a team for even more time saving benefits.

The Publish to Relativity Tool is distributed with the standard Relativity SDK and installed in the following directory by default:
C:\Program Files\kCura Corporation\Relativity SDK\Publish To Relativity

Note: For this demonstration we will use an upgrade we've been working on – the Publish to Relativity tool as a Visual Studio extension! If the window is not already open in your Visual Studio solution, open it under **View -> Other Windows -> Publish to Relativity**.



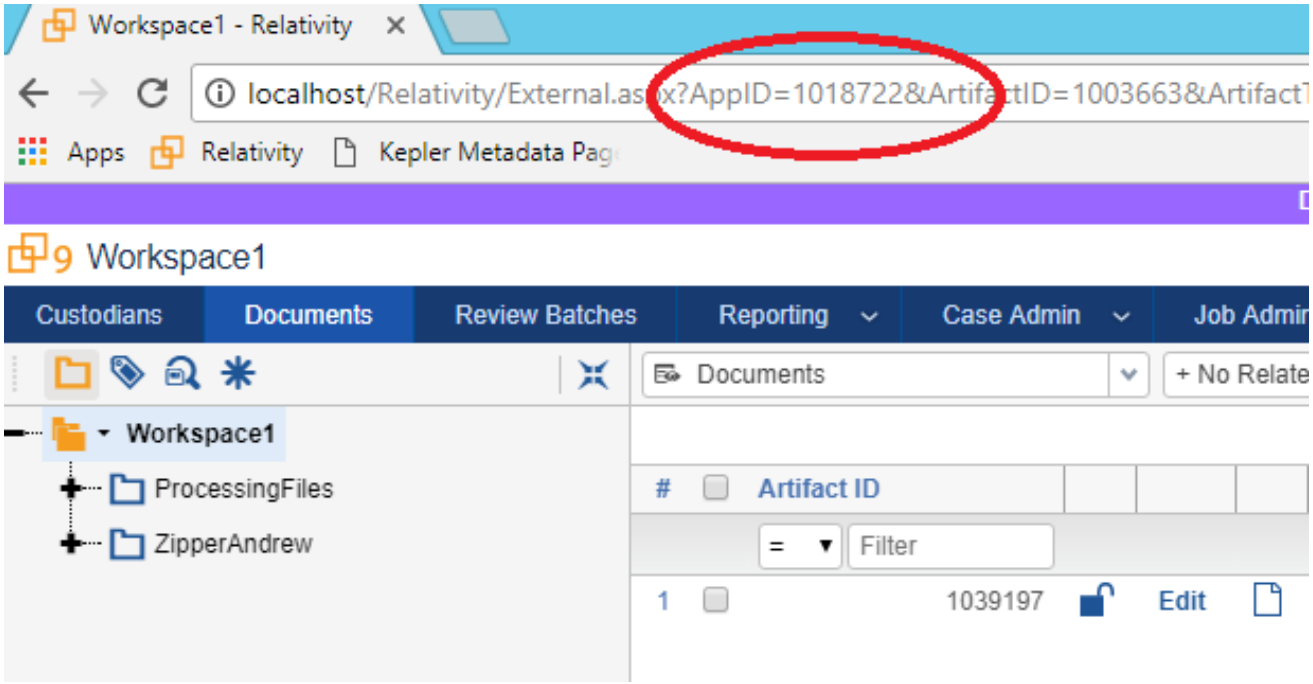
12.1 Application Settings

1. Enter the name **WikiLoaderConfig** for the Configuration Name.
2. Enter the Application Guid. You entered this GUID in the **GlobalConstants** class.

12.2 Connection Settings

12.2.1 Workspace Artifact ID

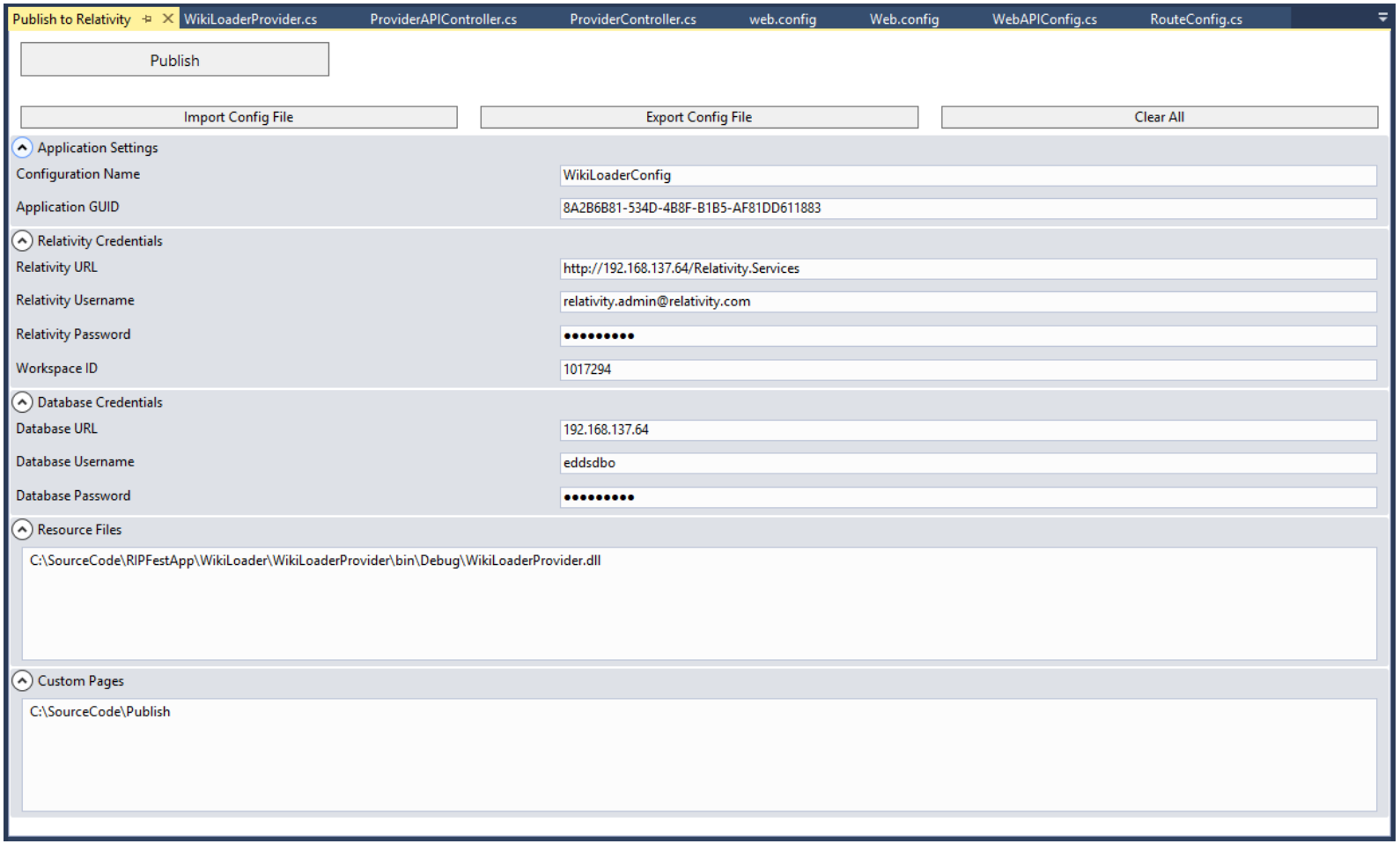
The Workspace ArtifactID is needed to complete the connection settings. To simplest way to find the workspace ArtifactID is to retrieve it from the AppID query string with the workspace open (in the image, Workspace1).



12.2.2 Credentials

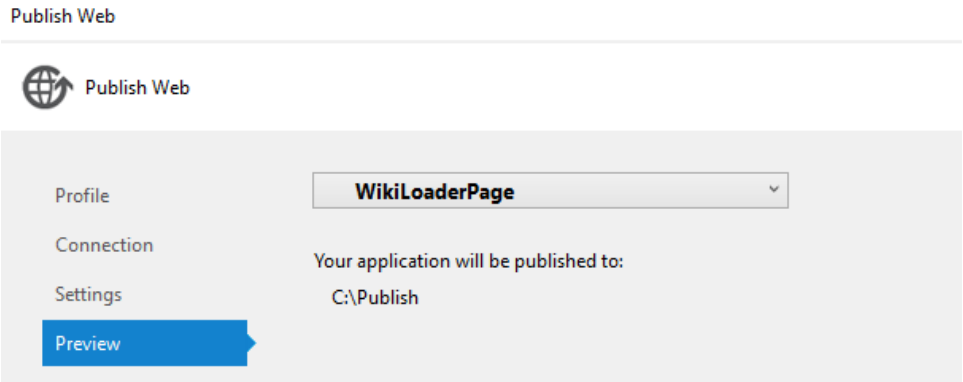
1. Enter a Relativity Admin Username and Password.
2. Enter Relativity URL in the following format: **http://<the IP address of server>/Relativity.services**

3. Enter the Database Username, Password and URL or IP Address.
4. Click **Export Config File** button and select a file path.

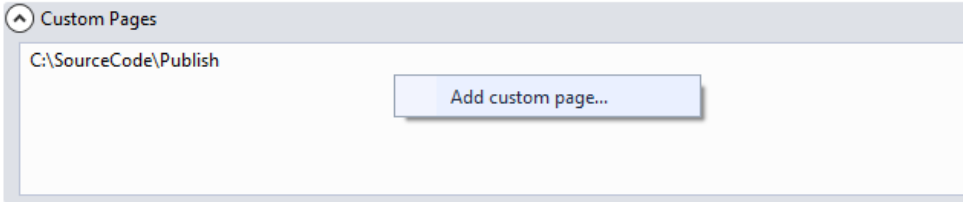


12.3 **Reference Custom Pages**

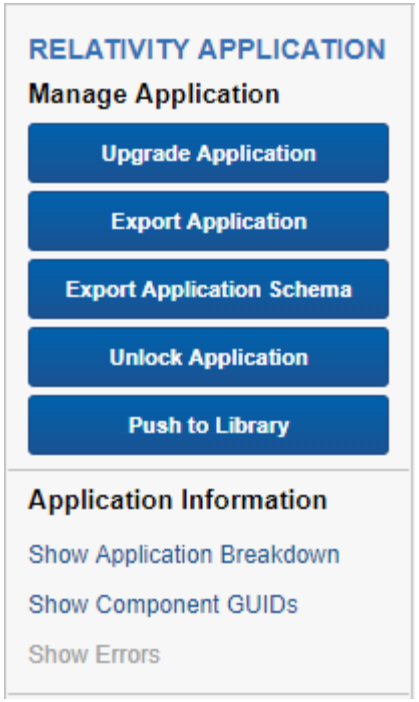
1. In Visual Studio 2015 right click on the WikiLoaderPage project and select **Publish**.
2. Ensure the publish file path reads C:\Publish.



3. Back on the Publish to Relativity window, right click the text box under Custom Pages and select **Add custom page...**



4. Enter the same file path as your publish folder from step 2.
5. To find the Custom Page GUID, navigate in Relativity to the application and click on **Show Component Guids** in the console.



12.4 **Reference Assemblies**

The final step to configure the Publish to Relativity application is to add the file paths to our provider DLL

1. Right click on the text box under Resource Files and select **Add resource files from open solution**. This will enter all the assemblies and their symbol files. In this case we're going to right click on each DLL and click **Remove file** for all DLLs except **WikiLoaderProvider.dll**.
2. Click **Export Config File** button and replace the existing configuration.

12.5 **Publish your projects**

1. Click the **Publish** button to update the application in the selected workspace.

Publish to Relativity | WikiLoaderProvider.cs | ProviderAPIController.cs | ProviderController.cs | web.config | Web.config | WebAPIConfig.cs | RouteConfig.cs

Publish

Import Config File | Export Config File | Clear All

Application Settings

Configuration Name	WikiLoaderConfig
Application GUID	8A2B6B81-534D-4B8F-B1B5-AF81DD611883

Relativity Credentials

Relativity URL	http://192.168.137.64/Relativity.Services
Relativity Username	relativity.admin@relativity.com
Relativity Password	•••••••••
Workspace ID	1017294

Database Credentials

Database URL	192.168.137.64
Database Username	eddsdbo
Database Password	•••••••••

Resource Files

C:\SourceCode\RIPFestApp\WikiLoader\WikiLoaderProvider\bin\Debug\WikiLoaderProvider.dll

Custom Pages

C:\SourceCode\Publish

13 Appendix B: Remote Debugging

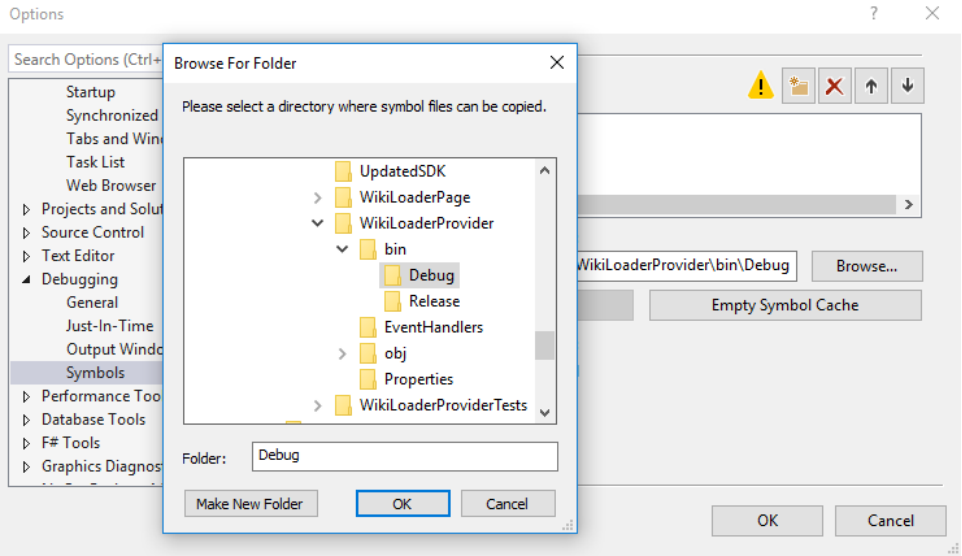
To enable remote debugging, you install or share remote debugging components on the remote computer that contains the application to be debugged. If your computers are protected by a firewall, you must take extra steps to enable communication between the remote computer and the computer that is hosting Visual Studio. You must also ensure that the debugger can locate the symbol files for the process to be debugged and that you have the correct permissions to access the process to be debugged.

13.1 Reference PDB Files Locally

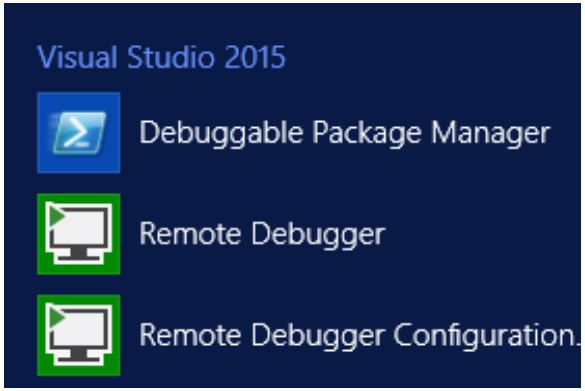
The Relativity Integration Points Framework does not allow PDB files in the same Relativity Application Domain as RIP Providers, therefore please follow the steps below to reference PDB files locally in Visual Studio. Symbol files contain the debugging information for compiled executables. The symbol files of the application to be debugged must be the files that were created when the application executables were compiled. The symbol files must also be located where the debugger can find them.

- The symbol files for native applications must be located on the Visual Studio host computer.
- The symbol files for managed applications must be located on the remote computer.
- The symbol files for mixed (managed and native) applications must be located on both the Visual Studio host computer and the remote computer.

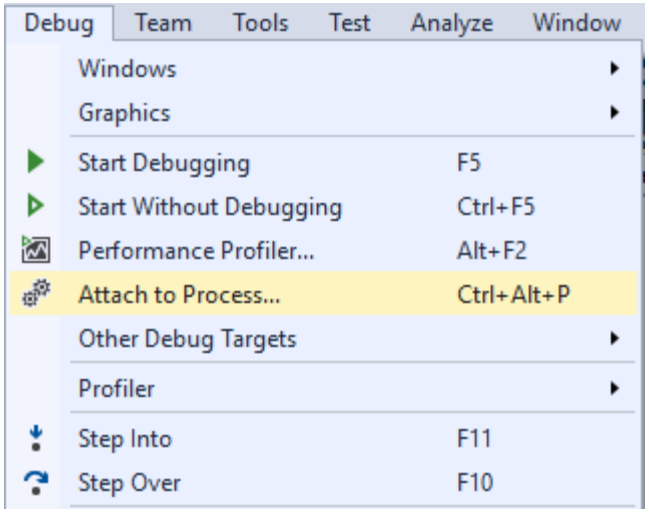
1. Open the WikiLoader Solution in Visual Studio 2015.
2. To reference the PDB files locally, click **Tools > Options > Debugging > Symbols**
3. Click Browse... and navigate to the debug folder in the build path of the current Solution.



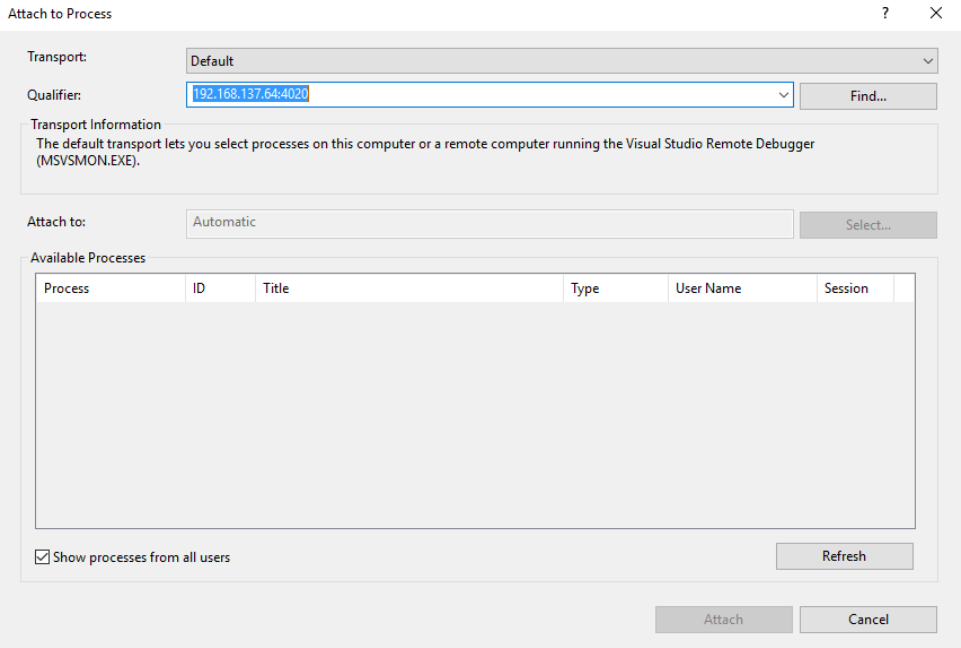
4. Login to the server running your Relativity instance and launch Visual Studio 2015 Remote Debugger as an **Administrator.**



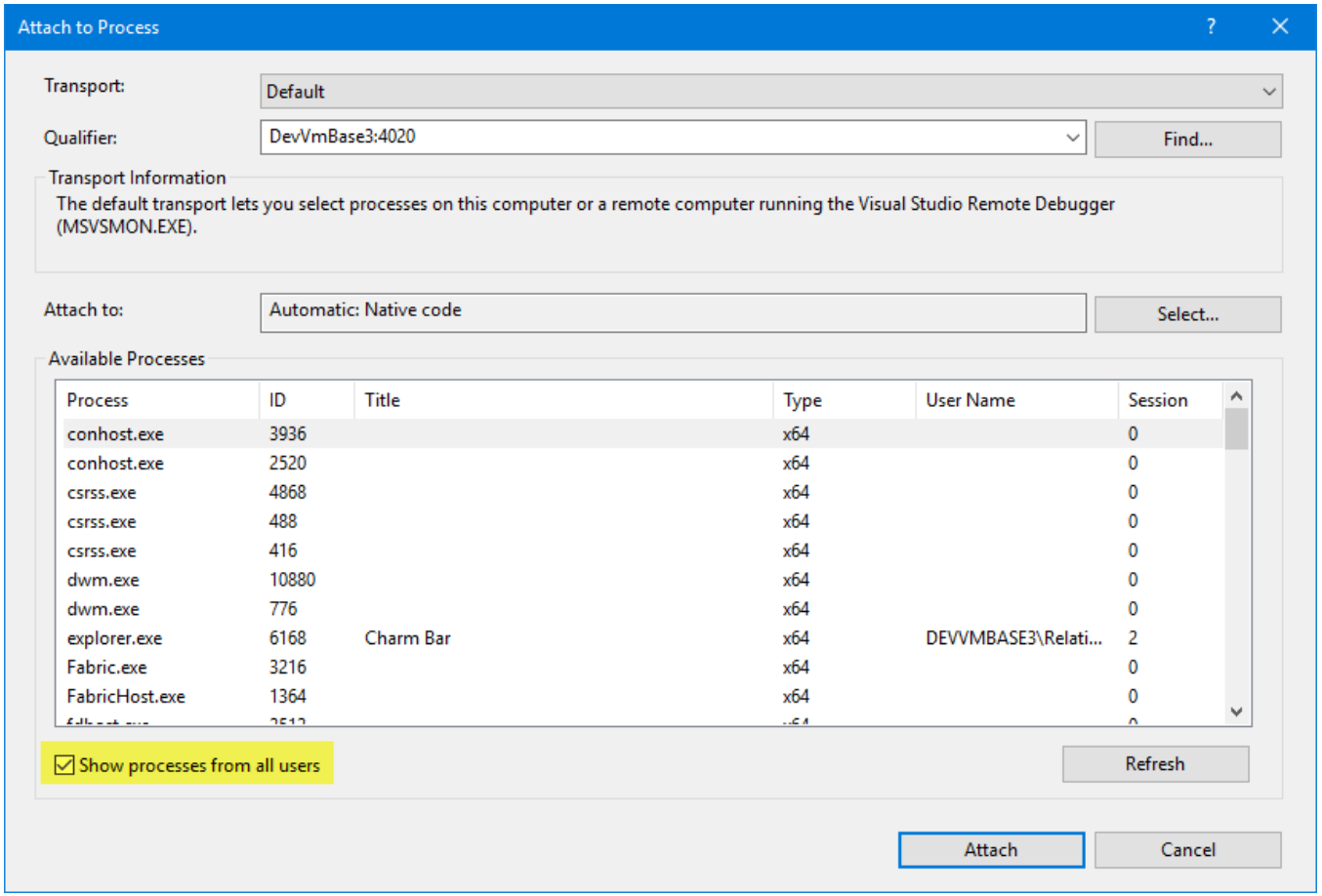
5. Open the **WikiLoaderProvider.cs** file.
6. Add a breakpoint to the first line of the following methods: **GetFields()**, **GetBatchableIDs**, **GetData()**
7. In Visual Studio 2015, go to Debug menu option and select Attach to Process.



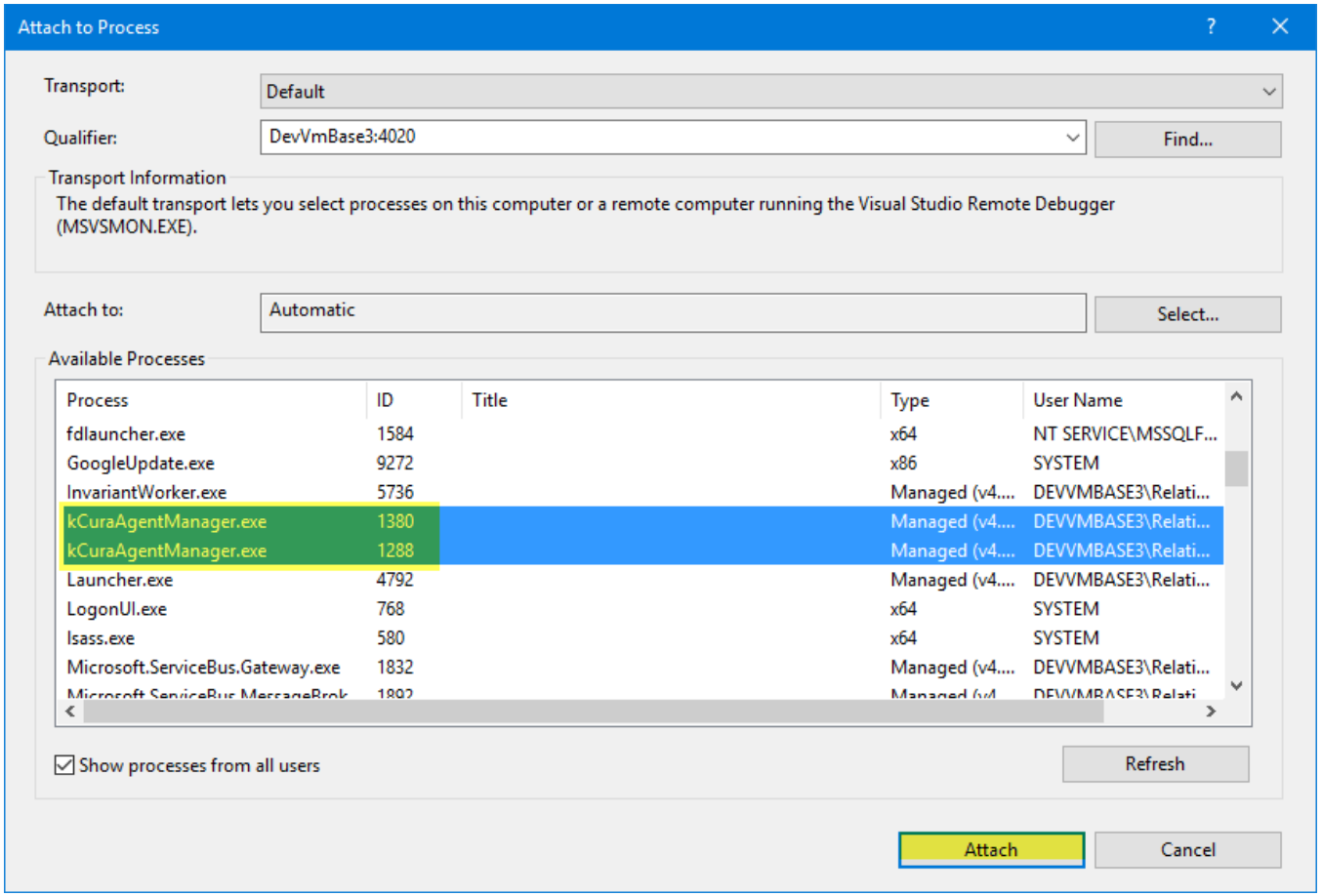
8. In the **Attach to Process** windows, enter your Relativity server name or IP address along with the port and press **Enter**.



9. If prompted for credentials, enter your Relativity Server credentials provided in the windows account section of this document.
10. Next check the box for option **Show processes from all users**.



11. Troubleshooting code for the **GetBatchableIDs()** method or **GetData()** method require the debugger to be attached to the **kCuraAgentManager.exe** process. Troubleshooting the **GetFields()** method requires the debugger to be attached to the **w3p.exe** process (select them all). After the process is selected, click on the **Attach** button as shown in the below screenshot.



12. The Breakpoint will change color as a notification that it has been hit

Note**: You can find more information on how to use remote debugging to troubleshoot event handlers and custom pages at this link - **https://platform.kcura.com

14 Appendix C: Automated Tests

Relativity Integration Points (RIP) Providers support dependency injection to encourage its use with automated testing. Please follow the steps below to write a test for the WikiLoader Provider's **GetFields()** method.

14.1 Test Scenario

The **GetFields()** method is passed in a DataSourceProviderConfiguration object which contains user settings from the Custom Page. A test will be written that verifies the correct fields are returned when the correct file path is passed in the DataSourceProviderConfiguration object.

14.2 Add New Test Project

1. Add new class library project named **WikiLoaderProviderUnitTests** to the solution.
2. Go to the Project Properties and make the Platform target **x64**
3. Add a reference to **kCura.IntegrationPoints.Contracts**
4. Add a reference to **WikiLoaderProvider**
5. Use NuGet Package Manager to install the following: **NUnit 3.10.1**
6. Rename the default class(class1.cs) to **WikiLoaderProviderUnitTests.cs**

14.3 Implement Test

1. First, copy the following code into **WikiLoaderProviderUnitTests.cs**
2. Next, replace the **xmlPath** property with the path to your wikidata.xml file.

```
using NUnit.Framework;
using kCura.IntegrationPoints.Contracts.Models;
using System.Collections.Generic;

namespace WikiLoaderProviderTests
{
    [TestFixture]
    public class WikiLoaderProviderUnitTests
    {
        Provider.WikiLoaderProvider provider = null;
        const string xmlPath = "C:\\SourceCode\\RIPFestApp\\WikiLoader\\WikiLoaderProviderTests\\Resource\\wikidata.xml";
        const int titleIndex = 0;
        const int urlIndex = 1;
```

```
        const int abstractIndex = 2;

        [SetUp]
        public void SetUp()
        {
            provider = new Provider.WikiLoaderProvider();
            if (!System.IO.File.Exists(xmlPath))
            {
                throw new System.IO.FileNotFoundException("Test load file not found - ending tests in setup");
            }
        }

        [Test]
        public void GetFieldsReturnsMoreThanZeroFieldEntries()
        {
            // Arrange
            var providerConfig = new DataSourceProviderConfiguration(xmlPath);

            // Act
            var result = (List<FieldEntry>)provider.GetFields(providerConfig);

            // Assert
            Assert.AreEqual(result.Count, 0);
        }
    }
}
```

- The test setup method instantiates an instance of the system under test (SUT), in this case **WikiLoaderProvider**
- To avoid a test failure because of a faulty file path, we confirm the file exists in the test setup.
- Our test is explicitly named in the following way: [name of the method under test][and expected result]
- The structure of all tests follow the pattern Arrange – Act – Assert.
    - In arrange, we create an instance of **GetFields()** required parameter, a DataSourceProviderConfiguration object. We pass this object the path of our XML file.
        - Note: For this to be a true unit test, we would mock the IDataSourceProviderConfiguration interface and replace the public property **Configuration** with the XML path. This is because our test should not depend on the DataSourceProviderConfiguration implementation in any way. We'd also have to rearchitect our **WikiLoaderProvider** class to abstract the file retrieval so we could mock that too. If this bothers you, read 'Integration Test' where we have written 'Unit Test'.
    - In act, we execute the method under test, passing it the required parameters.

- In assert, we confirm that exactly zero fields are returned.
- Some additional tests we could write for **GetFields()**
  - Confirm that exactly three fields are returned (no accidental duplicates)
  - Confirm the fields returned have all required properties
  - Confirm only one returned field has **IsIdentifier** set to true.

3. Run the test and it should... fail! Why is this? Am I playing with you?

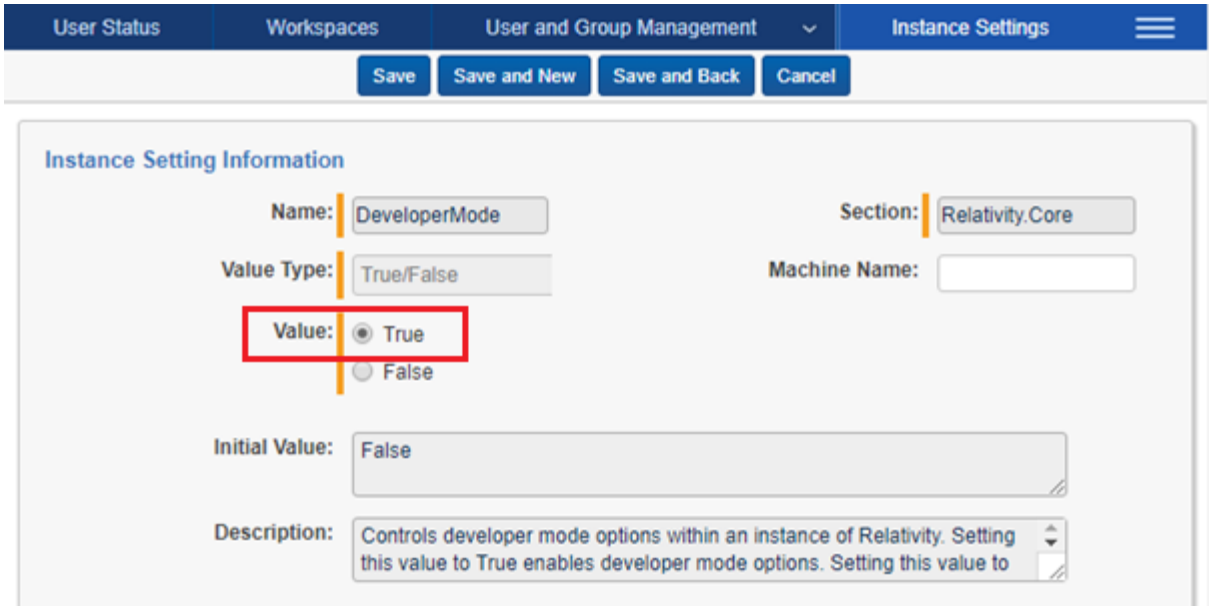
   No, I'm not playing with you. We've deliberately written a failing test first to confirm an assumption – that more than one field *does* return from **GetFields()**. While this is a contrived example, it's best to write a failing test first lest your passing test work, not because the code is correct, but because your assumptions are wrong. In the future, write the failing test *before* you write the code rather than writing an arbitrary failed test. Like I said, it's contrived.

   Now, to get the test passing (we hope), replace **Assert.AreEqual** with **Assert.Greater**.

15 Appendix D: Developer Mode

Enabling Developer Mode is recommended while developing application because it adds a link to the Relativity Application Page that makes the GUIDs of object associated with an application easy to access. However, please be advised that Developer Mode is a system-wide setting and should only be enabled in local test environments. Please follow the steps below to enable Developer Mode in your environment:

1. Log into Relativity with an Admin account.
2. Go to the **Instance Settings** tab.
3. Filter the Instance Setting by name to find the setting named: **DeveloperMode.**
4. Edit the **DeveloperMode** Instance Setting and make its value **true.**



5 Save, refresh your screen and you'll be able to confirm that Developer Mode is active by receiving confirmation with the purple bar that appears at the top of your screen.