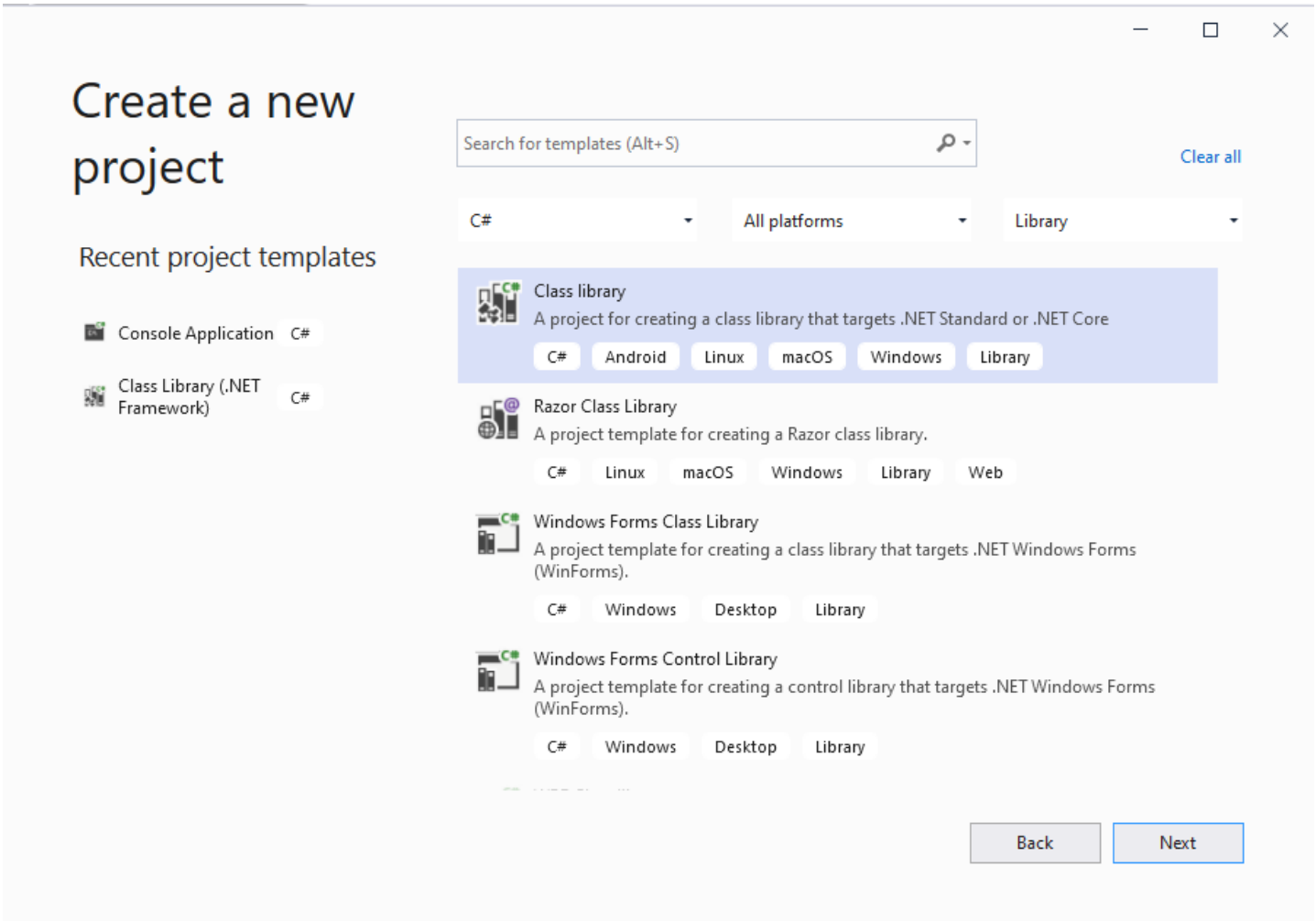
Creating the Example Converter DLL

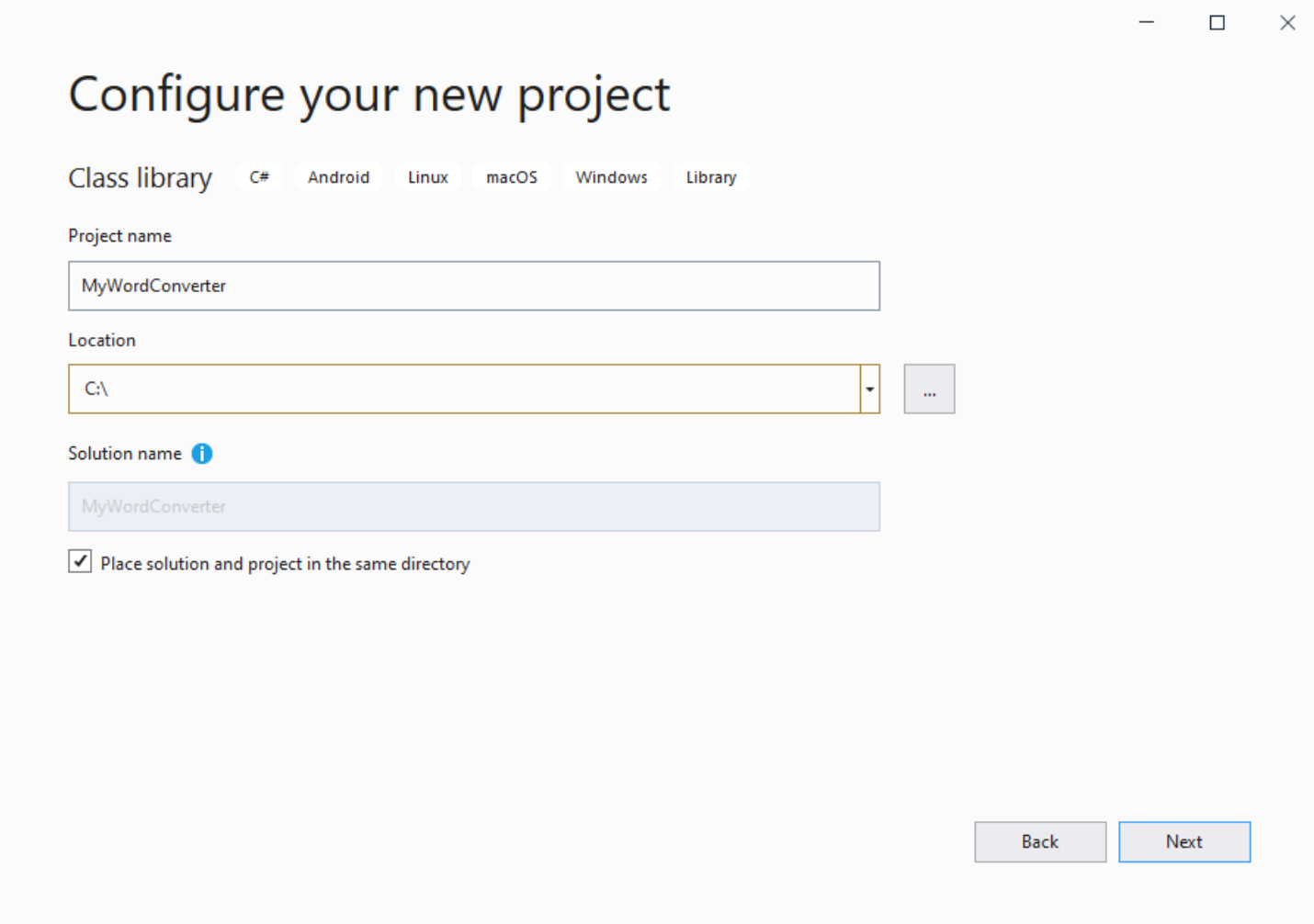
The following section describes the procedure for creating the example DLL to integrate a file converter with the Conversion Server.

Adding the Main Converter Class

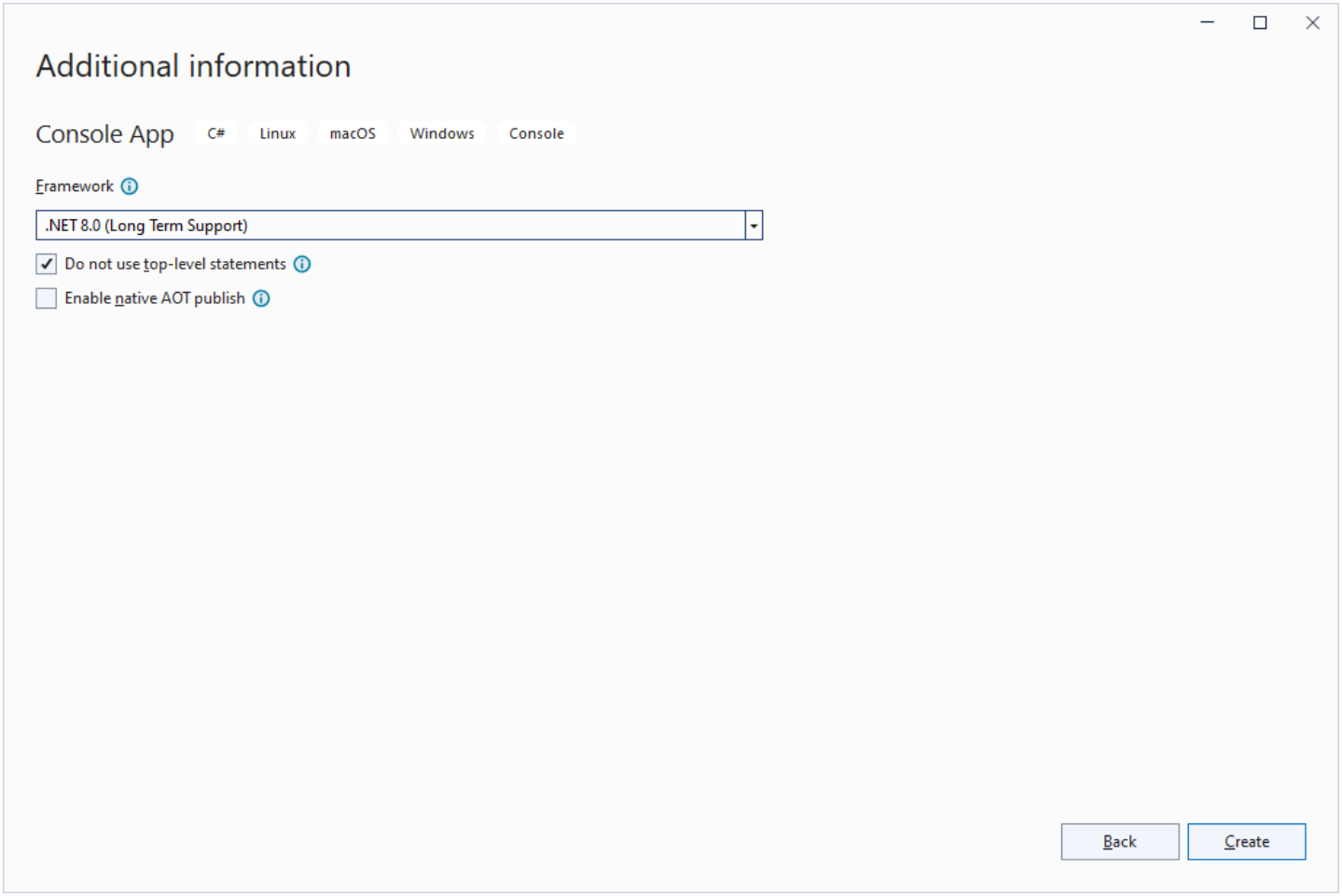
1. Open Microsoft Visual Studio (Visual Studio 2022 is used in this example).
2. Create a new Class Library Project using .NET Standard or .NET 8.0 (Visual C# is used in this example) and press the “Next” button.



3. Set the Solution name to **MyWordConverter** and its location and press the “Next” button.



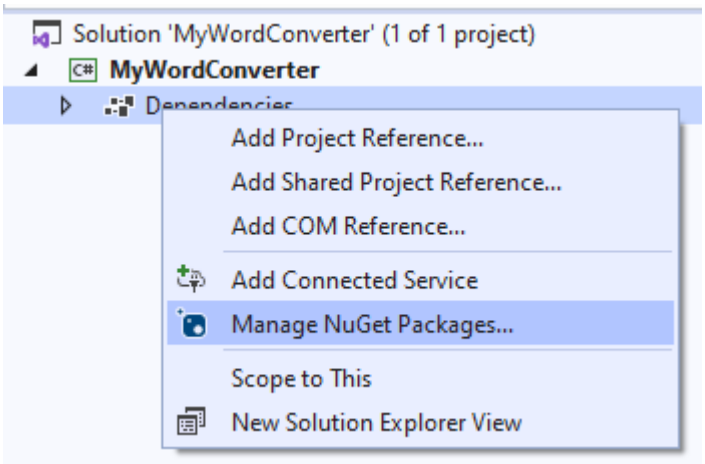
4. Select the appropriate .Net Standard or .Net 8.0 version and press the “Create” button. It’s recommended to use long-term supported versions.



5. Add any required references to the Solution. The following are required for this example:
    1. Right click on **References** in the Solution Explorer and select **Add References**.
    2. Select the **Browse** tab and return both IOM.dll and Conversion.Base.dll from the 'bin\Debug' folder of the Solution. You can copy the files this location from the '\ConversionServer\bin' folder of your Aras Innovator code tree.
    3. Right click on **References** in the Solution Explorer and select **Add References**.
    4. Select COM → Type Libraries

```
Microsoft Word 16.0 Object Library      8.7
```

5. Right click on theproject **Dependencies** in the Solution Explorer and select **Manage NuGet packages…**.



6. In the “Browse” tab start typing ConfigurationManager in search input until System.Configuration.ConfigurationManager package appears in the packages list.
7. Select the latest version and press the “Install” button

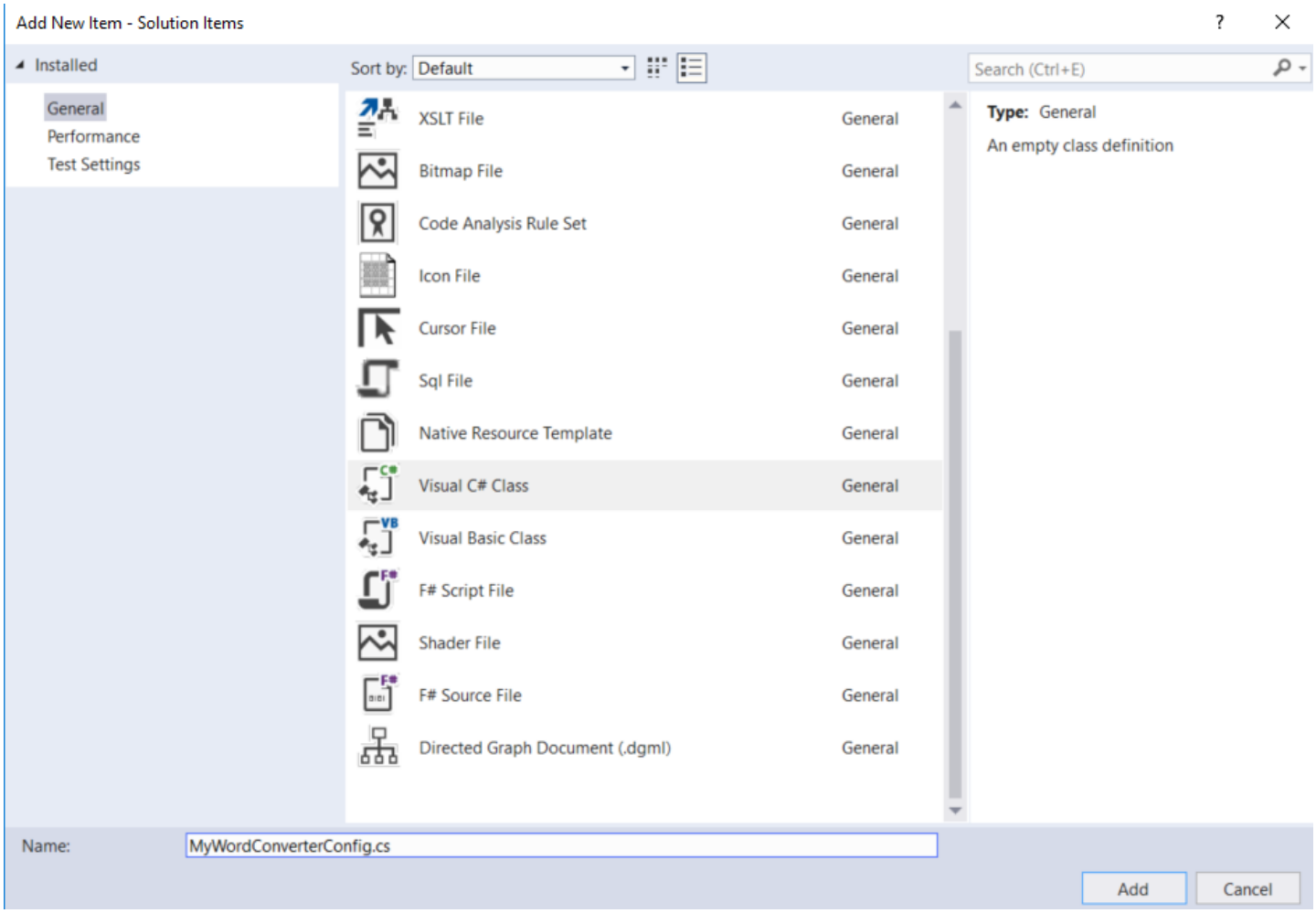
6. In the new .cs file, add the necessary Namespaces. The following are required for this example:

```
…
using System;
using System.Collections.Generic;
using System.IO;
using Microsoft.Office.Interop.Word;
using Aras.ConversionFramework.Converter;
```

Add the Configuration Class

1. Right click on the **MyWordConverter Project** in the Solution Explorer and select **Add→ New Folder**.
2. Name the new folder **Configuration**.
3. Right click on the Configuration folder and select **Add→ New Item**.

4. In the prompt, select a Visual C# Class and name it MyWordConverterConfig.cs.



5. Set the following code in this new file. This code sets which tags and parameters will be available for configuration in the Conversion Server configuration file.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Configuration;
namespace MyWordConverter.Configuration
{
class MyWordConverterConfig : ConfigurationSection
{
[ConfigurationProperty("Settings", IsRequired = true)]
public SettingsElement Settings
{
get
{
return (SettingsElement)this["Settings"];
}
set
{
this["Settings"] = value;
}
}
}
public class SettingsElement : ConfigurationElement
{
[ConfigurationProperty("pdfSuffix", IsRequired = true)]
public String PdfSuffix
{
get
{
return (String)this["pdfSuffix"];
}
set
{
this["pdfSuffix"] = value;
}
}
}
}
```

The above code corresponds with the following tag in the \ConversionServerConfig.xml in the root of your Aras Conversion Server installation:

```
<ConverterSettings>
<MyWordConverter>
<Settings pdfSuffix="_X" />
…
```

6. Save the Project and CS files.
7. Edit `MyWordConverter.cs` and add the new namespace:

```
…
using Aras.ConversionFramework.Converter;
using MyWordConverter.Configuration;
```

Add the Converter Interface

You can implement the following two interfaces for the converter class. This example uses the VaultFileConverter interface:

- IConverter: Used in cases where the code should download the necessary files rather than automatically download these files from the Aras Innovator vault. The main Convert function is as follows in this interface:

```
public IList<FileConversionResult> Convert(ConversionContext context)
```

- VaultFileConverter: Used in cases where all the files that are required for the conversion are located within the Aras Innovator vault. The main Convert function is as follows in this interface:

```
protected override IList<FileInfo> Convert(IDictionary<string, System.IO.FileInfo> filePaths)
```

You need to edit the MyWordConverter class in the example project to implement the desired interface:

```
…
namespace MyWordConverter
{
public class MyWordConverter : VaultFileConverter
{
#region Overrided Protected Methods
…
```

The primary method used for the conversion is the Convert method. This method should take in the past data for the conversion and return a list of the converted files in the format `‘List<FileInfo>‘.`

The basic design of the Convert method is as follows:

1. Create a list of IDs of the File Items that must be downloaded from Aras Innovator. This is done by the VaultFileConverter interface.

2. Download all files within the list defined in 1. This is done by the VaultFileConverter interface.
3. Pass the downloaded file(s) as well as other parameters to the third-party conversion tool.
4. Read the result from the third-party conversion tool.
5. Generate a list of the resulting converted files that contain the ID and the ‘kind’ of each uploaded File Item. The ‘kind’ is used in the onAfterConvert event of the Conversion Task to identify where to use each uploaded file.
6. Upload the new files to the Aras Innovator vault, creating new File Items for each file. This is done by the VaultFileConverter interface.

This example uses the VaultFileConverter interface. In addition to adding code to the `Convert` method, we need to override the `Dispose` and `GetFileKind` methods of the `VaultFileConverter` .

For this example, the `Dispose` method can be left empty.

The `GetFileKind` method must return a string that represents the resulting file’s type. This string sets in the Results tab of the Conversion Task Item.

The full sample code is available in Appendix B .

```
namespace MyWordConverter
{
public class MyWordConverter : VaultFileConverter
{
#region Overrided Protected Methods
protected override void Dispose(bool disposing)
{
}
protected override string GetFileKind(FileInfo file)
{
if (file == null)
{
throw new ArgumentNullException("file");
}
switch (file.Extension.ToUpperInvariant())
{
case ".PDF":
return "pdf";
default:
return String.Empty;
}
}
protected override IList<FileInfo> Convert(IDictionary<string, FileInfo> filePaths)
{
// Do the actual conversion ...
List<FileInfo> returnList = new List<FileInfo> { };
Application wordApplication = new Application();
Document wordDocument = null;
object paramSourceDocPath = filePaths[Context.ConversionTask.FileID].FullName;
string fileExt = filePaths[Context.ConversionTask.FileID].Extension;
object paramMissing = Type.Missing;
// Retrieve the settings from the ConversionServerConfig xml file
MyWordConverterConfig configuration = (MyWordConverterConfig)ConversionConfigurationManager.GetInstance()
string pdfSuffix = configuration.Settings.PdfSuffix;
string paramExportFilePath = filePaths[Context.ConversionTask.FileID].FullName.Replace(fileExt, pdfSuffix
FileInfo returnItem = new FileInfo(paramExportFilePath);
WdExportFormat paramExportFormat = WdExportFormat.wdExportFormatPDF;
bool paramOpenAfterExport = false;
WdExportOptimizeFor paramExportOptimizeFor = WdExportOptimizeFor.wdExportOptimizeForPrint;
WdExportRange paramExportRange = WdExportRange.wdExportAllDocument;
int paramStartPage = 0;
int paramEndPage = 0;
WdExportItem paramExportItem = WdExportItem.wdExportDocumentContent;
bool paramIncludeDocProps = true;
bool paramKeepIRM = true;
WdExportCreateBookmarks paramCreateBookmarks = WdExportCreateBookmarks.wdExportCreateHeadingBookmarks;
bool paramDocStructureTags = false;
bool paramBitmapMissingFonts = true;
bool paramUseISO19005_1 = false;
object objMissing = System.Reflection.Missing.Value;
object objConfirmConversions = false;
object objReadOnly = true;
object objAddToRecentFiles = false;
object objPasswordDocument = objMissing;
object objPasswordTemplate = objMissing;
object objRevert = true;
```

```
object objWritePasswordDocument = objMissing;
object objWritePasswordTemplate = objMissing;
object objFormat = WdOpenFormat.wdOpenFormatAuto;
object objEncoding = 20127;
object objVisible = false;
object objOpenConflictDocument = false;
object objOpenAndRepair = false;
object objDocumentDirection = WdDocumentDirection.wdLeftToRight;
object objNoEncodingDialog = true;
object objXmlTransform = objMissing;
object ofalse = false;
wordDocument = wordApplication.Documents.Open(ref paramSourceDocPath,
ref objConfirmConversions, ref objReadOnly, ref objAddToRecentFiles,
ref objPasswordDocument, ref objPasswordTemplate, ref objRevert,
ref objWritePasswordDocument, ref objWritePasswordTemplate,
ref objFormat, ref objEncoding, ref objVisible, ref objOpenAndRepair,
ref objDocumentDirection, ref objNoEncodingDialog, ref objXmlTransform);
// Export it in the specified format.
if (wordDocument != null)
wordDocument.ExportAsFixedFormat(paramExportFilePath,
paramExportFormat, paramOpenAfterExport,
paramExportOptimizeFor, paramExportRange, paramStartPage,
paramEndPage, paramExportItem, paramIncludeDocProps,
paramKeepIRM, paramCreateBookmarks, paramDocStructureTags,
paramBitmapMissingFonts, paramUseISO19005_1,
ref paramMissing);
// Quit Word and release the ApplicationClass object.
if (wordApplication != null)
{
wordApplication.Quit(ref paramMissing, ref paramMissing,
ref paramMissing);
wordApplication = null;
}
returnList.Add(returnItem);
return returnList;
}
#endregion
…
```

Build and Deploy the DLL

With the override of the VaultFileConverter implemented, the project should be ready to build. Once built, locate the new class library DLL, and copy the file to the ‘\ConversionServer\bin’ folder of your Aras Innovator code tree.