Aras Innovator 34

Example for Developing Conversion Server Tasks

Document #: *D-008105*

Last Modified: *2/11/2025*

Copyright Information

Aras Corporation
100 Brickstone Square
Suite 100
Andover, MA 01810
Phone: 978-691-8900

E-mail: support@aras.com

Website: https://www.aras.com/

Table of Contents

Send Us Your Comments

Aras Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for future revisions.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where and what level of detail?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, indicate the document title, and the chapter, section, and page number (if available).

You can send comments to us in the following ways:

Email:
TechDocs@aras.com
Subject: Aras Product Documentation

Or,

Postal service:
Aras Corporation
100 Brickstone Square
Suite 100
Andover, MA 01810
Attention: Aras Technical Documentation

If you would like a reply, provide your name, email address, address, and telephone number.

If you have usage issues with the software, visit https://www.aras.com/support/

1 Overview

The Aras Innovator Conversion Server is a framework to manage file conversion processes used to transform native files into other viewable formats such as TIFF or PDF files.

The following sections describe the procedures needed to create a DLL called ‘MyWordConverter’ that integrates a file converter for Microsoft Word 2016’s ‘Save to PDF’ feature as an example for use with the Conversion Server.

The steps required are:

1. Set up the required software
2. Create the custom converter DLL.
3. Deploy the project

A full code example can be found in Appendix B at the end of this document. The remainder of this document describes the procedures required for each step.

Note: This document is only an example for integrating a file conversion utility with the Conversion Server. The example in this document is not meant for production use. Microsoft Word is not intended for use as a file converter without the direct interaction of the end user.

2 Example Prerequisites

This section walks through configuring Microsoft Word 2016 to work with the Conversion Server. Before proceeding, take the time to review these requirements.

2.1 Aras Innovator Conversion Server

Before proceeding, you need to install the Conversion Server.

To install the conversion server, refer to the *Aras Innovator 14 - Conversion Server Setup Guide*.

2.2 Microsoft Visual C++ Redistributable Package

You must install The Microsoft Visual C++ 2015, 2017, 2019, and 2022 Redistributable Packages on the Conversion Server. The links to these downloads can be found at Microsoft's latest supported Visual C++:

Microsoft Visual C++ 2015, 2017, 2019, and 2022 Redistributable Packages (x64):

https://support.microsoft.com/en-us/help/2977003/the-latest-supported-visual-c-downloads

2.3 Microsoft Word 2016

For this example to work, Microsoft Word 2016 must be configured to launch using the interactive user and must have the correct permissions. This is configured in the DCOM Config under Component Services:

1. Select **Start --> Run**.
2. Type `dcomcnfg` and click **OK**.
3. Expand **Component Services** --> **Computers** --> **My Computer** --> **DCOM Config**.
4. Right click on the Microsoft Word 97 – 2003 Document and select **Properties**.
5. Click the **Security** tab.
6. Select **Customize** under **Launch and Activation Permissions** and click **Edit**.
   a. Add "IIS_IUSRS" and set **Allow** for **Local Launch** and 'Local Activation.'
   b. Click **OK**.
7. Select **Customize** under **Access Permissions** and click **Edit**.
   a. Add "IIS_IUSRS" and set **Allow** for **Local Access**.
   b. Click **OK**.
8. Select the **Identity** tab and select the **The interactive user** option.
9. Click **Apply** and then **OK**.
10. Close the Component Services window.

3 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.

3.1 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

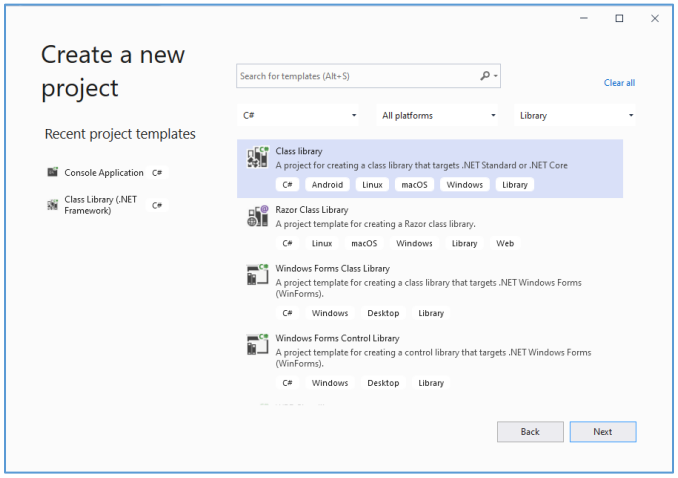

Figure 1.

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

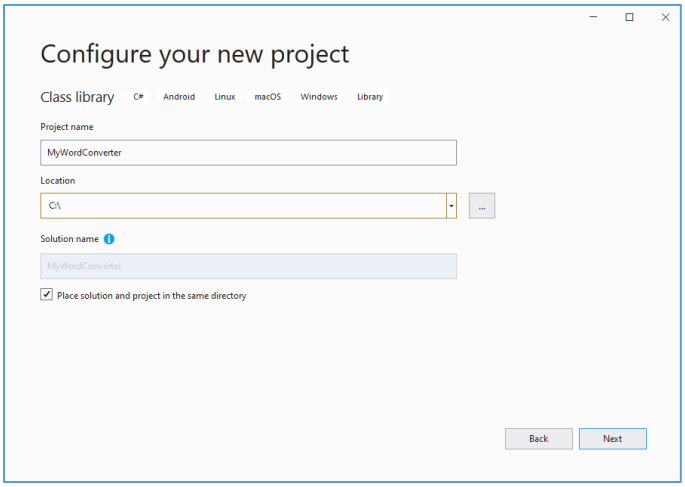

Figure 2.

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.

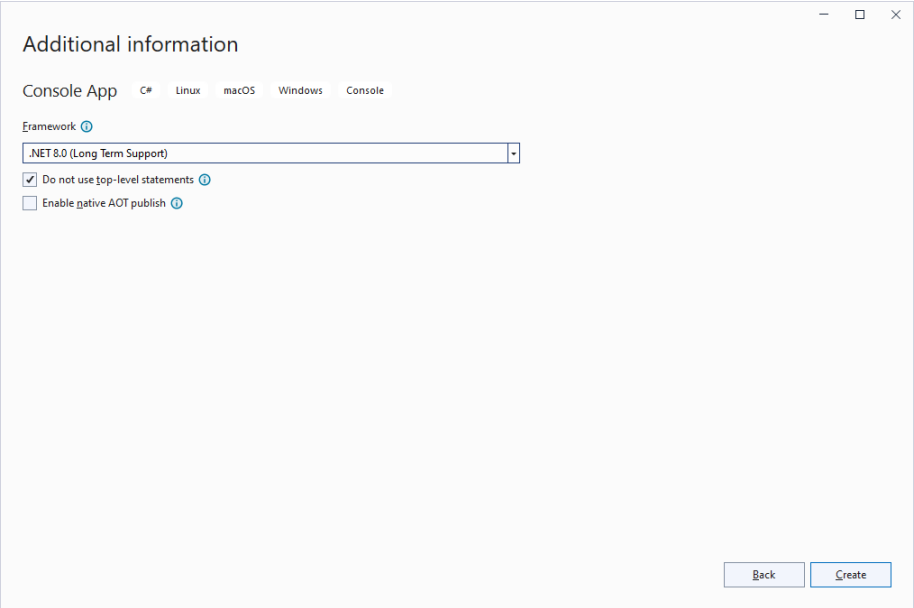

Figure 3.

5. Add any required references to the Solution. The following are required for this example:
   a. Right click on **References** in the Solution Explorer and select **Add References**.
   b. 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.
   c. Right click on **References** in the Solution Explorer and select **Add References**.
   d. Select COM → Type Libraries

      `Microsoft Word 16.0 Object Library  8.7`

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

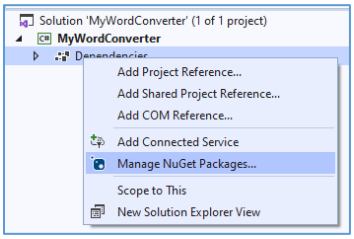

Figure 4.

   f. In the “Browse” tab start typing ConfigurationManager in search input until System.Configuration.ConfigurationManager package appears in the packages list.

g. Select the latest version and press the “Install” button

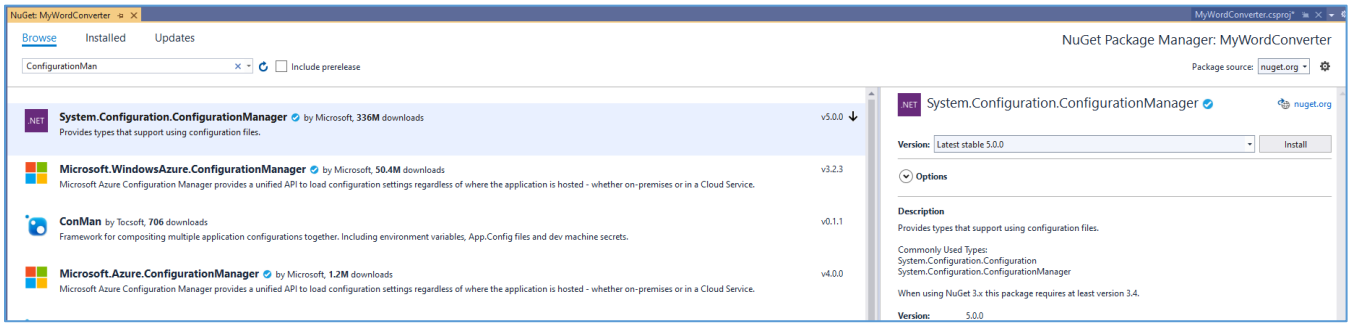

Figure 5.

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;
```

3.1.1 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**.

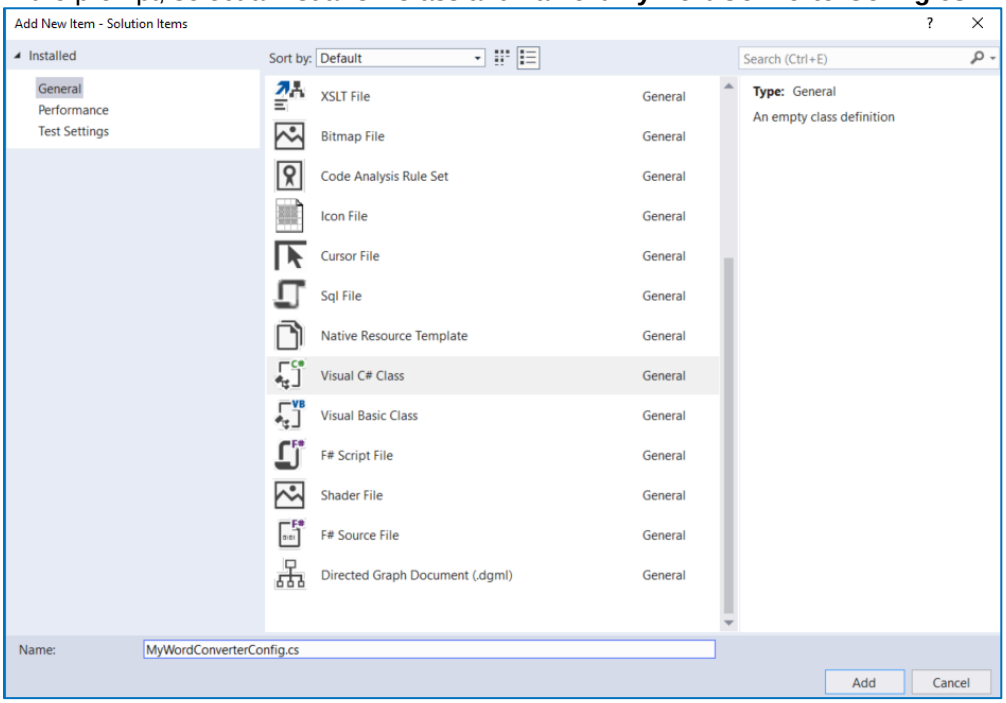

Figure 6.

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;
```

3.1.2 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().GetConverterConfigura
tion().GetSection("ConverterSettings/MyWordConverter");
            string pdfSuffix = configuration.Settings.PdfSuffix;

            string paramExportFilePath =
filePaths[Context.ConversionTask.FileID].FullName.Replace(fileExt, pdfSuffix + ".pdf");
            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
    …
```

3.2 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.

4 Deploying the Example Converter DLL

Once the converter is ready to use, it needs to be included in the Conversion Server setup. This section describes the steps to take when creating a new Conversion Rule using the Microsoft Word 2016 example.

4.1 Add the Converter to the ConversionServerConfig file

1. On the Conversion Server, open the **ConversionServerConfig.xml** in a text editor.
2. Within the existing `configSections` tags at the top of the `ConversionServerConfig.xml`, there is a section group defined for settings that are unique to each Converter.

   Add the following tag:

```
<configSections>
      <sectionGroup name="ConverterSettings">
         <section name="MyWordConverter"
type="MyWordConverter.Configuration.MyWordConverterConfig"></section>
      </sectionGroup>
</configSections>
```

3. Within the **Converters** tag, add a **Converter** tag for the new converter.

   This tag specifies the DLL for the Converter as well as what Class within the DLL performs the conversion:

```
<ConversionServer>
   <Converters>
      <Converter name="MyWordConverter"
type="MyWordConverter.MyWordConverter, MyWordConverter" />
      …
   </Converters>
</ConversionServer>
```

   The attributes of the tag are as follows:

   - **Name**: Name of the Converter. This value must match the ‘Conversion Type’ Item’s name.
   - **Type**: The text before the comma refers to the Class that is defined within the Converter DLL that performs the conversion.

     The text after the comma refers to the name of the converter’s DLL in the ‘ConversionServer\bin\’ folder.

     Note: Set the ‘url’ attribute of the ‘InnovatorServer’ tag to the ‘InnovatorServer.aspx’ page of the Aras Innovator URL that uses this Conversion Server.

4. Within the existing `ConverterSettings` tags, add a tag for the new converter. The tag must match the name configured in step 2.
   a. Set the `pdfSuffix` to some string that you wish to add to the end of the generated PDF file name.

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

5. All “type” attributes in both the “section” tag and “Converter” tag must contain the assembly name (“Conversion.Base” for ConversionServer, “ArasCadConverter” for cad converter etc).

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
        <configSections>
                <!-- Common converter service configuration -->
                <section name="ConversionServer"
type="Aras.ConversionFramework.ConversionServer.Configuration.ConversionServe
rConfigurationSection, Conversion.Base"/>
                <sectionGroup name="ConverterSettings">
                        <section name="ArasCadConverter"
type="Aras.ConversionFramework.Converter.Hoops.Configuration.HoopsConverterCo
nfiguration, ArasCadConverter">
                        </section>
                       <!-- … -->
                </sectionGroup>
        </configSections>
        <ConversionServer>
                <InnovatorServer
url="http://some_url/some_instance/Server/InnovatorServer.aspx"/>
                <Converters>
                        <Converter name="Aras CAD to PDF Converter"
type="Aras.ConversionFramework.Converter.Hoops.HoopsConverter,
ArasCadConverter" />
                </Converters>
                <!-- … -->
        </ConversionServer>
        <ConverterSettings>
                <!-- … -->
        </ConverterSettings>
</configuration>
```

6. Save and close the '`ConversionServerConfig.xml`'

4.2 Add the Converter within Aras Innovator

1. Log on to Aras Innovator as an administrator.
2. Go to **Administration --> File Handling --> Conversion Types**.
3. Add a new 'Conversion Type' with the name equal to the name that is set in the web.config (Step 2. in section 4.1) 'MyWordConverter'.
4. Click 

and 

to save, unclaim, and close the Conversion Type.
5. Go to **Administration --> File Handling --> Conversion Servers**.
6. Open the Default 'Conversion Server' Item for editing.
7. Under the 'Converters' relationship tab, add the newly created 'Conversion Type' Item.
8. Click 

and 

to save, unclaim, and close the 'Conversion Server'.
9. Go to **Administration --> File Handling --> Conversion Rules**.
10. Create a new 'Conversion Rule' and configure the following:
    a. **Name**: MyWordConverter
    b. **Conversion Type**: Search for and return 'MyWordConverter'
    c. **Description**: Converts doc and docx files to PDF
    d. **Timeout**: 60 (set in minutes)

       Number of minutes after which the actual converting of the physical file is considered as hung and the conversion task is marked as failed. In other words, this is the time allowed to the OnConvert event handler to return a result; suggested default is 60 minutes.

    e. **Delay**: 5 (set in minutes)

       The delay in minutes after which a failed task can be rerun; suggested default is 5 minutes.

    f. **Cutoff**: 24 (set in hours)

       The number of hours that is given to a failed conversion to finish successfully; If in cutoff hours after the event handler was first tried it still has status Failed, the conversion task is marked as failed and removed from processing by the framework. The suggested default value is 24.

    g. **Enabled**: 1 (Checked to enable the rule)

h. **Relationship Tabs**:

- **FileTypes**: Add all ‘File Type’ Items (file extensions) that trigger the Conversion Rule:

  If desired, you can create multiple ‘Conversion Rule’ Items for the same ‘Conversion Type’ to handle different FileTypes uniquely.

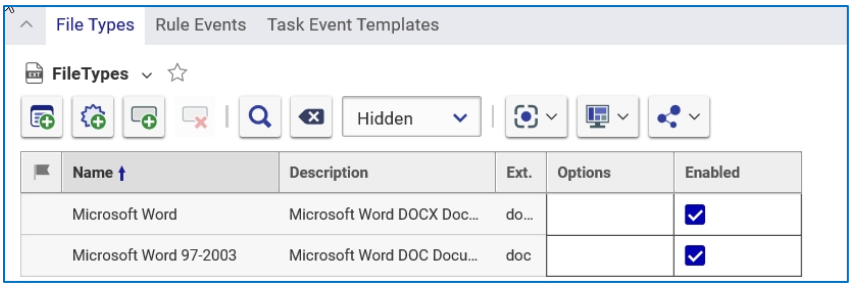

Figure 7.

- **Rule Events**: none (Methods placed ‘onTaskCreating’)

  Methods may be placed on this event to prevent the Conversion Task from being created if certain criteria are met.

- **Task Event Templates**: Add methods to be executed at various points during the conversion.

  For the MyWordConverter, the following methods need to be added. The methods are available in Appendix B at the end of the document.

  - **ArasWord_initDependencies**: Adds dependency on the Document Item that is the parent of the File that was added or updated. Cancels conversion if there is no Document Item found.

    Items listed added as dependencies are claimeded during events that have ‘Lock Dependencies’ set to True (‘1’).

  - **ArasWord_attachFile**: Attaches the resulting File from the Conversion to the same Document as the source Word file.

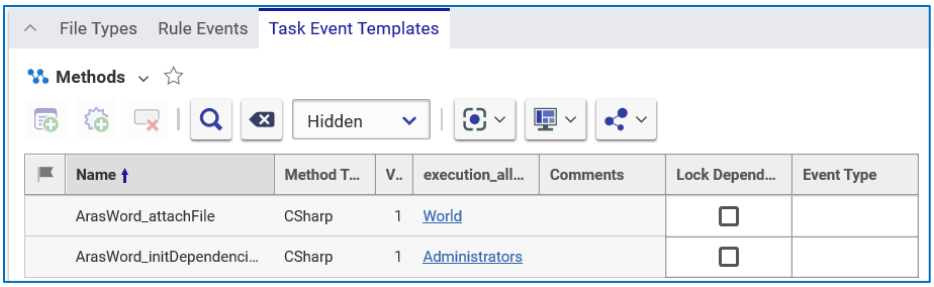

Figure 8.

11. Click 

and 

to save, unclaim, and close the new ‘Conversion Rule’ Item.

Once the configuration is complete, the system is ready to use the new converter. Simply add a Word document on the Document Files tab of a Document Item in Aras Innovator and it is automatically converted.

Note:	Remember that this document serves only as an example for integrating a file conversion utility with the Conversion Server. The example in this document is not meant for production use. Microsoft Word is not intended for use as a file converter without the direct interaction of the end user.

5 Appendix A

5.1 Conversion Data Model

This section describes the complete data model for the File Conversion feature.

5.1.1 ConversionServer

The 'ConversionServer' ItemType is where the URL for the Conversion Server is stored. In this regard, it is like the Vault ItemType.

Properties:

- **Name**: Unique name for the Conversion Server.

  For example, this could describe what types of Conversions are handled by this Conversion Server, i.e., 'Word Native Server.'

  Aras Innovator has the 'Default' Conversion Server defined for use with the Actify Converter by default.
- **URL**: HTTP path to the '/ConversionServer/ConversionService.asmx' file
- **Impersonation User**: Aras Innovator User that is used to perform the conversion. All Files created by the Conversion Server have their 'Created_by' as this User.

Note:	The Impersonation User either must be the Vault Admin User or must be a member of the Administrators Identity.

Relationships:

- **Converters** ('ConversionServerConverterType'): Relationship to 'ConverterType'

  Used to select which Converters are executed using this Conversion Server.

5.1.1.1 ConversionServerPriority

The 'ConversionServerPriority' ItemType is a Relationship from the 'Vault' ItemType to the 'ConversionServer' ItemType.

This Relationship specifies which Conversion Server to use based on the Vault that the File was added to or updated in.

5.1.2 ConverterType

The 'ConverterType' ItemType uses the name of the Converter to represent the Converter DLL inside of Aras Innovator.

Properties:

- **Name**: Unique name for the Converter. This name must match the name attribute set in the 'Converter' tag of the '\ConversionServerConfig.xml' or '\ConversionServer\web.config' file as follows:

```
<ConversionServer>
      …
      <Converters>
```

```
        <Converter name="MyWordConverter" … />
    </Converters>
</ConversionServer>
```

5.1.3 ConversionRule

The ‘ConversionRule’ ItemType is where the rules and events for the conversion are defined.

Properties:

- **Name**: Unique name to describe what the Rule does using the Conversion Server.
- **Conversion Type**: Points to the ‘ConverterType’ Item that performs the Conversion.
- **Description** (optional): Detailed description of the Conversion Rule.
- **Timeout**: Number of minutes after which the actual converting of the physical file is considered as hung and the conversion task is marked as failed.

  In other words, this is the time allowed to the OnConvert event handler to return a result; suggested default is 60 minutes.
- **Delay**: The delay in minutes after which a failed task can be rerun; suggested default is 5 minutes.
- **Cutoff**: The number of hours that is given to a failed Conversion Task to finish successfully.

  If the Conversion Task has status Failed for the specified number of hours, the conversion task is marked as failed and removed from processing by the framework. The suggested default value is 24.
- **Enabled**: Boolean to enable or disable the Conversion Rule.

Relationships:

- **File Types** (‘ConversionRuleFileType’): Relationship to ‘FileType’

  Used to specify which types of files trigger this Conversion Rule. Each FileType can be enabled or disabled using the ‘Enabled’ property on the Relationship.
- **Rule Events** (‘ConversionRuleEventHandler’): Relationship to ‘Method’

  Used to specify Server-side methods that are run prior to the creation of the Conversion Task. (Event Type = OnTaskCreating)

  These methods can then be used to check certain conditions such as the File size and prevent the Conversion Task from being created.
- **Task Event Templates**(‘ConversionRuleEventTemplate’): Relationship to ‘Method’
  - Used to specify Server-side methods that are run at different times while the Conversion Task is processing.
  - Conversion mode - blocking vs non-blocking (default):
    Non-blocking is intended to process long-running tasks (more than 30 minutes); the Innovator Server sends a request to the Conversion Server to execute a Task, and the Conversion Server notifies the Innovator Server about completion, so the Innovator doesn't wait for completion, which avoids the possibility of a timeout.

  Available Events (In order of execution):
  - **OnStartTaskProcessing**: Fired right after the conversion task is created and the processing cycle gets the task for processing.
  - **OnBeforeConvert**: Fired right before the conversion task is ready to be sent for conversion.

- **OnConvert**: Overrides the conversion event.
- **OnAfterConvert**: Fired on the processing cycle iteration after the framework sets result to ConversionTaskResult.

5.1.4 ConversionTask

'ConversionTask' Items are created automatically based on enabled Conversion Rules. Each 'Conversion Task' represents one File being converted by a Conversion Server.

All properties and relationships are automatically populated.

Properties:

- **Status**: Current status of the Task. Can be 'Not Started', 'In Progress', 'Succeeded', 'Failed', or 'Discarded'.
- **Rule**: Points to the Conversion Rule that was used for the creation of this Conversion Task.
- **File Type**: The type of File that is being converted.
- **File**: The ID of the File being converted.
- **Started On**: Date that the Conversion Task started.
- **Finished On**: Date that the Conversion Task finished.
- **User Data**: Text (most probably XML) that could be used by a specific converter to carry the converter-specific data that has to be eventually passed to the process that performs actual conversion of the physical file (e.g., conversion options; list of supplementary files that must be given to the converter in order to convert the file; etc.).
- **Error**: Error description in case the task status is Failed or Discarded.

Relationships:

- **Dependencies** ('ConversionTaskDependency'): Null Relationship

  Displays the Type and ID of Items that can be claimed by methods during the processing of the Conversion Task.

- **Results** ('ConversionTaskResult'): Null Relationship

  Displays the File IDs that have been created because of the Conversion Task and an additional string 'kind' to possibly describe why it was created / what it was created for.

- **Event Handlers** ('ConversionTaskEventHandler'): Relationship to 'Method'

  Events copied from the 'Task Event Templates' Relationship on the 'Conversion Rule' ItemType.

  Displays the status of each Event during the processing of the Conversion Task with the following properties:

  - **Error** […]: Error message set if the Method failed.
  - **User Data** […]: Used by the event handler to pass some information between different invocations of the event handler (e.g., in case the event handler failed it could save in this property some information about what it managed to finish before it failed so that the next invocation of the method does not try to do it again).
  - **Started On** […]: Date when the first invocation of the method started.
  - **Finished On** […]: Date when the last invocation of the method finished.

- **Execution Attempt**: The number of times the Method has been called by this Conversion Task.
- **Status**: Status of the Event. It can be Not Started, In Progress, Failed, or Succeeded.

6 Appendix B

6.1 Visual Studio C# Sample Code

6.1.1 MyWordConverter.cs

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

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().GetConverterConfigura
tion().GetSection("ConverterSettings/MyWordConverter");
            string pdfSuffix = configuration.Settings.PdfSuffix;

            string paramExportFilePath =
filePaths[Context.ConversionTask.FileID].FullName.Replace(fileExt, pdfSuffix + ".pdf");
            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
    }
}
```

6.1.2 MyWordConverterConfig.cs

```
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;
            }
        }
    }
}
```

6.2 Aras Innovator Sample C# Methods

6.2.1 ArasWord_initDependencies

```
Aras.ConversionFramework.Models.ConversionTask task = new
Aras.ConversionFramework.Models.ConversionTask()
{
  Item = this
};

Innovator inn = this.getInnovator();

String docQuery = @"
<AML>
  <Item type='Document' action='get'>
    <Relationships>
      <Item type='Document File' action='get'>
        <related_id>" + task.FileID + @"</related_id>
      </Item>
    </Relationships>
  </Item>
</AML>";

Item res = inn.applyAML(docQuery);

if(res.isError())
{
  return res;
}

task.AddDependency(res);

return task.Item;
```

6.2.2 ArasWord_attachFile

```
Aras.ConversionFramework.Models.ConversionTask task = new
Aras.ConversionFramework.Models.ConversionTask()
{
  Item = this
};

Innovator inn = this.getInnovator();
```

```
Item results =
task.Item.getItemsByXPath("Relationships/Item[@type='ConversionTaskRes
ult']");
String pdfFile = results.getItemByIndex(0).getProperty("file_id", "");

if(!String.IsNullOrEmpty(pdfFile))
{
   List<String> depIds = new List<String>();
    Item dependencies =
task.Item.getItemsByXPath("Relationships/Item[@type='ConversionTaskDep
endency']");
    for(int depIndex = 0; depIndex < dependencies.getItemCount();
depIndex++)
    {
      Item dependency = dependencies.getItemByIndex(depIndex);
      depIds.Add(dependency.getProperty("dependency_id", ""));
    }

    if(depIds.Count > 0)
    {
     String updateQuery = @"<AML>
     <Item type='Document' id='" + String.Join(",", depIds.ToArray()) +
@"' action='update' version='0'>
       <Relationships>
         <Item type='Document File' action='add'>
           <related_id>" + pdfFile + @"</related_id>
         </Item>
       </Relationships>
     </Item>
     </AML>";

      Item result = inn.applyAML(updateQuery);
      if(result.isError())
      {
        return result;
      }
    }
}

return task.Item;
```