PEERNET Document Conversion Service 3.0

.NET API Utility Quick Start Guide

Easily add document conversion into your own C# and VB.NET programs with the PEERNET.ConvertUtility.dll included with Document Conversion Service.

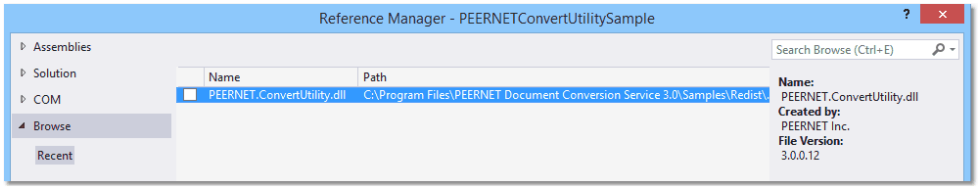
It only takes a few lines of code to add any of the conversion features below to your program:

- convert a single file or an entire folder of files
- convert a dynamic list of files
- combine multiple files or a folder of files into a single TIFF or PDF
- get full details of the results of the conversion

Adding Conversion to Your Program

> Before you begin! Document Conversion Service needs to be running to convert files. If the service is not started, the conversion methods will fail.

Add the reference to the PEERNET.ConvertUtility.dll to your project using the *References Manager*. The DLL is located in the \Samples\Redist folder under your Document Conversion Service installation location and is built against .NET Framework 4.5.



Finally, add the following *using* statement at the top of your code. You are now ready to call the conversion methods in your program.

```
using PEERNET.ConvertUtility;
```

Setting the Output Type

Each conversion method uses a *conversion profile* to control the type of file that will be created. Document Conversion Service includes over 60 conversion profiles to create many types of images and both vector and raster PDF files.

Each conversion profile is a simple XML-formatted text file containing a list of settings that determines the type of file created. These files can be copied and edited in any text editor to create new profiles as needed. See What are Conversion Profiles? to learn more about creating your own profiles.

Profiles are specified by name in the argument list to the conversion methods. If you want to keep custom profiles with your own program you can also pass profiles by using a full file path.

Reading the Results

The conversion methods return either a single PNConversionItem object, in the case of converting a single file, or an IList collection of *PNConversionItem* objects when converting a folder or a list of files.

The *PNConversionItem* class contains information about the original conversion request. It includes an inner object, PNConversionResult, that contains the list of converted files for successful conversions, and error messages if the conversion did not succeed.

The combine method returns a PNCombineItem object. It contains a collection of *PNConversionResult* items, one for each file in the supplied list of files added to the combined file. It also contains the list of files passed to be combined and a list of the resulting combined files.

> All underlined options in the included code snippets can be replaced with your own input file and folder paths, conversion profiles and output paths.

Convert a File

The ConvertFile method converts a single file and saves the new file in the same location.

A folder and file name can be passed in to save the new file in a different location. When saving to a different location the output folder must already exist.

If a file of the same name already exists the conversion returns with an error. You can disable this behavior by setting the overwrite option to *true*.

Below we are using the conversion profile '*TIFF 200dpi OptimizedColor*' to convert a PDF file to a TIFF image. Changing the profile used will change the type of file created.

This method does not return until conversion is complete or an error occurs.

```
PNConversionItem resultItem = null;

// This is the single call needed to convert a file
resultItem = PNConverter.ConvertFile(@"C:\Test\File.pdf",
                                     @"C:\Test\Output",
                                     @"ConvertedFromPDF",
                                     true, // overwrite existing
                                     true, // remove file ext
                                     false, // create log
                                     "TIFF 200dpi OptimizedColor",
                                     String.Empty, String.Empty,
                                     null, String.Empty,
                                     String.Empty, String.Empty);
```

The ConvertFile Result

The returned PNConversionItem contains all the information about the conversion request and the status and results of the actual conversion. It can be queried for success or failure and contains an inner PNConversionResult item used to iterate though the created files or any error information as needed.

```
if (resultItem != null) {
   // With single file conversion this is a single item
   // The PNConversionResultItem object contains the error and file list.
   // Failed items will have an error list > 0 and no output files.

   Console.WriteLine("Conversion Item: " + resultItem.SourceFilePath);
   Console.WriteLine("==========================================");

   if (resultItem.HasErrors()) {
      if (resultItem.ConversionresultItem.Errors.Count > 0) {

         Console.WriteLine("Errors occured during conversion: ");

         foreach (PNConversionresultItemError itemError in
                 resultItem.ConversionresultItem.Errors) {
            Console.WriteLine(itemError.Value);
         }
      }
   } else {
      if (resultItem.ConversionresultItem.OutputFiles != null) {
         if (resultItem.ConversionresultItem.OutputFiles.Count > 0) {

            Console.WriteLine("The following files where created: ");

            foreach (PNConversionresultItemOutputFile itemOutputFile in
                    resultItem.ConversionresultItem.OutputFiles) {
               listBox1.Items.Add(itemOutputFile.OutputFilePath);
            }
         } else {
            Console.WriteLine("No files were created.");
         }
      }
   }
} else {
   Console.WriteLine("Conversion module did not run.");
}
```

Convert a Folder of Files

The ConvertFolder method will convert all files in the given folder and optionally any subfolders as well.

Search pattern filters can be used to decide what files to include or exclude from the folders. The include filter, which defaults to all files (*.*) is applied first. The exclude filter defaults to no files and is applied after. Multiple filters can be combined using the pipe (|) character, such as *"*.doc|*.pdf"* to process only Word and PDF files. Files can be sorted by name, date created or modified, and ascending or descending order.

The structure of the input folder is mirrored in the output folder, which must exist before the method is called. If no output folder is given, a *.converted* folder is created in the root of the source folder and used as the output folder. If a file of the same name already exists that file's conversion will return with an error. You can disable this behavior by setting the overwrite option to *true*.

The conversion profile '*TIFF 200dpi Grayscale*' is used to convert all PDF and Word files in the folder and subfolders to grayscale TIFF images. This method will not return until all files meeting the criteria have been processed.

```
IList<PNConversionItem> results = new List<PNConversionItem>();

results = PNConverter.ConvertFolder(@"C:\Test\InputFiles",
                                    true, // include subfolders
                                    "*.pdf|*.doc|*.docx", // Word and PDF
                                    String.Empty, // exclude filter
                                    @"C:\Test\Output", // output folder
                                    true, // overwrite existing
                                    true, // remove file ext
                                    false, // create log
                                    "TIFF 200dpi Grayscale",
                                    String.Empty, String.Empty,
                                    null, String.Empty,
                                    String.Empty, string.Empty,
                                    PNFileSortMode.Name,
                                    PNFileSortOrder.Descending);
```

The ConvertFolder Results Collection

A collection of PNConversionItem objects is returned, one for each file found to convert in the folder and or any subfolders.

Each item contains all the information about the conversion request for each file found in the folder and the status and results of that file's actual conversion.

The item can be queried for success or failure and contains an inner PNConversionResult item used to iterate though the files created from that file or any error information as needed.

```
if (results != null) {
    foreach (PNConversionItem item in results) {
        if (item != null) {

            Console.WriteLine("Item: " + item.SourceFilePath);
            Console.WriteLine("      " + item.OutputDirectory);
            Console.WriteLine("      " + item.OutputBaseName);

            if (item.HasErrors() == false) {

                foreach (PNConversionResultOutputFile outputfile in
                         item.ConversionResult.OutputFiles) {

                    Console.Write("Converted to: ");
                    Console.WriteLine(outputfile.OutputFilePath);
                }
            } else {

                foreach (PNConversionResultError errorItem in
                         item.ConversionResult.Errors) {

                    Console.Write("    Error: ");
                    Console.WriteLine("errorItem.Value);
                }
            }
        }
    }
} else {
   Console.WriteLine("Conversion module did not run.");
}
```

Convert a List of Files

You can convert a list of files from different locations using the ConvertFileList method. Each file in the list can have different conversion settings and even different output locations.

To use the output folder and conversion profile provided as arguments to the *ConvertFileList* method, pass an IList collection of paths to each file.

```
IList<String> simpleFileList = new List<String>();

simpleFileList.Add(@"C:\Test\InputPDF\File1.txt");
simpleFileList.Add(@"C:\Test\InputPDF\File2.pdf");
simpleFileList.Add(@"C:\Test\InputWord\File1.doc");
```

For custom settings and output locations per file, an IList of PNConvertFileInfo items is used. Each item contains the path to the input file to be converted and optionally a separate output folder and a private settings collection. When not empty, the folder and settings update the arguments passed to *ConvertFileList*.

Custom settings are passed as a a collection of PNSetting items. Each item is a name-value pair of conversion setting like those used to when creating conversion profiles.

```
// Create file list to convert
IList<PNConvertFileInfo> fileList = new List<PNConvertFileInfo>();

// Uses the folder and settings in the ConvertFileList method
fileList.Add(new PNConvertFileInfo(@"C:\Test\InputPDF\File1.txt",
                                   String.Empty, null));

// Sets output folder, uses ssettings in the ConvertFileList method
fileList.Add(new PNConvertFileInfo(@"C:\Test\InputPDF\File2.pdf",
                                   @"C:\Test\Output\ConvertedPDF",
                                   null));

// Sets output folder, adds single setting to ConvertFileList method
IList<PNSetting> filesettings = new List<PNSetting>();
filesettings.Add(new PNSetting("Devmode settings;Resolution", "300"));
fileList.Add(new PNConvertFileInfo(@"C:\Test\InputWord\File1.doc",
                                   @"C:\Test\Output\ConvertedDocs",
                                   filesettings));
```

The last step before calling ConvertFileList is to create any missing output folders. If the output folder for a file does not not exist that file's conversion will fail.

```
// Test output directories, they need to exist
foreach (PNConvertFileInfo info in fileList) {
    if (!String.IsNullOrEmpty(info.OutputPath) &&
        !Directory.Exists(info.OutputPath)) {
        Directory.CreateDirectory(info.OutputPath);
    }
}

// Create main output folder
if (!Directory.Exists(@"C:\Test\Output\Converted")) {
    Directory.CreateDirectory(@"C:\Test\Output\Converted");
}
```

Converting the List

With the custom file list above, the first file will be saved in the output folder passed to the *ConvertFileList* method, and the last two will go to their respective folders as set in the *PNConvertFileInfo* items.

If the output file already exists that file's conversion will return an error. You can disable this behavior by setting the overwrite option to *true*.

The last file is also saved at a resolution of 300dpi instead of 200dpi specified by the conversion profile '*TIFF 200dpi OptimizedColor*' passed into the *ConvertFileList* method.

This method will not return until all files meeting the criteria have been processed.

```
IList<PNConversionItem> results = new List<PNConversionItem>();

results = PNConverter.ConvertFileList(fileList,
                                      @"C:\Test\Output\Converted",
                                      String.Empty, // use source base name
                                      true, // overwrite existing
                                      true, // remove file ext
                                      false, // create log
                                      "TIFF 200dpi OptimizedColor",
                                      String.Empty, String.Empty,
                                      null, String.Empty,
                                      String.Empty, String.Empty);
```

The ConvertFileList Results Collection

A collection of PNConversionItem objects is returned, one for each file in the list of files passed to be converted.

Each item contains all the information about the conversion request for that file and the status and results of the actual conversion.

The item can be queried for success or failure and contains an inner PNConversionResult item used to iterate though the files created from that file or any error information as needed.

```
if (results != null) {
    foreach (PNConversionItem item in results) {
        if (item != null) {

            Console.WriteLine("Item: " + item.SourceFilePath);
            Console.WriteLine("      " + item.OutputDirectory);
            Console.WriteLine("      " + item.OutputBaseName);

            if (item.HasErrors() == false) {

                foreach (PNConversionResultOutputFile outputfile in
                         item.ConversionResult.OutputFiles) {

                    Console.Write("Converted to: ");
                    Console.WriteLine(outputfile.OutputFilePath);
                }
            } else {

                foreach (PNConversionResultError errorItem in
                         item.ConversionResult.Errors) {

                    Console.Write("    Error: ");
                    Console.WriteLine("errorItem.Value);
                }
            }
        }
    }
} else {
   Console.WriteLine("Conversion module did not run.");
}
```

Combine a List of Files

CombineFiles will append a list of files into a single output file or a serialized sequence of single page output files. The files are converted and combined together in the order in which they are listed.

The files can be passed as a simple IList collection of paths to each file.

```
IList<String> simpleFileList = new List<String>();

simpleFileList.Add(@"C:\Test\PDF\InputFile1.pdf");
simpleFileList.Add(@"C:\Test\DOC\InputFile2.doc");
simpleFileList.Add(@"C:\Test\XLS\InputFile3.xls");
```

To combine a collection of files, each with their own settings, an IList collection of customized PNConvertFileInfo items is used. In this case only the input folder and settings properties on the PNConvertFileInfo items are used; the output folder property is ignored.

Each PNSetting item is a name-value pair of conversion setting like those used to when creating conversion profiles.

```
IList<PNConvertFileInfo> fileList = new List<PNConvertFileInfo>();
IList<PNSetting> filesettings = new List<PNSetting>();

// This file will print all pages and uses only the conversion
// settings from the profile - we aren't passing any additional settings.
fileList.Add(new PNConvertFileInfo(@"C:\Test\PDF\InputFile1.pdf",
                                   String.Empty, // Not used in combine
                                   null));

// This file only prints the first 3 pages
filesettings.Add( new PNSetting("PageRange", "1-3"));
fileList.Add(new PNConvertFileInfo(@"C:\Test\DOC\InputFile2.doc",
                                   String.Empty, // Not used in combine
                                   filesettings));

// This file only prints the first page
filesettings.Clear();
filesettings.Add( new PNSetting("PageRange", "1"));
fileList.Add(new PNConvertFileInfo(@"C:\Test\XLS\InputFile3.xls",
                                   String.Empty, // Not used in combine
                                   filesettings));
```

Combining the Files

Once the list of files is created, the last step before calling CombineFiles is to create any missing output folders.

When combining files, the output directory and final output file name must be provided and the directory must exist before the call is made.

```
// Create main output folder
if (!Directory.Exists(@"C:\Test\CombineOutput")) {
    Directory.CreateDirectory(@"C:\Test\CombineOutput");
}
```

The code sample below uses the *PNConvertInfo* file list created above to append all three files into a single multipaged TIFF image. The resulting file will contain all of the pages of the first file, the first three pages of the second file, and the first page of the last file.

If the output file already exists the combine will return an error. You can disable this behavior by setting the overwrite option to *true*.

The combined file will be created using the conversion settings from the profile *TIFF 200dpi OptimizedColor*. You can change this to use any profile you require. The final file is created in the output folder specified.

```
PNCombineItem resultsItem = null;

resultsItem =
    PNConverter.CombineFiles(fileList,  // files collection
                             @"C:\Test\CombineOutput", // output folder
                             "CombinedInput",  // name of combined file
                             true,       // overwrite
                             false,      // create results log
                             "TIFF 200dpi OptimizedColor",
                             String.Empty, String.Empty,
                             null, String.Empty,
                             String.Empty, String.Empty
                             );
```

The PNCombineItem Result

The returned PNCombineItem contains information about the original combine request and the input files used.

It can be queried for success or failure and can contain a list of created files or a collection of error messages detailing why the files were not combined.

It also contains collection of PNConversionResult objects that lists the results of the conversion for each input file before it was combined.

```
if (resultsItem != null) {

    Console.WriteLine("*******************************");
    Console.WriteLine("* Combined ITEM               *");
    Console.WriteLine("*******************************");
    Console.WriteLine("BaseName: " + resultsItem.OutputBaseName);
    Console.WriteLine("Directory: " + resultsItem.OutputDirectory);
    Console.WriteLine("Input Files:");

    foreach (String inputFile in resultsItem.InputFiles) {
        Console.WriteLine("  - " + inputFile);
    }

    Console.WriteLine("Combined Output:");
    if (resultsItem.CombinedOutputFileList.Count == 0) {
        Console.WriteLine("  - None");
    }

    foreach (String combinedFile in resultsItem.CombinedOutputFileList) {
        Console.WriteLine("  - " + combinedFile);
    }

    if ( resultsItem.HasErrors() == true) {
        foreach (PNConversionResultError errorItem in resultsItem.Errors) {
            Console.WriteLine("  - Error: " + errorItem.Value);
        }
   }
}
```

Combine a Folder of Files

CombineFolder combines all files, or all files matching a search filter pattern, into a single output file or a serialized sequence of single page output files. You can choose to include all files in subfolders under the starting folder as well.

Files can be sorted by name, date created or modified, and ascending or descending order. Files from the root of the input folder are sorted first, then all files from the subfolders when enabled. Subfolders are listed in alphabetical or creation time order, dependent on the file system.

Search pattern filters can be used to decide what files to include or exclude from the folder. The include filter, which defaults to all files (*.*) is applied first. The exclude filter defaults to no files and is applied after. Multiple filters can be combined using the pipe (|) character, such as *"*.doc|*.pdf"* to process only Word and PDF files.

Combining the Files in the Folder

Once you have determined any file filters and if you are including any subfolders, the last step before calling CombineFolder is to create any missing output folders.

When combining files from a folder, the output directory and final output file name must be provided and the directory must exist before the call is made.

```
// Create main output folder
if (!Directory.Exists(@"C:\Test\CombineOutput")) {
    Directory.CreateDirectory(@"C:\Test\CombineOutput");
}
```

The code sample below appends into a single PDF file all files in the folder and subfolders that start with *ABC_*, and excludes any existing PDF files.

If the output file already exists the combine will return an error. You can disable this behavior by setting the overwrite option to *true*.

The combined file will be created using the conversion settings from the profile *PDF 200dpi OptimizedColor*. You can change this to use any profile you require. The final file is created in the output folder specified.

```
PNCombineItem resultsItem = null;

resultsItem =
    PNConverter.CombineFolder(@"C:\Test\Input\",,  // input folder
                              true, // include subfolders
                              @"ABC_*.*", // process files matching this
                              @"*.pdf", // don't inlcude existing PDF
                             @"C:\Test\CombineOutput", // output folder
                             "CombinedInput",  // name of combined file
                             true,       // overwrite
                             false,      // create results log
                             "PDF 200dpi OptimizedColor",
                             String.Empty, String.Empty,
                             null, String.Empty,
                             String.Empty, String.Empty
                             );
```

The PNCombineItem Result

The returned PNCombineItem contains information about the original combine request and the input files found.

It can be queried for success or failure and can contain a list of created files or a collection of error messages detailing why the files were not combined.

It also contains collection of PNConversionResult objects that lists the results of the conversion for each input file before it was combined.

```
if (resultsItem != null) {

    Console.WriteLine("*******************************");
    Console.WriteLine("* Combined ITEM                *");
    Console.WriteLine("*******************************");
    Console.WriteLine("BaseName: " + resultsItem.OutputBaseName);
    Console.WriteLine("Directory: " + resultsItem.OutputDirectory);
    Console.WriteLine("Input Files:");

    foreach (String inputFile in resultsItem.InputFiles) {
        Console.WriteLine("  - " + inputFile);
    }

    Console.WriteLine("Combined Output:");
    if (resultsItem.CombinedOutputFileList.Count == 0) {
        Console.WriteLine("  - None");
    }

    foreach (String combinedFile in resultsItem.CombinedOutputFileList) {
        Console.WriteLine("  - " + combinedFile);
    }

    if ( resultsItem.HasErrors() == true) {
        foreach (PNConversionResultError errorItem in resultsItem.Errors) {
            Console.WriteLine("  - Error: " + errorItem.Value);
        }
   }
}
```

What are Conversion Profiles?

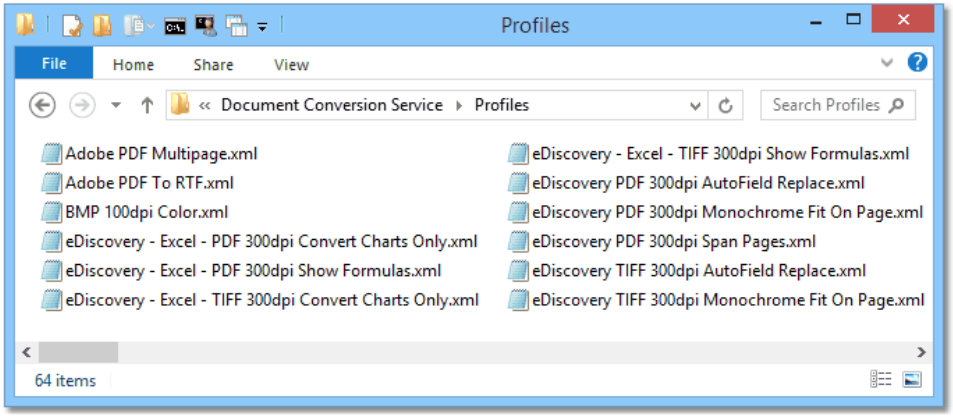
A *conversion profile* is a XML-formatted text file containing a list of settings that determines the type of file created. By passing the name of the profile into the method call you control the type of file created.

Document Conversion Service includes a large selection of profiles for common types of output. These sample profiles can be used as-is, or they can be edited or copied to create new profiles for your own use.

There are sample profiles included to create:

- vector or raster PDF files.
- TIFF, JPG, and BMP images at various resolutions and color models.
- fax format TIFF with CCITT Group4 compression.
- TIFF or PDF files with eDiscovery options for converting Word, Excel, and Powerpoint files.

To see a complete list of the included conversion profiles go to Programs - PEERNET Document Conversion Service 3.0 – Open Conversion Profiles Folder.



Adding or Changing Conversion Profiles

A large collection of settings is available to allow you to customize the provided conversion profiles to create the type of output you need.

The most commonly used settings are grouped below by category with links to the setting that controls it or a complete listings of settings for that category.

Category	Settings Collection
Change the type of file created:	*Save;Output File Format* in Save settings
Change the resolution of the created file:	*Devmode settings;Resolution* in Devmode settings settings
Creating TIFF images and Group 4 Fax TIFF	TIFF File Format Image Options
Creating vector or raster PDF Documents:	Adobe Reader Options PDF File Format PDF Security
Auto-rotate pages and other image processing actions:	Processing
Add eDiscovery settings for Office documents:	Word Converter Options Excel Converter Options PowerPoint Converter Options

For a full listing of all settings details see Conversion Settings in the online user guide.

Parallel Conversion

The number of files that can be converted in parallel is first controlled by the license level of Document Conversion Service that you own.

After that, PEERNET.ConvertUtility also submits the documents on parallel threads. We recommend that you allow this value to be determined automatically based on the number of CPU's and cores on your system for best performance.

If you do decide to customize how many documents to submit in parallel, the setting NumberOfDocumentsInParallel can be passed as part of an additional user setting collection to control how many parallel threads the PEERNET.ConvertUtility uses.

> Take Note! The parallel conversion setting only applies to the ConvertFolder and ConvertFileList methods where you are processing multiple files.

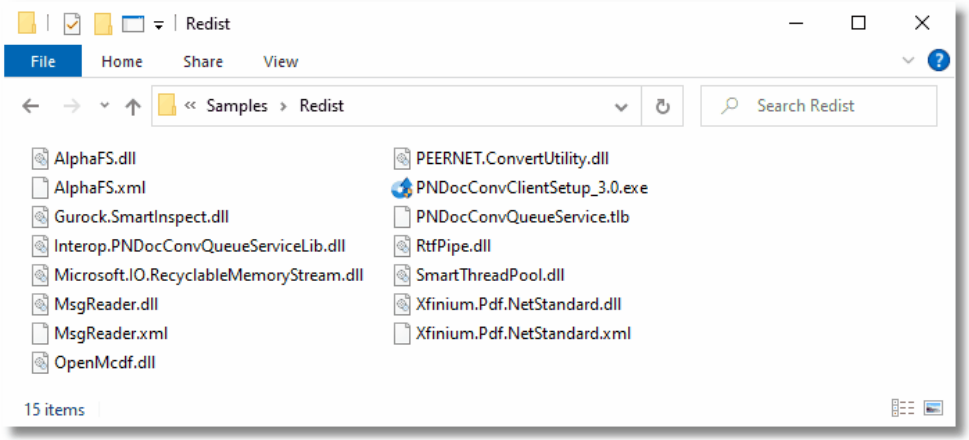
```
PNConversionItem resultItem = null;

// Convert 6 documents in parallel, requires DCS Level II or higher
Dictionary<String, String> customSettings = new Dictionary<String, String>(
customSettings[ "NumberOfDocumentsInParallel" ] = "6";

resultItem = PNConverter.ConvertFolder(@"C:\Test\InputFiles",
                                       true, // include subfolders
                                       "*.*", // all supported files
                                       String.Empty, // exclude filter
                                       @"C:\Test\Output", // output folder
                                       true, // overwrite existing
                                       true, // remove file ext
                                       false, // create log
                                       "TIFF 200dpi OptimizedColor",
                                       String.Empty, String.Empty,
                                       customSettings, // User settings
                                       String.Empty, String.Empty,String.Em
```

Deploying your application

When deploying applications built with PEERNET.ConvertUtility, the following files must be included with your application:

- PEERNET.ConvertUtility.dll
- Gurock.SmartInspect.dll
- SmartThreadPool.dll
- Xfinium.Pdf.Win.dll, Xfinium.Pdf.Win.xml starting with Document Conversion Service 3.27
  - XFinium.Pdf.Pcl.dll, XFinium.Pdf.Pcl.xml prior to Document Conversion Service 3.27
- AlphaFS.dll
- AlphaFS.xml

When you add the PEERNET.ConvertUtility.dll as a reference into your .NET application, set its Copy Local property to *True* to have this library and its dependencies automatically copied to your build output path. If you need to manually copy these files you can find them in the \Samples\Redist folder under the Document Conversion Service installation tree.



Other Files

Any custom conversion profiles that you may have created will also need to be deployed with your application.

Get Other Quick Start Guides

Document Conversion Service Quick Start Guide

A short and quick guide to help you get the conversion service running and converting documents on your server. The Document Conversion Service Quick Start Guide covers starting the conversion service for the first time, using the logging console and converting documents.

Watch Folder Conversion

Monitor any number of folders for documents to convert with the included Watch Folder Service utility. The Watch Folder Quick Start Guide introduces you to setting up folders, handling large volumes of files and running custom commands and other handy features. Perfect for companies with a small to medium volume of daily conversions on a single server.

Clustered Watch Folder for Large Data

For larger companies with a need to convert an existing, very large volume of data, or a large volume of files daily, multiple installations of Document Conversion Service and the included Watch Folder service can provide high-performance clustering and fail over protection with load-balancing and 100% conversion stability. The Clustered Watch Folder Conversion Quick Start shows how easy it is to set up.

Client Server Conversion Quick Start Guide

See how easy it is to setup client-server, or remote, conversion with Document Conversion Service. Remote conversion allows one or more client applications on separate computers to convert files by 'talking' to a server computer, where the actual file conversion takes place. Read through the Client Server Conversion Quick Start Guide to see how this can work for your company.

Command Line Tools Guide

For Java programmers and adding conversion into scheduled tasks, batch files, or any program that can call an external program, command line tools for converting files and folders are included. If this is what you are looking for, the Command Line Tools Quick Start Guide will get you started.

Our Support Team is Here to Help

Not sure where to go from here? Have questions? Our support team is always available to help you choose the correct solution for your conversion needs. Please contact us and we'll be in touch via e-mail, telephone or web meeting to help answer your questions.