

XF Rendering Server 2007 .NET SDK Tutorial and QuickStarts

Copyright© 2002-2006 ECRION Software. All Rights Reserved.

This document is designed to quickly acquaint .NET programmers with the programming model of XF Rendering Server 2007 .NET SDK (Software Development Kit).

Please contact Technical Support at support@ecrion.com if you need additional information about this product, or visit our Web Site at www.ecrion.com.

Table of Contents

About XF Rendering Server 2007	2
Product Features	2
Before You Get Started	2
Platform Requirements	2
Feedback and Support	2
XF Rendering Server 2007 .NET SDK Tutorial	3
Hello World!	4
Web Access	6
XSL Transformations	8
In Memory Operations	10
Selective Rendering and Thumbnails	11
Using XF Merge Contexts	12
XSL-FO Viewer .NET Control	13
Creating a test application	13
Sample Web Application Step-By-Step	15
Prerequisites	15
Tutorial Steps	15
Additional information	18

Last updated: August 2006

Important Notice: This document and the information within is furnished "as is" and is subject to change without notice. In no event shall the author be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) arising out of the use of or inability to use this product, even if the author has been advised of the possibility of such damages.

This document was generated using XF Rendering Server 2007. For the latest version, visit [Technical Resources](#) section on our web site.

About XF Rendering Server 2007

XF Rendering Server 2007 is a highly scalable, enterprise class rendering product. It can be used to automate the creation of electronic documents like technical manuals, brochures, proposals, business reports containing charts and graphs, by dynamically generating them from XML.

XF Rendering Server 2007 supports two major industry standards: XSL-FO (Extensible Style Language Formatting Objects) describing how an XML document should be formatted for a variety of media as well as SVG (Scalable Vector Graphics) used to describe two-dimensional vector and mixed vector/raster graphics in XML.

In addition, high quality, all-purpose charts can be generated directly from XML using **xChart** XML language. More information can be found in [xChart 1.0 Language Reference](#).

Product Features

- Supports XSL-FO, SVG, xChart as input.
- Produces PDF, Postscript, HTML, GIF, JPEG, PNG, BMP and other formats.
- Supports TrueType and Postscript Type 1 font embedding.
- Scalable server architecture that can run across multiple CPUs, meeting the high-performance needs of your applications.
- Is accessible from a multitude of development environments: C++, VB, ASP, .NET, Java.
- Includes XF Designer 2004 XSL-FO authoring tool.

Before You Get Started ...

To load and run the tutorial samples, you must have the server product installed. In addition, **XF Rendering Services for .NET** and the **.NET framework** have to be installed as well.

Platform Requirements

XF Rendering Server 2007 can be used in the Microsoft Windows® 2000 Professional and Server, Windows XP and Windows Server 2003 environments.

- Minimum Intel Pentium III, AMD Athlon 500 MHz or better. Intel Pentium IV 2.4 GHz recommended for development computers, dual XEON 3.0 GHz for production servers.
- Minimum 128 Mb RAM, 512 Mb recommended for development computers, 1Gb for productions servers.

Feedback and Support

Send error reports, feature requests, and comments about the SDK documentation or samples directly to the [Technical Support team](#).

Further information about support options can be found on the [Ecrion Web site](#).

XF Rendering Server 2007 .NET SDK Tutorial

The tutorial provides a step-by-step introduction to fundamental areas of programming using the XF Rendering Server 2007 .NET SDK.

SDK samples are included when you install the product. The location of the samples has a path similar to the following. Here, it is assumed that C: is the XF Rendering Server 2007 installation drive:

```
C:\Program Files\Ecrion Software\XF Rendering Server 2007\Samples\
```

The tutorial includes the following sections:

Hello World!	Demonstrates the basic code needed to create a console application using the SDK and generate a PDF document.
Web Access	Describes how to render PDF documents directly into IIS (Internet Information Server) output stream.
XSL Transformations	Demonstrates how to transform XML using a stylesheet and then feed this input into the rendering engine.
In Memory Operations	Demonstrates how to receive rendering output in a memory stream.
Selective Rendering and Thumbnails	Demonstrates how to generate thumbnails and render pages selectively.
Using Merge Contexts	Describes how to render multiple input documents into one output.
XSL-FO Viewer .NET Control	Demonstrates how to create a sample Windows application able to display XSL-FO documents.

Hello World!

The following console program is the PDF version of the traditional "Hello World!" program, which displays the string Hello World!.

```
using System;
using Ecrion.XF;

namespace Samples
{
    class HelloWorld
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            try
            {
                XFEngine engine = new XFEngine();
                engine.Render(
                    "<fo:root xmlns:fo='http://www.w3.org/1999/XSL/Format'>" +
                    "<fo:layout-master-set>" +
                    "<fo:simple-page-master master-name='all-pages'>" +
                    "<fo:region-body region-name='xsl-region-body'"
                    margin='0.7in' />" +
                    "</fo:simple-page-master>" +
                    "</fo:layout-master-set>" +
                    "<fo:page-sequence master-reference='all-pages'>" +
                    "<fo:flow flow-name='xsl-region-body'>" +
                    "<fo:block>Hello World!</fo:block>" +
                    "</fo:flow>" +
                    "</fo:page-sequence>" +
                    "</fo:root>", @"C:\HelloWorld.pdf");

                Console.WriteLine("Document rendered successfully!\n");
            }
            catch (Exception e)
            { // Report any errors that may occur
                Console.WriteLine(e);
            }
        }
    }
}
```



The complete source code for this example is located in the installation folder, under Code Samples/C#/HelloWorld.

The important points of this program are the following:

- Ecrion.XF is included through *using* directive, in order to use the types in the namespace without having to specify the namespace. Please note that a reference to Ecrion.XF.x.x.dll (x.x could be 1.1 or 2.0 depending on the .NET framework version that you use) must be added to the C# project before using the namespace. Use Project/Add Reference menu item in Microsoft Visual Studio .NET and look for Ecrion.XF.x.x.dll in the list of assemblies under .NET tab.
- A XFEngine object is created.
- We call *Render*, supplying an XSL-FO text and an output file name.

Remarks

When rendering in a file, the output format can be derived from the file's extension (in this case is PDF) if is not specified explicitly using *OutputFormat* property.

The method *Render* accepts a string or `System.IO.Stream` object as input. For output, use a string (denoting a file name), `System.IO.Stream` or `System.Web.HttpResponse`. There is also another method called *RenderUrl* if the document you want converted to PDF resides in a file.

Web Access

The following sample describes how to render PDF documents directly into IIS (Internet Information Server) output stream.

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
using Ecrion.XF;

namespace PDFExample
{
    public partial class Default: System.Web.UI.Page
    {
        protected void Page_Load(object sender, System.EventArgs e)
        {
            XFEngine engine = new XFEngine();

            engine.OutputFormat = OutputFormat.PDF;
            engine.Render(
                "<fo:root xmlns:fo='http://www.w3.org/1999/XSL/Format'>" +
                "<fo:layout-master-set>" +
                "<fo:simple-page-master master-name='all-pages'>" +
                "<fo:region-body region-name='xsl-region-body' margin='0.7in' />" +
                "</fo:simple-page-master>" +
                "</fo:layout-master-set>" +
                "<fo:page-sequence master-reference='all-pages'>" +
                "<fo:flow flow-name='xsl-region-body'>" +
                "<fo:block>Hello World!</fo:block>" +
                "</fo:flow>" +
                "</fo:page-sequence>" +
                "</fo:root>", Response);
        }

        #region Web Form Designer generated code
        override protected void OnInit(EventArgs e)
        {
            //
            // CODEGEN: This call is required by the ASP.NET Web Form Designer.
            //
            InitializeComponent();
            base.OnInit(e);
        }

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
        }
        #endregion
    }
}
```



The complete source code for this example is located in the installation folder, under Code Samples/ASPX/PDFExample.

The important points of this program are the following:

- A XFEngine object is created.

- We have to set the output format explicitly.
- Call *Render* passing the ASPX's intrinsic *Response* object as the second parameter.

Remarks

Other output formats are possible as well, including Postscript, HTML, PNG, JPG, etc. For the complete list, please see [XF Rendering Server Programmers Reference](#).

XSL Transformations

One typical usage scenario is when you have a stylesheet (XSL) file, and you want to use it to transform input XML into XSL-FO or XCHART and then generate PDF documents or JPG images.

In the example below, the stylesheet is hardcoded in C++ for demonstration purposes. In real life you probably have the stylesheet in a file. **XF Designer 2004** can be used to test transformation results and preview the generated PDFs (just open the XML, and hit F5).

```
using System;
using System.IO;
using System.Xml;
using System.Xml.XPath;
using System.Xml.Xsl;
using System.Text;
using System.Reflection;
using Ecrion.XF;

namespace Samples
{
    class XSLTransformations
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            try
            {
                String xmlString =
                    "<person name='Joe Doe' age='65' />";

                String xslString =
                    "<xsl:stylesheet version='1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>" +
                    " <xsl:template match='/'>" +
                    " <fo:root xmlns:fo='http://www.w3.org/1999/XSL/Format'>" +
                    " <fo:layout-master-set>" +
                    " <fo:simple-page-master master-name='all-pages'>" +
                    " <fo:region-body region-name='xsl-region-body' margin='0.7in' />" +
                    " </fo:simple-page-master>" +
                    " </fo:layout-master-set>" +
                    " <fo:page-sequence master-reference='all-pages'>" +
                    " <fo:flow flow-name='xsl-region-body'>" +
                    " <fo:block>Name: <xsl:value-of select='person/@name' /></fo:block>" +
                    " <fo:block>Age: <xsl:value-of select='person/@age' /></fo:block>" +
                    " </fo:flow>" +
                    " </fo:page-sequence>" +
                    " </fo:root>" +
                    " </xsl:template>" +
                    "</xsl:stylesheet>";

                XPathDocument xml = new XPathDocument(new StringReader(xmlString));
                XPathDocument xsl = new XPathDocument(new StringReader(xslString));

                //Create a new XslTransform object.
                XslTransform xslt = new XslTransform();

                xslt.Load(xsl, null, null);

                MemoryStream xslFo = new MemoryStream();

                //Transform the data and send the output in the memory stream.
                xslt.Transform(xml, null, new XmlTextWriter(xslFo, Encoding.Unicode), null);

                XFEngine engine = new XFEngine();

                engine.Render(xslFo, @"C:\XSLTransformations.pdf");
            }
        }
    }
}
```

```
        Console.WriteLine("Document rendered successfully!\n");
    }
    catch (Exception e)
    {
        Console.Out.WriteLine(e);
    }
}
}
```



The complete source code for this example is located in the installation folder, under Code Samples/C#/XSLTransformations.

The important points of this program are the following:

- First, we create and load two documents: the input XML and the stylesheet.
- Transform the xml data using the stylesheet and place the transformation result in a memory stream.
- Use the transformation result as input to the engine, and generate a PDF document.

In Memory Operations

In memory only operations are useful in certain special situations like storing the generated PDFs in a database BLOB, performing additional post-processing, etc.

The example below shows one way to create an in memory stream and then use it as output destination for the rendering process.

```
using System;
using System.IO;
using Ecrion.XF;

namespace Samples
{
    class InMemoryRendering
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            try
            {
                XFEngine engine = new XFEngine();
                MemoryStream stream = new MemoryStream();

                // The output format must be specified when rendering into a Stream.
                engine.OutputFormat = OutputFormat.PDF;
                // Render some XSL-FO text and place the output into our stream
                engine.Render(
                    "<fo:root xmlns:fo='http://www.w3.org/1999/XSL/Format'>" +
                    "<fo:layout-master-set>" +
                    "<fo:simple-page-master master-name='all-pages'>" +
                    "<fo:region-body region-name='xsl-region-body' margin='0.7in' />" +
                    "</fo:simple-page-master>" +
                    "</fo:layout-master-set>" +
                    "<fo:page-sequence master-reference='all-pages'>" +
                    "<fo:flow flow-name='xsl-region-body'>" +
                    "<fo:block>Hello World!</fo:block>" +
                    "</fo:flow>" +
                    "</fo:page-sequence>" +
                    "</fo:root>", stream);
                // Do something usefull with the result. In this example will just save it
                // to a file.
                stream.WriteTo(new FileStream(@"C:\InMemoryRendering.pdf", FileMode.Truncate));

                Console.WriteLine("Document rendered successfully!\n");
            }
            catch (Exception e)
            { // Report any errors that may occur
                Console.WriteLine(e);
            }
        }
    }
}
```



The complete source code for this example is located in the installation folder, under Code Samples/C#/InMemoryRendering.

The important points of this program are the following:

- We create a memory stream; the memory will be allocated incrementally, as more data is added to this stream.
- Output format must be set before calling *Render*.
- Call *Render* passing the stream as the second parameter.

Remarks

Be aware that using this feature for large documents (>1MB) can result in poor performance if the sever doesn't have enough memory - the operating system will swap memory pages to disk in order to accommodate your memory requests.

Selective Rendering and Thumbnails

XF Rendering Server 2007 also offers support for selective rendering of document's pages, as well as resizing the output. You can use these features to generate thumbnails, as shown below:

```
using System;
using Ecrion.XF;

namespace Thumbnail
{
    class Thumbnail
    {
        [STAThread]
        static void Main(string[] args)
        {
            XFDocument doc = null;
            doc = new XFDocument();
            doc.LoadUrl("http://www.ecrion.com/XF/Examples/XSLEExampleMulticolumn.xml");

            if (doc.GetPageCount() > 0)
            {
                double width = doc.GetPageWidth(0);
                double height = doc.GetPageHeight(0);
                double zoom;
                double thumbSize = 256;

                if (thumbSize/width < thumbSize/height)
                    zoom = thumbSize/width*100.0;
                else
                    zoom = thumbSize/height*100.0;
                doc.OutputFormat=OutputFormat.BITMAP;
                doc.SetProperty("Compression", 100);
                doc.SetProperty("Zoom", zoom);
                doc.SetProperty("Pages", "0");
                doc.ExportTo("c:\\TestPage0.bmp");
                doc.SetProperty("Pages", "1");
                doc.ExportTo("c:\\TestPage1.bmp");
            }
        }
    }
}
```



The complete source code for this example is located in the installation folder, under Code Samples/C#/Thumbnail.

The important points of this program are the following:

- Document is loaded with LoadUrl (or Load). The url in the code exists but can be any url.
- Once the document is loaded, you can find the number of pages, as well as the dimensions of each individual page.
- Use ExportTo to generate the output. You can call this method multiple times, for example to generate a thumbnail for each page, or to generate the document and a thumbnail of the first page, etc.

Using XF Merge Contexts

XF can render multiple input documents into one output file or stream.

This feature can very useful for printing jobs which require all inputs to be sent to a printer into one large file Postscript file. It can also be used for archiving purposes.

The following code fragment exemplifies this feature:

```
using Ecrion.XF;
static void mergeOutputs(string[] inputFiles, string outputFile,
                        OutputFormat fmt)
{
    using(XFMergeContext ctx = new XFMergeContext())
    {
        ctx.OutputFormat = fmt;
        ctx.SetOutput(outputFile);

        foreach(string inFile in inputFiles)
        {
            XFDocument doc = new XFDocument()
            doc.LoadUrl(inFile);
            doc.ExportTo(ctx);
        }
    }
}
```

The important points of this program are the following:

- A merge context object is created. The output destination, as well as the output format are set.

Please note that only PDF and POSTSCRIPT are valid output formats. More output formats may be added in the future.

- Each input is loaded into new instances of XFDocument, and then "exported" into the merge context.
- When the merge context is disposed, the renderer will perform closing tasks on the output. For PDF output this is equivalent with writing the trailer.

Remarks

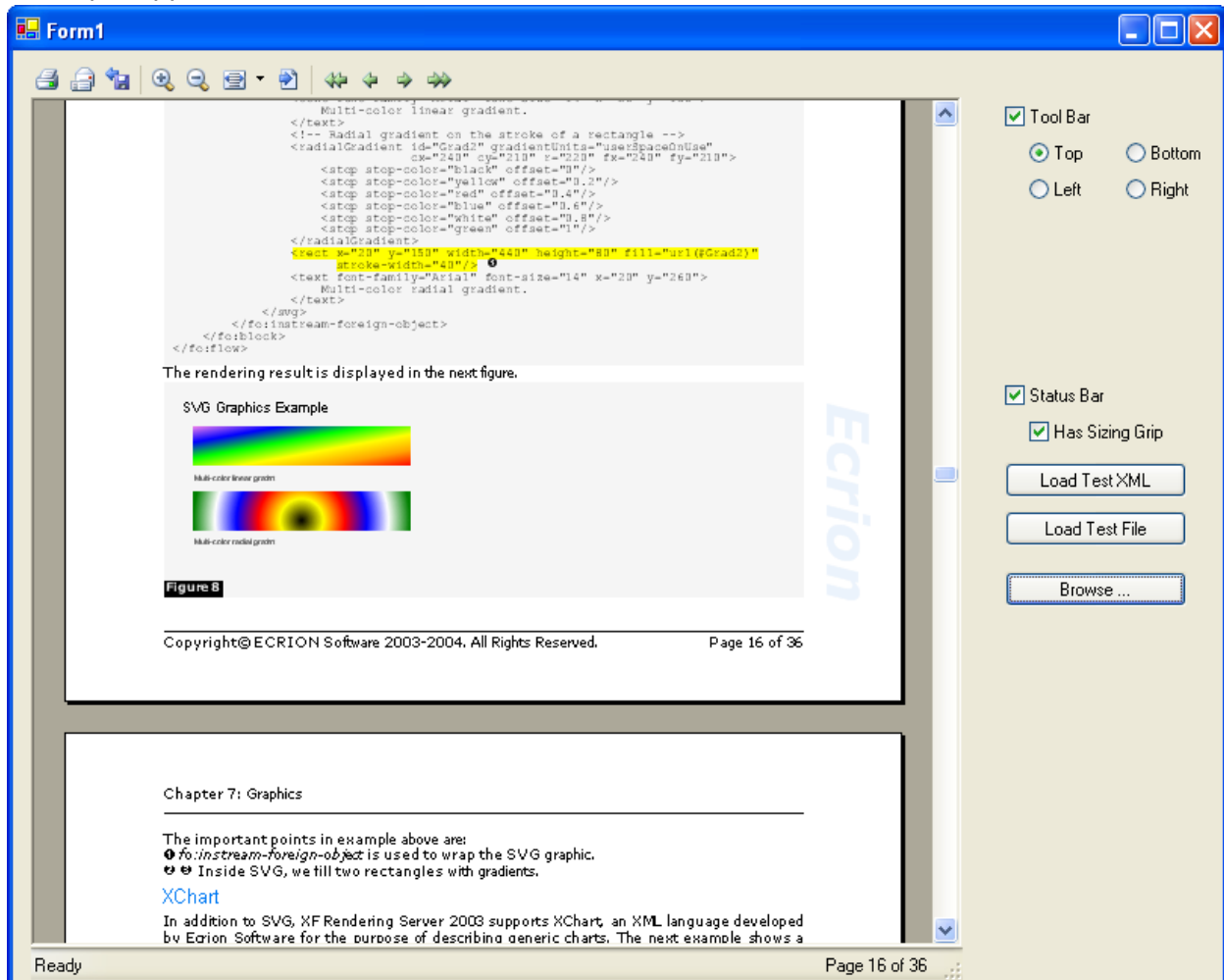
The example above describes a generic way to merge multiple inputs into one output. You can also use a System.IO.Stream object as parameter to SetOutput method.

Please note that the merge context must be closed gracefully in order to obtain a valid output file.

In the example above, the close operation is automatically performed by XFMergeContext's IDisposable implementation: 'using' statement defines the scope at the end of which the merge context is disposed.

XSL-FO Viewer .NET Control

Ecricion XSL-FO Viewer .NET Control is a component that allows displaying of XSL-FO documents in a .NET windows application. It includes support for exporting the document in PDF format, sending the PDF document attached in an email, printing, etc. A sample application is included in the distribution:



The complete source code for this example is located in the installation folder, under Samples/C#/XFViewerControlTest.

Creating a test application

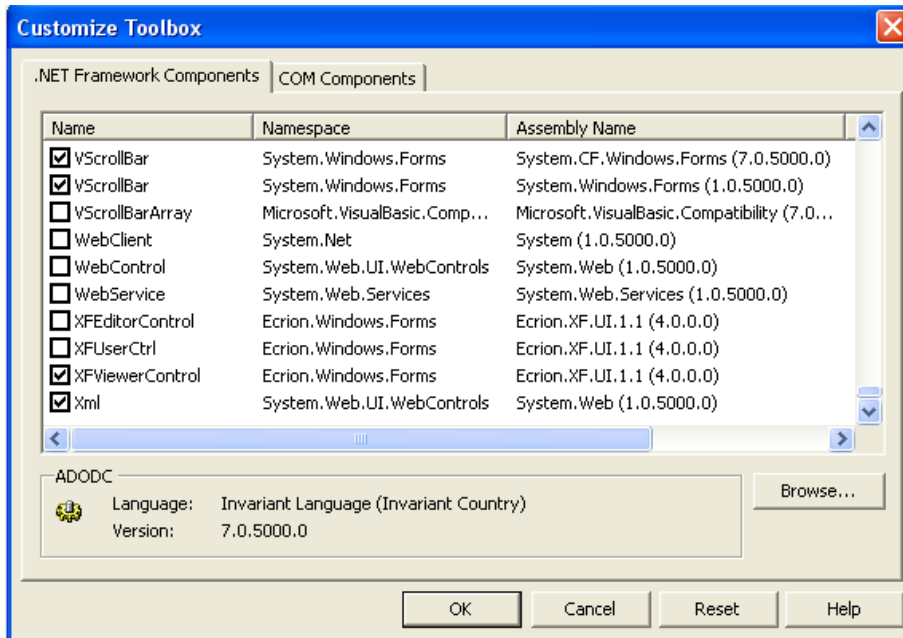
Create a new Windows Application

- Open Visual Studio .NET
- From file menu choose New/New Project.
- Select *Visual C# Projects* in Project Types list.
- Select *Windows Application* in Templates list.
- Click Ok. A new Windows application is created.

Add Ecricion.XF.UI.x.x.dll (x.x could be 1.1 or 2.0 depending on the .NET framework version that you use) in .NET Toolbox

- Open Form1.cs in the new created Windows Application in Design mode.

- Open the toolbox window if not already visible by choosing View/Toolbox.
- Right click *Windows Forms* group heading and choose *Add/Remove Items...* from the menu. *Customize Toolbox* window will appear.
- Click *.NET Framework Components* tab and find *XFViewerControl*. Check the checkbox and choose Ok.



- *XFViewerControl* will be added in *Windows Forms* list in the toolbox.

Now you can drag *XFViewerControl* in *Form1* form, set docking properties, etc. Use *LoadXML* or *LoadUrl* to load and display a XSL-FO document. For complete members list see [XF Rendering Server 2007 Programmers Reference](#).

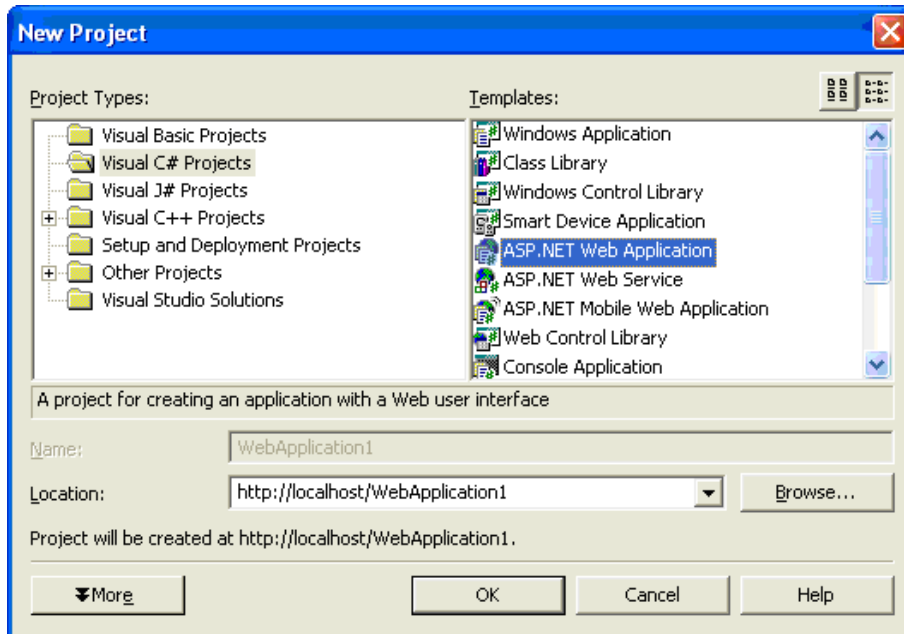
Sample Web Application Step-By-Step

Prerequisites

- Install XF Rendering Server 2007.
- Start XF Windows Service from Administrative Tools/Services if not already started.

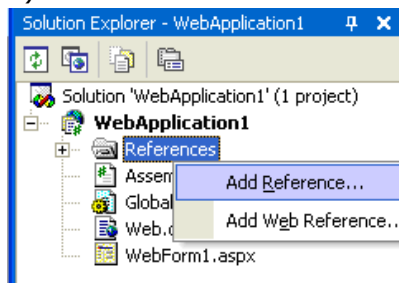
Tutorial Steps

- 1) Start Microsoft Visual Studio .NET
- 2) From File menu choose New Project to create a new Web Application

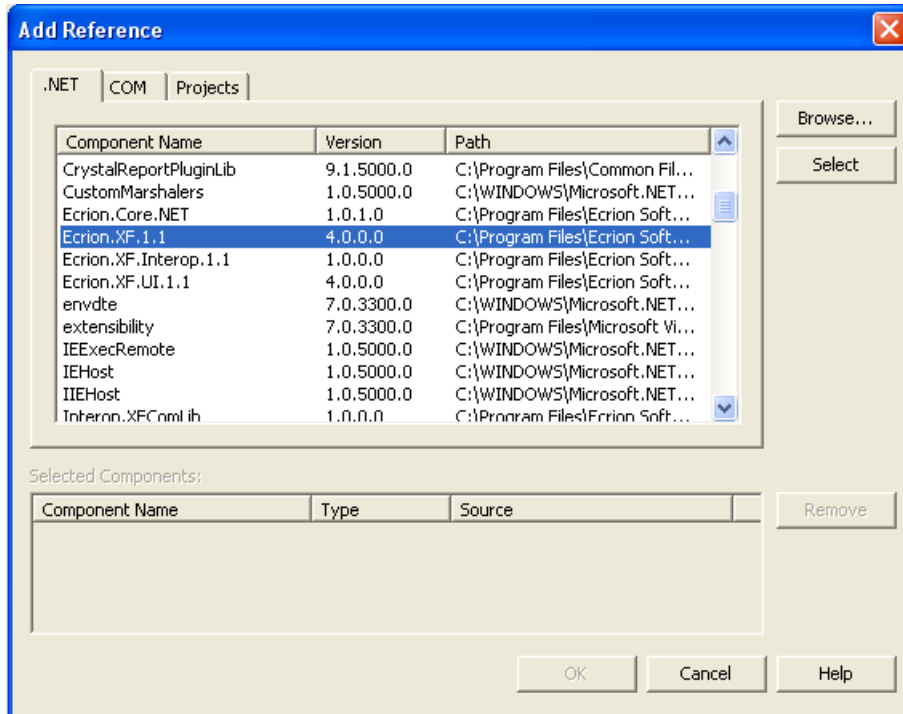


Click Ok. Web Application Project will be generated.

- 3) Add a reference to XF.NET library.

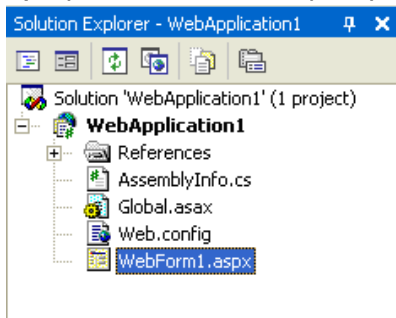


Add reference Dialog will appear. Find XF.NET in the list and select it. Click "Select" button, then click "OK".



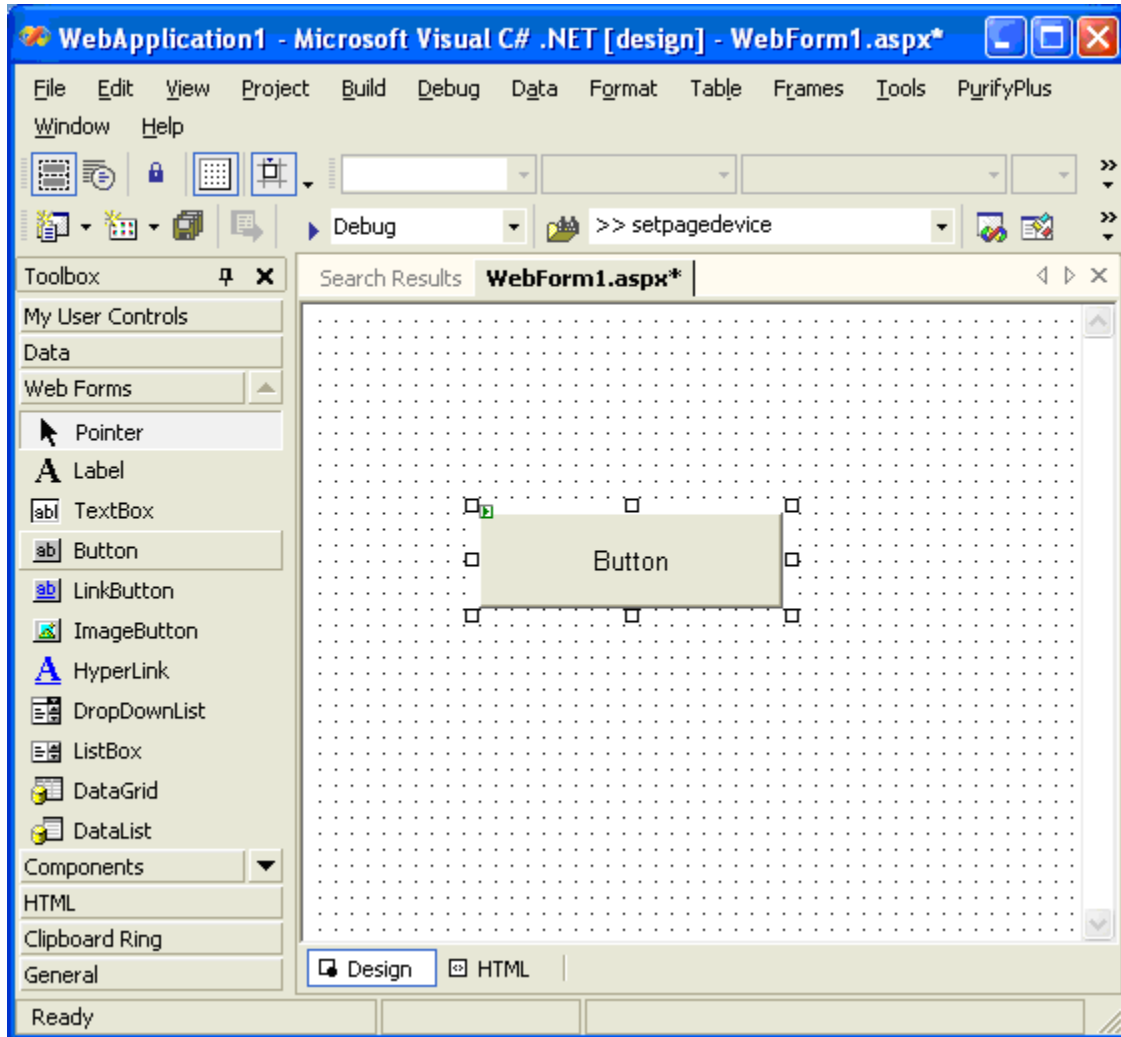
If you can't find XF.NET in this list, then XF Rendering Services for .NET are not installed. You have to run XFNetSetup.msi before continuing.

4) Open WebForm1.aspx by doubleclicking the file name in the Project View.



5) Activate the Toolbox (if not already visible) by choosing View/Toolbox.

6) Drag and drop a Button Web Forms control from Toolbox onto WebForm1.aspx



7) Double click the new created button to enter in text edit mode.

8) Replace the click handler Button1_Click with the following text.

```
protected void Button1_Click(object sender, EventArgs e)
{
    Ecrion.XF.XFEngine engine = new Ecrion.XF.XFEngine();

    engine.OutputFormat = Ecrion.XF.OutputFormat.PDF;
    engine.Render(
        "<fo:root xmlns:fo='http://www.w3.org/1999/XSL/Format'>" +
        "<fo:layout-master-set>" +
        "<fo:simple-page-master master-name='all-pages'>" +
        "<fo:region-body region-name='xsl-region-body' margin='0.7in'/>" +
        "</fo:simple-page-master>" +
        "</fo:layout-master-set>" +
        "<fo:page-sequence master-reference='all-pages'>" +
        "<fo:flow flow-name='xsl-region-body'>" +
        "<fo:block>Hello World!</fo:block>" +
        "</fo:flow>" +
        "</fo:page-sequence>" +
        "</fo:root>", Response);
}
}
```

9) Build the solution (Build/Build Solution).

10) Run the application (Debug/Start). When clicking the button, a PDF file should be generated and displayed in the browser.

Additional information

1) On Step 9, the code on Button1_Click can perform a server side redirect to another page, which in turn will execute the rendering in Page_Load method.

2) Instead of rendering directly in the Response stream which is more memory consuming, you can render into a file that resides in a folder mapped for Web access. After rendering, perform a server side redirect to the generated PDF file. The files will have to be cleaned up on a regular basis.

3) In the example above the XSL-FO document is supplied in a string. Look in C:\Program Files\Ecrion Software\XF Rendering Server 2007\Code Samples\C#\XSLTransformations\XSLTransformations.cs for an example of transforming XML with a stylesheet to produce XSL-FO, and then feed this input into XF Rendering Server.