Integrating RenderX XSL FO Technology with iText for High Performance Dynamic Forms Generation

This paper presents a complete, generic framework for creating modifiable, flowing PDF forms, based on XML content and standard XSL templates. It requires no programming to generate dynamic, fill-able, custom PDF forms by mapping some RenderX XSL FO extensions to form elements.

Recently RenderX has seen many opportunities arise for creating "dynamic" forms. We mean dynamic because both the shape and the content of the form may depend on the actual content. This paper presents a method used for generating dynamic forms using RenderX's XEP, an XSL FO formatting engine, combined with the iText library for PDF manipulation to produce dynamic, fill-able, custom forms in a high-volume, multi-threaded application.

Potential Applications

There are 1000s of potential applications that would benefit from such a solution. One real-life RenderX customer is creating over 100,000 monthly statements in PDF to be e-mailed to their customers. What is unique is that each statement is in reality a custom PDF Acroform for credit card submittal. The fields of the form must flow to any location in the document to be at the end of the statement and they must carry specific information in hidden fields. This includes such things as the account number, the amount, and the recipient's name and address. The RenderX customers' monthly statement run creates these dynamic, custom forms for presentation to the recipient through their on-line portal – not only as a "green" initiative but also providing a very convenient (and in many cases a more expedient) method for obtaining payment. They already were producing statements using XSL FO, the methods presented here allow them to dynamically generate unique custom forms for every recipient.

Another example customer is creating a dynamic insurance application form that is 100% custom to the applicant. The form is generated using information from a few upfront questions and known information from the customers' account. Whole sections and/or specific questions are included depending on the known information about the particular user (like previous insurance products, state of residence, etc.). Other fields are filled with known information and even hidden to prevent their modification. Based on a few simple questions asked at the web site, a dynamic form is generated on-demand that is custom for the applicant. The overall application process is streamlined, forgoing the many different applications for different types of users for one process to generate a custom form. It also eliminates mistakes and saves a lot of time for known prospects by pre-filling in information as well as hiding fields that should not be changed.

The Benefits of XSL FO for the Document and iText for the Form

XSL FO, a W3C standard for representing print information in a standard XML language, is a technology that is well suited to creating flowing text documents through no use of programming. However, XSL FO contains no structures for supporting input of information such as a form field. The iText library is an ideal solution for enhancing and modifying existing PDFs and in this case it is used for its ability to stamp form fields onto existing PDF documents given a known location and page.

Thus, one can use the ease of XML and XSL technology to compose PDF documents, while leveraging information from the RenderX XEP software during the composition phase to record the physical locations on the page for the form fields. RenderX is used to generate a "blank" PDF, one that would be a background as if the form fields where in the document. During the process, RenderX can record information about each of these fields – names, exact locations, pages, and other properties. A post-process of the resulting "blank" PDF and the form field information (obtained from the formatting process) using iText completes the application, creating a fully "dynamic" form with fields perfectly located within the document. The sample application created to demonstrate this capability has two classes. One creates the "background" form using RenderX software and records the information needed. The second class stamps the fields using iText. A standard set of templates in XSL completes the application. They are used to transform an XML element representing a form field. For the production application, the two classes were wrapped into a multi-threaded framework. The results were very high-speed production of totally custom form documents from source XML content and XSL instructions. ***The most important benefit of this solution is that it creates a complete, generic framework with modifiable form field elements, based on XML content and standard XSL templates. No further programming is necessary to generate dynamic, fill-able, custom PDF forms.***

How it Works – The Details

One particular customer requested a final implementation on Windows with .NET interfaces, so RenderX selected the XEPWin solution. XEPWin is RenderX's Java-based XSL FO formatting engine (XEP) that has a .NET wrapper to expose only .NET interfaces to the developer. The iText C# port (itextsharp) was selected for the stamping of the form fields.

RenderX XEPWin is an application that takes in XML as the source data and XSL as the rules for how that XML data will flow and appear on a page. XSL can contain logic to make decisions and create structures in the XSL FO that represent the page layout and all other visible appearances. The RenderX XSL FO core technology handles the flow of the content into the layout, making all the decisions about character placement, spacing, page endings, etc. based on the rules contained within the XSL FO Specification itself as well as the parameters in the XSL FO file.

RenderX technology can produce PDF, Postscript and AFP output. RenderX software composes the XSL FO document to an internal XML representation of the page which is normally not exposed to the end user but has always been available through programming. This XML format is called the XEP Intermediate Format (XEP format). Normally, this internal format is streamed directly to a backend program that converts the XEP format to the desired output, like PDF in this case. However, the XEP format can be obtained through the API and programmatically examined and even manipulated. RenderX has presented other papers discussing manipulation of this formation for things like inserting OMR marks, generating custom barcodes and Transpromo advertising.

The XEP format is well documented on the RenderX support web site. One can easily see the page structure in an XML format that is simple to understand. The XEP format contains such elements as <document>, <page>, <text>, <rectangle>, <line>, <font>, <cmyk-color>, etc. It also contains instructions like <rotate>, <clip> and <translate>. The key here is that RenderX produces an easily interpreted XML structure of entire documents and this XML file can be analyzed programmatically.

RenderX also supports an extension element to the XSL FO standard within it's own "rx:" namespace. This element is known as a <pinpoint>. By placing a <pinpoint> element in the source, a resulting <pinpoint> will appear in the XEP format at the exact page location where it would be formatted. In the XEP format, the <pinpoint> element is marked with attributes of the exact X,Y page coordinates where the "pin" is dropped. It can also contain a single label as an attribute to be used to identify the pin.

The XML and XSL Representations

The following figures show sample XML content with <formfield> elements, a sample XSL set of templates that can be included in any XSL file and a brief look at the resulting XEP format results.

```
        <test>
            <desc>Simple Textbox with Default Value</desc>
            <content><formfield name="field06" type="textbox" readonly="false" default="Change me"
font-family="Helvetica" font-size="10pt" font-weight="normal"/></content>
        </test>
        <test>
            <desc>Password Field</desc>
            <content><formfield name="field07" type="password"/></content>
        </test>
        <test>
            <desc>Option List</desc>
            <content><formfield name="field08" type="combobox"
options="option1;option2;option3;option4;option5;option6"  font-family="Helvetica" font-size="10pt"
font-weight="normal"/></content>
        </test>
        <test>
            <desc>Option List with Default Value</desc>
            <content><formfield name="field09" type="combobox" default="option5"
options="option1;option2;option3;option4;option5;option6"  font-family="Helvetica" font-size="8pt"
font-weight="normal" font-color="green"/></content>
        </test>
        <test>
            <desc>Select List (10 items, showing 5 lines, default to option7, bold and red font</desc>
            <content><formfield name="field28" type="selectlist" numlines="5"
options="option1;option2;option3;option4;option5;option6;option7;option8;option9;option10"  default="option7"
font-family="Helvetica" font-size="10pt" font-weight="bold" font-color="red"/></content>
        </test>
        <test>
            <desc>Checkbox Using "check"</desc>
          <content><formfield name="field10" type="checkbox" checktype="check" default="true"/></content>
        </test>
```

Figure 1: A Sample XML Fragment Showing Form Fields

```
<xsl:template match="formfield">
    <xsl:call-template name="process.formfield"/>
</xsl:template>
    <xsl:template name="process.formfield">
        <!-- Start the formfield -->
        <rx:pinpoint>
            <xsl:attribute name="value">start,formfield</xsl:attribute>
        </rx:pinpoint>
        <!-- Output each attribute as is -->
        <xsl:for-each select="attribute::node()">
            <rx:pinpoint>
                <xsl:attribute name="value"><xsl:value-of select="name(self::node())"
                        />,<xsl:value-of select="."/></xsl:attribute>
            </rx:pinpoint>
        </xsl:for-each>
        <xsl:choose>
            <xsl:when test="attribute::type='hidden'">
            </xsl:when>
            <xsl:otherwise>
                <fo:block-container background-color="white">
                    <xsl:choose>
                        <xsl:when test="attribute::type= 'textarea'">
                            <xsl:call-template name="form.line">
                                <xsl:with-param name="count">
                                    <xsl:value-of select="attribute::numlines"/>
                                </xsl:with-param>
                            </xsl:call-template>
                        </xsl:when>
                        <xsl:when test="attribute::type= 'selectlist'">
                            <xsl:call-template name="form.line">
                                <xsl:with-param name="count">
                                    <xsl:value-of select="attribute::numlines"/>
                                </xsl:with-param>
                            </xsl:call-template>
                        </xsl:when>
                        <xsl:otherwise>
                            <fo:block>
                                <fo:leader leader-pattern="space"/>
                            </fo:block>
                        </xsl:otherwise>
                    </xsl:choose>
                </fo:block-container>
            </xsl:otherwise>
        </xsl:choose>
        <rx:pinpoint>
            <xsl:attribute name="value">end,formfield</xsl:attribute>
        </rx:pinpoint>
    </xsl:template>
    <xsl:template name="form.line">
        <xsl:param name="count"/>
        <xsl:param name="iteration">1</xsl:param>
        <fo:block>
            <fo:leader leader-pattern="space"/>
```

```
        </fo:block>
        <xsl:if test="$iteration &lt; $count">
            <xsl:call-template name="form.line">
                <xsl:with-param name="count" select="$count"/>
                <xsl:with-param name="iteration" select="$iteration +1"/>
            </xsl:call-template>
        </xsl:if>
    </xsl:template>
```

Figure 2: Sample XSL Template to Convert into XSL FO with <pinpoint> Elements

The <formfield> Element

Essentially these two things – the <pinpoint> element and the easily interpreted XML format - are all that is needed to process the files. Any customer can now introduce a new XML element in the source document to describe a form field. This empty element is <formfield> and it carries various attributes that describe the actual form field type, content and appearance. Of course, many field properties are optional and the code itself has most items defaulted to standard PDF form conventions. However, for very fine control most of the appearance features of the field can be modified. The following table shows the various attributes supported for the sample application.

Table 1: The <formfield> Element Attributes

name	The name of the field in the output (must be unique across all form fields and is required)
type	The type of field, one of (textbox|password|combobox|selectlist|checkbox|textarea|radio|submit|reset|hidden)
numlines	the number of visible lines in a textarea, selectlist
options	a semi-colon separated list of options for a combobox or selectlist
default	the default value to the field
font-family	the font-family to use for the field
font-size	the font size to use for the field
font-weight	normal|bold
font-style	normal|italic
font-color	the color of the font represented as a common name like “SlateBlue” or through hex like “#F0F0F0”
borderstyle	one of (solid|beveled|dashed|inset|underline)
bordercolor	the color of the field border represented as a common name like “SlateBlue” or through hex like “#F0F0F0”
border	the thickness of the border
readonly	true|false to create a readonly field
radiogroup	A name to allow for grouping radio buttons into a collection. All radios with the same group (and on the same page) are grouped into a set of radios.
onstate	the checkbox onstate for a radio group
checktype	one of (diamond|check|circle|square|star|cross) for checkboxes or radio buttons
href	the link for a submit button
background-color	the color of the background of the submit or reset button represented as a common name like “SlateBlue” or through hex like “#F0F0F0”

After an XML file is processed through an XSL file with the addition of the templates for processing the <formfield> elements, the resulting XEP file contains all the information necessary to construct a form field using an automated process.

```
<xep:pinpoint x="377700" y="635100" value="start,formfield"/>
<xep:pinpoint x="377700" y="635100" value="name,field04"/>
<xep:pinpoint x="377700" y="635100" value="type,textbox"/>
<xep:pinpoint x="377700" y="635100" value="readonly,false"/>
<xep:pinpoint x="377700" y="635100" value="font-family,Helvetica"/>
<xep:pinpoint x="377700" y="635100" value="font-size,12pt"/>
<xep:pinpoint x="377700" y="635100" value="font-weight,bold"/>
<xep:pinpoint x="377700" y="635100" value="font-style,italic"/>
<xep:pinpoint x="377700" y="635100" value="font-color,red"/>
<xep:gray-color gray="1.0"/>
<xep:rectangle x-from="377700" y-from="620700" x-till="538500" y-till="635100"/>
<xep:pinpoint x="377700" y="620700" value="end,formfield"/>
```

Figure 3: A Sample of the XEP Intermediate Format to Used to Derive a Form Field

The original XML+XSL elements are processed in a program to the XEP file, represented by a MemoryStream. This stream is loaded to an XML Document and analyzed programmatically to find the <pinpoint> elements, extract the field attributes and the exact X,Y locations of the fields. The field size itself is obtained from a table cell whose background color is white. This table cell area results in a <rectangle> element in the XEP file and this <rectangle> element gives us the exact llx,lly,urx and ury locations of the rectangle to insert for the form field using iText (after unit and coordinate system conversion).

This XEP file MemoryStream is processed to PDF using RenderX software, utilizing one class to obtain the document background as a PDF. The MemoryStream and the PDF are then passed to another class used to analyze the XEP intermediate file and stamp the form fields using iText. These two classes now represent a complete solution for insertion of form fields into XSL FO data to be dynamically stamped into the output stream.

For the customer application and for high-volume testing, these two classes were wrapped in a multi-thread harness that can take a list of XML files and an XSL file and process the whole list in a configurable number of threads. The whole solution is very fast and can easily run 4, 8, 10 or more simultaneous rendering threads on a tested configuration of a dual-core laptop computer. One example document was developed that contains a mega-test of all different types of form fields. There are 28 fields on this document and results for the tested dual-core laptop show 60 forms of one page (each with 28 fields) generated in 4 threads in 9.2 documents/second. A single CPU dual-core license for a server-class machine should be sufficient to handle 20-30 simultaneous requests for a similar form and be able to serve that form is less than 1.5 seconds to the end-user in an on-demand situation.

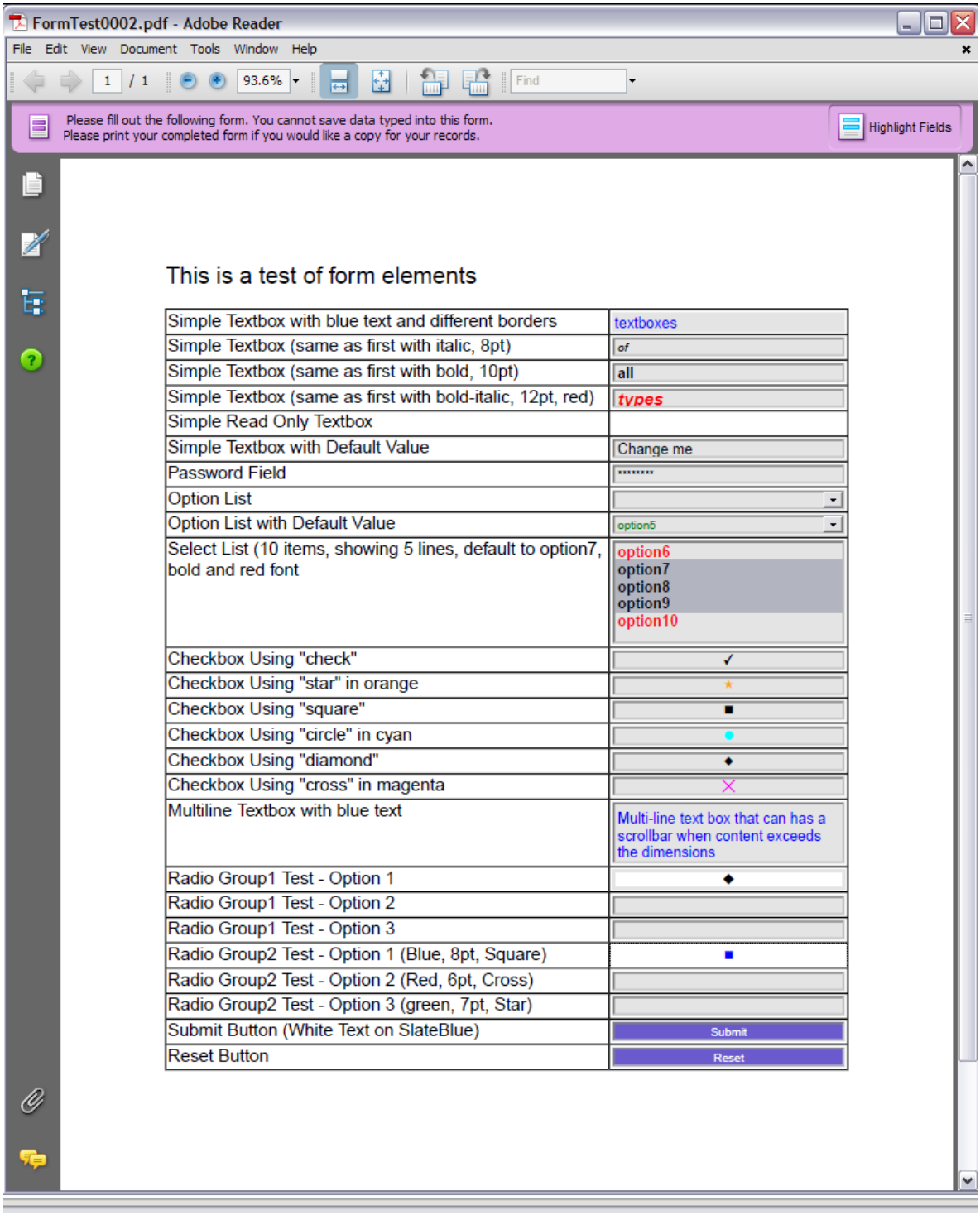

Figure 4: Mega-Test Sample PDF Form Generated

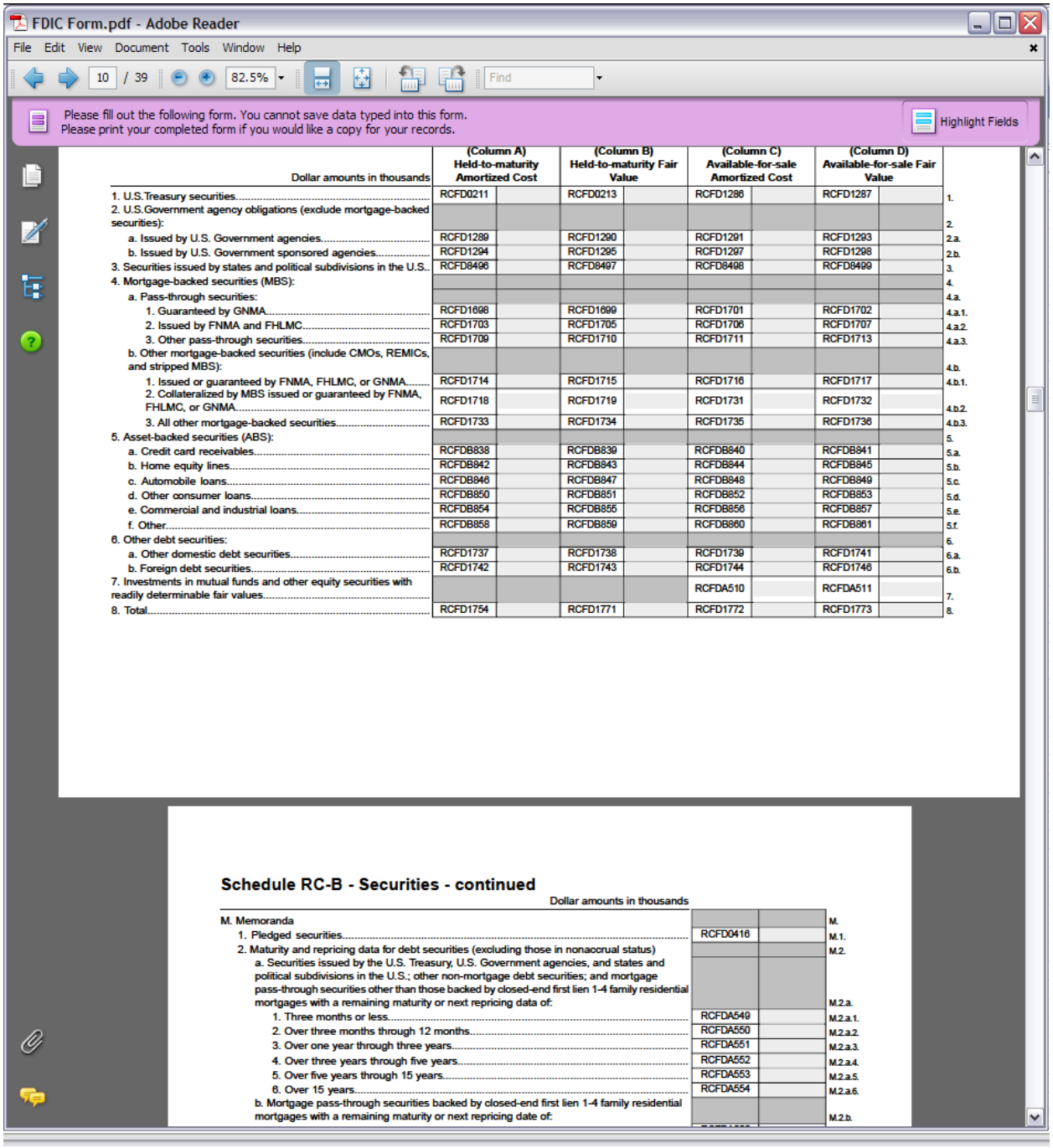

Figure 5: Sample Application (30 Page PDF with 1872 form fields in multiple layouts)

Conclusions

Marrying RenderX XEP and XSL FO technology with iText has resulted in a 100% generic way to generate dynamic PDF forms for everyday applications. Users only need to include a new style sheets template into their existing XSLs and add a new element into their XML where they wish form fields to be placed. Using code like the two classes developed, the <formfield> element is automatically processed, leveraging RenderX XEP and iText to create dynamic, fillable, custom forms in a muti-threaded, high volume application.

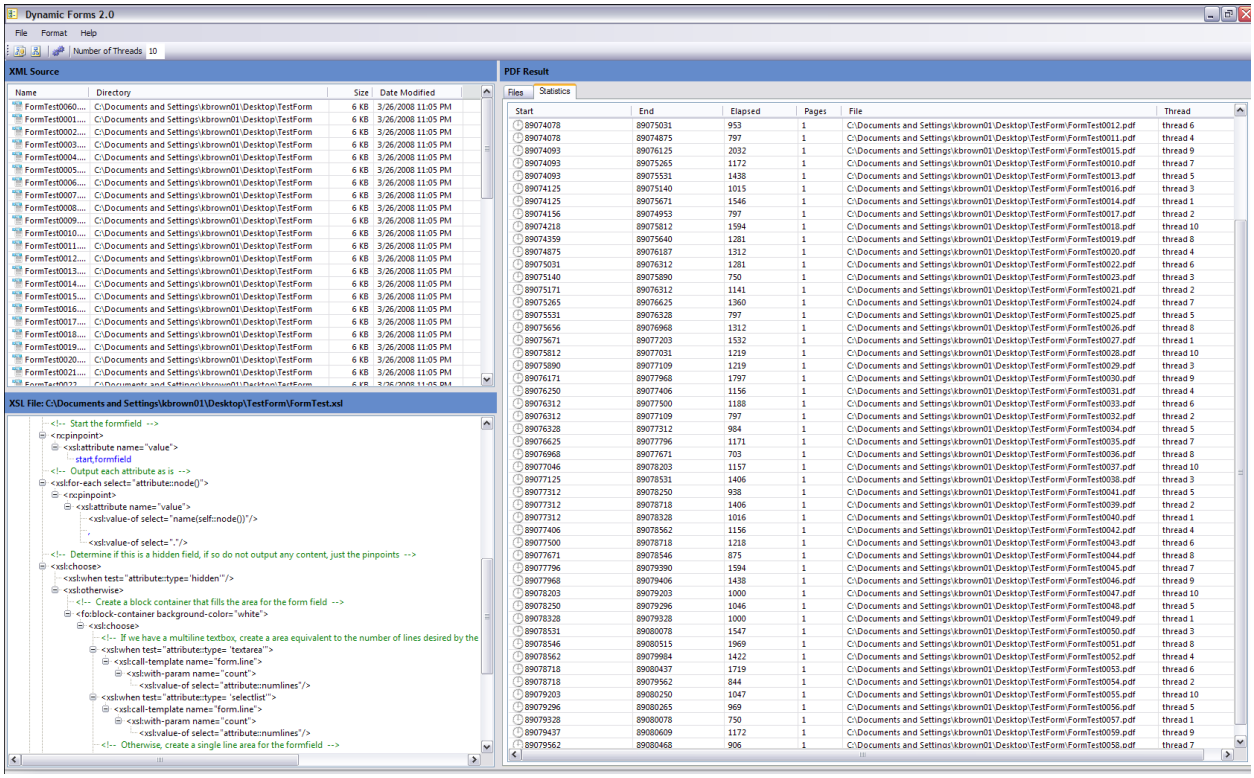

Figure 6: The Dynamic Forms Sample Application

Acknowledgements

This solution uses the C# port (itextsharp) of the iText Library for generating form fields in the document. The iText Library is Copyright (C) 1999-2008 by Bruno Lowagie and Paulo Soares. All rights reserved.

This solution uses RenderX XEPWin for dynamic formatting of XSL FO content. XEPWin is a product of RenderX, Copyright (c) 2004 - 2008 by RenderX, Inc. All rights reserved.

Sample code of the example application is available on request. For more information about this application, contact:

Kevin Brown
RenderX, Inc.
kevin@renderx.com