Using RenderX XSL FO Technology for Large Report Formatting

This paper presents concepts that can be applied to create a solution for very large report formatting. The concepts can be applied in parallel such that one can realize the creation of huge output documents with 100,000+ pages at very high performance.

Many RenderX customers are using our technology for formatting very large documents with input files that are multiple gigabytes of XML data. As RenderX XEP is an in-memory formatter and depending how things are integrated, it is possible that the input, internal representation of the document and other assets must be in memory. Using various techniques in the way RenderX XEP operates, it is possible to use our technology to *achieve huge document production* while *simultaneously optimizing performance*.

Potential Applications

There are specific applications where the techniques described here should be employed and there are also applications where they cannot be employed. The use of such techniques is governed by the nature of the document. These techniques should not be used for producing a huge manual or textbook where such things as PDF bookmarks, back of book indexes and intra-document cross-references are used. Such constructs currently require the whole document in memory to keep track of links. These techniques are suited for producing huge print files for such applications as a monthly statement run where the input is essentially something like 50,000 two page invoices ... all in one XML file or in separate XML files. Typically, in such document collections, one does not even have page numbering that continues from document to document. And of course these types of document collections don't have a Table of Contents or back indexes.

How it Works – The Details

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

The XEPOUT format is well documented on the RenderX support web site – XEP Reference/User Guide, Appendix E. The document structure serializes to an XML format that is really simple to understand. The XEPOUT 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 XEP produces an easily interpreted XML structure of entire documents and this XML file can be manipulated programmatically.

If one were to examine sample output from RenderX XEP in the XEPOUT format representation, it would look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<xep:document xmlns:xep="http://www.renderx.com/XEP/xep" producer="XEP 4.14 build
20081212" creator="RenderX" author="Kevin Brown" title="Sample Document">
	<xep:page width="419528" height="297638" page-number="1" page-id="1">
		<xep:word-spacing value="0"/>
		<xep:letter-spacing value="0"/>
		<xep:font-stretch value="1.0"/>
		<xep:font family="Helvetica" weight="400" style="normal" variant="normal"
size="10000"/>
		<xep:rgb-color red="0.0" green="0.0" blue="0.0"/>
		<xep:text value="A" x="42520" y="245563" width="12630"/>
	</xep:page>
</xep:document>
```

Figure 1: A Sample XEPOUT Document

The XEPOUT sample in Figure 1 is the result of a one page document (obvious as it has one <page> element). The highest level structure ... an <document> contains one or more <page> elements. Each <page> represents a page in the output document and is a self-contained unit for the most part. There are some higher -level constructs that apply to the whole document but for these purposes we can consider a simplified model. All of a page's content and appearance is represented within the <page> element.

It is also key to understand the overall process inside the RenderX XEP formatting engine. Getting from XML and XSL to final output form is a multi-step process. We should all know the first step, XML+XSL -> XSL FO ... or *Transformation*. This is an optional step and may take place separately or in some cases not at all as XSL FO can certainly be generated programmatically and not through transformation.

The second step is called *Formatting*. In the formatting step, XSL FO is converted to the internal representation. Serializing this internal representation to disk and one will have the XEPOUT format. This step that takes place all in a single thread and is the limited step in terms of memory. The more pages, the more memory required to hold the internal representation.

The final step is called *Generation*. The internal representation is converted to the final format required. This step takes little to no memory and is extremely fast. It can be as high as 300-500 pages per second or more.

All of these steps can be implemented separately. In previous white papers we have shown that a user could implement a solution to serialize XEPOUT format to disk, manipulate it and then send it back to RenderX XEP for *Generation* only. And there is nothing that says that this "manipulation" couldn't be actually done by just appending additional <page> elements into the serialized output of a document.

Typically, in such applications, the XSL style sheet is designed in such a way that it just iterates over some element (normally the second level tag) and creates a page sequence for each repeating document. For example, you might have something like:

```
<batch>
	<invoice>...</invoice>
	<invoice>...</invoice>
	<invoice>...</invoice>
	...
</batch>
```

The XSL has an <xsl:for-each> whose match is <invoice> and it generates a page sequence for each one. This means that one can simply split the input file into multiple input files at this element. So, given this and the description above, one *conceptually* could:

a) Write code to split the incoming file *automatically* at some user defined level, like every 10 invoices

b) Send each of this split files through XSLT to XSL FO and then to RenderX XEP requesting XEPOUT format and not the final output format

c) Write some code to merge all XEPOUT format files, simply by copying <xep:page> elements from one document to the end of the previous document

d) Once the very large XEPOUT format file is created, send it back through RenderX XEP to *generate* the final PDF, Postscript, XPS and AFP output format

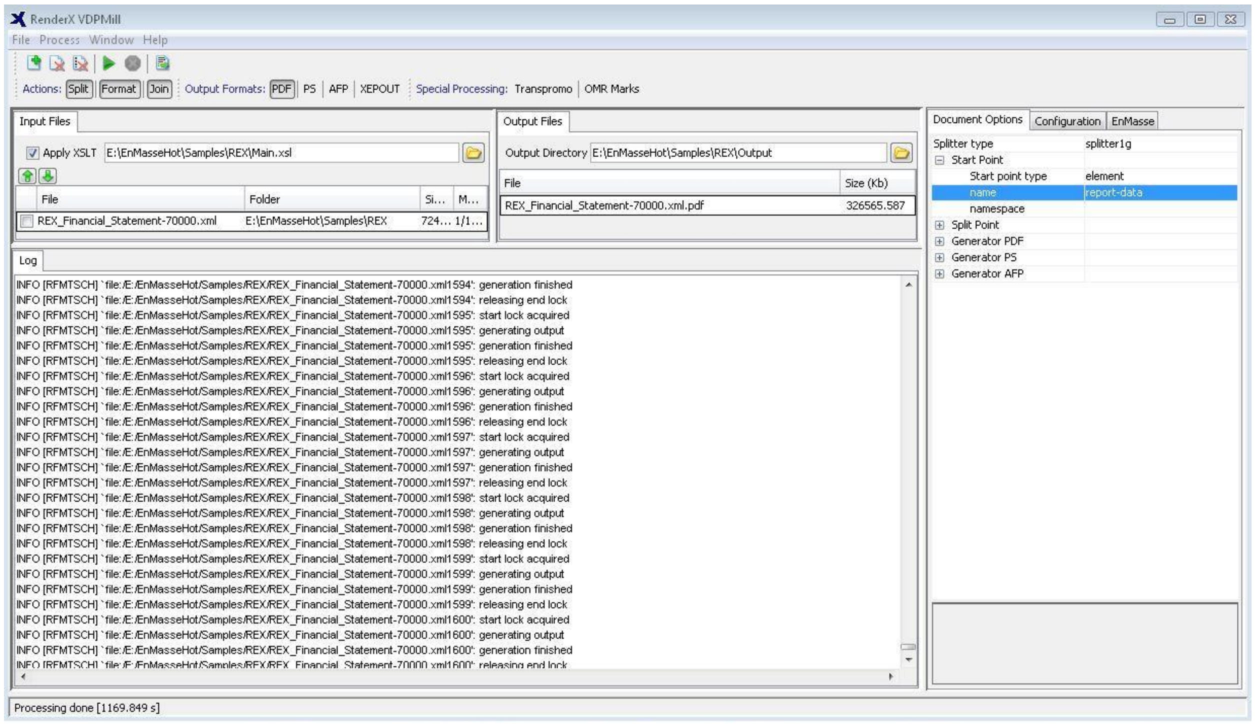
I say *conceptually* because there are even better ways to do this. It is possible to run RenderX XEP in a way that you send individual internal representations to it and tell XEP to put them together (i.e.: it does the merge for you). Or better yet, since splitting is extremely fast and generation is extremely fast, one can make the formatting process *parallel*. A solution can spread the memory and CPU intensive part of the process across multiple threads, multiple JVMs and even multiple machines. As long as you wrap the application with some processing control that splits, calls a multi-threaded grid with individual splits, and puts them back through RenderX XEP generation in order ... you have a complete huge document formatter that can meet any realistic performance and document size requirements.

VDPMill, the Document Factory

The good news is that one does not really have to build such an application. This is because RenderX has already done all of this for you and even more. VDPMill from RenderX, our newest product, is built to accomplish these tasks.

It has a *Splitter* which provides for both a 1-D (simple, one dimensional split at a certain element) and a 2-D (complex, two-dimensional split where parent data is appended onto individual splits). The 2-D *Splitter* can do such things as intelligently split a huge document that consists of one giant table. The *Splitter* is completely configurable allowing the user to control individual "chunk" size to maximize performance.

VDPMill can be used with a local, multi-thread RenderX XEP engine for the *Formatter*. Optionally, VDPMill plugs into RenderX EnMasse, our load-balancer application. Leveraging EnMasse, the *Formatter* spreads the work across a grid of RenderX XEP engines. The grid could be a single machine running multiple threads of rendering, in one or more JVMs and even spread across multiple machines. This delivers formatting in parallel that can be spread across memory and CPUs to maximize the throughput for any one document. Using multiple JVMs, one can use more available memory to overcome Java memory limitations. Using multiple threads can leverage all cores in CPUs and machines. More formatting nodes allows more chunks to be formatted in parallel.



Apply XSLT E:\EnMasseHot\Samples\REX\Main.xsl

File	Folder	Si...	M...
REX_Financial_Statement-70000.xml	E:\EnMasseHot\Samples\REX	724...	1/1...

Output Directory E:\EnMasseHot\Samples\REX\Output

File	Size (Kb)
REX_Financial_Statement-70000.xml.pdf	326565.587

Splitter type	splitter1g
Start Point	
Start point type	element
name	report-data
namespace	
Split Point	
Generator PDF	
Generator PS	
Generator AFP	

```
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1594': generation finished
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1594': releasing end lock
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1595': start lock acquired
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1595': generating output
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1595': generation finished
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1595': releasing end lock
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1596': start lock acquired
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1596': generating output
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1596': generation finished
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1596': releasing end lock
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1597': start lock acquired
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1597': generating output
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1597': generation finished
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1597': releasing end lock
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1598': start lock acquired
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1598': generating output
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1598': generation finished
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1598': releasing end lock
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1599': start lock acquired
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1599': generating output
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1599': generation finished
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1599': releasing end lock
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1600': start lock acquired
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1600': generating output
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1600': generation finished
INFO [RFMTSCH] `file:/E:/EnMasseHot/Samples/REX/REX_Financial_Statement-70000.xml1600': releasing end lock
```

Processing done [1169.849 s]

Figure 2: The VDPMill GUI

VDPMill has a *Joiner* which controls the *Splitter* and *Formatter* so it understands document order and can execute RenderX XEP by sending individual internal structures to a running XEP engine (or even multiple threads of XEP engine for simultaneous multi-format output). It has recovery and retry file-safe functions to ensure documents are produced as intended and/or incorrect output can be at least reported or even fixed and processed.

And at each integration point in the process flow, VDPMill allows for customization of the data stream. Such things as injecting OMR marks into the output stream or inserting Transpromo advertising can be put in the entire process flow and take advantage of known information available like the XML structure or the page sequence.

As we discussed, this type of processing is suited for creating very large documents. Of course one can also split and not merge to generate many separate documents if desired. And the documents processed with these techniques are ideal for our VisualXSL designer.

In the future, RenderX will be adding such things as JDF Job ticketing to control the whole process. VDPMill is a fully configurable huge document, high performance, scalable formatting core solution that supports simultaneous generation of output streams.

And the Results

With VDPMill, RenderX has reached new milestones in XSL FO formatting. One can create a 500,000 page Postscript file on a laptop within 1GB of memory for Java. We know, we have done it. We have created solutions to format such things as W-2s, 1099's at 250 documents/second on standard hardware. We have reached speeds of 150+ pages/second for even complex documents like a 401K statement by spreading formatting across two quad-core machines . We have shown that the proper path to multi-format generation can be accomplished by simultaneously creating a large Postscript file to print such statements along with individual PDFs for email to customers. as opposed to some complex post-formatting file conversions.

Many in the past have tried to say that XSL FO was not suited for high performance print. As RenderX and 1000s of our customers have shown, this is simply not true. RenderX are our 100% standards-based technology delivers high-performance, print file generation to our customers and partners every day.

For more information and to get a trial of VDPMill, contact us:

Kevin Brown
RenderX, Inc.
kevin@renderx.com