**ITM-594 : Special Projects in IT**

**by**

**Megha Ramachandra**

**Aim**:

To build a markup tool which will be used to structure the unstructured text transcript. When the application is launched, it should connect to the database and pull the required video transcript data and display it on the screen. This will be a drag and drop application which provides the functionality to drop tags around a required transcript text displayed which will then render the tags as XML markup. The final XML document will be used with Docbook tools to convert it into required format such as PDF, HTML, Epub. Finally, the XML document data will be stored back into the database.

**Application Explanation:**

Foremost, I divided the project into two sub-projects namely i) Drag and Drop application and ii) Docbook to convert XML document to required format. This was done for convenience and to keep track of the goal to be achieved. Both project’s implementation explained below:

**Drag and Drop Application:**

The Drag and Drop Application is built on .Net platform using Visual Studio 2010. For now, I am using the local MYSQL database which will store video transcripts. So, when DragandDrop application is executed it connects to the local database MYSQL and pulls the required video transcript data from the table ‘Transcript’ and displays it in TextArea of the document. (Note: I have installed the MYSQL and created the table name ‘Transcript’ which has starttime, endtime, transcriptdata columns out of which transcriptdata column’s data is displayed in TextArea). Since I am using MYSQL, we need to add Mysql.Data dll into the project to connect to the database and pull the required data.

DragandDrop application consists of a single Default.aspx file which implements the user interface and has three JavaScript file namely EventHelpers, DragDropHelpers, setDataGetData.The JavaScript file plays an important role as it provides the functionality to drag the XML tag image which the user wants to insert as XML tags and add XML tags around the required text.

**EventHelpers**: This JavaScript library contains helper routines to assist HTML5 Drag and Drops behave consistently among browsers. I have add the code to work efficiently with browsers such as Firefox, Internet explorer, Google chrome. It handles several functions such has Set the mouse move event of a document, get the X and Y coordinate given the mouse pointer, Find the HTML object that fired an Event, Add an event to the document etc.

**DragDropHelpers**: This JavaScript library contains helper routines to assist HTML5 Drag and Drop behaves consistently among browsers. I have add the code to work efficiently with all browsers such as Firefox, Internet explorer, Google chrome. This handles several functions such as creating drag events like setting events for the respective mentioned node, dragStartEvent, DragEvent, DragEndEvent, getEventsCoordinates. This JavaScript is executed on the document only after EventHelpers JavaScript is loaded.

**setDataGetData**: This JavaScript library contains helper routines to add respective XML tags around the required text in text area. This JavaScript is triggered when an XML image to be inserted is dragged and dropped. EventHelpers and DragDropHelpers initiate the required function in this JavaScript to complete the required action. It handles several functions such as dragObject (XML tags images), DragStartEvent, Dropevent(inserts tags around text in TextArea), DragOverEvent etc where DragStartEvent triggers and checks whether the image selected for drag and drop is the correct image to be dropped.

**Default.aspx**: This implements the graphical user interface of the application which in backend connects to MYSQL database, pulls transcript data and displays it in text area. In the graphical user interface I have several XML tags images at the top from which XML tags are inserted in the textarea around the text when it is dragged and dropped. The XML images tags is set as dropTarget id which will be checked by JavaScripts to check whether the drag and drop target is valid image file to be dropped.

**Docbook**

**Introduction:**

DocBook is a [semantic](http://en.wikipedia.org/wiki/Semantics#Computer_science) [markup language](http://language) for technical documentation. It was originally intended for writing technical documents related to computer hardware and software but it can be used for any other sort of documentation. As a semantic language, DocBook enables its users to create [document content in a presentable neutral form](http://content) that captures the logical structure of the content; that content can then be published in a variety of formats, including [HTML](http://en.wikipedia.org/wiki/HTML), [XHTML](http://en.wikipedia.org/wiki/XHTML), [EPUB](http://en.wikipedia.org/wiki/EPUB), [PDF](http://format), man pages, [Web help](http://help) and [HTML Help](http://help), without requiring users to make any changes to the source.

First, I tried to implement DocBook files using eclipse which has ant built by default. The steps and tools needed to implement the DocBook in eclipse is explained below:

**Required Tools:**

To convert the xml files into other formats using Docbook we need the following tools:

* The DocBook DTD (I have used version 5.0) which defines the structure of a *DocBook* document.
* An XSLT stylesheet to convert your *DocBook* file into another format.
* An XSLT parser : I have used Saxon as the XSLT parser
* Eclipse IDE : used as xml editor
* Apache Ant: for XSLT transformation.

Note: Eclipse has Apache Ant integrated.

**Download Links:**

[DocBook DTD version](http://www.oasis-open.org/docbook/xml/4.5) 5.0: <http://www.docbook.org/xml/5.0/>

[XSLT stylesheets](http://sourceforge.net/projects/docbook/files/docbook-xsl/): <http://sourceforge.net/projects/docbook/files/docbook-xsl/>

Saxon: <http://saxon.sourceforge.net/>

Fop for PDF: <http://xmlgraphics.apache.org/fop/>

EPub jar: <http://grepcode.com/snapshot/repo1.maven.org/maven2/org.idpf/epubcheck/3.0>

**Steps I followed to convert Docbook to Html5 Using Eclipse:**

In Eclipse navigate the menu to *File* → *New* → *Project* entry and select *General* →*Projects*

The new project is called *“your-own-required-project-name”*

Create the following folder structure:

* input
* input/images
* css
* output
* docbook-xml-5.0
* docbook-xsl
* lib

Place the DocBook DTD (docbook-xml-5.0) and the XSLT stylesheets(docbook-xsl) into the corresponding directories. Copy the *jar* files from *Saxon* download into the *lib* folder.

In *input* folder place the xml files which you want to convert to other formats. Place the images used by xml files in *input/images* folder.

Next I created the buildhtml.xml file with required code, which is a script to convert the xml files to required format. We need to run the script file by right-clicking on it and then by selecting Run as -> ant-build. The required Html5 file of input files will be created in *output* folder of the project. The buildhtml.xml code dataflow is as shown below:

Declare project path

Define base properties i.e., input and output dir, stylesheets path.

Make saxon.jar available

Copy the stylesheets and images to the target directory

Including xml files, stylesheets and docbook convert xml into HTML.

**Steps I followed to convert Docbook to epub Using Eclipse:**

In Eclipse navigate the menu as *File* → *New* → *Project* entry and select *General* →*Projects*

The new project is called *“your-own-required-project-name”*

Create the following folder structure:

* input
* input/images
* css
* output
* docbook-xml-5.0
* docbook-xsl
* lib

Place the DocBook DTD (docbook-xml-5.0) and the XSLT stylesheets(docbook-xsl) into the corresponding directories. Copy the *jar* files from *Saxon* download into the *lib* folder.

Also, create a folder "epubinput" with a file "mimetype". This file should have the following content: "application/epub+zip" need to be the only content in the file and it needs to be in the first line of the file. In *input* folder place the xml files which you want to convert to other formats. Place the images used by xml files in *input/images* folder.

The *epubcheck-3.0.jar* should be downloaded and added into the *lib* folder

Next I created the buildhtml.xml file with required code, which is a script to convert the xml files to required format. We need to run the script file by right-clicking on it and by selecting Run as -> ant-build. The required epub file of input files will be created in *output* folder of the project.

Declare project path

Define base properties i.e., input and output dir, stylesheets path.

Make saxon.jar and epubcheck.jar available

Copy the stylesheets and images to the target directory

Iterates through a directory and transforms .xml files into .epub files using the DocBook XSL.

We create temporary zips so that minetype is the first one in the **final** zip.

Finally, Make sure the epubcheck lib has a subfolderlib with saxon.jar

**Steps I followed to convert Docbook to pdf using Eclipse:**

In Eclipse navigate the menu as *File* → *New* → *Project* entry and select *General* →*Projects*

The new project is called *“your-own-required-project-name”*

Create the following folder structure:

* input
* input/images
* css
* output
* docbook-xml-5.0
* docbook-xsl
* lib

Place the DocBook DTD (docbook-xml-5.0) and the XSLT stylesheets(docbook-xsl) into the corresponding directories. Copy the *jar* files from *Saxon* download into the *lib* folder.

In *input* folder place the xml files which you want to convert to other formats. Place the images used by xml files in *input/images* folder.

In addition to the existing setup Apache FOP library is required. I downloaded the binary format and copied all the jars from the FOP distribution in *lib*. Then add libs to the ant build path.

Next I created the buildhtml.xml file with required code, which is a script to convert the xml files to required format. We need to run the script file via right-click on it and by selecting Run as -> ant-build. The required pdf file of input files will be created in *output* folder of the project.

Declare project path

Define base properties i.e., input and output dir, stylesheets path.

Make Apache FOP jars available in lib folder.

Copy the stylesheets and images to the target directory

Iterates through a directory and transforms .xml files into .fo files using the DocBook XSL.

Convert DocBook Files into FO. Finally, convert FO files to pdf.

Integrating the ant build script file at the backend of the drag and drop application turned out very tedious. It included installing Nant and extra dll which would become very tedious as Nant requires changing path and also running the script file at the backend would require extra java related dll. So, I decided to implement the docbook directly in .Net. I was able to implement the convertion of docbook to HTML and PDF successfully.

The steps involved to convert Docbook to HTML in .Net are as follows:

**Loading the stylesheets:**

We want the book in HTML format, so we have to use the ‘docbook.xsl’ stylesheet (located in the html folder), which generates a single HTML output file. Basically I am creating a **XmlReader** object pointed to the stylesheet we want to use and passing it to the Load method of a **XslCompiledTransform** instance. This is pretty straightforward, but we need to set a couple of options to make it work:

* By default DTD processing is prohibited, so we’ll get an **XmlException** when loading the stylesheet. To fix this we set the ProhibitDtd property to false.
* The document() function is disabled by default. This is used by the stylesheets, so we need to enable it or we’ll get yet another exception. This is done where we create a new **XsltSettings** object and set its EnableDocumentFunction property to true.

**Applying the Transform:**

* This is the step when XML files will be converted into HTML. First, we set the conformance level to ConformanceLevel.Auto through a **XmlWriterSettings** instance. For some reason if you don’t do this an **InvalidOperationException** is thrown complaining about the validity of the output file, even though the HTML file looks fine.
* We can give parameters to the stylesheet. In this case, I’m defining the name of the CSS stylesheet to use (this will generate a link tag in the output file).

It turns out **XslCompiledTransform** doesn’t support XInclude. Luckily there’s a project called [Mvp.Xml](http://www.codeplex.com/MVPXML/) that provides this functionality through a class called **XIncludingReader**. If you read the Mvp.Xml documentation in this link <http://mvpxml.codeplex.com/> it looks as it’s only a matter of returning a **XIncludingReader** instance from the CreateReader method to make everything work, but it’s not. For some reason, a **XmlException** is thrown complaining about the DTD processing prohibition again. The solution is to subclass **XIncludingReader** and overrides its Settings property, making sure the returned object has the ProhibitDtd property set to false. None of the constructors in the **XIncludingReader** class accepts a **XmlReaderSettings** object, so it’s the only way.

It took me a while to figure the subclassing thing out, because internally the**XIncludingReader** class uses a normal XmlReader where the DTD prohibition is disabled. I don’t know how the **XslCompiledTransform** class interacts with the reader it receives, but it (or some other class) must be looking at this property at some point because otherwise the exception wouldn’t be thrown.

The steps involved to convert Docbook to PDF in .Net are as follows:

I used IKVM.net and Apache FOP to implement this. IKVM.NET it is not a pdf renderer. IKVM.NET is an implementation of Java for Mono and the Microsoft .NET Framework. It includes the following components:

* a Java Virtual Machine implemented in .NET
* a .NET implementation of the Java class libraries
* tools that enable Java and .NET interoperability

In the simplest form IKVM.NET allows to use a Java code library in the C# code and vice versa. The download links for IKVM.NET and Apache FOP respectively is as follows:

<http://www.ikvm.net/download.html>

<http://xmlgraphics.apache.org/fop/download.html>

From the FOP directory copy all the \*.jar’s files from lib and build catalogs to some location, e.g.In my case e:\fop.

Second step is to build the \*.dll library from these files. On the console execute the following comand:

ikvmc –target:library –out:e:\fop\fop.dll –recurse:e:\fop

The ikvmc is located in the bin subdirectory where you unpacked the IKVM.NET. You must execute this command from this catalog, add this path to the global variable PATH or specify the full path to the bin subdirectory.

If no error occurred during this process, the fop.dll library should be created. Then we need to add reference to the fop.dll and the IKVM dll’s : IKVM.OpenJDK.Core and IKVM.OpenJDK.XML.API.

Then we need to initialize the Apache FOP and then using the Docbook xsl convert it into fo template and then by reading the fo template we create the .NET MemoryStream stream to the generated pdf file. I also created one more cs file named DotNetOutputMemoryStream which is my wrapper for the Java OutputStream. I have created it to have the possibility to exchange data between the .NET and Java objects.

**Target Achieved So Far:**

|  |  |
| --- | --- |
| 1. Connect to database, pull transcript data and display it. | Done |
| 2. Drag and Drop XML icons | Done |
| 3. After dropping the tags update the text with xml tags around the required text | Done |
| 4. Docbook implementation at the back end to convert the xml file to required format. | Not yet integrated. |
| 5. Save back the XML data back to the database | Not Done |

**References**:

<http://docbook.org/>

<http://en.wikipedia.org/wiki/Docbook>

<https://developer.gnome.org/gtk-doc-manual/unstable/documenting_docbook.html.en>

<http://www.codeproject.com/Articles/4481/Documention-with-DocBook-on-Win32>

<http://msdn.microsoft.com/en-us/magazine/cc135985.aspx>

<http://social.msdn.microsoft.com/Forums/en-US/96a299cc-f42a-43e1-b364-37b157b9e6d1/convert-xml-file-to-pdf-format-using-c>