MULTILINGUAL USE OF LUCENE

KARL MUTCH

UCB INTRODUCTION TO WEB ALGORITHMS, 2010.

GOAL:

This initial project was done to exercise my knowledge of the basic environment and gain some practical experience.

The goal of this weeks effort is to gain an understanding of the use of UTF-8 within the context of Lucene and web crawling technology while running the standard scripts from Chapter 2 and then to modify them.

I am investigating this with the hope of being able to create a collection of searchable Coptic and Demotic documents consisting eventually of historical records of commercial and legal documents from the Ptolemaic period of Egyptian history. Doing this will allow academic scholars and investigators to build a corpus of documents to make generalized queries on textual information for documents rather than existing systems which require very constrained criteria when searching documents. This project will eventually be used to assist a site survey in Egypt starting this December.

INTENDED RESULTS:

Eventually I hope to create an online system for entering documents and performing searches on these along with performing clustering based characterization of documents.

WEEK 1 : TASKS

This week I hope to be able to determine how I can both crawl and store UTF-8 documents, initially using Coptic religious texts, in a Lucene database, and then perform naïve searches of the same in the spirit of Chapter 2.

Some of what I will investigate is how to undertake document conversion to recover Coptic text from ASCII files and perform conversions to render them as genuine Coptic UTF-8. These will then be formatted into HTML and crawled. Once stored in Lucene these will then be searched using BeanShell scripts accepting UTF-8 input so that queries can be correctly encoded in the source files using native Coptic script.

The final step will be to perform searches successfully.

WEEK 1 : DIARY

UTF-8 AND MULTILINGUAL STARTER STEPS

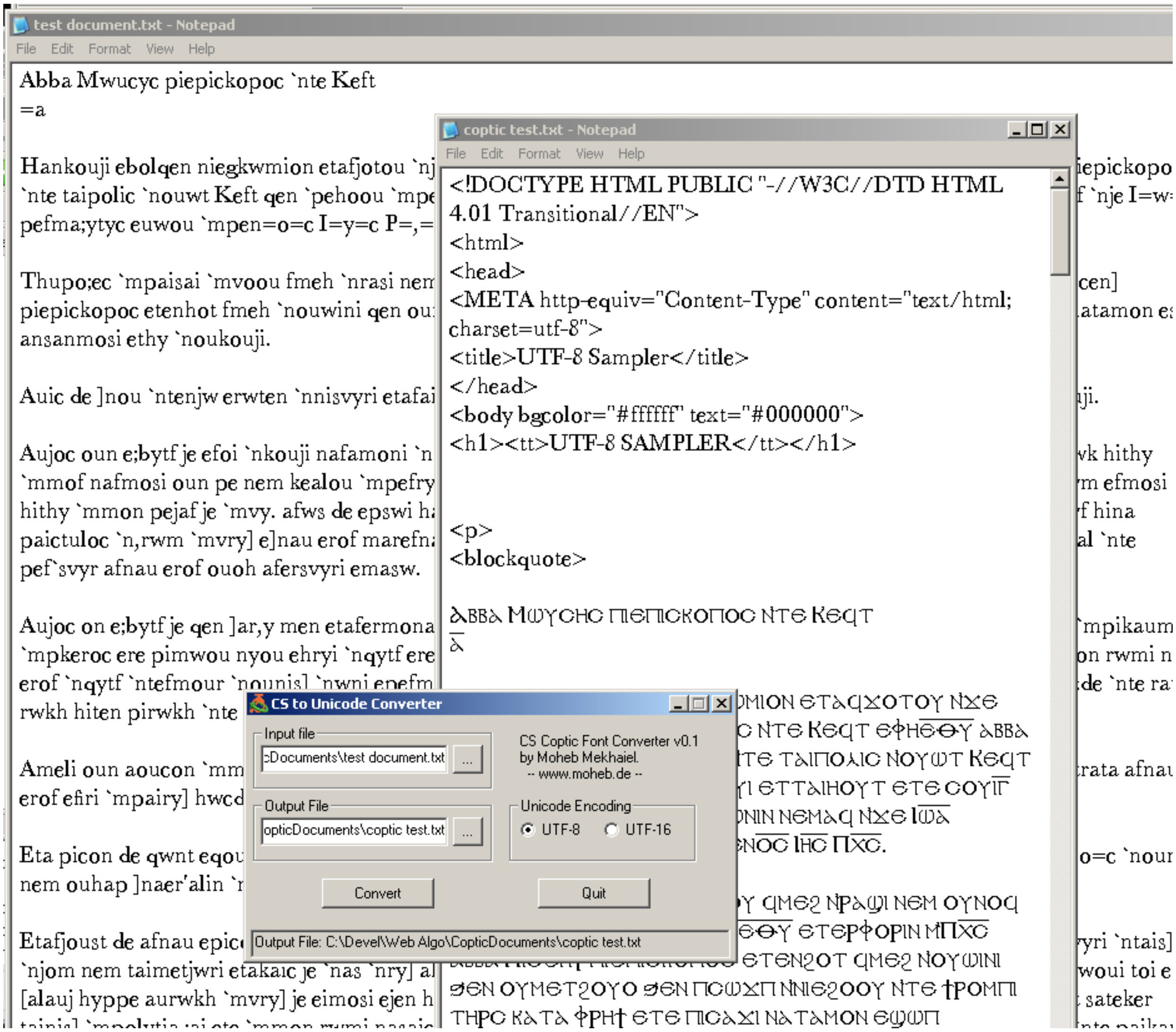
The first task for attempting to undertake the project is to select a number of tools for preparing input documents. To do this you will need a copy of a Unicode Font called “New Athena Unicode”. This font will allow you to view text in Unicode pages for archaic languages including Coptic. This font can be found at http://apagreekkeys.org/NAUdownload.html.

Having installed a suitable font a conversion utility is now needed to take existing Coptic texts encoded as raw ASCII into UTF-8 encoded Coptic texts. This is done using the tool “CS to Unicode Converter”, this can be downloaded from the web page http://www.moheb.de/cs_converter.html. There you will find three different converters for tools such as Microsoft Word etc. I am using the most primitive of these.

When you use the New Athena Unicode font to view files in Microsoft Notepad, or other UTF-8 aware editors you will see English and Coptic text cleanly. Be aware that editors that do not comprehend Unicode will often treat UTF-8 text files as a binary file. I recommend using Notepad++ which allows good UTF style editing including modifying the Byte Order Mark (***BOM***), http://notepad-plus-plus.org/release/5.7.

To convert files you should start with clean text and then the CS Converter tool will produce an output file that can be edited in Notepad++ and have HTML added to the output as English. Be sure to set the Encoding menu to Encoding to UTF-8 without BOM.

The following figure shows an example of an input and output document with the converter being used to produce each. The Coptic text.txt file contains both the UTF-8 coptic text and HTML that I have added to allow the HTML crawler to be used with the text.


test document.txt - Notepad
File Edit Format View Help
coptic test.txt - Notepad
File Edit Format View Help
CS to Unicode Converter
Convert
Quit

```
Abba Mwucyc piepickopoc `nte Keft
=a
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
4.01 Transitional//EN">
<html>
<head>
<META http-equiv="Content-Type" content="text/html;
charset=utf-8">
<title>UTF-8 Sampler</title>
</head>
<body bgcolor="#ffffff" text="#000000">
<h1><tt>UTF-8 SAMPLER</tt></h1>


<p>
<blockquote>

ⲀⲂⲂⲀ ⲘⲰⲨⲤⲎⲤ ⲠⲈⲠⲒⲤⲔⲞⲠⲞⲤ ⲚⲦⲈ ⲔⲈϤⲦ
ⲁ̅
```

Input file: :Documents\test document.txt ...

Output File: opticDocuments\coptic test.txt ...

CS Coptic Font Converter v0.1
by Moheb Mekhaiel.
-- www.moheb.de --

Unicode Encoding: ● UTF-8 ○ UTF-16

Output File: C:\Devel\Web Algo\CopticDocuments\coptic test.txt

For our test I produced several Coptic religious texts that are commonly read by students, are easily accessible, and fairly regular. It is my intention that in a future project I will add other categories' of texts and attempt to use clustering techniques to analyze texts possibly in future weeks.

I hope to replace this manual mechanism for preparing input texts with an analyzer customized to handle the languages desired at a later date.

In the mean time, I processed three texts for use in our project using the above method.

Keft Biography.txt

Father Shenoute.txt

Father John.txt

Prior to using beanShell there is a deficiency that I discovered with using UTF-8 input streams. The Input stream reader is not able to handle the Unicode Byte Order marker (BOM). To get around this there is a solution within the Bug Description, http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4508058. It should be noted however that, some aspects of java are not compatible with BeanShell. Including such features as, private static final variables, and certain types of array usage.

So although this could be done a lot more cleanly one can go with defining a naive function to read these types of files and to read the first three BOM bytes before presenting the file to BeanShell or software written for BeanShell.

For example when running scripts that use Coptic characters during searches we have to use a function similar to the following:

```
public sourceUTF8(URL url)
{
   in  = new InputStreamReader(url.openStream(), "UTF-8");

   char [] bom = new char[3];
   in.read(bom, 0, 3);
   this.interpreter.eval(in, this.caller.namespace,
                         "URL: "+url.toString());
   in.close();
}
```

The other option is to remove the BOM marker from the files. Doing this avoids having to re-implement classes used within the iWeb2 environment and makes prototyping easier. This is the approach I took for the documents themselves to reduce the complexity of coding during the early stages of this project.

To run the script UTF-8 source, if it uses the BOM, in the UTF-8 reader and then do the following:

```
sourceUTF8(new File("C:/Devel/Web Algo/project-1.bsh").toURL());
```

Otherwise without the BOM we use the following version of this function:

```
public sourceUTF8(URL url)
{
   in  = new InputStreamReader(url.openStream(), "UTF-8");

   this.interpreter.eval(in, this.caller.namespace,
                         "URL: "+url.toString());
   in.close();
}
```

INITIAL SCRIPT

In order to begin our simple test we code up a script for BeanShell using Notepad++ as follows:

```
// Initialize a Root  directory into which crawler and Lucene Data will be written

FetchAndProcessCrawler c = new FetchAndProcessCrawler("C:/Devel/Web Algo/", 5, 200);

// Create a filter for the iWeb2 crawler so that we can use file system based input rather than
// http
URLFilter urlFilter = new URLFilter();
urlFilter.setAllowFileUrls(true);
urlFilter.setAllowHttpUrls(false);
c.setUrlFilter(urlFilter);
```

```
// Define documents to be crawled and indexed
c.addUrl(new File("C:/Devel/Web Algo/Keft Biography.html").toURI().toASCIIString());
c.addUrl(new File("C:/Devel/Web Algo/Father Shenoute.html").toURI().toASCIIString());
c.addUrl(new File("C:/Devel/Web Algo/Father John.html").toURI().toASCIIString());

// Process our documents
c.run();

// generate Lucene index for these documents
LuceneIndexer lidx = new LuceneIndexer(c.getRootDir());
lidx.run();

// Initiate a test search of the documents indexed
MySearcher oracle = new MySearcher(lidx.getLuceneDir());
oracle.search("ⲁⲡⲁ",5);
```

Unfortunately BeanShell has patchy UTF-8 support. Using BeanShell will show the Coptic word for father (ⲁⲡⲁ) as "???". Here is output from the run of this script.

```
BeanShell 2.0b4 - by Pat Niemeyer (pat@pat.net)
bsh % public sourceUTF8(URL url)
{
   in  = new InputStreamReader(url.openStream(), "UTF-8");
   this.interpreter.eval(in, this.caller.namespace,
                         "URL: "+url.toString());
   in.close();
}
bsh % sourceUTF8(new File("C:/Devel/Web Algo/project-1.bsh").toURL());
Starting url group: 1, current depth: 0, total known urls: 3, maxDepth: 5, maxDo
cs: 200, maxDocs per group: 50, pause between docs: 500(ms)
Finished url group: 1, urls processed in this group: 3, current depth: 0, total
urls processed: 3
Starting url group: 2, current depth: 0, total known urls: 3, maxDepth: 5, maxDo
cs: 200, maxDocs per group: 50, pause between docs: 500(ms)
Finished url group: 2, urls processed in this group: 0, current depth: 0, total
urls processed: 3
Starting url group: 3, current depth: 1, total known urls: 3, maxDepth: 5, maxDo
cs: 200, maxDocs per group: 50, pause between docs: 500(ms)
Finished url group: 3, urls processed in this group: 0, current depth: 1, total
urls processed: 3
Timer (s): [Crawler processed data] --> 0.922
Starting the indexing ... Indexing completed!


Search results using Lucene index scores:
Query: ???


bsh %
```

Now it is time to try and switch to using the console. Upon doing this I discovered that the standard console, bsh.util.JConsole is not able to interpret UTF-8 strings for display. The following figure shows this:

```
bsh % print("ⲁⲡⲁ");
// Error: EvalError: Ambigous class names: [java.lang.String,
com.sun.org.apache.xpath.internal.operations.String] : at Line: 35 : in file:
/bsh/commands/print.bsh : String .valueOf ( arg )

Called from method: print : at Line: 1 : in file: <unknown file> : print ( "ⲁⲡⲁ" )
bsh %
```

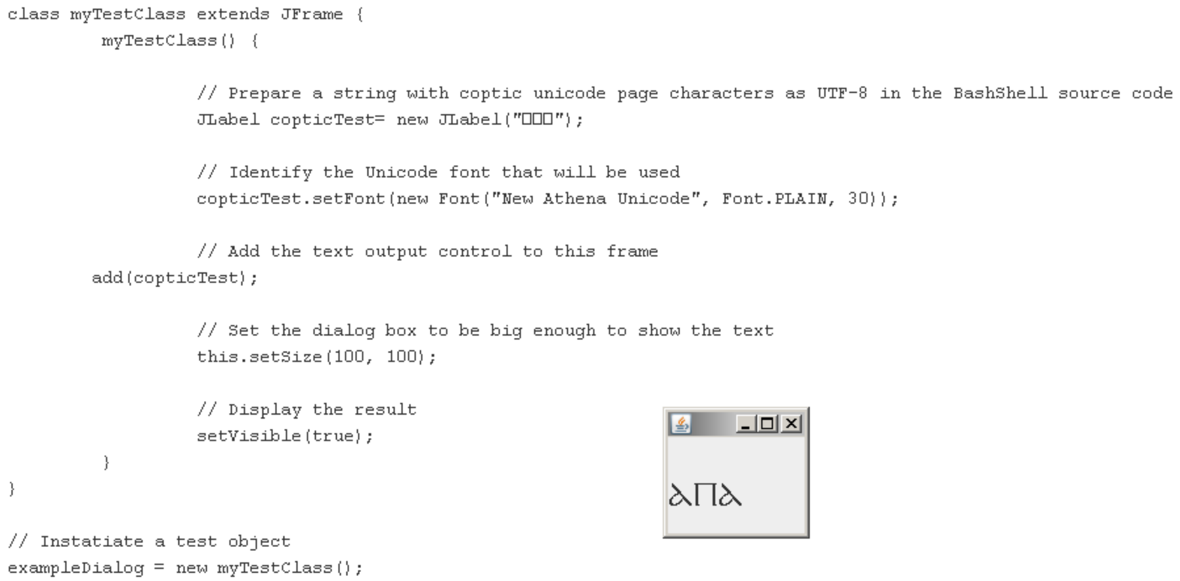
The alternative then is to make use of swing to display output something the iWeb2 libraries are not well suited to do. One alternative calls for doing this using the System.setOut and System.setErr. One other option was tried and that involved using the "File" menu in the console mode to "Capture System in/out/err". Although they both resulted in the text being printed and going to the console they both still failed to handle the UTF-8 output resulting in the same “???” output as previously.

UTF-8 DISPLAY SOLUTION

To test the ability of the BashShell to do this I created and ran the following script:

```
// A Class to help test that UTF-8 encoded strings can be correctly rendered for Coptic
// demotic and other code pages such as Hieroglyphic from BashShell based scripts

class myTestClass extends JFrame {
        myTestClass() {

                // Prepare a string with coptic unicode page characters as UTF-8 in the BashShell
source code
                JLabel copticTest= new JLabel("ⲁⲡⲁ");

                // Identify the Unicode font that will be used
                copticTest.setFont(new Font("New Athena Unicode", Font.PLAIN, 30));

                // Add the text output control to this frame
                add(copticTest);

                // Set the dialog box to be big enough to show the text
                this.setSize(100, 100);

                // Display the result
                setVisible(true);
        }
}

// Instatiate a test object
exampleDialog = new myTestClass();
```

The output appeared as follows:



```
class myTestClass extends JFrame {
        myTestClass() {

                // Prepare a string with coptic unicode page characters as UTF-8 in the BashShell source code
                JLabel copticTest= new JLabel("□□□");

                // Identify the Unicode font that will be used
                copticTest.setFont(new Font("New Athena Unicode", Font.PLAIN, 30));

                // Add the text output control to this frame
        add(copticTest);

                // Set the dialog box to be big enough to show the text
                this.setSize(100, 100);

                // Display the result
                setVisible(true);
        }
}

// Instatiate a test object
exampleDialog = new myTestClass();
```

ⲁⲡⲁ

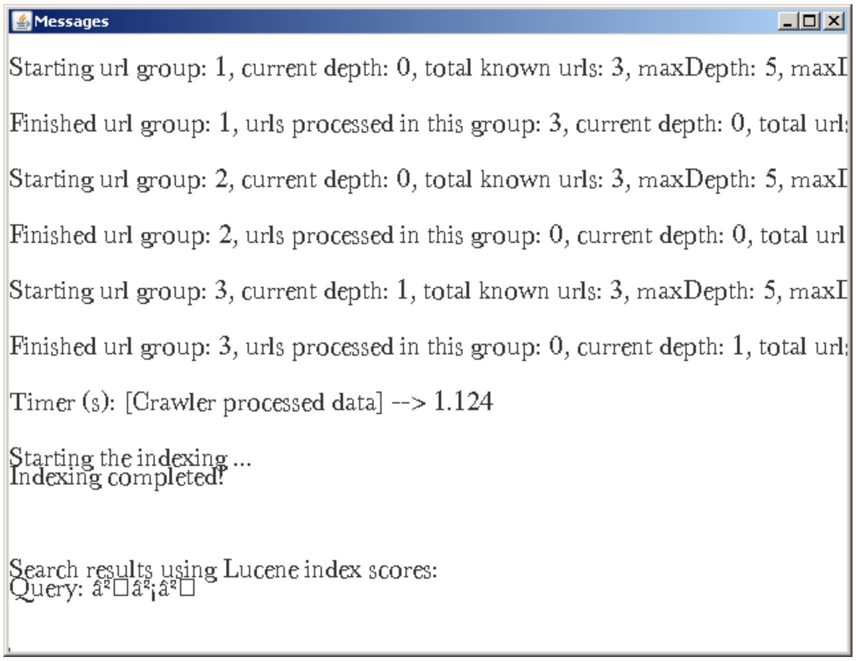
Now that we have a pattern to follow for output formatting we can hook up the console to the text output widget, allowing output from Lucene queries to be done.

```
/** A test class for exercising the UTF-8 features of Swing to
  * ensure Unicode Font compatibility */

public class RedirectedFrame extends JFrame {

    // Create an output widget for rendering unicode text */
    JTextArea aTextArea = new JTextArea();

    //  Use a font with Unicode pages relevant to our queries etc
    aTextArea.setFont(new Font("New Athena Unicode", Font.PLAIN, 20));

    // Add stream handling that will allow UTF-8 output and processing
    // when non UTF-8 devices are present such as those used by System.out
    PrintStream aPrintStream  =
       new PrintStream(
         new FilteredStream(
           new ByteArrayOutputStream()), true, "UTF-8");

    // Add a constructor that redirects the System.out and System.err to
    // our panel for display using swing
    RedirectedFrame() {

        // Direct output to a print stream
        System.setOut(aPrintStream);
        System.setErr(aPrintStream);

        // Decorate the window appropriately and set the size
        setTitle("Messages");
        setSize(500,300);
        setLayout(new BorderLayout());

        // Include the text widget in the Frame for display
        add("Center" , aTextArea);

        // Include a listener for closing the window and cleaning up
        addWindowListener
          (new WindowAdapter() {
             public void windowClosing(WindowEvent e) {
                dispose();
                }
             }
          );
       }

    // Define a method for output of UTF-8 strings to the text display
    void print(java.lang.String output)
    {
        aTextArea.append("\n" + output);
    }

    // Inner class for dealing with Byte oriented processing.
    // This class allows bye oriented data to be handled as
    // UTF-8 Strings
    class FilteredStream extends FilterOutputStream {
        public FilteredStream(ByteArrayOutputStream aStream) {
            super(aStream);
        }

        // Overridden byte array output method
        public void write(byte [] b) throws IOException {
            aString = new java.lang.String(b);
            aTextArea.append("\n" + aString);
        }

        // Overriden byte range output method.  This style of method
        // can be combined with PushBackInputStream when BOM UTF-8 file
        // handling is required
        public void write(byte [] b, int off, int len) throws IOException {
            aString = new java.lang.String(b , off , len);
            aTextArea.append("\n" + aString);
         }
    }
}
```

Now if we combine the scripts we have so far mentioned we end up with the following results:

```
BeanShell 2.0b4 - by Pat Niemeyer (pat@pat.net)
bsh % public sourceUTF8(URL url)
{
   in  = new InputStreamReader(url.openStream(), "UTF-8");
   this.interpreter.eval(in, this.caller.namespace,
                         "URL: "+url.toString());
   in.close();
}
bsh % sourceUTF8(new File("C:/Devel/Web Algo/RedirectedFrame.bsh").toURL());
bsh % outputTest = new RedirectedFrame(true);
bsh % sourceUTF8(new File("C:/Devel/Web Algo/project-1.bsh").toURL());
bsh %
```

Including the following dialog window:



```
Starting url group: 1, current depth: 0, total known urls: 3, maxDepth: 5, maxI

Finished url group: 1, urls processed in this group: 3, current depth: 0, total url:

Starting url group: 2, current depth: 0, total known urls: 3, maxDepth: 5, maxI

Finished url group: 2, urls processed in this group: 0, current depth: 0, total url

Starting url group: 3, current depth: 1, total known urls: 3, maxDepth: 5, maxI

Finished url group: 3, urls processed in this group: 0, current depth: 1, total url:

Timer (s): [Crawler processed data] --> 1.124

Starting the indexing ...
Indexing completed!



Search results using Lucene index scores:
Query: â²□â²¡â²□
```

As can be seen from this output the use of System.out and System.err have a constraint in that they do not support UTF-8, however output using the print methods do give UTF-8 support.

UTF-8 SEARCHING AND PROCESSING

Having arrived at this point it is clear that the Lucene engine has not been able to process the input as there are no results. To successfully deal with UTF-8 some changes are going to have to be made.

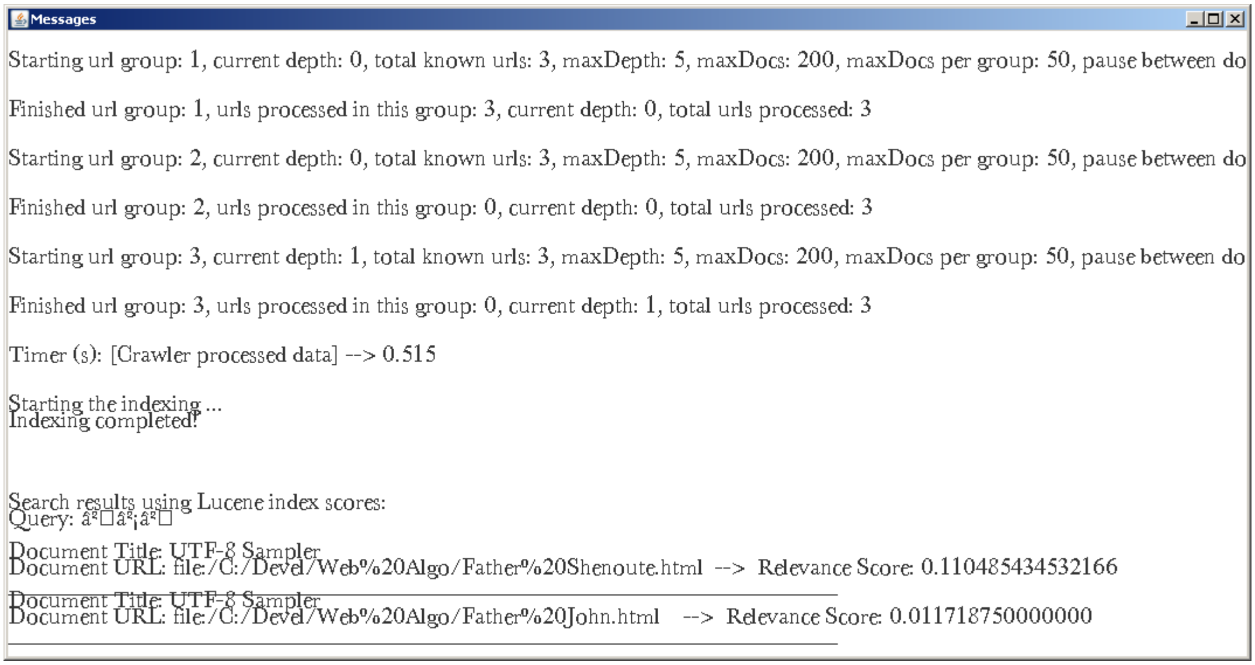
Here is a list of the things that need to change.

1. We need to convert from the StandardAnalyzer to the WhiteSpaceAnalyzer in MySearcher.java from C:\Devel\iWeb2\src\iweb2\ch2\shell.

2. The same change needs to happen to the C:\Devel\iWeb2\src\iweb2\ch2\lucene\LuceneIndexBuilder.java file as well.

It is critically important that the same analyzer is used between the indexing and search passes. Having done this we run ant in the build directory of the iWeb2 tree.

Now we restart the search process with these changes in place with the following result:



```
Starting url group: 1, current depth: 0, total known urls: 3, maxDepth: 5, maxDocs: 200, maxDocs per group: 50, pause between do

Finished url group: 1, urls processed in this group: 3, current depth: 0, total urls processed: 3

Starting url group: 2, current depth: 0, total known urls: 3, maxDepth: 5, maxDocs: 200, maxDocs per group: 50, pause between do

Finished url group: 2, urls processed in this group: 0, current depth: 0, total urls processed: 3

Starting url group: 3, current depth: 1, total known urls: 3, maxDepth: 5, maxDocs: 200, maxDocs per group: 50, pause between do

Finished url group: 3, urls processed in this group: 0, current depth: 1, total urls processed: 3

Timer (s): [Crawler processed data] --> 0.515

Starting the indexing ...
Indexing completed!


Search results using Lucene index scores:
Query: â²□â²¡â²□

Document Title: UTF-8 Sampler
Document URL: file:/C:/Devel/Web%20Algo/Father%20Shenoute.html  -->  Relevance Score: 0.110485434532166
Document Title: UTF-8 Sampler
Document URL: file:/C:/Devel/Web%20Algo/Father%20John.html   -->  Relevance Score: 0.011718750000000
```

The differences between the two documents is that the Shenoute document uses the Egyptian word for father while the biography of John use the more Semitic word (abba). Of course the biography of Keft which makes no reference to this word or a similar one is not in the search results.

LESSONS LEARNED

While offering a great prototyping environment BeanShell has some unusual incompatibilities with java that can be difficult to deal with. The best approach to dealing with complex classes is to first generate jar files and then use these to package code not directly used within the BeanShell scripts.

Dealing with input documents is a critical area for using Lucene effectively. Identifying and characterizing input is an art. For example in our case many documents needed where encoded using ASCII characters with proprietary mappings to fonts and required specialized converters to generate suitable Unicode documents.

Lucene is surprisingly intuitive to use for document based content and is optimized for this use case. Attempting to integration across different types of content look to be difficult and this is where solr might come in useful through content unification.

FUTURE WORK

Coptic is typically written in document without word breaks and punctuation. Coptic is a language that is not agglutinative, for example unlike German. Typically introduction of lexicographic word separators helps understanding and eases translation but may also introduce unwanted artifacts. It would be interesting to see if and analyzer and tokenizer could be introduced that can deal with these characteristics.