# NEETS Domain Modeling v 2

Previously we attempted this using an IKVM version of TIKA, but were unable to extract images properly. This new version uses Beaker 1.7.1 which is able to load TIKA classes properly, which Beaker 1.6 was unable to do. However 1.7.1 has problems with Groovy and Scala coexisting (maybe Java too). So while the Java below for using TIKA must run in 1.7.1 for class loading purposes, it may be necessary to run the rest in 1.6 until the Beaker bug is fixed.

# Using TIKA, convert PDF to HTML and extract images

Creates a folder for each pdf with extracted images and html. The images are a mixture of pngs and tifs. Tifs will not render in html, so we must follow this with a tif to png conversion

## Use TIKA to convert PDF to HTML and extract images

Jv **Java**

```java
/**
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

package org.apache.tika.example;


import java.io.IOException;
```

```
22 import java.io.FileWriter;
23 import java.io.PrintWriter;
24 import java.io.File;
25 import java.io.FileInputStream;
26 import java.io.FileOutputStream;
27 import java.io.OutputStreamWriter;
28 import java.io.BufferedWriter;
29 import java.nio.charset.Charset;
30
31 import org.apache.tika.config.TikaConfig;
32 import org.apache.tika.detect.Detector;
33 import org.apache.tika.exception.TikaException;
34 import org.apache.tika.extractor.EmbeddedDocumentExtractor;
35 import org.apache.tika.extractor.ParsingEmbeddedDocumentExtractor;
36 import org.apache.tika.io.FilenameUtils;
37 import org.apache.tika.metadata.Metadata;
38 import org.apache.tika.mime.MediaType;
39 import org.apache.tika.mime.MimeTypeException;
40 import org.apache.tika.parser.AutoDetectParser;
41 import org.apache.tika.parser.ParseContext;
42 import org.apache.tika.parser.Parser;
43 import org.apache.tika.sax.BodyContentHandler;
44 import org.xml.sax.ContentHandler;
45 import org.xml.sax.SAXException;
46
47 import org.apache.tika.sax.ToXMLContentHandler;
48 import org.apache.tika.parser.pdf.PDFParserConfig;
49
50 public class ExtractEmbeddedFiles {
51
52   private Parser parser = new AutoDetectParser();
53   private Detector detector = ((AutoDetectParser)parser).getDetector();
54   private TikaConfig config = TikaConfig.getDefaultConfig();
55
56   //public void extract(InputStream is,  Path outputDir) throws
   SAXException, TikaException, IOException {
57   public void extract(String inputPath) throws SAXException,
   TikaException, IOException {
58     /*
59     Metadata m = new Metadata();
60     ParseContext c = new ParseContext();
61     ContentHandler h = new BodyContentHandler(-1);
62
63     c.set(Parser.class, parser);
64     EmbeddedDocumentExtractor ex = new
   MyEmbeddedDocumentExtractor(outputDir, c);
65     c.set(EmbeddedDocumentExtractor.class, ex);
66
67     parser.parse(is, h, m, c);
68     */
69     File inputFile = new File(inputPath);
```

```
72      InputStream inputStream = new FileInputStream(inputFile);
73      File outputDirectory = new File(parentDirectory, inputFile.getName() +
    "-extracted" );
74      System.err.println(outputDirectory.getPath()); //junk
75
76      Parser parser = new AutoDetectParser();
77      ToXMLContentHandler handler = new
    org.apache.tika.sax.ToXMLContentHandler();
78
79      PDFParserConfig pdfConfig = new PDFParserConfig();
80      pdfConfig.setExtractInlineImages(true);
81
82      ParseContext parseContext = new ParseContext();
83
84      parseContext.set(PDFParserConfig.class, pdfConfig);
85      parseContext.set(Parser.class, parser);
86
87      EmbeddedDocumentExtractor ex = new
    MyEmbeddedDocumentExtractor(outputDirectory.toPath(), parseContext);
88      parseContext.set(EmbeddedDocumentExtractor.class, ex);
89
90      Metadata metadata = new Metadata();
91
92      parser.parse(inputStream, handler, metadata, parseContext);
93
94      String text = handler.toString().trim();
95
96      File outputFile = new File(outputDirectory, inputFile.getName() +
    ".xhtml" );
97      System.err.println(outputFile.getPath()); //junk
98      //PrintWriter printer = new PrintWriter( new FileWriter(
    outputFile.getPath() ));
99
100      PrintWriter printer = new PrintWriter( new BufferedWriter (new
    OutputStreamWriter(
101        new FileOutputStream( outputFile.getPath()  ),
102        Charset.forName("UTF-8").newEncoder()
103   )));
104     printer.print( text );
105     printer.close();
106
107   }
108
109   private class MyEmbeddedDocumentExtractor extends
    ParsingEmbeddedDocumentExtractor {
110     private final Path outputDir;
111     private int fileCount = 0;
112
113     private MyEmbeddedDocumentExtractor(Path outputDir, ParseContext
    context) {
114       super(context);
```

```
118     @Override
119     public boolean shouldParseEmbedded(Metadata metadata) {
120       return true;
121     }
122
123     @Override
124     public void parseEmbedded(InputStream stream, ContentHandler handler,
    Metadata metadata, boolean outputHtml)
125       throws SAXException, IOException {
126
127       //try to get the name of the embedded file from the metadata
128       String name = metadata.get(Metadata.RESOURCE_NAME_KEY);
129
130       if (name == null) {
131         name = "file_" + fileCount++;
132       } else {
133         //make sure to select only the file name (not any directory paths
134         //that might be included in the name) and make sure
135         //to normalize the name
136         name = FilenameUtils.normalize(FilenameUtils.getName(name));
137       }
138
139       //now try to figure out the right extension for the embedded file
140       MediaType contentType = detector.detect(stream, metadata);
141
142       if (name.indexOf('.')==-1 && contentType!=null) {
143         try {
144           name += config.getMimeRepository().forName(
145           contentType.toString()).getExtension();
146         } catch (MimeTypeException e) {
147           e.printStackTrace();
148         }
149       }
150       //should add check to make sure that you aren't overwriting a file
151       Path outputFile = outputDir.resolve(name);
152
153       //get parent, convert to file, make directory
154       //outputFile.getParent().toFile().mkdir();
155       //do a better job than this of checking
156       Files.createDirectories(outputFile.getParent());
157       Files.copy(stream, outputFile);
158
159
160     }
161   }
162 }
163
```

Run

Jv Java

```
1 import org.apache.tika.example.ExtractEmbeddedFiles;
2 import java.io.File;
3 import java.nio.file.Files;
4 import java.nio.file.Path;
5 import java.io.InputStream;
6 import java.io.FileInputStream;
7
8 File pdfDirectory = new
  File("/z/aolney/research_projects/braintrust/materials/NEETS/pdf/");
9 File[] pdfFiles = pdfDirectory.listFiles((d,name) ->
  name.endsWith(".pdf"));
10 for ( File pdfFile : pdfFiles ) {
11   ExtractEmbeddedFiles ex = new ExtractEmbeddedFiles();
12   ex.extract( pdfFile.getPath() );
13 }
```

Run

**Error**
2 lines of stderr, BeakerDisplay

## Use ImageMagick to convert TIF to PNG

Jv Java

```
1 import java.io.File;
2 import java.nio.file.Files;
3 import java.nio.file.Path;
4 import java.io.InputStream;
5 import java.io.FileInputStream;
6 import java.lang.ProcessBuilder;
7 import java.io.IOException;
8 import java.io.PrintWriter;
9 import java.io.OutputStreamWriter;
10 import java.io.Writer;
11 import java.io.FileOutputStream;
12
```

```
16     File tempScript = createTempScript(commands);
17
18     try {
19         ProcessBuilder pb = new ProcessBuilder("bash",
   tempScript.toString());
20         pb.inheritIO();
21         Process process = pb.start();
22         process.waitFor();
23     } catch (java.lang.InterruptedException e ){
24        System.err.println( e );
25     } finally {
26         tempScript.delete();
27     }
28 }
29
30 private static File createTempScript(String commands ) throws IOException {
31     File tempScript = File.createTempFile("script", null);
32
33     Writer streamWriter = new OutputStreamWriter(new FileOutputStream(
34             tempScript));
35     PrintWriter printWriter = new PrintWriter(streamWriter);
36
37     printWriter.println("#!/bin/bash");
38     printWriter.println(commands);
39
40     printWriter.close();
41
42     return tempScript;
43 }
44 }
45
```

Run

```
com.twosigma.beaker.javash.bkrfb7a9607.Bash
```

Jv Java

```
1 import java.io.File;
2 import java.nio.file.Files;
3 import java.nio.file.Path;
4 import java.io.InputStream;
5 import java.io.FileInputStream;
6
```

```
8
9  for ( File directory : htmlDirectories ) {
10       //do tif to png conversion here, assumes imagemagick is installed and
   we have bash shell
11       Bash.executeCommands("cd '" + directory.getPath() + "';for f in
   *.tif; do  echo \"Converting $f\"; convert \"$f\"  \"$(basename \"$f\"
   .tif).png\"; done");
12 }
```

Run

# Apply domain model, regenerate HTML a JSON

## Domain model

This was developed in `monodevelop` and pasted below. Code below runs fine, but the complexity of the code is such that it is advisable develop in the orginal solution and treat this as archival.

Two modifications must be made

- Comment out EntryPoint
- Add `main [||]` as last line so main is called

CSS for the output HTML was hand authored.

F# FSharp

```
1  type PageType =
2    | TOCPage
3    | TitlePage
4    | PrefacePage
5    | MainPage
6    | AppendixPage
7    | IndexPage
8    | AssignmentPage
9    with override this.ToString() = sprintf "%A" this
10
11 type Header =
12   | Chapter of string * int
13   | LearningObjectives of string
14   | Section of string
```

```
18   with override this.ToString() = sprintf "%A" this
19
20 type PageElement =
21   | ImageCaption of string
22   | ImageURL of string
23   | Header of Header
24   | Question of string
25   | Answer of string
26   | LearningObjective of string
27   | Paragraph of string
28   with override this.ToString() = sprintf "%A" this
29
30 //Helpers to classfy page type
31 type Page =
32   {
33     Number : int option; //preface pages are negative
34     NumberString : string;
35     Type : PageType
36     Elements : ResizeArray<PageElement>
37   }
38   with override this.ToString() =
39         let pageNumber =
40           match this.Number with
41           | Some( x ) -> x.ToString()
42           | None -> "no-number"
43         pageNumber + " " + this.Type.ToString() + " " + System.String.Join(
   " ", this.Elements)
44
```

Run

F# FSharp

```
1 #r "/z/aolney/repos/HtmlAgilityPack.1.4.9.5/lib/Net40/HtmlAgilityPack.dll"
2 #r "/z/aolney/repos/Newtonsoft.Json.9.0.1/lib/net40/Newtonsoft.Json.dll"
3
4 open HtmlAgilityPack
5
6 let (|IsTOC|_|) (elements:ResizeArray<PageElement>) =
7     elements |> Seq.tryFind(
8         function
9         | Header (Section hs) when hs.Contains("TABLE OF CONTENTS") ->
  true
```

```
13
14 let (|IsAppendix|_|) (elements:ResizeArray<PageElement>) =
15     elements |> Seq.tryFind(
16         function
17         | Header (Section hs) when hs.Contains("APPENDIX") -> true
18         | _ -> false
19         )
20
21 let (|IsIndex|_|) (elements:ResizeArray<PageElement>) =
22     elements |> Seq.tryFind(
23         function
24         | Header (Section hs) when hs.Contains("INDEX") -> true
25         | _ -> false
26         )
27
28 let (|IsAssignment|_|) (elements:ResizeArray<PageElement>) =
29     elements |> Seq.tryFind(
30         function
31         | Header (Section hs) when hs.StartsWith("ASSIGNMENT") ||
   hs.StartsWith("Assignment")  -> true
32         | _ -> false
33         )
34
35 //helpers to classify page elements
36 let (|IsImageCaption|_|) (text:string,html:string, _) =
37     if text.StartsWith("Figure") && text.Contains("–") then
38         Some(text)
39     else
40         None
41
42 let (|IsImageUrl|_|) (text:string,html:string,_) =
43     if html.Contains("img src=") then
44         Some( html )
45     else
46         None
47
48 //It appears that headers never have clause final punctuation
49 let (|IsHeader|_|) (rawText:string,html:string,_) =
50     let text = rawText.Trim()
51     if text = "" || text.Contains(".") || text.Contains("?") ||
   text.Contains("!") || text.EndsWith(":") then
52         None
53     else
54         Some(text)
55
56 let questionRegex = new System.Text.RegularExpressions.Regex("^\s*Q\d+")
57 //ignores that multiple questions can be in one pargraph
58 let (|IsQuestion|_|) (text:string,html:string,_) =
59     if questionRegex.IsMatch(text) then
60         Some(text)
61     else
```

```
65 let (|IsAnswer|_|) (text:string,html:string,_) =
66     if answerRegex.IsMatch( text ) then
67         Some( text)
68     else
69         None
70
71 let learningObjectiveRegex = new
   System.Text.RegularExpressions.Regex("^\s*\d+")
72 let (|IsLearningObjective|_|) (text:string,html:string,lastHeader:Header)
   =
73     let trimmed = text.Trim()
74     let patternMatch = learningObjectiveRegex.IsMatch( trimmed ) &&
   trimmed.EndsWith(".")
75     match lastHeader,patternMatch with
76     | Header.LearningObjectives(text),true -> Some(text)
77     | _ -> None
78
79
80 let capitalLetterRegex = new System.Text.RegularExpressions.Regex("^[A-
   Z\s-]+$")
81 let pageNumberRegex = new System.Text.RegularExpressions.Regex("^\s*
   [ivxIVXANDEX0-9-]+\s*$") //may need adjusting for appendix
82
83 //assumes we have already tested this is in fact a header
84 let HeaderClassifier (text:string) (html:string) =
85     if text.StartsWith("CHAPTER") then
86         let s = text.Split(' ')
87         match System.Int32.TryParse( s.[1] ) with
88         | true, number -> Header.Chapter (text,number)
89         | false, _ -> Section text
90     else if text.StartsWith("LEARNING OBJECTIVES") then
91         Header.LearningObjectives text
92     else if text.StartsWith("SUMMARY") then
93         Header.Summary text
94     else if pageNumberRegex.IsMatch(text) then
95         PageNumber text
96     else if capitalLetterRegex.IsMatch(text) then
97         Section text
98     else
99         Subsection text
100
101 let GetElementByChapterList( pageList : ResizeArray<Page> ) (
   elementMatcher : PageElement -> string option ) =
102     //for each chapter, build a list of following sections
103     let mutable sectionList = new ResizeArray<string*int>()
104     let mutable chapterList = new
   ResizeArray<string*int*ResizeArray<string*int>>()
105     let chapterHash = new System.Collections.Generic.HashSet<string>()
   //specifically to solve chapters appearing in appendix
106     let mutable currentChapter = (0,"")
107
```

```
111             //then see if our page number is still increasing
112             match pageList.[index].Number,pageList.[index+1].Number with
113             | Some(current),Some(next) when next > current -> true
114             | _ -> false
115         else
116             false
117 
118     let mutable i = 0
119     while( i < pageList.Count - 1 ) do //ignore last page
120         let chapterText = pageList.[i].Elements |> Seq.tryPick( function |
    Header ( Chapter (text,_) ) -> Some( text ) | _ -> None)
121 
122         if chapterText.IsSome && chapterHash.Contains(
    chapterText.Value.Trim() ) |> not then //b/c closures and mutation
123             currentChapter <- (i,chapterText.Value)
124 
125             while ShouldContinue i pageList do
126                 //this only works because on a page, sections never come
    before chapters
127                 let sections = pageList.[i].Elements |> Seq.choose(
    elementMatcher )
128                 for sectionText in sections do  sectionList.Add(
    sectionText, i )
129                 i <- i + 1
130 
131             chapterList.Add( currentChapter |> snd, currentChapter |> fst,
    sectionList )
132             chapterHash.Add( (currentChapter |> snd).Trim() ) |> ignore
133             sectionList <- new ResizeArray<string*int>()
134             i <- i + 1
135         else
136             i <- i + 1
137     //return
138     chapterList
139 
140 let whitespaceRegex = new System.Text.RegularExpressions.Regex("\s+")
141 
142 //render an entire file to html
143 let RenderToHTML (fileName : string) (pageList : ResizeArray<Page> ) =
144 
145     //helpers for internal link ids
146     let SectionLinkId (text:string) (page:Page) =
147         whitespaceRegex.Replace("section-" + text + " " +
    page.NumberString, "-" )
148     let PageLinkId (page:Page)=
149         whitespaceRegex.Replace("page-" + page.Type.ToString() + " " +
    page.NumberString,"-")
150 
151     //helpers for images
152     let imageNamePrefix = fileName.Replace(".xhtml","")
```

```
155         |  "image0.png"  |  "image1.png"  ->  "image-logo"
156         | _ -> "image-regular"
157
158     //html for an element
159     let GetElementHTML (page:Page) (element:PageElement) =
160         match element with
161         | ImageCaption text -> "<p class=\"imagecaption\">" + text + "
    </p>"
162         | ImageURL html ->
163           let regexMatch = imageUrlRegex.Match( html )
164           //we've already converted tifs to pngs at this stage
165           let imageFile = regexMatch.Groups.
    ["imageFile"].Value.Replace(".tif",".png")
166           "<img class='" + (ImageClass imageFile) + "' src=\"" + imageFile
    + "\" alt=\"" + imageFile + "\" />"
167         | Header ( Chapter (text,_) ) -> "<h1 class=\"chapter\" id=\"" +
    (SectionLinkId text page) + "\">" + text + "</h1>"
168         | Header ( LearningObjectives text ) -> "<h2
    class=\"learningobjectivesection\">" + text + "</h2>"
169         | Header ( Section text ) ->  "<h2 class=\"section\" id=\"" +
    (SectionLinkId text page)  + "\">" + text + "</h2>"
170         | Header ( Subsection text ) -> "<h3 class=\"subsection\">" + text
    + "</h3>"
171         | Header ( PageNumber text ) -> "<p class=\"pagenumber\">" + text
    + "</p>"
172         | Header ( Summary text ) -> "<h1 class=\"summary\">" + text + "
    </h1>"
173         | Question text -> "<p class=\"question\">" + text + "</p>"
174         | Answer text -> "<p class=\"answer\">" + text + "</p>"
175         | LearningObjective text -> "<p class=\"learningobjective\">" +
    text + "</p>"
176         | Paragraph text -> "<p class=\"paragraph\">" + text + "</p>"
177
178     //page helper function
179     let RenderPageElementsToHTML fileName ( page : Page ) =
180
181         let sb = System.Text.StringBuilder(page.Elements.Count)
182
183         //split out image elements from rest
184         let imageList,otherList = page.Elements |> Seq.toList |>
    List.partition( function | ImageCaption _ -> true | ImageURL _ -> true | _
    -> false )
185         let urlList,captionList = imageList |> List.partition(function |
    ImageURL _ -> true | _ -> false )
186         let urlElements = new ResizeArray<PageElement>( urlList )
187         let captionElements = new ResizeArray<PageElement>( captionList )
188
189         //create image div for all image elements on this page, it goes at
    top
190         sb.Append("<div class=\"imageblock\">") |> ignore
191         //for each image/caption, create a div
```

```
195             //add a caption if it exists
196             if i < captionElements.Count then
197                 sb.Append( GetElementHTML page captionElements.[i] ) |>
    ignore
198             sb.Append("</div>") |> ignore
199         sb.Append("</div>") |> ignore
200 
201         //add remaining page elements
202         //for element in page.Elements do
203         for element in otherList do
204             sb.Append( GetElementHTML page element ) |> ignore
205         //return everything in the string builder
206         sb.ToString()
207 
208     //body helper function
209     let RenderHTMLBody fileName (pageList : ResizeArray<Page> ) =
210         let GetPageNav ( index : int ) (pageList : ResizeArray<Page> ) =
211             let previousPageId = if index > 0 then (index-1).ToString()
    else "" //PageLinkId pageList.[index-1] else ""
212             let nextPageId = if index < pageList.Count - 1 then
    (index+1).ToString() else ""  //PageLinkId pageList.[index+1] else ""
213             """<div class="centered">
214             <div class="pagination">
215             <a href='#""" + previousPageId + """'>❮</a>
216             <a href='#""" + nextPageId + """'>❯</a>
217             </div></div>"""
218 
219         let sb = System.Text.StringBuilder(pageList.Count)
220         //the body is just a series of pages
221         for i = 0 to pageList.Count - 1 do
222             let page = pageList.[i]
223             //create an internal link for navigation purposes
224             //each page is a div with classes for page and pageType along
    with a pagination based internal link
225             sb.Append("<div class=\"page " +
    page.Type.ToString().ToLower() + "\" id=\"" + i.ToString() + "\">") |>
    ignore
226             sb.Append( GetPageNav i pageList ) |> ignore
227             sb.Append( RenderPageElementsToHTML fileName page) |> ignore
228             sb.Append("</div>\n") |> ignore
229         //return everything in the string builder
230         sb.ToString()
231 
232 
233     //get a list of chapters and sections within them
234     let chapterList = GetElementByChapterList pageList ( function | Header
    ( Section (text) ) -> Some( text ) | _ -> None)
235 
236     //return main html with body expanded
237     let sb = System.Text.StringBuilder()
238     sb.Append(
```

```
242         <html xmlns= http://www.w3.org/1999/xhtml >
243         <head>
244         <title>""" + fileName.Replace(".pdf.xhtml","") + """</title>
245         <link rel="stylesheet" type="text/css" href="../neetstyle.css"/>
246         </head>
247         <body>
248         <!-- Site navigation menu -->
249         <header>
250         <div class="nav">
251         <ul>""" ) |> ignore
252 
253     //add a heading for each chapter
254     for text,index,sections in chapterList do
255         sb.Append("<li class='main-menu'><a href='#" + index.ToString() +
    "'>" + text + "</a>" ) |> ignore
256         //add a sublist to heading for each section
257         sb.Append("<ul>") |> ignore
258         for sectionText,sectionIndex in sections do
259             sb.Append("<li><a href='#" + sectionIndex.ToString() + "'>" +
    sectionText + "</a></li>" ) |> ignore
260         sb.Append("</ul>") |> ignore //close sublist
261         sb.Append("</li>") |> ignore //close heading
262     sb.Append("</ul>") |> ignore //close all headings
263     sb.Append("</div>") |> ignore //close nav
264     sb.Append(
265         (RenderHTMLBody fileName pageList) +
266         """</body>
267         </html>""") |> ignore
268     sb.ToString()
269 
270 
271 // Learn more about F# at http://fsharp.net
272 // See the 'F# Tutorial' project for more help.
273 //[<EntryPoint>]
274 let main argv =
275 
276     let directoryInfo = new
    System.IO.DirectoryInfo(@"/z/aolney/research_projects/braintrust/materials
    /NEETS/xhtml")
277     let files = directoryInfo.GetDirectories() |> Array.collect( fun x ->
    x.GetFiles("*.xhtml") )
278 
279     //delete existing pretty files
280     let prettyFiles = directoryInfo.GetDirectories() |> Array.collect( fun
    x -> x.GetFiles("*pretty*" ) )
281     for file in prettyFiles do
282         file.Delete()
283 
284     for file in files do
285         let filePath = file.FullName
286 
```

```
290             doc.Load filePath
291             doc.DocumentNode.Element("html").Element("body")
292 
293 
294         //each div is a page
295         //our goal is to transform this div list into a sequence of
    structured page objects
296         let divList = new ResizeArray<HtmlNode>( seq { for node in
    body.Elements("div") do yield node }  )
297         let pageList = new ResizeArray<Page>()
298 
299         //we need some state because a section may only be marked on the
    first page
300         let mutable firstPage = true
301         let mutable prefacePage = 0
302         let mutable lastPageType = TitlePage
303 
304         //loop over all divs to construct pages
305         for div in divList do
306 
307             //prepare a list of pageElements corresponding to nodes on
    page
308             let pageElementList = new ResizeArray<PageElement>()
309 
310             //get a list of nodes we can traverse sanely, ignoring #text
    nodes
311             let nodeList = ResizeArray<HtmlNode>(  seq { for node in
    div.ChildNodes do if node.Name <> "#text" then yield node } )
312 
313             //construct a page element for each node
314             let mutable lastHeader = Summary "" //we need some state for
    learning objectives, seed with junk
315 
316             for node in nodeList do
317                 //construct a PageElement
318                 let html = node.OuterHtml
319                 let text = node.InnerText
320                 let pageElement =
321                     match (text,html,lastHeader) with
322                     | IsImageCaption t -> ImageCaption( text )
323                     | IsImageUrl t -> ImageURL(html)
324                     | IsHeader t ->
325                         let header =  HeaderClassifier text html
326                         lastHeader <- header
327                         Header( header )
328                     | IsQuestion t -> Question(text)
329                     | IsAnswer t -> Answer(text)
330                     | IsLearningObjective t -> LearningObjective( text )
331                     | _ -> Paragraph(text)
332                 pageElementList.Add(pageElement)
333 
```

```
337             let pageNumber,pageNumberString =
338                 let pageNumberElement =
339                     pageElementList |> Seq.tryFind(
340                         function
341                         | Header (PageNumber pn)  -> true
342                         | _ -> false
343                         )
344 
345                 match pageNumberElement with
346                 | Some( Header (PageNumber pn) ) ->
347                    let s = pn.Split('-')
348                    if s.Length > 1 then
349                         match System.Int32.TryParse( s.[1] ) with
350                         | true,page -> Some page,pn
351                         | false,_ -> prefacePage <- prefacePage - 1; Some
    prefacePage,pn
352                    else if s.Length = 1 then
353                         match System.Int32.TryParse( s.[0] ) with
354                         | true,page -> Some page,pn
355                         | false,_ -> prefacePage <- prefacePage - 1; Some
    prefacePage,pn
356                    else
357                         prefacePage <- prefacePage - 1; Some
    prefacePage,pn
358                 | _ -> None,""
359 
360 
361 
362             let pageType =
363                 //have we hit a page transition
364                 let transitionPage =
365                     match pageElementList with
366                     | IsTOC e -> TOCPage
367                     | IsAppendix e -> AppendixPage
368                     | IsIndex e -> IndexPage
369                     | IsAssignment e -> AssignmentPage
370                     | _ -> MainPage //default
371 
372                 //special cases
373                 if firstPage then
374                     firstPage <- false
375                     TitlePage
376                 else if pageNumber.IsSome && pageNumber.Value < 0 &&
    transitionPage <> TOCPage then
377                     PrefacePage
378                 else if transitionPage = AppendixPage then
379                     AppendixPage
380                 else if transitionPage = IndexPage then
381                     IndexPage
382                 else if transitionPage = AssignmentPage then
383                     AssignmentPage
```

```
386                     AppendixPage
387                 else if transitionPage = MainPage && lastPageType =
    IndexPage then
388                     IndexPage
389                 else if transitionPage = MainPage && lastPageType =
    AssignmentPage then
390                     AssignmentPage
391                 else if transitionPage = TOCPage then
392                     TOCPage
393                 else
394                     MainPage
395 
396             pageList.Add(
    {Number=pageNumber;NumberString=pageNumberString;Type=pageType;Elements=pa
    geElementList})
397 
398             lastPageType <- pageType
399 
400         //Serialize to json
401         let json = Newtonsoft.Json.JsonConvert.SerializeObject(pageList,
    Newtonsoft.Json.Formatting.Indented )
402         System.IO.File.WriteAllText( filePath + ".json", json,
    System.Text.Encoding.UTF8 )
403 
404         //generate a "nice" html page
405         let html = RenderToHTML file.Name pageList
406         System.IO.File.WriteAllText( filePath + "-pretty.html", html,
    System.Text.Encoding.UTF8)
407 
408     printfn "%A" argv
409     0 // return an integer exit code
```

Run

F# FSharp

Run

Most reasonable approach is to section the text by chapters and parse those (but not headers). Parsing at the section level is possible but seems a bit too severe. The exact grain size/unit for parsing at this point is debateable. It is only relevant for coreference and discourse parsing; everything else is sentence level.

**NOTE THE TEXT HAS WHITESPACE STRIPPED AND IS UNIDECODED, SO IT NO LONGER IS BINARY MATCHED TO THE ORIGINAL**

F# FSharp

```
#r "/z/aolney/repos/Newtonsoft.Json.9.0.1/lib/net40/Newtonsoft.Json.dll"
#r "/z/aolney/repos/UnidecodeSharp.1.0.0.0/lib/net35/UnidecodeSharp.dll"
open BinaryAnalysis.UnidecodeSharp

///tuples so we can align parsed text to pages
let cleanRegex = new System.Text.RegularExpressions.Regex("[\s\r\n]+")
let GetTextPageTuples ( sections : ResizeArray<string*int> ) =
  sections
  |> Seq.map( fun (text,page) -> cleanRegex.Replace( text, " ").Trim().Unidecode(), page.ToString() )
  |> Seq.filter( fun (text,page) -> text.Length > 0)

let directoryInfo = new System.IO.DirectoryInfo(@"/z/aolney/research_projects/braintrust/materials/NEETS/xhtml")
let files = directoryInfo.GetDirectories() |> Array.collect( fun x -> x.GetFiles("*.json") )

for file in files do
  let filePath = file.FullName

  //go from file to json to object
  let json = System.IO.File.ReadAllText(filePath)
  let pageList = Newtonsoft.Json.JsonConvert.DeserializeObject<ResizeArray<Page>>(json)

  //get only paragraphs of text we want, write to file
  let chapterList = GetElementByChapterList pageList ( function | Paragraph (text) -> Some( text ) | _ -> None)

  //for each chapter in chapter list, create a parseable file
  for text,index,sections in chapterList do
    let texts,pages =  sections |> GetTextPageTuples |> Seq.toList |> List.unzip
    System.IO.File.WriteAllLines( filePath + "-" + text.Replace("\n","") + ".toparse", texts)
```

Run

## Use CLU on input text (WARNING: THIS TAKES ABOUT 1 H(

**ONLY PERFORM IF YOU NEED TO REGENERATE SER**

Using .toparse files prepared by fsharp

- load each file
- parse using clu
- serialize as .ser

Sc Scala

```
1 import java.io.File
2 def recursiveListFiles(f: File): Array[File] = {
3   val these = f.listFiles
4   these ++ these.filter(_.isDirectory).flatMap(recursiveListFiles)
5 }
6
```

Run

```
import java.io.File
recursiveListFiles: (f: java.io.File)Array[java.io.File]
```

Sc Scala

```
1 import edu.arizona.sista.processors.corenlp.CoreNLPProcessor
2 import java.io._
3 import edu.arizona.sista.utils.Files
4 import scala.collection.mutable.ListBuffer
5 import edu.arizona.sista.processors.Processor
6 import edu.arizona.sista.utils.Serializer
7 import java.io.File
8 import scala.io.Source
```

```
12 val filesToParse = recursiveListFiles( new File( rootDirectory ) ).filter(
   f=> """.*\.toparse$""".r.findFirstIn(f.getName).isDefined)
13
14 //300 word limit is empirically determined based on some sample sentences
15 val proc:Processor = new CoreNLPProcessor(withDiscourse =
   true,maxSentenceLength = 300)
16
17 //ASSUMES EACH LINE IS A SENTENCE -- CLASS5 ASSUMPTION
18 /*
19 def fileToSentences(file:File):List[String] = {
20   val sents = new ListBuffer[String]
21   io.Source.fromFile(file).getLines().foreach(sents += _)
22   sents.toList
23 }
24 */
25
26 for( file <- filesToParse ) {
27   println(s"Parsing and serializing $file...")
28   //val doc = proc.annotateFromSentences( fileToSentences( file ) )
29   val text = Source.fromFile(file, "utf-8").getLines().mkString(" ")
30   val doc = proc.annotate( text )
31   Serializer.save(doc, file + ".ser")
32 }
33
34
```

Run

9 lines of stdout, 10 lines of stderr

# Map Parse to JSON and JSON to F#

TODO add a bit here to add SRL annotation (or do the SRL parse). See /z/aolney/research_projects/csal/analysis/cloze-memory/cloze-generation-0416.bkr

**Scala to JSON**

```
1  import java.io.PrintWriter
2  import java.io.File
3  import org.json4s._
4  import org.json4s.JsonDSL._
5  import org.json4s.native.JsonMethods._
6  import edu.arizona.sista.discourse.rstparser.DiscourseTree
7  import edu.arizona.sista.discourse.rstparser.RelationDirection
8  import edu.arizona.sista.discourse.rstparser.TokenOffset
9  import scala.collection.mutable.ListBuffer
10 import edu.arizona.sista.utils.Serializer
11 import edu.arizona.sista.struct.DirectedGraphEdgeIterator
12 import edu.arizona.sista.utils.Files
13
14
15 def dTreeJSON (dTree:DiscourseTree) : JValue = {
16   val json = dTreeJSONRec( dTree, JObject(), true, true, 0 )
17   return json
18   //compact( render( json ))
19 }
20
21 def dTreeJSONRec ( dTree:DiscourseTree, argJson:JValue,
   printChildren:Boolean, printKind:Boolean, depth:BigInt): JValue = {
22     var json = argJson
23
24     if (printKind) {
25       json = json merge JObject(JField("kind",
   JString(dTree.kind.toString())))
26     }
27
28     if (dTree.relationLabel.length > 0) {
29       json = json merge JObject(JField("relLabel",
   JString(dTree.relationLabel)))
30       if (dTree.relationDirection != RelationDirection.None) {
31         json = json merge JObject(JField("relDir",
   JString(dTree.relationDirection.toString())))
32       }
33     }
34
35     json = json merge JObject(JField("depth", JInt( depth )))
36     json = json merge JObject(JField("sentence",
   JInt(dTree.firstToken.sentence)))
37     json = json merge JObject(JField("firstToken",
   JInt(dTree.firstToken.token)))
38     json = json merge JObject(JField("lastToken",
   JInt(dTree.lastToken.token)))
39
40     if (dTree.rawText != null) {
41       json = json merge JObject(JField("text", JString(dTree.rawText)))
42     }
43
44     if (printChildren) {
```

```
47          if (kids.length > 0)
48            json = json merge JObject(JField("kids", JArray(kids.toList)))
49        }
50      }
51      return json
52    }
53
54  def safeGet(x: Option[Array[String]], i: Int) = x match {
55        case Some(s) => s(i)
56        case None => ""
57     }
58
59  //-----------------------------------------------
60  //loop over docs in working directory
61
62  //val parsedFiles = Files.findFiles(beaker.workingDirectory.toString(),
    "ser")
63  val rootDirectory =
    "/z/aolney/research_projects/braintrust/materials/NEETS/xhtml"
64  val parsedFiles = recursiveListFiles( new File( rootDirectory ) ).filter(
    f=> """.*\.ser$""".r.findFirstIn(f.getName).isDefined)
65
66
67  for( file <- parsedFiles ) {
68    //deserialize parse
69    val doc =
    Serializer.load[edu.arizona.sista.processors.corenlp.CoreNLPDocument](
    file.getPath() )
70
71    //we will write the parse to JSON file
72    var jsonOutput = JObject()
73
74    //output discourse parse as json; others
    https://github.com/clulab/processors/blob/37392ced3a0ebdaf0a7481dccf0ce269
    91820dba/src/main/scala/edu/arizona/sista/processors/visualizer/DiscourseP
    arserRunner.scala
75    val jsonDiscourseTree = doc.discourseTree map { dTree => dTreeJSON(
    dTree )} // { dTree => dTree.visualizerJSON()}
76    jsonOutput = jsonOutput merge JObject(JField("discourse",
    jsonDiscourseTree ))
77
78    //coref as json
79    var chainCorefJson = ListBuffer[JObject]()
80    var chainId = 0
81    doc.coreferenceChains.foreach(chains => {
82      for (chain <- chains.getChains) {
83        //random color for each chain
84        //val color = new Color( (Math.random() *
    0x1000000).asInstanceOf[Int])
85        //val colorString = "rgb(" + color.getRed() + "," + color.getGreen()
    + "," + color.getBlue() + ")"
```

```
89          //hack: hope that text is distinctive; store associated color; we don't have indices in dTree json
90          val text = doc.sentences(mention.sentenceIndex).words.slice(mention.startOffset, mention.endOffset).mkString( " " )
91          //json = json merge JObject(JField("color", JString(colorString)))
92          json = json merge JObject(JField("sentence", JInt(mention.sentenceIndex)))
93          json = json merge JObject(JField("head", JInt(mention.headIndex)))
94          json = json merge JObject(JField("start", JInt(mention.startOffset)))
95          json = json merge JObject(JField("end", JInt(mention.endOffset)))
96          json = json merge JObject(JField("chainLength", JInt(length)))
97          json = json merge JObject(JField("chainId", JInt(chainId)))
98          json = json merge JObject(JField("text", JString(text)))
99          chainCorefJson += json
100       }
101         chainId += 1
102     }
103   })
104
105   jsonOutput = jsonOutput merge JObject(JField("coreference", chainCorefJson ))
106
107   /*
108   //TODO add SRL with MatePlus
109   //output srl as json
110 var srlJson = new ListBuffer[ListBuffer[JObject]]()
111 var sentenceCount = 0
112 for (sentence <- srlDoc.sentences) {
113   srlJson += new ListBuffer[JObject]()
114   sentence.semanticRoles.foreach(dependencies => {
115     val iterator = new DirectedGraphEdgeIterator[String](dependencies)
116     while(iterator.hasNext) {
117       val dep = iterator.next
118       var json = JObject()
119       json = json merge JObject(JField("sentence", JInt(sentenceCount)))
120       json = json merge JObject(JField("head", JInt(dep._1)))
121       json = json merge JObject(JField("token", JInt(dep._2)))
122       json = json merge JObject(JField("label", JString(dep._3)))
123       srlJson(sentenceCount) += json
124     }
125   })
126
127   sentenceCount += 1
128 }
129 */
130
131   //word level as json
132   var wordJson = new ListBuffer[ListBuffer[JObject]]()
133   var sentenceCount = 0
```

```
137       var json = JObject()
138       json = json merge JObject(JField("token",
    JString(sentence.words(i))))
139       json = json merge JObject(JField("lemma",
    JString(safeGet(sentence.lemmas,i))))
140       json = json merge JObject(JField("tag", JString(
    safeGet(sentence.tags,i))))
141       json = json merge JObject(JField("entity", JString(
    safeGet(sentence.entities,i))))
142       wordJson( sentenceCount ) += json
143     }
144     sentenceCount += 1
145   }
146
147   jsonOutput = jsonOutput merge JObject(JField("word", wordJson ))
148
149   //syntactic dep as json
150   var synJson = new ListBuffer[ListBuffer[JObject]]()
151   sentenceCount = 0
152   for (sentence <- doc.sentences) {
153     synJson += new ListBuffer[JObject]()
154     sentence.dependencies.foreach(dependencies => {
155       val iterator = new DirectedGraphEdgeIterator[String](dependencies)
156       while(iterator.hasNext) {
157         val dep = iterator.next
158         var json = JObject()
159         json = json merge JObject(JField("sentence", JInt(sentenceCount)))
160         json = json merge JObject(JField("head", JInt(dep._1)))
161         json = json merge JObject(JField("dependent", JInt(dep._2)))
162         json = json merge JObject(JField("label", JString(dep._3)))
163         synJson(sentenceCount) += json
164       }
165     })
166
167     sentenceCount += 1
168   }
169
170   jsonOutput = jsonOutput merge JObject(JField("dependencies", synJson ))
171
172   //write the combined object of all features for this session Id to file
173   //NOTE: this will require some post processing to make features
174   val pw = new PrintWriter(new File(  file.getPath() + ".json" ))
175   pw.write( pretty(render(jsonOutput)) )
176   pw.close
177 }
```

Run

F# FSharp

```
type DiscourseTree = {
  kind : string
  relLabel : string
  relDir : string
  sentence : int
  firstToken : int
  lastToken : int
  text : string
  kids : DiscourseTree array
}

type Coreference = {
  sentence : int
  head : int
  start : int
  ``end`` : int
  chainLength : int
  chainId : int
  text : string
}

type Word = {
  token : string
  lemma : string
  tag : string
  entity : string
}

type Dependency = {
  sentence : int
  head : int
  dependent : int
  label : string
}

type CluResult = {
  discourse : DiscourseTree
  coreference : Coreference array
  word : Word array array
  dependencies: Dependency array array
}

type PageParagraphSentenceIndex = {
    page : int array
    paragraph : int array
    sentence : int array
```

**Fsharp clean up Scala JSON and output final objects we will use**

F# **FSharp**

```
///Trivially we can traverse the tree and map to sentence indices. However, since kind is broken, we must generate a new tree during traversal with kind corrected
let MapDiscourseTreeToSentences( clu : CluResult ) =
  let sentenceToTreeMap = System.Collections.Generic.Dictionary<int,ResizeArray<DiscourseTree>>()

  let rec discourseDFS( tree:DiscourseTree) (isNucleus:bool) =
    let kind =
        if isNucleus then
            "nucleus"
        else
            "satellite"

    //for now, only map leaves. This excludes the structural information, e.g. relLabel
    //if we wanted we could track the rellabel path to the leaf; see class5 code for example
    if tree.text <> null then
        if sentenceToTreeMap.ContainsKey( tree.sentence ) |> not then sentenceToTreeMap.Add( tree.sentence, new ResizeArray<DiscourseTree>() )
        sentenceToTreeMap.[tree.sentence].Add( { tree with kind = kind } )

    if tree.kids <> null then
      discourseDFS tree.kids.[0] (tree.relDir = "LeftToRight")
      discourseDFS tree.kids.[1] (tree.relDir = "RightToLeft")

  //kick off traversal of tree
  discourseDFS clu.discourse true

  //return the map
  sentenceToTreeMap

///Clu returns a mess of coref. Index by sentence and chain
let MapCorefToSentencesAndChains( clu : CluResult ) =
```

```
33          |> Map.ofSeq
34
35      let chainToCorefMap =
36          clu.coreference
37          |> Seq.groupBy( fun x -> x.chainId )
38          |> Map.ofSeq
39
40      //
41      sentenceToCorefMap,chainToCorefMap
42
43
44  let SerializeToJson (filePath:string) ( o ) =
45      let json = Newtonsoft.Json.JsonConvert.SerializeObject(o,
    Newtonsoft.Json.Formatting.Indented )
46      System.IO.File.WriteAllText( filePath, json, System.Text.Encoding.UTF8
    )
47
48  //-------------
49
50  let xhtmlDirectory =
    @"/z/aolney/research_projects/braintrust/materials/NEETS/xhtml"
51
52  //builds pretty html and json in preparation of parsing
53  //TraverseHTML (xhtmlDirectory)
54
55  //after parsing, read in JSON from parser and build f# structures
56  let outputList = new ResizeArray<string>()
57
58  let jsonFiles = System.IO.Directory.GetFiles(xhtmlDirectory,"*.ser.json",
    System.IO.SearchOption.AllDirectories)
59
60  for jsonFile in jsonFiles do
61    //load up parse
62    let clu = Newtonsoft.Json.JsonConvert.DeserializeObject<CluResult>(
    System.IO.File.ReadAllText( jsonFile ) )
63
64    //linearize the discourse tree, align with sentences
65    let discourseBySentences = MapDiscourseTreeToSentences clu
66    SerializeToJson (jsonFile.Replace(".toparse.ser.json",".discourse-
    sentence"))  discourseBySentences
67
68    //align coref with sentences; separately align with chains
69    let sentenceCorefMap,chainCorefMap = MapCorefToSentencesAndChains clu
70    SerializeToJson (jsonFile.Replace(".toparse.ser.json",".coref-sentence"))
    sentenceCorefMap
71    SerializeToJson (jsonFile.Replace(".toparse.ser.json",".coref-chain"))
    chainCorefMap
72
73    //now map the parse output to pages (i.e. map sentences to pages
74    let mutable paragraphIndex = 0
75    let mutable lengthAccumulator = 0
```

```
78   for sentenceIndex = 0 to clu.word.Length - 1 do
79       let fauxSentence = clu.word.[sentenceIndex] |> Seq.map( fun x ->
   x.token ) |> String.concat "" //remove spaces b/c of tokenization
80       lengthAccumulator <- lengthAccumulator + fauxSentence.Length
81       sentenceToParagraphArray.[sentenceIndex] <- paragraphIndex
82       if lengthAccumulator >= toParseLines.[paragraphIndex].Length then
83         lengthAccumulator <- 0
84         paragraphIndex <- paragraphIndex + 1
85   //write page, paragraph, sentence index to file
86   let paragraphPageMap = jsonFile.Replace(".toparse.ser.json",".page") |>
   System.IO.File.ReadAllLines |> Array.map System.Int32.Parse
87   let pageArray,paragraphArray,sentenceArray =
88       sentenceToParagraphArray
89       |> Seq.mapi( fun sentence paragraph -> paragraphPageMap.[paragraph],
   paragraph, sentence )
90       |> Seq.toArray
91       |> Array.unzip3
92   let pageParagraphSentenceIndex =
   {page=pageArray;paragraph=paragraphArray;sentence=sentenceArray}
93   SerializeToJson (jsonFile.Replace(".toparse.ser.json",".page-paragraph-
   sentence"))  pageParagraphSentenceIndex
94
```

Run

Insert FSharp Cell | code | text | section