Lab 5: Advanced collection configuration

5.1. Pointing to documents on the web

1. Open up your **webtudor** collection, and in the **Gather** panel inspect the files you dragged into it. The first folder is *englishhistory.net*, which opens up to reveal *tudor*, and so on. The files represent a complete sweep of the pages (and supporting images) that constitute the *Tudor citizens* section of the *englishhistory.net* web site. They were downloaded from the web in a way that preserved the structure of the original site. This allows any page's original URL to be reconstructed from the folder hierarchy.

2. In the **Design** panel, select the **Document Plugins** section, then select the **plugin HTMLPlug** line and click <**Configure Plugin...**>. A popup window appears. Locate the **file_is_url** option (about halfway down the first block of items) and switch it on. While you are there, switch off the **smart_block** option so that stray images are not processed. Click <**OK**>.

   Setting this option to the **HTMLPlug** means that Greenstone sets an additional piece of metadata for each document called **URL**, which gives its original URL.

   It is important that the files gathered in the collection start with the web domain name (*englishhistory.net* in this case). The conversion process will not work if you dragged over a subfolder, for example the *tudor* folder, because this will set **URL** metadata to something like

   > http://tudor/citizens/...

   rather than

   > http://englishhistory.net/tudor/citizens/...

   If you have copied over a subfolder previously, delete it and make a fresh copy. Drag the folder in the right-hand side of the **Gather**panel on to the trash can in the lower right corner. Then obtain a fresh copy of the files by dragging across the *englishhistory.net*folder from the **Downloaded Files**folder on the left-hand side.

3. To make use of the new URL metadata, the icon link must be changed to serve up the original URL rather than the copy stored in the digital library. Go to the **Format** panel, select the **Format Features** section and edit the **VList** format statement by replacing

   ```
   [link][icon][/link]
   ```

   with

   ```
   [weblink][webicon][/weblink]
   ```

4. Switch to the **Create** panel and **build** and **preview** the collection. Note that the document icons have changed. The collection behaves exactly as before, except that when you click a document icon your web browser retrieves the original document from the web (assuming it is still there by the time you do this exercise!). If you are working offline you will be unable to retrieve the document.

5.2. Enhanced PDF handling

Greenstone converts PDF files to HTML using third-party software: *pdftohtml.pl*. This lets users view these documents even if they don't have the PDF software installed. Unfortunately, sometimes the formatting of the resulting HTML files is not so good.

This exercise explores some extra options to the PDF plugin which may produce a nicer version for display. Some of these options use the standard pdftohtml program, others use ImageMagick and Ghostscript to convert the file to a series of images. Ghostscript is a program that can convert Postscript and PDF files to other formats. You can download it from http://www.cs.wisc.edu/~ghost/ (follow the link to the current stable release).

1. In the Librarian Interface, start a new collection called "PDF collection" and base it on **-- New Collection --**.

   In the **Gather** panel, drag just the PDF documents from *sample_files → Word_and_PDF → Documents* into the new collection. Also drag in the PDF documents from *sample_files → Word_and_PDF → difficult_pdf*.

   Go to the **Create** panel and build the collection. Examine the output from the build process. You will notice that one of the documents could not be processed. The following messages are shown: "The file pdf05-notext.pdf was recognised but could not be processed by any plugin.", and "3 were processed and included in the collection. 1 was rejected".

2. Preview the collection and view the documents. *pdf05-notext.pdf* does not appear as it could not be processed. *pdf06-weirdchars.pdf* was processed but looks very strange. The other PDF documents appear as one long document, with no sections.

Modes in the Librarian Interface

*The Librarian Interface can operate in different modes. The default mode is **Librarian** mode. We can use **Expert** mode to work out why the pdf file could not be processed.*

3. Use the **Preferences...** item on the **File** menu to switch to **Expert** mode and then build the collection again. The **Create** panel looks different in **Expert** mode because it gives more options: locate the **<Build Collection>** button, near the bottom of the window, and click it. Now a message appears saying that the file could not be processed, and why. Amongst all the output, we get the following message: "Error: PDF contains no extractable text. Could not convert pdf05-notext.pdf to HTML format". pdftohtml.pl cannot convert a PDF file to HTML if the PDF file has no extractable text.

4. We recommend that you switch back to **Librarian** mode for subsequent exercises, to avoid confusion.

Splitting PDFs into sections

5. In the **Document Plugins** section of the **Design** panel, configure **PDFPlug**. Switch on the **use_sections** option.

In the **Search Indexes** section, check the **section** checkbox to build the indexes on section level as well as document level.

Build and **preview** the collection. View the text versions of some of the PDF documents. Note that these are now split into a series of pages, and a "go to page" box is provided. The format is still a bit ugly though, and pdf05-notext.pdf is still not processed.

Using image format

6. If conversion to HTML doesn't produce the result you like, PDF documents can be converted to a series of images, one per page. This requires ImageMagick and Ghostscript to be installed.

7. In the **Document Plugins** section, configure **PDFPlug**. Set the **convert_to** option to one of the image types, e.g. **pagedimg_jpg**. Switch off the **use_sections** option, as it is not used with image conversion.

8. **Build** the collection and **preview**. All PDF documents (including pdf05-notext.pdf) have been processed and divided into sections, but each section displays "This document has no text.". For the conversion to images for PDF documents, no text is extracted.

9. In order to view the documents properly, you will need to modify the format statement. In the **Format Features** section on the **Format** panel, select the **DocumentText** format statement. Replace

   ```
   [Text]
   ```

   with

   ```
   [srcicon]
   ```

10. Preview the collection. Images from the document are now displayed instead of the extracted text. Both *pdf05-notext.pdf* and *pdf06-weirdchars.pdf* display nicely now.

    In this collection, we only have PDF documents and they have all been converted to images. If we had other document types in the collection, we should use a different format statement, such as:

    ```
    {If}{[parent:FileFormat] eq PDF,[srcicon],[Text]}
    ```

    ***FileFormat** is an extracted metadata item which shows the format of the source document. We can use this to test whether the documents are PDF or not: for PDF documents, display [srcicon], for other documents, display [Text].*

Using process_exp to control document processing (advanced)

11. Processing all of the PDF documents using an image type may not give the best result for your collection. The images will look nice, but as no text is extracted, searching the full text will not be available for these documents. The best solution would be to process most of the PDF files as HTML, and only use the image format where HTML doesn't work.

12. We achieve this by putting the problem files into a separate folder, and adding another **PDFPlug** plugin with different options.

13. Go to the **Gather** panel. Make a new folder called "notext": right click in the collection panel and select **New folder** from the menu. Change the **Folder Name** to "notext", and click **<OK>**.

    Move the two pdf files that have problems with html (*pdf05-notext.pdf* and *pdf06-weirdchars*.pdf) into this folder by drag and drop. We will set up the plugins so that PDF files in this *notext* folder are processed differently to the other PDF files.

14. Change to **Library Systems Specialist** mode so that you can add two of the same plugin, and use regular expressions in the plugin options (**File → Preferences... → Mode**).

    For version 2.71, you'll need to close GLI now then restart it to get the list of plugins to update properly.

15. Switch to the **Document Plugins** section of the **Design** panel. Add a second PDF plugin by selecting **PDFPlug** from the **Select plugin to add:** drop-down list, and clicking **<Add Plugin...>**. This plugin will come after the first PDF plugin, so we configure it to process PDF documents as HTML. Set the **convert_to** option to **html**, and switch on the **use_sections** option. Click **<OK>**.

16. Configure the first PDF plugin, and set the **process_exp** option to '**notext.*\.pdf**'.

17. The two PDF plugins should have options like the following:

    ```
    plugin PDFPlug -convert_to pagedimg_jpg -process_exp "notext.*\.pdf"
    plugin PDFPlug -convert_to html -use_sections
    ```

    The *paged_img* version must come earlier in the list than the *html* version. The **process_exp** for the first **PDFPlug** will process any PDF files in the *notext* directory. The second **PDFPlug** will process any PDF files that are not processed by the first one.

    Note that all plugins have the **process_exp** option, and this can be used to customize which documents are processed by which plugin. This option is only visible in **Library Systems Specialist** and **Expert** modes.

    Change back to **Librarian** mode.

18. Edit the **DocumentText** format statement. PDF files processed as HTML will not have images to display, so we need to make sure they get text displayed instead. Change `[srcicon]` to `{If} {[NoText] eq "1",[srcicon],[Text]}`.

19. Build and preview the collection. All PDF documents should look relatively nice. Try searching this collection. You will be able to search for the PDFs that were converted to HTML (try e.g. "bibliography"), but not the ones that were converted to images (try searching for "FAO" or "METS").

Opening PDF files with query terms highlighted

20. Next we'll customize the **SearchVList** format statement to highlight the query terms in a PDF file when it is opened from the search result list. This requires Acrobat Reader 7.0 version or higher, and currently only works on a Microsoft Windows platform.

21. The search terms are kept in the macro variable **_cgiargq_**, and we append **#search="_cgiargq_"** to the end of a PDF file link to pass the query terms to the PDF file.

    PDFPlug renames each PDF file as **doc.pdf** and saves it in a unique directory for that document, so we use

    ```
    _httpcollection_/index/assoc/[archivedir]/doc.pdf
    ```

    to refer to the PDF source file. (However, if you used the **-keep_original_filename** option to **PDFPlug** when building the collection, the original name of the PDF file is kept, and we use

    ```
    _httpcollection_/index/assoc/[archivedir]/[Source]
    ```

    instead to locate the PDF source file.)

22. Select **SearchVList** from the list of assigned formats. We need to test whether the file is a PDF file before linking to doc.pdf, using `{If}{[ex.FileFormat] eq 'PDF',,}`. For PDF files, we use the above format instead of the `[ex.srclink]` and `[ex./srclink]` variables to link to the file.

    The resulting format statement is:

    ```
    <td valign="top">[link][icon][/link]</td>
    <td valign="top">{If}{[ex.FileFormat] eq 'PDF', <a href=
    \"_httpcollection_/index/assoc/[archivedir]/doc.pdf#search=&quot;
    _cgiargq_&quot;\">[ex.srcicon]</a>,
    [ex.srclink][ex.srcicon][ex./srclink]}</td>
    <td valign="top">[highlight]
    {Or}{[dc.Title],[ex.Title],Untitled}
    [/highlight]{If}{[ex.Source],<br><i>([ex.Source])</i>}</td>
    ```

    When the PDF icons are clicked in the search results, Acrobat will open the file with the search window open, and the query terms highlighted.

5.3. Enhanced Word document handling

The standard way Greenstone processes Word documents is to convert them to HTML format using a third-party program, wvWare. This sometimes doesn't do a very good job of conversion. If you are using Windows, and have Microsoft Word installed, you can take advantage of Windows native scripting to do a better job of conversion. If the original document was hierarchically structured using Word styles, these can be used to structure the resulting HTML. Word document properties can also be extracted as metadata.

1. In your digital library, preview the **reports** collection. Look at the HTML versions of the Word documents and notice how they have no structure-they have been converted to flat documents.

Using Windows native scripting

2. In the Librarian Interface, open up the **reports** collection. Switch to the **Design** panel and select the **Document Plugins** section on the left-hand side. Double click the **WordPlug** plugin and switch on the **windows_scripting** option.

   In the **Search Indexes** section, check the **section** checkbox to build the indexes on section level as well as document level.

3. **Build** the collection. You will notice that the Microsoft Word program is started up for each Word document—the document is saved as HTML from Word itself, to get a better conversion. **Preview** the collection. In the **Titles** list, notice that *word03.doc* and *word06.doc* now have a book icon, rather than a page icon. These now appear with hierarchical structure.

   The default behaviour for **WordPlug** with **windows_scripting** is to section the document based on "Heading 1", "Heading 2", "Heading 3" styles. If you open up the *word03.doc* or *word06.doc* documents in Word, you will see that the sections use these Heading styles.

   Note, to view style information in Word, you can select **Format → Styles and Formatting** from the menu, and a side bar will appear on the right hand side. Click on a section heading and the formatting information will be displayed in this side bar.

4. Some of the documents do not use styles (e.g. *word01.doc*) and no structure can be extracted from them. Some documents use user-defined styles. **WordPlug** can be configured to use these styles instead of Heading 1, Heading 2 etc. Next we will configure **WordPlug** to use the styles found in *word05.doc*.

Modes in the Librarian Interface

5. The Librarian Interface can operate in four modes. Go to **File → Preferences... → Mode** and see the four modes and what functionality they provide access to. **Librarian** is the default mode.

6. Change the mode to **Library Systems Specialist** because you will need to use regular expressions to set up the style options in the next part of the exercise.

Defining styles

7. Open up *word05.doc* in Word (by double-clicking on it in the **Gather** pane), and examine the title and section heading styles. You will see that various user-defined header styles are set such as:

    - *ManualTitle*: Title of the manual
    - *ChapterTitle*: Level 1 section heading
    - *SectionHeading*: Level 2 section heading
    - *SubsectionHeading*: Level 3 section heading
    - *AppendixTitle*: Appendix section title

8. In the **Document Plugins** section of the **Design** panel, select **WordPlug** and click <**Configure Plugin...**>. Four types of header can be set which are:

    - `level1_header (level1Header1|level1Header2|...)`
    - `level2_header (level2Header1|level2Header2|...)`
    - `level3_header (level3Header1|level3Header2|...)`
    - `title_header (titleHeader1|titleHeader2|...)`

    These header options define which styles should be considered as title, level 1, level 2 and level 3 styles.

    Ensure that the **windows_scripting** option is checked, and set the options as follows (spaces in the Word styles are removed when converting to HTML styles, and these options must match the HTML styles):

    ```
    level1_header:(ChapterTitle|AppendixTitle)
    level2_header: SectionHeading
    level3_header: SubsectionHeading
    title_header: ManualTitle
    ```

    *If you can't see these options in the **WordPlug** configuration pane, check that you are in **Library Systems Specialist** mode as described above.*

    Once these are set, click <**OK**>.

9. Close any documents that are still open in Word, as this can prevent the build process from completing correctly.

10. **Build** the collection and **preview** it. Look in particular at *word05.doc*. You will see that this document is now also hierarchically structured.

    If you have documents with different formatting styles, you can use `(...|...)` to specify all of the different styles.

Removing pre-defined table of contents

11. If you look at the HTML versions of *word05.doc* and *word06.doc*, you will see that it now has two tables of contents. One is generated by Greenstone based on the document's styles, the other was already defined in the Word document. WordPlug can be configured to remove predefined tables of contents and tables of figures. The tables must be defined with Word styles in order for this to work.

12. To remove the tables of contents and figures from *word06.doc* and the table of contents from *word05*.

doc, switch on the **delete_toc** option in **WordPlug**. Set the **toc_header** option to `(MsoToc1|MsoToc2|MsoToc3|MsoTof|TOA)`. In this document, the table of contents and list of figures use these four style names. Click <**OK**>.

13. **Build** and **preview** the collection. Both *word05.doc* and *word06.doc* should now have only one table of contents.

14. Switch the Librarian Interface back to **Librarian** mode (**File → Preferences... → Mode**).

Extracting document properties as metadata

15. When the **windows_scripting** option is set, word document properties can be extracted as metadata. By default, only the Title will be extracted. Other properties can be extracted using the **metadata_fields** option.

16. In the **Enrich** panel, look at the metadata that has been extracted for *word05.doc* and *word06.doc*. Now open the documents in Word and look at what properties have been set (**File → Properties**). They have Title, Author, Subject, and Keywords properties. **WordPlug** can be configured to look for these properties and extract them.

17. In the **Design** panel, under **Document Plugins**, configure **WordPlug** once again. Switch on the configuration option **metadata_fields**. Set the value to

    ```
    Title,Author<Creator>,Subject,Keywords<Subject>
    ```

    This will make **WordPlug** try to extract Title, Author, Subject and Keywords metadata. Title and Subject will be saved with the same name, while Author will be saved as Creator metadata, and Keywords as Subject metadata.

18. Make sure you have closed all the documents that were opened, then **rebuild** the collection.

19. Look at the metadata for the two documents again in the **Enrich** panel. You should now see ex.Creator and ex.Subject metadata items. This metadata can now be used in display or browsing classifiers etc.

5.4. Section tagging for HTML documents

1. In a browser, take a look at the Greenstone demo collection. Browse to one of the documents. This collection is based on HTML files, but they appear structured in the collection. This is because these HTML files were tagged by hand into sections.

2. Using a text editor (e.g. WordPad) open up one of the HTML files from the demo collection: *Greenstone → collect → demo → import → fb33fe →fb33fe.htm*. You will see some HTML comments which contain section information for Greenstone. They look like:

```
<!--
<Section>
  <Description>
    <Metadata name="Title">Farming snails 1: Learning about snails;
    Building a pen; Food and shelter plants</Metadata>
  </Description>
-->

<!--
</Section>
<Section>
  <Description>
    <Metadata name="Title">Dew and rain</Metadata>
  </Description>
-->
```

   When Greenstone encounters a `<Section>` tag in one of these comments, it will start a new subsection of the document. This will be closed when a `</Section>` tag is encountered. Metadata can also be added for each section—in this case, **Title** metadata has been added for each section. In the browser, find the **Farming snails 1** document in the demo collection (through the *Titles* browser). Look at its table of contents and compare it to the `<Section>` tags in the HTML document.

3. Add a new Section into this document. For example, lets add a new subsection into the **Introduction** chapter. In the text editor, add the following just after the Section tag for the **Introduction** section:

```
<!--
<Section>
  <Description>
    <Metadata name="Title">Snails are good to eat.</Metadata>
  </Description>
-->
```

   Then just before the next section tag (**What do you need to start?**), add the following:

```
<!--
</Section>
-->
```

   The effect of these changes is to make a new subsection inside the **Introduction** chapter.

4. Open the Greenstone demo collection in the Librarian Interface. In the **Document Plugins** section of

the **Design** panel, note that **HTMLPlug** has the **description_tags** option set. This option is needed when `<Section>` tags are used in the source documents.

5. **Build** and **preview** the collection. Look at the **Farming snails 1** document again and check that your new section has been added.