

Greenstone tutorial exercises (2023)

Modified for Greenstone version: 3.11

If you are working from a Greenstone Tutorial CD-ROM, DVD or USB flash drive, the sample files for these exercises are in the folder `sample_files`; otherwise they can be downloaded from [sourceforge](#).

The text sometimes uses Windows terminology, but the exercises work equally well on other systems if you make appropriate changes to the pathnames.

[Building a small collection of HTML files](#)

- Running the Greenstone Librarian Interface
- Starting a new collection
- Adding documents to the collection
- Building the collection
- Viewing the extracted metadata
- Viewing the internal links and external links
- Setting up a shortcut in the Librarian interface

[A simple image collection](#)

- Adding Title and Description metadata
- Change Format Features to display new metadata
- Changing the size of image thumbnails
- Adding a browsing classifier based on Description metadata
- Creating a searchable index based on Description metadata

[An image collection with GPS metadata](#)

- Extracting embedded metadata
- Adding in a map view to browsing
- Adding in a map view to a document

[A collection of Word and PDF files](#)

- Viewing the extracted metadata
- Manually adding metadata to documents in a collection
- Document Plugins
- Search indexes
- Browsing classifiers

[Formatting the Word and PDF collection](#)

- Tidying up the default format statement
- Linking to the Greenstone version or original version of documents
- Making bookshelves show how many items they contain
- Displaying multi-valued metadata
- Advanced multi-valued metadata

[Enhanced PDF handling](#)

- Using image format
- Using **process_exp** to control document processing (advanced)
- Customising the table of contents section heading display
- Opening PDF files with query terms highlighted

[Enhanced Word document handling](#)

- Using Windows native scripting
- Modes in the Librarian Interface
- Defining styles
- Removing pre-defined table of contents
- Extracting document properties as metadata
- Processing docx files

[Associated files: combining different versions of the same document together](#)

- Associating one document with another

Linking to associated documents

[A large collection of HTML files—Tudor](#)

Extracting more metadata from the HTML

Looking at different views of the files in the **Gather** and **Enrich** panels

[Enhanced collection of HTML files—Tudor](#)

Adding hierarchically-structured metadata and a **Hierarchy** classifier

Partitioning the full-text index based on metadata values

Controlling the building process

[Formatting the HTML collection—Tudor](#)

[Section tagging for HTML documents](#)

[Downloading files from the web](#)

[Pointing to documents on the web](#)

[Bibliographic collection](#)

Using fielded searching

Exploding the database

Reformatting the collection to use the exploded metadata

[CDS/ISIS collection](#)

[Looking at a multimedia collection](#)

[Building a multimedia collection](#)

Manually correcting metadata

Browsing by media type

Using switch statements

Using **AZCompactList** rather than **List**

Making bookshelves show how many items they contain

Branding the collection with an image

Using **UnknownPlugin**

Cleaning up a title browser using regular expressions

Using different icons for different media types

Building a full-size version of the collection

[Scanned image collection](#)

Grouping documents by series title and displaying dates within each group

Browsing documents by Date.

Searching at page level

[Advanced scanned image collection](#)

Adding another newspaper to the collection

XML based item file

[Open Archives Initiative \(OAI\) collection](#)

Tweaking the presentation with format statements

[Setting up your Greenstone OAI Server](#)

Validating the Greenstone OAI server

[Downloading over OAI](#)

Downloading using the Librarian Interface

Downloading using the command line

Building the downloaded documents in GLI

[Using the UnknownConverterPlugin to make unsupported document formats searchable](#)

Working with DjVu documents in Greenstone

Extracting the text from DjVu documents with DjVuLibre's djvutxt

Processing DjVu documents with the UnknownConverterPlugin

Associating an icon with DjVu documents in Greenstone

[Use METS as Greenstone's Internal Representation](#)

[Moving a collection from DSpace to Greenstone](#)

Adding indexing and browsing capabilities to match DSpace's

[Moving a collection from Greenstone to DSpace](#)

Using Greenstone from the command line

[Editing metadata sets](#)

Running GEMS

Creating a new metadata set

Adding a new element to a metadata set

[Building and searching with different indexers](#)

Build with Lucene

Search with Lucene

Build with MGPP

Search with MGPP

Use search mode hotkeys with query term

A quick reference of the search mode hotkeys in MGPP

[Incrementally building a collection using the command line](#)

Incrementally adding some additional new documents to a collection

Incrementally deleting some documents from a collection

Editing a document's text and metadata, and then incrementally rebuilding the collection

Automatic incremental indexing

[Customization: Themes](#)

Using Greenstone Visual Themes

Creating a custom theme using ThemeRoller (advanced)

[Collection-Specific Themes](#)

Creating a custom theme

Setting a collection's theme

[Customizing your home page](#)

Changing the library's home page

Adding the list of collections

Adding a cross-collection search box

Login Links

Adding your library's site name

Changing your library's site name

[Defining Libraries](#)

Exploring libraries

Creating a new site

Defining a new library

Adding and using a new interface

Changing default settings for the Greenstone server and GLI

[Designing a new interface: Part 1](#)

Creating a new interface

Defining a new library

Gathering files

[Designing a new interface: Part 2](#)

Changing the home page content

Changing the HTML header content

[Designing a new interface: Part 3](#)

Examining main.xsl

Adding a navigation bar

Adding functionality to the quick search box

Adding the library name and login links

[Using WebSwing GLI \(Web GLI\)](#)

Creating a user account

Accessing WebSwing GLI: a Greenstone Librarian Interface (GLI) application accessible over your browser

Building a small collection of HTML files

Sample files: [simple_html.zip](#)

Devised for Greenstone version: 2.60|3.06

Modified for Greenstone version: 2.87|3.11

You will need some HTML files, such as those in the `simple_html` folder in `sample_files`.

Running the Greenstone Librarian Interface

1. Start the Greenstone Librarian Interface:

Start → All Programs → Greenstone-3.11 → Greenstone Librarian Interface (GLI)

For current versions of Greenstone, do not install Greenstone into Program Files or any other System area that requires special permissions. Instead, install it into your user area, C:\Users \<username>.

For older versions of Greenstone: If you are using Windows Vista or Windows 7 and have installed Greenstone into C:\Program Files\Greenstone, a User Account Control dialog may appear as you try to start the Greenstone Librarian Interface, click <Yes> to continue.

On Windows 8 and onwards you may see a Windows Firewall message about Java. If you have admin permissions, choose the option "Domain networks, such as a workplace network" then press Allow Access to allow Java to communicate on the selected networks.

After a short pause a startup screen appears, and then after a slightly longer pause the main Greenstone Librarian Interface appears. (A command prompt is also opened in the background.)

On a Mac, one of the ways you can launch GLI is to click on the green `gli` application icon located in your Greenstone installation folder. (On MacOS version 10.14.4, also known as Mojave, when you first run GLI in this manner, you may see a warning that future MacOS versions won't support this application. Dismiss the warning and proceed.)

On linux as well as mac, you can run GLI by using a terminal to go into the Greenstone installation folder and then running the command `./gli/gli.sh` from there.

Starting a new collection

2. Start a new collection within the Librarian Interface:

File → New...

3. You will create a collection based on a few HTML web pages from the Tudor collection.

A window pops up. Fill it out with appropriate values—for example,

Collection title: Small HTML Collection

Description of content: A small collection of HTML pages.

Leave the setting for **Base this collection on:** at its default: -- **New Collection** --, and click <OK>.

4. Next you must gather together the files that will constitute the collection. A suitable set has been prepared ahead of time in `sample_files` → `simple_html` → `html_files`. Using the left-hand side of the Librarian Interface's **Gather** panel, interactively navigate to the `sample_files` → `simple_html` folder.

Adding documents to the collection

- Now drag the *html_files* folder from the left-hand side and drop it on the right. The progress bar at the bottom shows some activity. Gradually, duplicates of all the files will appear in the collection panel. A popup may appear saying that *geov2.js* is an unrecognised filetype and can't be processed by GLI. Tick the checkbox to no longer see this message again.

You can inspect the files that have been copied by double-clicking on the folder in the right-hand side.

- Since this is our first collection, we won't complicate matters by manually assigning metadata or altering the collection's design. Instead we rely on default behaviour. So pass directly to the **Create** panel by clicking its tab.

Building the collection

- To start building the collection, click the **<Build Collection>** button.
- Once the collection has built successfully, a window pops up to confirm this. Click **<OK>**.
- Click the **<Preview Collection>** button to look at the end result. This loads the relevant page into your web browser (starting it up if necessary).

Note: If you're working with a Safari browser and find that pressing the Preview button has no effect, then go into Safari's Preferences → Websites → Pop up Windows menu and set the value for its **localhost** entry to **allow** to prevent Safari from blocking pages you want to Preview (and from blocking web-supported documents you've double-clicked in GLI's Gather or Enrich panels).

Viewing the extracted metadata

- Back in the Librarian Interface, click the **Enrich** tab to view the metadata associated with the documents in the collection.
- Presently there is no manually assigned metadata, but the act of building the collection has extracted metadata from the documents. Double click the *html_files* folder to expand its content. Then single-click *aragon.html* to display all its metadata in the right-hand side of the panel. The initial fields, starting "dc.", are empty. These are Dublin Core metadata fields for manually entered data.
- Use the scroll bar on the extreme right to view the bottom part of the list. There you will see fields starting "ex." that express the extracted metadata: for example **ex.Title**, based on the text within the HTML Title tags, and **ex.Language**, the document's language (represented using the ISO standard 2-letter mnemonic) which Greenstone determines by analyzing the document's text.
- Close the collection by clicking **File** → **Close**. This automatically saves the collection to disk.

Viewing the internal links and external links

- Hyperlinks in a Greenstone collection work like this: If the link is to a document that is also in the collection, clicking it takes you to that document in the collection. If the link is to a document that is *not* in the collection, clicking it takes you to that document on the web.

Go back to the web browser and click the *titles* link near the top of the page. Open the file *boleyn.html* and look for the link to *Katharine of Aragon* (in the 5th paragraph of the *Biography* section). This links to a document inside the collection--*aragon.html*. View this document by clicking the link. For an external link, return to *boleyn.html* and click *letters written by Anne* (in the *Primary Sources* section). This takes you out on to the web.

Setting up a shortcut in the Librarian interface

15. To set up a shortcut to the source files, in the **Gather** panel navigate to the folder in your local file space that contains the files you want to use—in our case, the *sample_files* folder. Select this folder and then right-click it, and choose **Create Shortcut** from the menu. In the **Name** field, enter the name you want the shortcut to have, or accept the default *sample_files*. Click **<OK>**. Close all the folders in the file tree in the left-hand pane, and you will see the shortcut to your source files.

A simple image collection

Sample files: [images.zip](#)

Devised for Greenstone version: 2.60|3.06

Modified for Greenstone version: 2.87|3.11

In this tutorial, we create a new collection that is based on the configuration of another collection.

1. In a file browser, locate the folder *sample_files* → *images* → *image-e*. Copy this entire folder into your *Greenstone* → *web* → *sites* → *localsite* → *collect* folder.
2. In the Librarian Interface, start a new collection (**File** → **New...**) called **backdrop**. Fill out the fields with appropriate information. For **Base this collection on:**, select the item **Simple image collection** from the pull-down menu.

When you base a collection on an existing one, it inherits all the settings of the old one, including which metadata sets (if any) the collection uses.

3. Copy the images provided in *sample_files* → *images* (avoiding the README.TXT file and any folders) into your newly-formed collection.
4. Change to the **Create** panel and **build** the collection.
5. **Preview** the result.
6. Click on **Browse** in the navigation bar to view a list of the photos ordered by filename and presented as a thumbnail accompanied by some basic data about the image. The structure of this collection is the same as **Simple image collection**, but the content is different.
7. Back in the Librarian Interface, change to the **Enrich** panel and view the extracted metadata for *Bear.jpg*.

Adding Title and Description metadata

8. We work with just the first three files (*Bear.jpg*, *Cat.jpg* and *Cheetah.jpg*) to get a flavour of what is possible. First, we need to add the Dublin Core metadata set which is not used in the **Simple image collection** collection. Click the **<Manage Metadata Sets...>** button beneath the Collection file tree. A new window pops up showing the metadata sets used by current collection. Click the **<Add...>** button to bring up another window showing the available metadata sets. Select the "Dublin Core Metadata Element Set" from the list and click **<Add>**. Click **<Close>** to return to the **Enrich** panel.

First, set each file's **dc.Title** field to be the same as its filename but without the filename extension.

Click on *Bear.jpg* so its metadata fields are available, then click on its **dc.Title** field on the right-hand side. Type in **Bear**.

Repeat the process for *Cat.jpg*, *Cheetah.jpg* and so on.

9. Add a description for each image as **dc.Description** metadata.

What description should you enter? To remind yourself of a file's content, the Librarian Interface lets you open files by double-clicking them. It launches the appropriate application based on the filename extension, Word for .doc files, Acrobat for .pdf files and so on.

Double-click *Bear.jpg*: on Windows, the image will normally be displayed by Windows Photo Viewer (although this depends on how your computer has been set up).

Note: If you're working with a Safari browser and find that double-clicking on *Bear.jpg* has no effect, then go to Safari's Preferences → Websites → Pop up Windows menu and set the value for its **localhost** entry to **allow** to prevent Safari from blocking documents you've double-clicked (and from blocking Greenstone collection pages when you press Preview).

Back in the **Enrich** pane, make sure that *Bear.jpg* is selected in the collection tree on the left hand side. Enter the text **Bear in the Rocky Mountains** as the value for the **dc.Description** field.

Repeat this process for *Cat.jpg* and *Cheetah.jpg*, adding a suitable description for each.

- Go to the **Create** panel and click **<Build Collection>**. Once it has finished building, **preview** the collection. You will not notice anything new. That's because we haven't changed the design of the collection to take advantage of the new metadata.

Change Format Features to display new metadata

- Now we customize the collection's appearance. Go to the **Format** panel and select **Format Features** from the left-hand list.

Click on the **browse** Format Feature. Find the section under **documentNode** where it says

```
<td valign="top">
  <gsf:displayText name="ImageName"/><gsf:metadata name="Image"/><br/>
  <gsf:displayText name="Width"/><gsf:metadata name="ImageWidth"/><br/>
  <gsf:displayText name="Height"/><gsf:metadata name="ImageHeight"/><br/>
  <gsf:displayText name="Size"/><gsf:metadata name="ImageSize"/>
</td>
```

Edit the text as follows:

- Change `<gsf:displayText name="ImageName"/>` to `Title:`
- Change `<gsf:metadata name="Image"/>` to `<gsf:metadata name="dc.Title"/>`
- After `<gsf:metadata name="dc.Title"/>
` add `Description: <gsf:metadata name="dc.Description"/>
`

Metadata names are case-sensitive in Greenstone: it is important that you capitalize "Title" and "Description" (and don't capitalize "dc").

- The first substitution alters the fragment of text that appears to the right of the thumbnail image, the second alters the item of metadata that follows it. The addition displays the description after the Title.
- Preview the collection by clicking the **<Preview Collection>** button. When you click on **Browse** in the navigation bar the presentation has changed to "Title: Bear" and so on. Each image's description should appear beside the thumbnail, following the title.

After the first three items, the Title and Description become blank because we have only assigned Dublin Core metadata to these first three. (To get a full listing you would enter all the metadata.)

*Changes in the **Format** panel take place immediately and you can see the result straightaway by clicking the **Preview Collection** button. If you modify anything in the **Gather**, **Enrich** or **Design** panels, you will need to rebuild the collection.*

Changing the size of image thumbnails

- Let's change the size of the thumbnail image and make it smaller. Thumbnail images are created by the **ImagePlugin** plug-in, so we need to access its configuration settings. To do this, switch to the **Design** panel and select **Document Plugins** from the list on the left. Double-click **ImagePlugin** to pop up a window that shows its settings. (Alternatively, select **ImagePlugin** with a single click and

then click **<Configure Plugin...>** further down the screen). Currently most options are off, so standard defaults are used. Select **thumbnailsize**, set it to **50**, and click **<OK>**.

15. **Build** and **preview** the collection.
16. Once you have seen the result of the change, return to the **Design** panel, select the configuration options for **ImagePlugin**, and switch the **thumbnailsize** option off so that the thumbnail reverts to its normal size when the collection is re-built.

Adding a browsing classifier based on Description metadata

17. Now we'll add a new browsing option based on the descriptions. In the **Design** panel, select **Browsing Classifiers** from the left-hand list. Set the menu item for **Select classifier to add** to **List**, then click **<Add Classifier...>**.
18. A window pops up to control the classifier's options. Set the **metadata** option to **dc.Description**. Next, click the **partition_type_within_level** check box and choose **none** from the drop-down list. Click **<OK>**.
19. **Build** the collection, and **preview** it. Choose the new *Description* link that appears in the navigation bar.

*Only three items are shown, because only items with the relevant metadata (**dc.Description** in this case) appear in the list. The original browse list includes all photos in the collection because it is based on **ex.Image**, extracted metadata that reflects an image's filename, which is set for all images in the collection.*

Creating a searchable index based on Description metadata

20. Now we'll add an index so that the collection can be searched by descriptions. Switch to the **Design** panel and select **Search Indexes** from the left-hand list. Click the **<New Index>** button. Select **dc.Description** from the list of metadata to include in the index and click **<Add Index>**. Leave **Indexing Levels** at its default, "document".
21. Switch to the **Create** panel, **build** the collection, then **preview** it. There is now a **Search** button in the navigation bar. As an example, search for the term "bear" in the *dc.Description* index (which is the only index at this point).
22. To change the text that is displayed for the index (*dc.Description*), go to the **Format** panel back in the Librarian Interface. Select **Search** from the left-hand list. This panel allows you to change the text that is displayed on the search form. Set the **Display text** for the "dc.Description" index to "image descriptions" (or other suitable text). Press the **<Preview Collection>** button. In the browser, you should now see your new text appear for the displayed index name in the search form.

Note that if you use text instead of macros in the search metadata display text, you will need to do any translations yourself.

An image collection with GPS metadata

Prerequisite: [A simple image collection](#)

Sample files: [images_gps.zip](#)

Devised for Greenstone version: 3.06

Modified for Greenstone version: 3.11

In this tutorial, we'll be looking at building a collection that takes advantage of the GPS metadata embedded in image files. Using this data, we can plot the images on a map based on where they were taken.

In doing this tutorial, if the maps are not available for viewing at all, you will need to have a Google Maps API key. This is done through <https://console.cloud.google.com/apis> Only for the duration of this tutorial, set up a Google API key and restrict it to `localhost:8383/greenstone3/` (You'll want to disable it again afterward for security purposes.) Having created an API key, follow the remaining instructions given in your Greenstone3 installation's `resources/web/servLets.xml.in` file for the param-name "googlemaps_api_key".*

Note: Even though Google provides a free tier for usage, at the time of writing they still require you to register a credit card with your Google developer account.

1. Create a new collection in GLI called *Images-GPS*. In the **Gather** panel, drag and drop the 4 folders in *sample_files* → *images_gps* from the Workspace view on the left into the Collection view on the right.
2. Since the images are organised by folder, we can easily assign *folder-level* metadata to the images which will help with classifying them. In the **Enrich** panel, select the *eiffel-tower* folder, and in its **dc.Title** field type *Eiffel Tower*. Since this metadata is assigned at folder level, it is inherited as **dc.Title** metadata by all the images in the folder.

When setting folder-level metadata like this, the default setting in GLI is to produce a popup window alerting you to the fact that the assigned metadata will be assigned to all files and sub-folders contained in the selected folder. For this collection, this is what we want, so press *OK* for the action to proceed.

Note: if you prefer this popup not to appear each time you assign folder-level metadata, there is a tick-box in the lower left-hand corner of the popup window ("do not show this warning again") that allows you to control this. If you choose to suppress the popup, then it can be turned back on through *File* → *Preferences* and then clicking on the *Warnings* tab. The various options are alphabetically sorted, with the option that controls this popup: "About to add folder level metadata".

3. Now select each of the remaining folders of images in turn, and assign the appropriate values for their **dc.Title**: *Parc de Luxembourg*, *Musée d'Orsay* and *Panthéon district*.
4. Go to the **Design** panel. In the **Browsing Classifiers** section, choose **AZCompactList** from the **Select classifier to add** dropdown box and press **<Add Classifier...>**. In the configuration dialog that appears, set the metadata field to **dc.Title** and tick the *buttonname* option and set its value to `locations`. This will create a classifier labelled `locations` that groups all images under Eiffel Tower into one bookshelf and similarly creates bookshelves for the other 3 categories. Click **<OK>** to close the plugin configuration dialog.
5. Now go to the **Create** panel and press **Build Collection**.
6. Preview the collection and click the **Locations** tab in the web browser. Expand the bookshelf icons displayed in the web browser to see thumbnail pictures of the photos gathered under the displayed metadata heading. Explore further, and click on a thumbnail photo or two to view larger versions of

the photos in the document view.

Extracting embedded metadata

- Each of these image files has metadata embedded in it—including GPS data—generated by the smartphone when the photo was taken. We can extract this metadata when the collection is built, and in particular, make use of the GPS metadata to provide map-based views of the collection to the user.

In the **Document Plugins** section of the **Design** panel, go down to the **Select plugin to add:** and choose the **EmbeddedMetadataPlugin**. Press the **<Add Plugin...>** button, and then click **<OK>** to add it to the plugin list. Select this plugin in the list, then use the **<Move Up>** button to shift it upwards until it comes just after the **GreenstoneXMLPlugin**.

- Go to the **Create** panel and press **<Build Collection>**.
- Now go to the **Enrich** panel, expand the **eiffel-tower** folder and select the first image. Scroll down to see the metadata extracted during the building process. Among the extracted metadata, you will find several pieces of Latitude and Longitude metadata, which we will be taking advantage of shortly: **ex.Latitude**, **ex.Longitude**, **ex.LatShort**, and **ex.LngShort**.

Adding in a map view to browsing

Greenstone has a map view, based on the Google Maps API, that can make use of this location metadata. The map view can be controlled to appear in different parts of the interface: as part of a collection's search results page; when browsing the collection; when viewing a document. For this view to be operational in a Greenstone, it is necessary for the collection to index the GPS metadata.

- In the **Search Indexes** section of the **Design** panel, press the **<New Index>** button. Scroll down and tick the box for **ex.LatShort** and press **<Add Index>** to create an index on it. In like manner, create an index on **ex.Latitude**. Then another on **ex.LngShort**. And finally one on **ex.Longitude**.
- Now go to the **Create** panel and press **<Build Collection>**.
- To enable the map, go to the **Format Features** section of the **Format** panel. Select the **browse** format feature in the drop down for **Choose Feature**. In the editor below it, enter the following format statement *above* the **documentNode** template:

```
<gsf:option name="mapEnabled" value="true" />
```

- Also in the **Choose Feature** section, select the **searchType** feature. Add **raw** to the comma-separated **searchType** list.
- In the **Format** panel press **<Preview Collection>**, and click on the new browsing classifier (**locations**) and then click on a bookshelf icon (not the plus next to each bookshelf icon). The page that opens up shows a Google map, with the locations of the images in the collection pinpointed on it. The map view can also scroll through all the images, locating each place and associated image in turn.

By adding the `<gsf:option name="mapEnabled" value="true" />` statement to the **browse** format feature, all of the classifiers built will have the map view enabled. It is also possible to activate the map view on individual classifiers, and on the search results page by adding the *mapEnabled* statement to the *search* format feature.

Adding in a map view to a document

- To activate a map view when viewing the document, go to the **Format Features** section of the

Format panel, and select the **display** format feature in the **Choose Feature** dropdown. In the editor below, enter the following format statement *after* the line `<gsf:option name="TOC" value="true" />`:

```
<gsf:option name="mapEnabled" value="true" />
```

16. If you want the map to display nearby documents, then add the `mapEnabled` option to the **search** format feature also.
17. Still in the **Format** panel press the **<Preview Collection>** button, browse or search to locate a document, and then view the document. The page that opens up shows a Google map, shows the location of the document (where the photo was taken), in addition to the screen-sized photo.

A collection of Word and PDF files

Sample files: [Word_and_PDF.zip](#)

Devised for Greenstone version: 2.60|3.06

Modified for Greenstone version: 2.87|3.11

You will need some source files like those in the *sample_files* → *Word_and_PDF* folder.

1. Start a new collection called **reports** (**File** → **New...**) and base it on **-- New Collection --**.
2. Copy all the .doc, .rtf, .pdf and .ps files from *sample_files* → *Word_and_PDF* → *Documents* into the collection. There are 9 files in all: you can select multiple files by clicking on the first one and shift-clicking on the last one, and drag them all across together. (This is the normal technique of multiple selection.)
3. Switch to the **Create** panel, and **build** and **preview** the collection.

Viewing the extracted metadata

4. Again, this collection contains no manually assigned metadata. All the information that appears—title and filename—is extracted automatically from the documents themselves. Because of this the quality of some of the title metadata is suspect.
5. Back in the Librarian Interface, click the **Enrich** tab to view the automatically extracted metadata. You will need to scroll down to see the extracted metadata, which begins with "ex.".
6. Check whether the **ex.Title** metadata is correct for some of the documents by opening them. You can open a document from the Librarian Interface by double clicking on it.
7. The extracted Title metadata for some documents is incorrect. For example, the Titles for *pdf01.pdf* and *word03.doc* (the same document in different formats) have missed out the second line. The Title for *pdf03.pdf* has the wrong text altogether.

Manually adding metadata to documents in a collection

8. In the **Enrich** panel, manually add Dublin Core **dc.Title** metadata to those documents which have incorrect **ex.Title** metadata. Select *word03.doc* and double-click to open it. Copy the title of this document ("Greenstone: A comprehensive open-source digital library software system") and return to the Librarian Interface. Scroll up or down in the metadata table until you can see **dc.Title**. Click in the value box and paste in the metadata.

Note: On unix systems, such as Ubuntu and Mac, Greenstone may not be set up out of the box to launch the default application associated with many file extensions. To be able to double click on a document in GLI and have it open in the associated application you would need to enter the preferred applications to launch various file formats through the *GLI File* → *File Associations* menu's **Edit File Associations** dialog, where you would set the **Launch Command** field to a preferred application to associate with a file extension followed by %1.

If you prefer to use the system default application to open a file format in, Google for the default file open command for your unix operating system (e.g it's *xdg-open* on Ubuntu, and *open* on Mac) and enter this in GLI's **Edit File Associations** dialog as follows: click on a file extension to select it, then you will be able to set the value in the **Launch Command** field for it. In this field, enter the default file open command for the operating system you found, followed by %1. e.g on Ubuntu you would enter the Command *xdg-open %1* as **Launch Command** for each file extension listed. Whereas on Mac, you would enter *open %1* for each file extension. Finally, press the **Add** button to set the **Command** column's value for that extension. Move on to the next file extension you want to set the

Launch Command for, and repeat. If the file extension you want to set the default viewing application for isn't listed, enter the new file extension in the field labelled **for fields ending**, then type its **Launch Command** value (e.g. `open %1` on a Mac) as before, and press the **Add** button again.

9. Now add **dc.Creator** information for the same document. You can add more than one value for the same field: when you press **Enter** in a metadata value field, a new empty field of the same type will be generated. Add each author separately as **dc.Creator** metadata.
10. Close the document (in Microsoft Word) when you have finished copying metadata from it. External programs opened when viewing documents must be closed before building the collection, otherwise errors can occur.
11. Next add **dc.Title** and **dc.Creator** metadata for a few of the other documents.
12. You will notice as you add more values, they appear in the **Existing values for ...** box below the metadata table. If you are adding the same metadata value to more than one document, you can select it from this list. For example, *pdf01.pdf* and *word03.doc* share the same Title; and many documents have common authors.

*If you build and preview your collection at this point, you will see that the **titles** list now shows your new Titles. However, the **dc.Creator** metadata is not displayed. You need to alter the collection design to use this metadata.*

Document Plugins

13. In the Librarian Interface, look at the **Document Plugins** section of the **Design** panel, by clicking on this in the list to the left. Here you can add, configure or remove plugins to be used in the collection. There is no need to remove any plugins, but it will speed up processing a little. In this case we have only Word, PDF, RTF, and PostScript documents, and can remove the **ZIPPlugin**, **TextPlugin**, **HTMLPlugin**, **EmailPlugin**, **PowerPointPlugin**, **ExcelPlugin**, **ImagePlugin**, **ISISPlug**, **NULPlugin** and **OAIPlugin** plugins. To delete a plugin, select it and click **<Remove Plugin>**. **GreenstoneXMLPlugin** is required for any type of source collection and should not be removed.

Search indexes

14. The next step in the **Design** panel is **Search Indexes**. These specify what parts of the collection are searchable (e.g. searching by title and author). Delete the **ex.Source** index, which is not particularly useful, by selecting it and clicking **<>**.
15. By default the titles index (**dc.Title,ex.dc.Title,ex.Title**) includes **dc.Title**, **ex.dc.Title** and **ex.Title**. Searching this index will search **dc.Title**, **ex.dc.Title** and **ex.Title** metadata. If you wanted to restrict searching to just the manually added **dc.Title** metadata, you would edit this index and deselect **ex.dc.Title** and **ex.Title** from the list of metadata.
16. You can add indexes based on any metadata. Add a new index based on **dc.Creator** by clicking **<New Index>**. Select **dc.Creator** in the list of metadata, and click **<Add Index>**.

Browsing classifiers

17. The **Browsing Classifiers** section adds "classifiers," which provide the collection with browsing functions. Go to this section and observe that Greenstone has provided two *List* classifiers, based on **dc.Title;Title** and **ex.Source** metadata. These correspond to the **titles** and **filenames** buttons on the collection's access bar.

Remove the **ex.Source** classifier by selecting it and clicking **<Remove Classifier>**.

18. Now add an **AZCompactList** classifier for **dc.Creator**. Select **AZCompactList** from the **Select classifier to add** drop-down list and click **<Add Classifier...>**. A popup window for **Configuring Arguments** appears. Select **dc.Creator** from the **metadata** drop-down list and click **<OK>**.
19. Switch to the **Create** panel, and **build** the collection.
20. Next, go to the **Format** panel, and select the **Search** section to the left. On the right, set the display text value for **Index: dc.Creator** to

creators

21. Press the **<Preview Collection>** button. Check that all the facilities work properly. There should be three full-text indexes, called *text*, *titles*, and *creators*. The *titles* list should display all the document Titles. The *creators* list should show one bookshelf for each author you have assigned as **dc.Creator**, and clicking on that bookshelf should take you to all the documents they authored.

*The titles list shows all documents which have been assigned **dc.Title** metadata, or have automatically extracted **ex.Title**. For many documents, extracted Titles may be fine, and it is impractical to add the same metadata again as **dc.Title**. Specifying a list of metadata names in the classifier allows us to use both.*

22. If you have already done the [Enhanced Word document handling](#) exercise, some of the documents will have extracted **ex.Creator** metadata, and some will have **dc.Creator**. To use both of these in the Creators classifier, make the **metadata** field read **dc.Creator,ex.Creator**.

Build the collection again and **preview** it. Now extracted Creators should appear in the *creators* list.

We will play around with the format statements and customize the outlook of this collection in the [Formatting the Word and PDF collection](#) exercise.

Formatting the Word and PDF collection

Prerequisite: [A collection of Word and PDF files](#)

Devised for Greenstone version: 2.70w|3.06

Modified for Greenstone version: 2.87|3.11

In this exercise, we play around with the format statements in the Word and PDF collection.

1. Open the **reports** collection in the Librarian Interface and go to the **Format Features** section of the **Format** panel.

Tidying up the default format statement

2. In this part of the exercise, we make the format statement simpler without changing the resulting display.

Greenstone's default format statement is complex because it is designed to produce something reasonable under almost any conditions, and also because for practical reasons it needs to be backwards compatible with legacy collections. For this collection, we don't need all of the complexity.

Make sure that the **Browse** format statement is selected in the list of formats.

An excerpt from the default **Browse** format statement for **documentNode** looks like the following:

```
<td valign="top">
  <gsf:link type="source">
    <gsf:choose-metadata>
      <gsf:metadata name="thumbicon"/>
      <gsf:metadata name="srcicon"/>
    </gsf:choose-metadata>
  </gsf:link>
</td>
```

This format statement is the default used for the **documentNode** vertical lists under classifiers.

```
<gsf:choose-metadata>
  <gsf:metadata name="thumbicon"/>
  <gsf:metadata name="srcicon"/>
</gsf:choose-metadata>
```

chooses *ex.thumbicon* metadata if it's there, otherwise chooses *ex.srcicon* metadata. If neither are present, nothing is displayed. For this collection there is no *ex.thumbicon* metadata so the choice is not needed.

Replace the longer excerpt above with

```
<td valign="top">
  <gsf:link type="source">
    <gsf:metadata name="srcicon"/>
  </gsf:link>
</td>
```

Next edit the **global** format features: there is no *exp.Title* metadata, so remove that element from the following

```
<gsf:choose-metadata>
  <gsf:metadata name="dc.Title"/>
  <gsf:metadata name="exp.Title"/>
  <gsf:metadata name="ex.dc.Title"/>
  <gsf:metadata name="Title"/>
  <gsf:default>Untitled</gsf:default>
</gsf:choose-metadata>
```


Preview the collection to make sure the display hasn't changed. You shouldn't notice any difference when looking at search results, classifiers etc.

Linking to the Greenstone version or original version of documents

- For collections with documents that undergo a conversion process during importing (e.g. Word, PDF, PowerPoint documents, but not text, HTML documents), the original file is stored in the collection along with the converted version. The default **Browse** format statement links to both versions, but the format statement for **Search** links only to the converted version of the original file. That is, this format statement:

```
<td>
  <gsf:link type="document">
    <gsf:icon type="document"/>
  </gsf:link>
</td>
```

links to the Greenstone HTML version, while

```
<td>
  <gsf:link type="source">
    <gsf:metadata name="srcicon"/>
  </gsf:link>
</td>
```

links to the original.

Choose **Search** in **Format Features**. Experiment with removing and restoring either of the two links from the format statement, previewing the effect of each change.

To see the results of your changes, preview the collection and do a search. You are making changes to **documentNodes** under **Search**, which means the changes will only apply to search results.

Storing and displaying the original allows users to see the correct format, but requires the user to have the relevant program installed. It also increases the size of the collection. The Greenstone version can be viewed in a browser, but may not look as nice.

Making bookshelves show how many items they contain

- Next, we'll customize the format statement for the *creators* list. Classifier bookshelves have only a few pieces of metadata to display: *ex.Title* and *numleafdocs*. Whatever metadata the classifier has been built on, the bookshelf label is always stored as *ex.Title*. This is why a Creator is printed out for each bookshelf even though *dc.Creator* is not specified in the format statement.

Make each bookshelf in the Creator classifier show how many entries it contains. In the **Format Features** section of the **Format** panel, select the **Browse** format statement. This consists of three parts: the first `gsf:template` is the format statement defining the display of a **documentNode**, the second one is the format statement that controls the appearance of **VList classifierNodes** (which appear as bookshelves here), while the final `gsf:template` block is the format statement defining the display of **HList classifierNodes**.

Scroll down to the end of the second format statement, which is the one for the **VList** classifiers and appears just before the start of the format statement for **HList** classifiers. Then insert the line highlighted below, which will display the number of leaf documents inside a classifier bookshelf:

```
<gsf:template match="classifierNode[@classifierStyle = 'VList']">
  ...
  <td valign="top"><gsf:metadata name="numleafdocs"/></td>
</gsf:template>
<gsf:template match="classifierNode[@classifierStyle = 'HList']">
  <gsf:link type="classifier">
    <gsf:metadata name="Title"/>
```

```
</gsf:link>
</gsf:template>
```

Preview the collection. Click on the *creators* list and notice that the bookshelves now display how many documents they contain.

This revised format statement has the effect of specifying in brackets how many items are contained within a bookshelf.

Displaying multi-valued metadata

- Next we modify the document entries in the Creator classifier to display all authors. Back in **Format Features**, select the **Browse** format in the list of assigned formats. Edit the format statement for **documentNode** after the part where it displays the Title metadata, so that it now additionally contains the new line highlighted below. This will display the dc.Creator metadata.

```
<td valign="top">
  <gsf:link type="document">
    <xsl:call-template name="choose-title"/>
    <gsf:switch>
      <gsf:metadata name="Source"/>
      <gsf:when test="exists">
        <br/>
        <i>(<gsf:metadata name="Source"/></i>
      </gsf:when>
    </gsf:switch>
  </gsf:link>
  <br/>
  <gsf:metadata name="dc.Creator" />
</td>
```

The format statement as it is above will now display the Greenstone link, the link to the original, then the Title as before. Since it's defined for **documentNodes**, it will display all the Authors (Creators), and the source document for documents. Preview the *creators* list and make sure that all authors are displayed for documents.

The additional line `<gsf:metadata name="dc.Creator" />` displays all the Creator metadata for the document, separated by a comma (", "). All the metadata values for dc.Creator will be returned by default. If you wish to retrieve only the *first*, *last* or *nth* value for a metadata, you would use the *pos* attribute. For example, `<gsf:metadata name="dc.Creator" pos="first"/>` (or alternatively, `<gsf:metadata name="dc.Creator" pos="1"/>`) displays only the first author.

In sectioned documents, you can allow users to browse and search on the section level. In this case, you may want to request metadata for the *parent*, *ancestors*, or *root* (i.e. document) of the current section. For instance, if you want to display the title of the section's parent you can use `<gsf:metadata name="Title" select="parent"/>`.

- You can change the separator between the authors. Modify the format statement, and replace `<gsf:metadata name="dc.Creator" />` with `<gsf:metadata name="dc.Creator" separator=" "/>`. This will add a space after each author. Preview the *creators* list. However, if you want a newline to separate each author, it requires a little more in order to escape the HTML newline (`
`) element:

```
<gsf:metadata name="dc.Creator"><separator><br /></separator></gsf:metadata>
```

If you have done exercise [Enhanced Word document handling](#), the collection will have both dc.Creator and ex.Creator metadata. To display the metadata values for both, you can use

```
<gsf:metadata name="dc.Creator" />, <gsf:metadata name="Creator" />
```

To display dc.Creator if it is present, otherwise display ex.Creator, use

```
<gsf:choose-metadata>
  <gsf:metadata name="dc.Creator" />
```

```
<gsf:metadata name="Creator" />
</gsf:choose-metadata>
```

Advanced multi-valued metadata

7. You may notice that the **AZCompactList** classifier's configuration dialog has two options after the **metadata** option: **firstvalueonly** and **allvalues**. Manually added metadata can be used to replace or enhance automatically extracted metadata, and these options control exactly which pieces of metadata a document is classified by.

For example, say we have two documents. Document 1 has four Creators specified (dc.Creator = dcA, dc.Creator = dcB, ex.Creator = exA, ex.Creator = exB), while document 2 has three (ex.Creator = exA, ex.Creator = exB, ex.Creator = exC). The following table shows which metadata values each document is classified by, for the different classifier options:

<u>AZCompactList options</u>	<u>Document 1</u>	<u>Document 2</u>
-metadata dc.Creator,ex.Creator	dcA, dcB	exA, exB, exC
-metadata dc.Creator,ex.Creator -firstvalueonly	dcA	exA
-metadata dc.Creator,ex.Creator -allvalues	dcA, dcB, exA, exB	exA, exB, exC

8. We'll now set the **firstvalueonly** option for the *creators* classifier. Switch to the **Browsing Classifiers** section of the **Design** panel, select the **AZCompactList** for **dc.Creator** metadata in the **Assigned Classifiers** box and click **<Configure Classifier...>**. Select the **firstvalueonly** option.

Rebuild and **preview** the collection. Now the *creators* list classifies documents based on the first author appearing in the **dc.Creator** metadata.

If you set the **metadata** field of **AZCompactList** to *dc.Creator,ex.Creator* in the [A collection of Word and PDF files](#) exercise, now the *creators* list will classify based on the first author appearing in either the **dc.Creator** metadata or the **ex.Creator** metadata.

Enhanced PDF handling

Sample files: [Word_and_PDF.zip](#)

Devised for Greenstone version: 3.09

Modified for Greenstone version: 3.11

*Prior to Greenstone 3.09, Greenstone shipped with a plugin called **PDFPlugin**. It was the plugin Greenstone used to convert PDF files to HTML using the third-party software `pdftohtml.pl`. **PDFPlugin** allowed users to view PDF documents even if they didn't have the PDF software installed. Unfortunately, sometimes the formatting of the resulting HTML files was not so good. Earlier versions of this tutorial would provide some instruction on extra options to the **PDFPlugin** for producing a nicer version for display. The older `pdftohtml` process could however not cope with much newer versions of PDF unless **PDFPlugin**'s `pdfbox_conversion` option was switched on.*

*Starting with Greenstone 3.09, some older pdf processing functionality has been restructured into **PDFv1Plugin**, while shifting the `pdfbox_conversion` option into **PDFv2Plugin**. **PDFv2Plugin** further makes use of third-party software `xpdf-tools`, which better copes with newer PDFs, thus no longer requiring activating the `pdfbox_conversion` option when dealing with newer PDFs. **PDFv2Plugin** comes with several new preconfigured settings to produce output files in `html`, `text`, `image` or `image and text` formats, that can better reflect the appearance of an input PDF document's pages. Behind the scenes, **PDFv2Plugin** is configured to use the third-party `xpdf-tools` or `pdfbox` software for each output setting.*

*From Greenstone 3.09 onwards, **PDFv2Plugin** is added to a new collection's Document Plugins pipeline by default, in place of the now defunct **PDFPlugin**. In any instance where you particularly prefer the original **PDFPlugin**'s HTML output for a PDF, you can now use **PDFv1Plugin** instead, as it still retains this functionality.*

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`.

In the **Document Plugins** section of the **Design** panel, you should find **PDFv2Plugin** in the plugins list (in place of the deprecated **PDFPlugin** that would have been present in the plugins list in older versions of Greenstone).

Go to the **Create** panel and build the collection. Examine the output from the build process.

If you had built the same collection with **PDFv1Plugin** instead of **PDFv2Plugin**, the build output would inform you that one of the documents could not be processed and you'd have seen the following building messages: "The file `pdf05-notext.pdf` was recognised but could not be processed by any plugin.", and "3 documents were processed and included in the collection. 1 was rejected".

However, since you built the collection of 4 pdfs with **PDFv2Plugin**, you will notice that all 4 documents could be processed.

2. Preview the collection and view the documents. Inspect `pdf01` and `pdf03` first. There's a table of contents is provided to the right. Clicking on a page in the table of contents will scroll to that page. Another way of navigating can be found to the left, where individual pages are listed vertically by page number and clicking the "plus" box next to a page will expand its contents. The pdfs have been sectionalised into groups of 10 pages, each group further containing a section for each individual page. If your pdf contained 10 or fewer pages, there won't be two levels of sectionalising, just one.

If you visit a given page and try to select and copy the text, you can. These are not entirely images of the pdf's pages (like screenshots of a pdf page), but are HTML pages that combine images of the background of each pdf page with the actual text of that page superimposed. The latter is what makes the text selectable.

If you return to GLI's Design pane and double click on PDFv2Plugin in Document Plugins, then you will see that the `convert_to` option is set to `paged_pretty_html`. This is the default PDF `convert_to` type and produces the kind of sectionalised HTML pages consisting of background images and superimposed text that you see with *pdf01* and *pdf03*.

3. Next preview *pdf05-notext.pdf*. This is also similarly sectionalised, but the text is not selectable. That's because the original PDF file *pdf05-notext.pdf* contained no text, only images of text.
4. Now preview *pdf06-weirdchars.pdf*. Although also sectionalised, its contents look very strange. The reason for this will become apparent if you open the original document by double-clicking *pdf06-weirdchars.pdf* in GLI's Gather pane. Then in the open PDF, select as much of the text on its first page as possible. Copy that text and paste it in a text editor. You should see strange characters. This is why Greenstone's PDFv2Plugin wasn't able to extract legible text either.

Although Greenstone has processed all 4 documents, *pdf06-weirdchars.pdf* can be made to look better.

Using image format

5. PDF documents can be converted to a series of images, one per page. This uses the bundled ImageMagick and Ghostscript.
6. In the **Document Plugins** section, configure **PDFv2Plugin**. Set the `convert_to` option to one of the image types, e.g. `pagedimg_jpg`.
7. **Build** the collection and **preview**. All PDF documents have been processed again, still divided into a series of page sections, but this time one image per page. Images from the document are now displayed instead of the extracted text. That means there's no selectable text for any of the 4 documents this time. The table of contents on the right now displays a horizontal scroller containing thumbnails of each page. *pdf06-weirdchars.pdf* displays nicely now.

Using process_exp to control document processing (advanced)

8. 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.
9. We achieve this by putting the problem files into a separate folder, and adding another **PDFv2Plugin** plugin with different options.
10. 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>**.

Note: To see the right click context menu on the Mac, you would hold down the Ctrl key while pressing the (right) mouse button. If attempting to right click on the Mac does not produce any context menu, go into your Mac's Apple menu → System Preferences → Mouse and then tick the **Secondary click** box and then try right clicking the document in GLI as described.

Move *pdf06-weirdchars.pdf* (which has problems with html) and also *pdf05-notext.pdf* (which has no extractable text) 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.

11. Switch to the **Document Plugins** section of the **Design** panel. Add a second instance of **PDFv2Plugin** by selecting **PDFv2Plugin** from the **Select plugin to add:** drop-down list, and clicking **<Add Plugin...>**. This plugin will come after the first PDFv2Plugin instance, so we configure it to process PDF documents as sectionalised HTML by leaving the **convert_to** option on the default, **paged_pretty_html**. Click **<OK>**.
12. Configure the first PDF plugin, and set the **process_exp** option to **"notext.*\pdf"**.
13. The two PDF plugins should have options like the following:

```
plugin PDFv2Plugin -convert_to pagedimg_jpg -process_exp "notext.*\pdf"
plugin PDFv2Plugin -convert_to paged_pretty_html
```

The *paged_img* version must come earlier in the list than the *html* version. The **process_exp** for the first **PDFPlugin** will process any PDF files in the *notext* directory. The second **PDFPlugin** 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.

14. 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").

Customising the table of contents section heading display

15. In the table of contents (on the right), a section number and section title are displayed by default. For documents like these where the section titles are the same as the section numbers, this doesn't make much sense, as you end up with headings like "1 1". We can hide the section number from the display by adding some CSS style information.
16. Click on the **display** format statement in the **Format Features** list. Add the following to the start of the content:

```
<gsf:template name="additionalHeaderContent-collection">
  <style>span.tocSectionNumber { display: none; }</style>
</gsf:template>
```

17. Note that if you'd rather hide the title instead, you can use *span.tocSectionTitle* in the above CSS code instead of *span.tocSectionNumber*.

Opening PDF files with query terms highlighted

18. Next we'll customize the **search** 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 has been confirmed to work with Firefox browsers on Windows, Linux and Mac systems, but is known to not work with Safari browsers at present. Other browsers are as yet untested and may or may not support the PDF query term highlighting syntax used in this exercise.
19. To highlight the query terms in a PDF document, we need to pass them into the PDF file by appending **#search="query"** to the end of the document link. We need to create the link ourselves rather than using `<gsf:link type="source"/>` in the format statement.

PDFPlugin saves each PDF file in a unique directory for that document, and we can use

```
<gsf:metadata name="httpPath" type="collection"/>/index/assoc/<gsf:metadata name="archivedir"/>
/<gsf:metadata name="srclinkFile"/>
```

to refer to the PDF source file. The search terms can be found in the "q" cgi parameter. You can access this using `<gsf:cgi-param name="q"/>`.

20. Select **search** in **Format Features** for editing. We need to test whether the file is a PDF file before linking to it, using a test on whether the Greenstone extracted FileFormat metadata is PDF. For PDF files, we now generate the link explicitly.

The resulting format statement is:

```
<td valign="top">
  <gsf:link type="document">
    <gsf:icon type="document"/>
  </gsf:link>
</td>

<td valign="top">
<gsf:switch>
  <gsf:metadata name="FileFormat"/>
  <gsf:when test="equals" test-value="PDF">
    <a<xsl:attribute name="href"><gsf:metadata name="httpPath" type="collection"/>/index/assoc
/<gsf:metadata name="archivedir"/>/<gsf:metadata name="srclinkFile"/>#search=&quot;<gsf:cgi-param
name="query"/>&quot;</xsl:attribute>
      <gsf:choose-metadata>
        <gsf:metadata name="thumbicon"/>
        <gsf:metadata name="srcicon"/>
      </gsf:choose-metadata>
    </a>
  </gsf:when>
  <gsf:otherwise>
    <gsf:link type="source">
      <gsf:choose-metadata>
        <gsf:metadata name="thumbicon"/>
        <gsf:metadata name="srcicon"/>
      </gsf:choose-metadata>
    </gsf:link>
  </gsf:otherwise>
</gsf:switch>
</td>

<td valign="top">
...
```

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

For example, **Preview** and try searching for `bibliography`. Click on a PDF icon in the search results. The PDF will be opened on the page with the first instance of the word `bibliography`, with the word highlighted.

Enhanced Word document handling

Prerequisite: [A collection of Word and PDF files](#)

Devised for Greenstone version: 2.70w|3.06

Modified for Greenstone version: 2.87|3.11

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 **WordPlugin** plugin and switch on the **windows_scripting** option.

In the **Search Indexes** section, check the **section** checkbox, if not already the case, 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 **WordPlugin** 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 2003, you can select **Format** → **Styles and Formatting** from the menu, and a side bar will appear on the right hand side. (In Word 2007 and later, find the **Change Styles** button on the far right of the menu ribbon. Click on the tiny **Expand** icon to its bottom right to display the styles side bar.) 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. **WordPlugin** can be configured to use these styles instead of Heading 1, Heading 2 etc. Next we will configure **WordPlugin** to use the styles found in *word05.doc*.

Modes in the Librarian Interface

5. The Librarian Interface operates in three modes. Go to **File** → **Preferences...** → **Mode** and see the modes and what functionality they provide access to. **Librarian** is the default mode. Check that this is indeed the currently active mode.

Defining styles

6. 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

7. In the **Document Plugins** section of the **Design** panel, select **WordPlugin** 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 4 header options to the values highlighted in the following (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
```

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

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

10. If you look at the HTML version *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. **WordPlugin** 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.
11. To remove the tables of contents and figures from *word06.doc*, switch on the **delete_toc** option in **WordPlugin**. 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>**.
12. **Build** and **preview** the collection. *word06.doc* should now have only one table of contents.

Extracting document properties as metadata

13. 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.
14. 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** for Word 2003. In Word 2007/2010, click the Word Icon on the top left, then choose **Prepare** → **Properties**. In Word 2013, **File** → **Info**; the Properties section is on the right.) They have Title,

Author, Subject, and Keywords properties. **WordPlugin** can be configured to look for these properties and extract them.

- In the **Design** panel, under **Document Plugins**, configure **WordPlugin** once again. Switch on the configuration option **metadata_fields**. Set the value to the following (but make sure not to enter any trailing spaces)

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

This will make **WordPlugin** 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.

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

Processing docx files

- Drag and drop the *sample_files* → *Word_and_PDF* → *extra_docx* → *testword.docx* file, or any Word doc you have with docx file extension, into the collection. In the **Document Plugins** section of the **Design** panel, use the **<Move Down>** to move the **UnknownConverterPlugin** in the plugins list to below the **WordPlugin** in the document plugin pipeline. **Build** the collection. With **windows_scripting** turned on, docx files, which are the newer version of word documents, will now also be processed during build. **Preview** the collection and have a look at the document view of the newly added word document in the collection, to see what the generated html version of the file looks like. (testword.docx is a very basic docx file, containing a few sentences and an image.)
- Now turn off **windows_scripting** in the **Design** panel, and **rebuild** the collection again. All the documents should still be processed, because Greenstone's document plugin pipeline is now set up with an **UnknownConverterPlugin** configured to use *Apache Tika* to extract text from Word documents by default (including docx files). **Preview** the collection and revisit the document view of the docx file. This time, the html produced should look very different: much more basic. This is because *Tika* supports extracting text from different document formats, including word documents, but is not optimised for html presentation. However, this does mean full text searching will be available for docx files too when Greenstone is installed out-of-the-box.

So at a pinch, you can always use Greenstone's now default document plugins setup, to process a collection that includes docx files, to at least support full text searching of the contents of docx files, even if the document view (the HTML view) of docx files processed with *Tika* may not look as formatted as the original source document. Presentation may be of secondary importance, since by default Greenstone will anyway provide a link to the original source document in its original format (in this case, a link to the docx file).

*Above, we shifted the **UnknownConverterPlugin** that uses *Apache Tika* to below the **WordPlugin** in the document plugin pipeline, because we want to force **WordPlugin** to attempt to process all word documents first, when it recognises them. *Apache Tika* can always process Word documents, but we favour **WordPlugin** to try processing them first, including the newer docx files, which it can do when on Windows machines with Word installed and **windows_scripting** turned on. Turning off **windows_scripting** instructs the **WordPlugin** not to make use of Word to convert doc(x) files to html, and so **WordPlugin** is not able to process docx files. As a result, the document plugins in the pipeline pass the unprocessed docx file further down the pipeline to the **UnknownConverterPlugin** that is able to process the docx file as it's pre-configured to make use of *Apache Tika* to extract text from Word documents.*

Associated files: combining different versions of the same document together

Prerequisite: [A collection of Word and PDF files](#)

Devised for Greenstone version: 2.85|3.06

Modified for Greenstone version: 2.87|3.11

This tutorial demonstrates how to link different versions of the same document together in Greenstone. As an example, two identical articles about Greenstone are used; one is in PDF format, the other in Word.

1. Start a new collection called **Associated Files Example**, by selecting File → New. Enter an appropriate description for your collection.
2. Copy the files pdf01.pdf and word03.doc provided in *sample_files* → *Word_and_PDF* → *Documents* into your new collection. Do this by dragging these files across from the filesystem view on the left of the **Gather** panel into the **Collection view** on the right.
3. In the collection view, right-click on each file and select **Rename**, renaming them greenstone1.pdf and greenstone1.doc, respectively.
4. In the **Enrich** panel, assign appropriate **dc.Title** and **dc.Creator** metadata to the documents. Since the contents are identical, you can select both documents and set metadata for them simultaneously.

Associating one document with another

5. In **Document Plugins**, select the **WordPlugin** and press the **<Configure Plugin...>** button. In the resulting popup, scroll down to find the `associate_ext` option, and set this option to *pdf*. Now, for Word documents, Greenstone will look for documents with the exact same name but the PDF file extension. These PDFs will not be processed separately; instead, they will be associated with their equivalent Word documents. (Alternatively, you could make the PDF document the primary document, by setting the `associate_ext` option in the **PDFPlugin** to *doc*.)
6. Build the collection. Notice that only one document was considered for processing and included in the collection. Since the PDF version of the document is an associated document, it is not processed.

Linking to associated documents

7. Greenstone has internally associated the PDF version with the Word version of the document. However, with the default format statement, the end-user will have no idea that the PDF version exists. The collection built at this point (with default settings) only gives the user the choice of viewing either the Word version or the Greenstone-generated HTML version of the document. They are not given the option to view the PDF version.

To allow users to view the PDF version of the document, edit the **documentNode** template of the **Browse** Format Feature in the **Format** panel, to reference the `equivDocIcon` with a link to the PDF document `equivDocLink` as follows.

Change:

```
<gsf:template match="documentNode">
  <td valign="top">
    <gsf:link type="document">
      <gsf:icon type="document"/>
    </gsf:link>
  </td>
  <td valign="top">
    <gsf:link type="source">
      <gsf:choose-metadata>
```

To:

```
<gsf:template match="documentNode">
  <td valign="top">
    <gsf:link type="document">
      <gsf:icon type="document"/>
    </gsf:link>
  </td>
  <td valign="top">
    <gsf:link type="source">
      <gsf:choose-metadata>
```

```

        <gsf:metadata name="thumbicon"/>
        <gsf:metadata name="srcicon"/>
    </gsf:choose-metadata>
</gsf:link>
</td>


    <td valign="top">
        <gsf:link type="document">
<!--
Defined in the global format statement
-->
        <xsl:call-template name="choose-
title"/>
        <gsf:switch>
        <gsf:metadata name="Source"/>
        <gsf:when test="exists">
            <br/>
            <i><gsf:metadata
name="Source"/></i>
        </gsf:when>
        </gsf:switch>
        </gsf:link>
    </td>
</gsf:template>

        <gsf:metadata name="thumbicon"/>
        <gsf:metadata name="srcicon"/>
    </gsf:choose-metadata>
</gsf:link>
<td valign="top">
        <gsf:metadata name="equivDocLink"/>
        <gsf:metadata name="equivDocIcon"/>
        <gsf:metadata name="/equivDocLink"/>
    </td>
    <td valign="top">
        <gsf:link type="document">
<!--
Defined in the global format statement
-->
        <xsl:call-template name="choose-
title"/>
        <gsf:switch>
        <gsf:metadata name="Source"/>
        <gsf:when test="exists">
            <br/>
            <i><gsf:metadata
name="Source"/></i>
        </gsf:when>
        </gsf:switch>
        </gsf:link>
    </td>
</gsf:template>

```

The above change to the browse format statement, adds the equivalent document icon (a PDF icon in this case) next to the source icon (Word icon) for general classifiers. Preview the collection and browse the collection either by Titles or by Filenames.

Preview the collection.

Note: When Greenstone encounters a file that matches the provided `associate_ext` value (`pdf` in our case), it sets the metadata value **ex.equivDocIcon** for that document to be the macro `_iconXXX_`, where `XXX` is whatever the filename extension is (so `_iconpdf_` in our case). As long as there is an existing macro defined for that combination of the word `icon` and the filename extension, then a suitable icon will be displayed when the document appears in a VList. For `pdf` the displayed icon will be .

8. Go to Format Features → search and you will see:

```

<gsf:template match="documentNode">
    <td valign="top">
        <gsf:link type="document">
            <Tab n="3"/><gsf:icon type="document"/>
        </gsf:link>
    </td>
    <td>
        <gsf:link type="document">
            <xsl:call-template name="choose-title"/>
        </gsf:link>
    </td>
</gsf:template>

```

The above will only display search results where there is a link to the Greenstone generated HTML version of the original source document, followed by the title of the document.

Change the above to:

```

<gsf:template match="documentNode">
    <td valign="top">
        <gsf:link type="document">
            <Tab n="3"/><gsf:icon type="document"/>
        </gsf:link>
    </td>

    <td valign="top">

```

```
<gsf:link type="source">
  <gsf:choose-metadata>
    <gsf:metadata name="thumbicon"/>
    <gsf:metadata name="srcicon"/>
  </gsf:choose-metadata>
</gsf:link>
</td>
<td valign="top">
  <gsf:metadata name="equivDocLink"/>
  <gsf:metadata name="equivDocIcon"/>
  <gsf:metadata name="/equivDocLink"/>
</td>

<td>
  <gsf:link type="document">
    <xsl:call-template name="choose-title"/>
  </gsf:link>
</td>
</gsf:template>
```

Now, following the link to Greenstone's HTML document, there is a link to the source document (the doc file) and a link to its equivalent doc (the equivalent PDF file in our example).

A large collection of HTML files—Tudor

Sample files: [tudor.zip](#)

Devised for Greenstone version: 2.60|3.06

Modified for Greenstone version: 2.87|3.11

You will need the files in the `sample_files` → `tudor` folder.

1. Invoke the Greenstone Librarian Interface (from the Windows *Start* menu) and start a new collection called **tudor** (use the **File** menu), based on the default -- **New Collection --**.
2. In the **Gather** panel, open the *tudor* folder in *sample_files*.
3. Drag *englishhistory.net* from the left-hand side to the right to include it in your **tudor** collection. (This material is from Marilee Hanson's Tudor England Collection at <https://englishhistory.net/tudor/>, distributed with her permission.)
4. Switch to the **Create** panel and click **<Build Collection>**.
5. When building has finished, **preview** the collection.

Extracting more metadata from the HTML

6. The browsing facilities in this collection (**titles** and **filenames**) are based entirely on extracted metadata. Switch to the **Enrich** panel in the Librarian Interface and examine the metadata that has been extracted for some of the files.
7. Many HTML documents contain metadata in `<meta>` tags in the `<head>` of the page. Open up the *englishhistory.net* → *tudor* → *monarchs* → *boleyn.html* file by navigating to it in the tree on the left hand side, and double clicking it. This will open it in a web browser. View the HTML source of the page (**View** → **Source** in Internet Explorer, **Tools** → **Web Developer** → **Page Source** in Mozilla, and press Ctrl+U in Microsoft Edge). You will notice that this page has *page_topic*, *content* and *author* metadata.
8. By default, **HTMLPlugin** only looks for Title metadata. Configure the plugin so that it looks for the other metadata too. Switch to the **Design** panel and select the **Document Plugins** section. Select the **plugin HTMLPlugin** line and click **<Configure Plugin...>**. A popup window appears. Switch on the **metadata_fields** option, and set the value to

Title,Author,Page_topic,Content

Click **<OK>**.

9. Switch to the **Create** panel and **rebuild** the collection. Go back to the **Enrich** panel and look at the extracted metadata for some of the HTML files in *englishhistory.net* → *tudor* → *monarchs*. The new metadata should now be visible.

Looking at different views of the files in the Gather and Enrich panels

10. Switch to the **Gather** panel and on the right-hand side open *englishhistory.net* → *tudor*.
11. Change the **Show Files** menu for the right-hand side from **All Files** to **HTM & HTML**. Notice the files displayed above are filtered accordingly, to show only files of this type.
12. Change the **Show Files** menu to **Images**. Again, the files shown above alter.
13. Now return the **Show Files** setting back to **All Files**, otherwise you may get confused later.

Remember, if the **Gather** or **Enrich** panels do not seem to be showing all your files, this could be the problem.

Enhanced collection of HTML files—Tudor

Prerequisite: [A large collection of HTML files—Tudor](#)

Devised for Greenstone version: 2.60|3.06

Modified for Greenstone version: 2.87|3.11

We return to the Tudor collection and add metadata that expresses a subject hierarchy. Then we build a classifier that exploits it by allowing readers to browse the documents about Monarchs, Relatives, Citizens, and Others separately.

Adding hierarchically-structured metadata and a Hierarchy classifier

1. Open up your **tudor** collection (the original version, not the **webtudor** version, in case you've already done that tutorial), switch to the **Enrich** panel and select the *citizens* folder (a subfolder of *englishhistory.net* → *tudor*). Set its **dc.Subject** metadata to **Tudor period|Citizens**. The vertical bar ("|") is a hierarchy marker. Selecting a *folder* and adding metadata has the effect of setting this metadata value for all files contained in this folder, its subfolders, and so on. A popup alerts you to this fact. Click **<OK>** to close the popup.
2. Repeat for the *monarchs* and *relative* folders, setting their **dc.Subject** metadata to **Tudor period|Monarchs** and **Tudor period|Relatives** respectively. Note that the hierarchy appears in the **Existing values for dc.Subject and Keywords** area.

If you don't want to see the popup each time you add folder level metadata, tick the **Do not show this warning again** checkbox; it won't be displayed again.

3. Finally, select all remaining files—the ones that are not in the *citizens*, *monarchs*, or *relative* folders—by selecting the first and shift-clicking the last. Set their **dc.Subject** metadata to **Tudor period|Others** and click outside the cell for the metadata to be assigned. This is done in a single operation (there is a short delay before it completes).

When multiple files are selected in the left hand collection tree, all metadata values for all files are shown on the right hand side. Items that are common to all files are displayed in black—e.g. **dc.Subject**—while others that pertain to only one or some of the files are displayed in grey—e.g. any extracted metadata.

Metadata inherited from a parent folder is indicated by a folder icon to the left of the metadata name. Select one of the files in the *relative* folder to see this.

4. Switch to the **Design** panel and select **Browsing Classifiers** from the left-hand list. Set the menu item for **Select classifier to add** to **Hierarchy**; then click **<Add Classifier...>**.
5. A window pops up to control the classifier's options. Change the **metadata** to **dc.Subject** and then click **<OK>**.
6. For tidiness' sake, **remove** the **classifier** for **Source** metadata (included by default) from the list of currently assigned classifiers, because this adds little to the collection.
7. Now switch to the **Create** panel, **build** the collection, and **preview** it. Choose the new **subjects** link that appears in the navigation bar, and click the bookshelves to navigate around the four-entry hierarchy that you have created.

Partitioning the full-text index based on metadata values

*Next we partition the full-text index into four separate pieces. To do this we first define four subcollections obtained by "filtering" the documents according to a criterion based on their **dc.Subject** metadata. Then*

an index is assigned to each subcollection. This will enable users to restrict a search to a subset of the documents.

8. Switch to the **Design** panel, and click **Partition Indexes**.
9. Ensure that the **Define Filters** tab is selected (the default). Define a subcollection filter with name **monarchs** that matches against **dc.Subject**, and type **Monarchs** as the regular expression to match with. Click **<Add Filter>**. This filter includes any file whose **dc.Subject** metadata contains the word *Monarchs*.
10. Define another filter, **relatives**, which matches **dc.Subject** against the word **Relatives**. Define a third and fourth, **citizens** and **others**, which matches it against the words **Citizens** and **Others** respectively.
11. Having defined the subcollection filters, we partition the index into corresponding parts. Click the **Assign Partitions** tab. Select the citizens subcollection and click **<Add Partition>**. Next select monarchs, and click **<Add Partition>**. Repeat for the other two subcollections, so that you end up with four partitions, one based on each subcollection filter.

The order they appear in the **Assigned Subcollection Partitions** list is the order they will appear in the drop down menu on the search page. You can change the order by using the **<Move Up>** and **<Move Down>** buttons.
12. **Build** and **preview** the collection.
13. The **search** section includes a pulldown menu that allows you to select one of these partitions for searching. For example, try searching the *relatives* partition for *mary* and then search the *monarchs* partition for the same thing.
14. To allow users to search the collection as a whole as well as each subcollection individually, return to the **Partition Indexes** section of the **Design** panel and select the **Assign Partitions** tab. Select all four subcollections by either checking their boxes or press the **Select All** button, and click **<Add Partition>**.
15. To ensure that the combined index appears first in the list on the reader's web page, use the **<Move Up>** button to get it to the top of the list here in the **Design** panel. Then **build** and **preview** the collection.
16. The text in the drop down box on the search page is based on the filters each partition was built on. To change the text that is displayed, go to the **Search** section of the **Format** panel. The single filter partitions have sensible default text, but the combined partition does not. Set the **Display text** for the combined partition to "all". **Preview** the collection.
17. Search for the term *Mary* again, as that is likely to be common in all five index partitions, and check that the numbers of words (not documents) in the search results for the 4 individual indexes add up to the number of words for the *all* index.

Controlling the building process

*Finally we look at how the building process can be controlled. Developing a new collection usually involves numerous cycles of building, previewing, adjusting some enrich and design features, and so on. While prototyping, it is best to temporarily reduce the number of documents in the collection. This can be accomplished through the **maxdocs** parameter to the building process.*

18. Switch to the **Create** panel. Expand the top panel to be able to see the options for collection building. Scroll to view them all. Select **maxdocs** and set its numeric counter to **3**. (When in GLI's **Expert Mode**, the **maxdocs** option for the import process are located under the **Import Options** of

the **Create** panel.) Now **build**.

19. Preview the newly rebuilt collection's **titles** page. Previously this listed more than a dozen pages per letter of the alphabet, but now there are just three—the first three files encountered by the building process.
20. Go back to the **Create** panel and turn off the **maxdocs** option. **Rebuild** the collection so that all the documents are included.

Formatting the HTML collection—Tudor

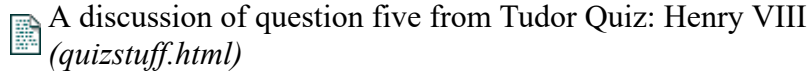
Prerequisite: [A large collection of HTML files—Tudor](#)

Devised for Greenstone version: 2.60|3.06

Modified for Greenstone version: 2.87|3.11

1. Open up your **tudor** collection, go to the **Format** panel (by clicking on its tab) and select **Format Features** from the left-hand list. Select the **browse** format feature and inspect its long format statement.

This displays something that looks like this:



for a particular document whose *Title* metadata is **A discussion of question five from Tudor Quiz: Henry VIII** and whose *Source* metadata is **quizstuff.html**.

This format appears in the **titles** list and also when you get down to individual documents in the **subjects** hierarchy. This is Greenstone's default format statement used in the **browse** format features.

Greenstone's default format statement is complex because it is designed to produce something reasonable under almost any conditions, and also because for practical reasons it needs to be backwards compatible with legacy collections.

2. In the **browse** format feature, replace the default **documentNode** template shown on the left with the simpler version on the right. (*The highlighted lines on the left are the ones that have been removed or modified. Those highlighted on the right have been newly added.*)

Change:

```
<gsf:template match="documentNode">
  <td valign="top">
    <gsf:link type="document">
      <gsf:icon type="document"/>
    </gsf:link>
  </td>
  <td valign="top">
    <gsf:link type="source">
      <gsf:choose-metadata>
        <gsf:metadata name="thumbicon"/>
        <gsf:metadata name="srcicon"/>
      </gsf:choose-metadata>
    </gsf:link>
  </td>
  <td valign="top">
    <gsf:link type="document">
<!--
Defined in the global format statement
-->
    <xsl:call-template name="choose-title"/>
    <gsf:switch>
      <gsf:metadata name="Source"/>
      <gsf:when test="exists">
        <br/>
        <i><gsf:metadata
name="Source"/></i>
      </gsf:when>
    </gsf:switch>
    </gsf:link>
  </td>
</gsf:template>
```

To:

```
<gsf:template match="documentNode">
  <td valign="top">
    <gsf:link type="document">
      <gsf:icon type="document"/>
    </gsf:link>
  </td>
  <td valign="top">
    <gsf:link type="document">
      <gsf:metadata name="Title"/>
      <br/>
      <i><gsf:metadata
name="Source"/></i>
    </gsf:link>
  </td>
</gsf:template>
```

Preview the result (you don't need to build the collection, because changes to format statements

take effect when you press the **Preview Collection** button). Look at the **titles** list. It is just the same as before! Under most circumstances this far simpler format statement is entirely equivalent to Greenstone's more complex default.

- Since we edited the **browse** format feature, the same format statements are used for documents listed in the **titles** list classifier and under each of the bookshelf nodes in the **subject** hierarchy classifier. The **Choose Feature** menu can be used to restrict a format statement to a specific classifier and its nodes. We will override this format statement for the hierarchical *subject* classifier. In the **Choose Feature** menu, scroll down to the item that says

CL2 Hierarchy -metadata dc.Subject

and select it. This is the format statement that affects the second classifier (i.e., "CL2"), which is a **Hierarchy** classifier based on **dc.Subject** metadata.

Click **<Add Format>** to add this format statement to the collection.

Edit the **HTML Format String** box, replacing the **documentNode** template on the left with the one on the right. (The changes are mostly the same as before, but without reference to the Source document name.)

Change:

```
<gsf:template match="documentNode">
  <td valign="top">
    <gsf:link type="document">
      <gsf:icon type="document"/>
    </gsf:link>
  </td>
  <td valign="top">
    <gsf:link type="source">
      <gsf:choose-metadata>
        <gsf:metadata name="thumbicon"/>
        <gsf:metadata name="srcicon"/>
      </gsf:choose-metadata>
    </gsf:link>
  </td>
  <td valign="top">
    <gsf:link type="document">
      <xsl:call-template name="choose-title"/>
    </gsf:link>
    <gsf:switch>
      <gsf:metadata name="Source"/>
      <gsf:when test="exists">
        <br/>
        <i>(<gsf:metadata name="Source"/>)</i>
      </gsf:when>
    </gsf:switch>
  </td>
</gsf:template>
```

To:

```
<gsf:template match="documentNode">
  <td valign="top">
    <gsf:link type="document">
      <gsf:icon type="document"/>
    </gsf:link>
  </td>
  <td valign="top">
    <gsf:link type="document">
      <gsf:metadata name="Title"/>
    </gsf:link>
  </td>
</gsf:template>
```

- Preview** the **subjects** list in the collection. When you get down to a list of documents in the subject hierarchy, the filename does not appear beside the title, because **Source** is not specified in the format statement and this format statement applies to all **documentNodes** in the *subject* classifier. Note that the **titles** classifier has not changed: it still displays the filename underneath the title.
- Let's change the search results' format statement so that **dc.Subject** metadata is displayed below the title. Select the **search** format feature. Insert the highlighted line:


```
<gsf:template match="documentNode">
  <td valign="top">
    <gsf:link type="document">
      <gsf:icon type="document"/>
    </gsf:link>
```

```

</td>
<td>
  <gsf:link type="document">
    <xsl:call-template name="choose-title"/>
  </gsf:link>
  <br /><gsf:metadata name="dc.Subject"/>
</td>
</gsf:template>

```

6. **Preview** the collection. Documents in the search results list will be displayed like this:

 A discussion of question five from Tudor Quiz: Henry VIII
Tudor period|Others

(The vertical bar appears because this **dc.Subject** metadata is hierarchical metadata. Unfortunately there is no easy way to get at individual components of the hierarchy. For most metadata, such as title and author, this isn't a problem.)

7. Finally, let's return to the **subjects** hierarchy and modify the bookshelf display. Reselect the format feature for

CL2 Hierarchy -metadata dc.Subject

First modify the **documentNode** template so that it now looks like:

```

<gsf:template match="documentNode">
  <td valign="top">
    <gsf:link type="document">
      <gsf:icon type="document"/>
    </gsf:link>
  </td>
  <td valign="top">
    <b>Title:</b>
    <gsf:metadata name="Title"/>
  </td>
</gsf:template>

```

Next, scroll down to the **VList classifierNode** template, and add in the highlighted statement:

```

<gsf:template match="classifierNode[@classifierStyle = 'VList']">
  <td valign="top">
    <gsf:link style="static" type="classifier">
      <gsf:icon type="classifier"/>
    </gsf:link>
  </td>
  <td valign="top">
    <b>Bookshelf title:</b>
    <gsf:link type="classifier">
      <gsf:metadata name="Title"/>
    </gsf:link>
  </td>
</gsf:template>

```

Preview the collection and examine the subject hierarchy again to see the effect of your changes. Bookshelves should now say **Bookshelf title:** and then the subject name (which is the "title" of the bookshelf), while documents will display **Title:** and the title.

Section tagging for HTML documents

Devised for Greenstone version: 2.70w|3.06

Modified for Greenstone version: 2.87|3.11

1. In a browser, visit the Greenstone demo collection, **Demo Collection (lucene-jdbm-demo)**, and have a look at it. 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: *Greenstone3* → *web* → *sites* → *localsite* → *collect* → *lucene-jdbm-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 highlighted text just after the tag for the **Introduction** section:

```
<!--
<Section>
  <Description>
    <Metadata name="Title">Introduction</Metadata>
  </Description>
-->
<!--
<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 highlighted section:

```
<!--
</Section>
-->
<!--
<Section>
  <Description>
    <Metadata name="Title">What do you need to start?</Metadata>
  </Description>
-->
```

Save the edited file and close it. 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 **HTMLPlugin** 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.

Downloading files from the web

Devised for Greenstone version: 2.60|3.06

Modified for Greenstone version: 2.87|3.11

The Greenstone Librarian Interface's Download panel allows you to download individual files, parts of websites, and indeed whole websites, from the web.

1. Start a new collection called **webtudor**, and base it on -- **New Collection** --.
2. In a web browser, visit <https://englishhistory.net>, follow the link to *The Tudors*. You should be at the URL

<https://englishhistory.net/tudor/>

This is where we started the downloading process to obtain the files you have been using for the **tudor** collection. You could do the same thing by copying this URL from the web browser, pasting it into the **Download** panel, and clicking the **<Download>** button. However, several megabytes will be downloaded, which might strain your network resources—or your patience! For a faster exercise we focus on a smaller section of the site.

3. Go to the **Download** panel by clicking its tab. There are five download types listed on the left hand side. For this exercise, we only use the **Web** type. Make sure this is selected in the list.

Enter this URL

<https://englishhistory.net/tudor/citizens/>

into the **Source URL** box. There are several other options that govern how the download process proceeds. To see a description of an option, hover the mouse over it and a tooltip will appear. To copy just the *citizens* section of the website, switch on the **Only files below URL** option by checking its box and set the **Download Depth** option to 1. If you don't do this (or if you miss out the terminating "/" in the URL), the downloading process will follow links to other areas of the *englishhistory.net* website and grab those as well. Also switch on the **Only files within site** option to avoid downloading any items on the site pages that actually emanate from outside it (like google ads).

4. If your computer is behind a firewall or proxy server, you will need to edit the proxy settings in the Librarian Interface. Click the **<Configure Proxy...>** button. Switch on the **Use proxy connection?** checkbox. Enter the proxy server address and port number in the **HTTP Proxy Host:** and **Port:** boxes.

URLs that start with *https*, or URLs that resolve to *https*, will additionally need the **HTTPS Proxy Host:** and corresponding **Port:** filled in too, before web pages can be downloaded from there.

Websites at *https* URLs often have a security certificate, but not always. For instance, <https://englishhistory.net> does not have one. To instruct GLI to nevertheless download pages from *https* URLs that don't have a security certificate, you'll also need to switch on the **No certificate checking for HTTPS downloads** checkbox.

Once you've finished configuring the proxy settings, click **<OK>** to close the dialog.

5. Now click **<Download>**. If you have set proxy information in **Preferences...**, a popup will ask for your user name and password. If you're on Windows Vista or later, Windows may show a popup message asking whether you wish to block or unblock the download. In such a case, choose to unblock. With proxy settings turned on, it may take a short while before GLI starts downloading.

Once the download has started, a progress bar appears in the lower half of the panel that reports on how the downloading process is doing.

*More detailed information can be obtained by clicking <View Log>. The process can be stopped altogether by clicking <Close>. Downloading can be a lengthy process involving multiple sites, and so Greenstone allows additional downloads to be queued up. When new URLs are pasted into the **url** box and <Download> clicked, a new progress bar is appended to those already present in the lower half of the panel. When the currently active download item completes, the next is started automatically.*

6. Downloaded files are stored in a top-level folder called **Downloaded Files** that appears on the left-hand side of the **Gather** panel. You may not need all the downloaded files, and you choose which you want by dragging selected files from this folder over into the collection area on the right-hand side, just like we have done before when selecting data from the *sample_files* folder. In this example we will include everything that has been downloaded.

Select the *englishhistory.net* folder within **Downloaded Files** and drag it across into the collection area. Once you've dropped the folder into the collection area, you may see popup dialogs, one for each file extension that is not recognised by GLI. Either keep clicking <OK> to confirm for each unrecognised filetype, or, in the popup, you can tick the checkbox to not see the same message again.

7. Switch to the **Create** panel to **build** and **preview** the collection. It is smaller than the previous collection because we included only the *citizens* files. However, these now represent the latest versions of the documents.

Pointing to documents on the web

Prerequisite: [Downloading files from the web](#)

Devised for Greenstone version: 2.60|3.06

Modified for Greenstone version: 2.87|3.11

1. Open up your **tudor** 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* 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 **HTMLPlugin** 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. Click **<OK>**.

Setting this option to the **HTMLPlugin** 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

```
https://englishhistory.net/tudor/citizens/...
```

If you had 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 *sample_files* → *tudor* folder (or the **Downloaded Files** folder if you have done exercise [Downloading files from the web](#)) 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 **documentNode** template of the **browse** format statement by replacing

```
<gsf:link type="document">
  <gsf:icon type="document"/>
</gsf:link>
```

with

```
<gsf:link type="web">
  <gsf:icon type="web"/>
</gsf:link>
```

4. Switch to the **Create** panel and **build** and **preview** the collection. Note that the document icons have changed. Try clicking on *boleyn.html*. 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.

Bibliographic collection

Sample files: [marc.zip](#)

Devised for Greenstone version: 2.60|3.06

Modified for Greenstone version: 2.87|3.11

This exercise looks at using fielded searching in a collection. Fielded searching is best used for metadata rich collections. Here we use bibliographic data in MARC format.

1. Start a new collection called **Papers Bibliography** which will contain a collection of example MARC records of the working papers published at the [Computer Science Department, Waikato University](#). Enter the requested information and base it on -- **New Collection** --.
2. In the **Gather** panel, open the *sample_files* → *marc* folder, drag *CMSwp-all.marc* into the right-hand pane and drop it there. A popup window asks whether you want to add **MARCPlugin** to the collection to process this file. Click **<Add Plugin>**, because this plugin will be needed to process the MARC records.
3. Now select **Browsing Classifiers** within the **Design** panel and **remove** the default classifier for **Source** metadata.
4. In the **Search Indexes** section, **remove** the **ex.Source** index. In this collection all records are from the same file, so **ex.Source** metadata, which is set to the filename, is not particularly interesting or useful.
5. Switch to the **Create** panel, **build** the collection, and **preview** it. Browse through the *titles* classifier and view a record or two. Try searching—for example, find items that include **graphics**.
6. Back in the Librarian Interface, go to the **Browsing Classifiers** section of the **Design** panel. Select **AZCompactList** from the **Select classifier to add** drop down menu, and click **<Add Classifier...>**. In the popup window, select **dc.Subject** as the metadata item. Click **<OK>**.

AZCompactList is like List, except that terms that appear multiple times in the hierarchy are automatically grouped together and a new node, shown as a bookshelf icon, is formed.

7. **Build** the collection and **preview** the result.

Using fielded searching

8. Now let's look at fielded searching. In the browser, press the **fielded search** button below the usual search form. This will present a fielded search form.
9. You can specify which search form types are available for a particular collection, and which one is the default, using the **searchType** format statement. In the **Format** panel, select **Format Features** from the left-hand list. Select the **searchType** format statement from the list of assigned formats, and set the contents to just **simpleform**. This will make only fielded searching available for this collection.

*Search type options include **plain**, **simpleform** (for fielded searching) and **advancedform** (for fielded searching with boolean operations). You can specify any combination of these, separated by a comma. If the **plain** search type is specified, it will be available in the search area at the top of each page of the collection.*

10. **Preview** the collection again. Notice that the collection's pages no longer includes a query box. (This is because the search form is too big to fit here nicely.) To search, you have to click **form search** in the navigation bar. Note that **text search** is no longer offered.

11. Look at the search form in the collection. There are two fields that can be searched: *text* and *titles*. Add some more fields to search on by going back to the Librarian Interface.
12. In the **Design** panel, go to the **Search Indexes** section. Add a new index based on **dc.Subject** by clicking **<New Index>**, selecting **dc.Subject** in the list of metadata, and clicking **<Add Index>**.
13. **Rebuild** the collection and **preview** the results. Notice the extra field in the **in field** drop-down menus in the search form. You can do quite complicated queries by searching for words in different fields at the same time.
14. To change the text that is displayed in the drop-down menus of the search form, you would go to the **Search** section of the **Format** panel. Here you can change the display text for the indexes.

Exploding the database

15. Go to the **Enrich** panel and try to see the metadata. It doesn't appear! This is because the metadata is associated with records inside the file, not the file itself.

Metadata file types, such as MARC, CDS/ISIS, BibTex etc. can be imported into Greenstone but their metadata cannot be viewed in the Librarian Interface. To edit any metadata you need to go back to the program that created the file.

Greenstone provides a way of *exploding* a metadata database so that each record appears as an individual document, with viewable and editable metadata. This process is irreversible: once this step has been done, the database is deleted and can no longer be used in its original program.

16. In the **Gather** panel, you may notice that the MARC database has a different coloured icon to other files. A metadata database that can be exploded will be displayed with this green icon. Right-click on the file and choose **Explode Metadata Database** from the menu. A new window opens, containing options for the exploding process. A description of each option can be obtained by hovering the mouse over the option.

If it's not already on, turn on the **metadata_set** option by checking its box. This option indicates which metadata set to explode the metadata into. The default set is the "Exploded Metadata Set"—a metadata set which initially has no elements in it, but will receive a new element for each metadata field retrieved from the database.

17. Click **<Explode>** to start the exploding process. This may take a short while, depending on the size of the database.
18. Once exploding has finished, the MARC database file will have been deleted, and three folders created in its place. These folders contain an empty file for each record in the original database. The metadata for these records can be viewed and edited by switching to the **Enrich** panel.
19. Because the MARC file is no longer present, and the collection contains empty (.nul) files, we need to change the list of plugins. In the **Document Plugins** section of the **Design** panel, remove **MARCPlugin**.
20. **Rebuild** and **preview** the collection. You will notice that the *subjects* classifier is empty, searching no longer returns any results, and the document display is useless as the linked .nul documents don't exist, resulting in "Not Found" messages from the server.

Although the *titles* classifier was built on **ex.Title**, it still displays the correct titles, but in the **Enrich** panel you can see the **ex.Title** metadata are actually the filenames rather than titles of the MARC records. This is because the default **browse** format uses the **exp.Title** metadata. In the **Format Features** section of the **Format** panel, select **global** in the list of assigned format statements. The format statement looks like:

```
<gsf:template name="choose-title">
  <gsf:choose-metadata>
    <gsf:metadata name="dc.Title"/>
    <gsf:metadata name="exp.Title"/>
    <gsf:metadata name="ex.dc.Title"/>
    <gsf:metadata name="Title"/>
    <gsf:default>Untitled</gsf:default>
  </gsf:choose-metadata>
</gsf:template>
```

The above **choose-title** template, defined in the **global** format features, is included by the **browse** format statements. Since there is no **dc.Title** metadata and because **exp.Title** comes before **ex.Title**, the exploded titles will be displayed.

Reformatting the collection to use the exploded metadata

The collection previously used extracted (ex.) metadata, but now it uses exploded (exp.) metadata. The *subjects* classifier and search indexes were built on ex metadata, which is why they no longer work properly.

There is also no longer any text in the documents. Previously, **MARCPlugin** stored the raw record as the "text" of each record. Now that the metadata is in the Librarian Interface, there is no longer the concept of raw record, and so there is no text.

We need to modify the collection design to take note of these changes.

21. In the **Design** panel's **Search Indexes** section, change the Title index to use **exp.Title**: select the Title index in the **Assigned Indexes** list and click **<Edit Index>**. Deselect **dc.Title** and **ex.Title** in the list of metadata, and select **exp.Title**. Click **<Replace Index>**.
22. Remove the **dc.Subject** index by selecting it in the **Assigned Indexes** list and clicking **<>**. Add an index on **exp.Subject**: click **<New Index>**, select **exp.Subject** in the metadata list, and click **<Add Index>**.
23. The text index is no longer any use, so remove that index too.
24. To enable combined searching across all indexes at once, click **<New Index>**, tick the **Add combined searching over all assigned indexes (allfields)** checkbox, and click **<Add Index>**. Move this to the top of the list using the **<Move Up>** button, so that it appears first in the drop down list. Click **<Set Default>** on the right so that it becomes the default field for searching.
25. To explicitly use the **exp.Title** metadata, in the **Browsing Classifiers** section, change the **dc.Title;Title List** to use **exp.Title** metadata. Double click the **dc.Title;Title List** in the **Assigned Classifiers** list, and change the **metadata** option to use **exp.Title**. Click **<OK>**. Do the same thing for the Subject **AZCompactList**, changing **dc.Subject** to **exp.Subject**.
26. **Rebuild** and **preview** the collection. The classifiers should be back to normal and searching should now work.
27. Switch to the **Format Features** section of the **Format** panel to make the following adjustments.
 - There is no dc (or ex.dc) metadata for this collection, so in the **global** format feature's **choose-title** template, replace the following

```
<gsf:choose-metadata>
  <gsf:metadata name="dc.Title"/>
  <gsf:metadata name="exp.Title"/>
  <gsf:metadata name="ex.dc.Title"/>
  <gsf:metadata name="Title"/>
  <gsf:default>Untitled</gsf:default>
</gsf:choose-metadata>
```

with

```
<gsf:choose-metadata>
  <gsf:metadata name="exp.Title"/>
  <gsf:metadata name="Title"/>
  <gsf:default>Untitled</gsf:default>
</gsf:choose-metadata>
```

- There are no source or thumb icons, so in the **documentNode** template of the **browse** format features remove the occurrence of the following section:

```
<td valign="top">
  <gsf:link type="source">
    <gsf:choose-metadata>
      <gsf:metadata name="thumbicon"/>
      <gsf:metadata name="srcicon"/>
    </gsf:choose-metadata>
  </gsf:link>
</td>
```

- The ex.Source metadata is set to the nul filename, so remove that from the display. Once again, from both browsing classifiers (**browse** and **CL1 List**), remove:

```
<gsf:switch>
  <gsf:metadata name="Source"/>
  <gsf:when test="exists">
    <br/>
    <i><gsf:metadata name="Source"/></i>
  </gsf:when>
</gsf:switch>
```

28. **Preview** the collection. In the *titles* list, click on one of the documents. This will take you to the document's display page. Exploding the database has left this document display useless. Only the record Title (in this case, the generated filename) is displayed. We will make two changes to improve the document display. First, we will remove the record Title, since it is not useful in this instance. To remove the record Title, we need to override the default **documentHeading** format statement with one that does not do anything. Go to the **display** format feature in the **Format Features** section of the **Format** panel and add the following just after `<gsf:option name="TOC" value="true"/>`:

```
<gsf:template name="documentHeading"/>
```

29. Next, override the default **documentContent** behaviour by creating a format statement for this. Still in the **display** format features, after the **documentHeading**, add the following format statement (which can be copied from *sample_files* → *marc* → *format_tweaks* → *document_content.txt*):

```
<gsf:template name="documentContent">
  <table>
    <tr>
      <td>Title:</td>
      <td><gsf:metadata name="exp.Title"/></td>
    </tr>
    <tr>
      <td>Subject:</td>
      <td><gsf:metadata name="exp.Subject"/></td>
    </tr>
    <tr>
      <td>Publisher:</td>
      <td><gsf:metadata name="exp.Publisher"/></td>
    </tr>
  </table>
</gsf:template>
```

30. Press the **<Preview Collection>** button to preview the collection and see how the document display has improved.

CDS/ISIS collection

Sample files: [isis.zip](#)

Devised for Greenstone version: 2.70w|3.06

Modified for Greenstone version: 2.87|3.11

This exercise is similar to the [Bibliographic collection](#) exercise, except that a CDS/ISIS database is used instead of a MARC database, and we do not explode the database.

1. Start a new collection called **ISIS Collection** (base it on **New Collection**).
2. Drag the files from *sample_files* → *isis* (excluding the *format_tweaks* folder and the README.txt file) into the collection.
3. **Build** and **preview** the collection. The default indexes, classifiers and formats are not very useful for this data. There is no metadata searching, and the *titles* classifier is completely empty. The *filenames* classifier is useless because all records come from the same file.
4. In the **Search Indexes** section of the **Design** panel, remove the useless Source and Title indexes, and add new indexes for **ex.Photographer^all**, **ex.Country^all** and **ex.Notes^all** metadata. In the **Search** section of the **Format** panel, you can set the display text for these indexes to "photographer", "country" and "notes".

CDS/ISIS metadata has subfields, and these are represented using ^.

5. In the **Browsing Classifiers** section of the **Design** panel, remove the existing (useless) classifiers for **dc.Title;ex.Title** and **ex.Source**, and add a new **List** for **ex.Photographer**.
6. **Rebuild** and **preview** the collection.
7. In the **Format Features** section of the **Format** panel, change the **browse** format statement to display **Photographer** and **Notes** metadata. Change its **documentNode** template to look like:

```
<gsf:template match="documentNode">
  <td valign="top">
    <gsf:link type="document">
      <gsf:icon type="document"/>
    </gsf:link>
  </td>
  <td valign="top">
    <b><gsf:metadata name="Photographer^all"/></b>
    <br/><gsf:metadata name="Notes^all"/>
  </td>
</gsf:template>
```

The above format can be copied from *sample_files* → *isis* → *format_tweaks* → *browse_tweak.txt*. If you want search results to be displayed in a similar manner, make the same changes to the **documentNode** template of the **search** format features too.

8. Go to the **display** format feature in the **Format Features** section of the **Format** panel and add the following just after `<gsf:option name="TOC" value="true"/>`:

```
<gsf:template name="documentHeading"/>
```

ISISPlug stores a nicely formatted version of the record as the document text, and this is what is displayed when we view a record. Let's tidy it up a little more.

9. We'll link to the raw record, which is stored as **ISISRawRecord** metadata. Add the following to the **display** format statement (which can be copied from *sample_files* → *isis* → *format_tweaks* → *document_content.txt*), to adjust the **documentContent**. This now makes use of a predefined

Greenstone javascript function to toggle between displaying and hiding the raw record.

```
<gsf:template name="documentContent">
  <p>
    <xsl:call-template name="wrappedSectionText"/>
  </p>

  <a href="javascript:;" id="cdsreclink">Show/Hide CDS Record</a>
  <div id="cdsrecord">
    <b>CDS Record:</b>
    <br/>
    <tt>
      <gsf:metadata name="ISISRawRecord"/>
    </tt>
  </div>

  <script type="text/javascript">
    <xsl:text disable-output-escaping="yes">
      var link=document.getElementById("cdsreclink");
      var div=document.getElementById("cdsrecord");
      gs.functions.makeToggle(link, div);
    </xsl:text>
  </script>
</gsf:template>
```

10. Preview the collection.

Looking at a multimedia collection

Sample files: [beatles.zip](#)

Devised for Greenstone version: 2.60|3.06

Modified for Greenstone version: 2.87|3.11

1. Copy the entire folder

sample_files → *beatles* → *advbeat_large*

(with all its contents) into your Greenstone *collect* folder. If you have installed Greenstone in the usual place, this is

My Computer → *Local Disk (C:)* → *Users* → *<Username>* → *Greenstone3* → *web* → *sites* → *localsite* → *collect*

where *<Username>* is the username under which Greenstone is installed.

Put *advbeat_large* in there. Then go into the *advbeat_large* folder and delete its *index* subfolder.

2. Next, go into the *advbeat_large* folder's *etc* subfolder and rename the Greenstone 2 specific **collect.cfg** file there to **collect.cfg.bak**. Doing so will allow GLI to detect only the Greenstone 3 collection configuration file, **collectionConfig.xml**, enabling GLI to rebuild the collection based on that.
3. Start up GLI and open the *Advanced Beatles -- large* collection. Switch to the **Create** panel and **build** the collection. **Preview** the result.
4. Explore the Beatles collection. Note how the **Browse** button divides the material into seven different types. Within each category, the documents have appropriate icons. Some documents have an audio icon: when you click these you hear the music (assuming your computer is set up with appropriate player software). Others have an image thumbnail: when you click these you see the images.
5. Look at the *titles* browser. Each title has a bookshelf that may include several related items. For example, *Hey Jude* has a MIDI file, lyrics, and a discography item.
6. Observe the low quality of the metadata. For example, the five items under **A Hard Day's Night** (under "H" in the *titles* browser) have different variants as their titles. The collection would have been easier to organize had the metadata been cleaned up manually first, but that would be a big job. Only a tiny amount of metadata was added by hand—fewer than ten items. The original metadata was left untouched and Greenstone facilities were used to clean it up automatically. (You will find in [Building a multimedia collection](#) that this is possible but tricky.)
7. In the file browser, take a look at the files that make up the collection, in the

sample_files → *beatles* → *advbeat_large* → *import*

folder. What a mess! There are around 450 files under seven top-level sub-folders. Organization is minimal, reflecting the different times and ways the files were gathered. For example, *html_lyrics*, *discography* and *tablature_txt* are excerpts of web sites, and *images* contains various images in JPEG format. For each type, drill down through the hierarchy and look at a sample document.

Building a multimedia collection

Prerequisite: [Looking at a multimedia collection](#)

Sample files: [beatles.zip](#)

Devised for Greenstone version: 2.60|3.06

Modified for Greenstone version: 2.87|3.11

We will proceed to reconstruct from scratch the Beatles collection that you have just looked at. We develop the collection using a small subset of the material, purely to speed up the repeated rebuilding that is involved.

1. Start a new collection (**File** → **New...**) called **small beatles**, basing it on the default -- **New Collection** --. (Basing it on the existing Advanced Beatles collection would make your life far easier, but we want you to learn how to build it from scratch!)
2. Copy the files and folders provided in

sample_files → *beatles* → *advbeat_small*

into your new collection. Do this by opening up *advbeat_small*, selecting the eight items within it (from *discography* to *beatles_midi.zip*), and dragging them across. Because some of these files are in MP3 and MARC formats you will be asked whether to include **MP3Plugin** and **MARCPlugin** in your collection. Click **<Add Plugin>**.

3. Change to the **Enrich** panel and browse around the files. There is no metadata—yet. Recall that you can double-click files to view them.

(There are no MIDI files in the collection: these require more advanced customisation because there is no MIDI plugin. We will deal with them later.)

4. Change to the **Create** panel and **build** the collection.
5. **Preview** the result.

Manually correcting metadata

6. You might want to correct some of the metadata—for example, the atrocious misspelling in the titles "MAGICAL MISTERY TOUR." These documents are in the discography section, with filenames that contain the same misspelling. Locate one of them in the **Enrich** panel. Notice that the extracted metadata element **ex.Title** is now filled in, and misspelt. You cannot correct this element, for it is extracted from the file and will be re-extracted every time the collection is re-built.
7. Instead, add **dc.Title** metadata for these two files: "Magical Mystery Tour." In the **Enrich** panel, open the discography folder and drill down to the individual files. Set the **dc.Title** value for the two offending items.
8. **Build** the collection again, and **preview** it.

Extracted metadata is unreliable. But it is very cheap! On the other hand, manually assigned metadata is reliable, but expensive. The previous section of this exercise has shown how to aim for the best of both worlds by using extracted metadata but correcting it when it is wrong.

Browsing by media type

9. First let's remove the **List** classifier for filenames, which isn't very useful, and replace it with a browsing structure that groups documents by category (discography, lyrics, audio etc.). Categories

are defined by manually assigned metadata.

- Change to the **Enrich** panel, select the folder *discography* and set its **dc.Format** metadata value to "Discography". Setting this value at the folder level means that all files within the folder inherit it.
- Repeat the process. Assign "Lyrics" to the *html_lyrics* folder, "Images" to *images*, "MARC" to *marc*, "Audio" to *mp3*, "Tablature" to *tablature_txt*, and "Supplementary" to *wordpdf*.
- Switch to the **Design** panel and select the **Browsing Classifiers** section.
- Delete the **ex.Source** classifier (the second one).
- Add a **List** classifier and select **dc.Format** as the **metadata** field. Click the **bookshelf_type** and select **always** in the drop-down list. Click the **partition_type_within_level** check box and choose **none** from the drop-down list. Click the **sort_leaf_nodes_using** checkbox, and select **ex.Title** in the drop-down list: this will make the classifier display documents in alphabetical order of title. Specify **browse** as the **buttonname**.

Build the collection again and **preview** it.

*Note how we assigned **dc.Format** metadata to all documents in the collection with a minimum of labour. We did this by capitalizing on the folder structure of the original information. Even though we complained earlier about how messy this folder structure is, you can still take advantage of it when assigning metadata.*

Using switch statements

10. Alongside the Audio files there is an MP3 icon, which plays the audio when you click it. There is also a document icon, which doesn't make much sense with either audio or image files. We can modify the format statement to display different icons depending on the value of the **dc.Format** metadata field.

- Change to the **Format** panel and select the **Format Features** section.
- Ensure that the **browse** format feature is selected, and make the changes highlighted below to its **documentNode** template. Change:

```
<td valign="top">
  <gsf:link type="document">
    <gsf:icon type="document"/>
  </gsf:link>
</td>
<td valign="top">
  <gsf:link type="source">
    <gsf:choose-metadata>
      <gsf:metadata name="thumbicon"/>
      <gsf:metadata name="srcicon"/>
    </gsf:choose-metadata>
  </gsf:link>
</td>
```

to this:

```
<td valign="top">
  <gsf:switch>
    <gsf:metadata name="dc.Format"/>
    <gsf:when test='equals' test-value='Audio'>
      <gsf:link type="source"><gsf:metadata name="srcicon"/></gsf:link>
    </gsf:when>
    <gsf:when test='equals' test-value='Images'>
      <gsf:link type="source"><gsf:metadata name="thumbicon"/></gsf:link>
    </gsf:when>
    <gsf:when test='equals' test-value='Supplementary'>
      <gsf:link type="source"><gsf:metadata name="srcicon"/></gsf:link>
      <gsf:link type="document"><gsf:icon type="document"/></gsf:link>
    </gsf:when>
    <gsf:otherwise>
      <gsf:link type="document"><gsf:icon type="document"/></gsf:link>
    </gsf:otherwise>
```

```

    </gsf:switch>
  </td>

```

To make this easier for you we have prepared a plain text file that contains the new text. In WordPad open the following file. (Do not use Notepad, because Notepad does not display the line breaks correctly.)

sample_files → *beatles* → *format_tweaks* → *audio_tweak_3.txt*

The `gsf:switch` statement allows you to display different things depending on the value (or existence) of a metadata field. This switch statement is based on the value of **dc.Format**. If `dc.Format` equals *Audio*, then the source icon will be displayed, linking to the source document. (For the MP3 files, the MP3 icon will display, and clicking the icon will play the MP3.) If `dc.Format` equals *Images*, the image thumbnail will appear, linking to the full-size image. If `dc.Format` equals *Supplementary*, both the source and document icons will appear, linking to the source document and the document display page, respectively. Finally, the `gsf:otherwise` statement says what to do in all other cases.

Preview the result. You may need to click the browser's **<Reload>** button to force it to re-load the page.

11. While we're at it, let's remove the source filename from where it appears after each document.

- In the **documentNode** template of the **browse** format feature, delete the following text:

```

<gsf:switch>
  <gsf:metadata name="Source"/>
  <gsf:when test="exists">
    <br/>
    <i><gsf:metadata name="Source"/></i>
  </gsf:when>
</gsf:switch>

```

Preview the result (you don't need to rebuild the collection.)

Using *AZCompactList* rather than *List*

12. There are sometimes several documents with the same title. For example, *All My Loving* appears both as lyrics and tablature (under *ALL MY LOVING*). The **titles** browser might be improved by grouping these together under a bookshelf icon. This is a job for an **AZCompactList**. Earlier in this tutorial we showed how to use the **bookshelf_type** option in **List** classifier to group documents with the same metadata value (**dc.Format** in that case) in one bookshelf. Here we use **AZCompactList** instead.

- Change to the **Design** panel and select the **Browsing Classifiers** section.
- Remove the **dc.Title;Title** classifier (at the top)
- Add an **AZCompactList** classifier, and enter **dc.Title,ex.Title** as its metadata.
- Finish by pressing **<OK>**.
- Move the new classifier to the top of the list (**<Move Up>** button).

Build the collection again and **preview** it. Both items for *All My Loving* now appear under the same bookshelf. However, many entries haven't been amalgamated because of non-uniform titles: for example *A Hard Day's Night* appears as several different variants. We will learn below how to amalgamate these.

Making bookshelves show how many items they contain

13. Make the bookshelves show how many documents they contain by modifying the **VList classifierNode** template of the **browse** format feature in the **Format Features** section of the **Format** panel. Insert the highlighted statements:

```

<gsf:template match="classifierNode[@classifierStyle = 'VList']">
  ...
  <gsf:link type="classifier">
    <gsf:metadata name="Title"/>
  </gsf:link>
</td>
<td valign="top">
  (<gsf:metadata name="numleafdocs"/>)
</td>
</gsf:template

```

The complete format statement for the **VList classifierNode** template of the **browse** format feature can be copied from *sample_files* → *beatles* → *format_tweaks* → *show_num_docs_3.txt*.

Preview the result (you don't need to build the collection.) Bookshelves in the titles and browse classifiers should show how many documents they contain.

Branding the collection with an image

14. To complete the collection, let's give it a new image for the link from the main page. Go to the **General** section of the **Format** panel. Use the browse button of the **URL to 'about page' image:** to select the following image:

sample_files → *beatles* → *advbeat_large* → *images* → *tile.jpg*

You can also set an image for the link to the collection's home page here. For this, use the browse button of **URL to 'home page' image:** to select the following image:

sample_files → *beatles* → *advbeat_large* → *images* → *beatlesmm.png*

Preview the collection, and make sure the new image appears on the collection's about page.

Also go to the digital library home page by clicking on the *My Greenstone Library* link at the top left. On the home page, look through the links to all the collections in your digital library to find the one to the Small Beatles collection. This link should now be denoted by an image bearing the text "BeatlesMultimedia".

Using UnknownPlugin

In this section we incorporate the MIDI files. Greenstone has no MIDI plugin (yet). But that doesn't mean you can't use MIDI files!

15. **UnknownPlugin** is a useful generic plugin. It knows nothing about any given format but can be tailored to process particular document types—like MIDI—based on their filename extension, and set basic metadata.

In the **Document Plugins** section of the **Design** panel:

- add **UnknownPlugin**;
- activate its **process_extension** field and set it to "mid" to make it recognize files with extension *.mid*;
- Set **file_format** to "MIDI" and **mime_type** to "audio/midi".

In this collection, all MIDI files are contained in the file *beatles_midi.zip*. **ZIPPlugin** (already in the list of default plugins) is used to unpack the files and pass them down the list of plugins until they reach **UnknownPlugin**.

16. **Build** the collection and **preview** it. Unfortunately, the MIDI files don't appear as Audio under the *browse* button. That's because they haven't been assigned **dc.Format** metadata.

- Back in the **Enrich** panel, click on the file *beatles_midi.zip* and assign its **dc.Format** value to "Audio"—do this by clicking on "Audio" in the **Existing values for dc.Format** list. All files extracted from the Zip file inherit its settings.

Cleaning up a title browser using regular expressions

We now clean up the **titles** browser.

17. We are going to use the **removesuffix** classifier option. The aim is to amalgamate variants of titles by stripping away extraneous text. For example, we would like to treat "ANTHOLOGY 1", "ANTHOLOGY 2" and "ANTHOLOGY 3" the same for grouping purposes. To achieve this:
 - Go to the Title **AZCompactList** under **Browsing Classifiers** on the **Design** panel;
 - Click the **<Configure Classifier...>** button on it. Activate its **removesuffix** option and set it to:

```
(?i)(\s+\d+)|(\s+[[:punct:]].*)
```

Build the collection and **preview** the result. Observe how many more times similar titles have been amalgamated under the same bookshelf. Test your understanding of regular expressions by trying to rationalize the amalgamations. (Note: `[[:punct:]]` stands for any punctuation character.)

*One powerful use of regular expressions in the exercise was to clean up the **titles** browser. Perhaps the best way of doing this would be to have proper title metadata. The metadata extracted from HTML files is messy and inconsistent, and this was reflected in the original **titles** browser. Defining proper title metadata would be simple but rather laborious. Instead, we have opted to use regular expressions in the **AZCompactList** classifier to clean up the title metadata. This is difficult to understand, and a bit fiddly to do, but if you can cope with its idiosyncrasies it provides a quick way to clean up the extracted metadata and avoid having to enter a large amount of metadata.*

Using different icons for different media types

To put finishing touches to our collection, we add some decorative features

18. Close the collection in the Librarian Interface (**File** → **Close**).
19. Using your file browser outside Greenstone, locate the folder

sample_files → *beatles* → *advbeat_large*

20. Open up another file browser, and locate the small beatles collection in your Greenstone installation:

Greenstone3 → *web* → *sites* → *localsite* → *collect* → *smallbea*

smallbea is the folder name generated by Greenstone for this collection. You can determine what the folder name is for a collection by looking at the title bar of the Librarian Interface: the folder name is displayed in brackets after the collection name.

21. Using the file browser, copy the *images* folder from the *advbeat_large* folder into the *smallbea* folder. (It's OK to overwrite the existing *images* folder: the image in it is included in the folder being copied.) The *images* folder includes some useful icons.
22. Open the collection in GLI again and update the previously edited portion of the **documentNode** format statement of the **browse** format feature (in **Format Features** on the **Format** panel) to be the following. You can copy this text from the file *sample_files* → *beatles* → *format_tweaks* → *multi_icons_3.txt*. Change:

```

<td valign="top">
  <gsf:switch>
    <gsf:metadata name="dc.Format"/>
    <gsf:when test='equals' test-value='Audio'>
      <gsf:link type="source"><gsf:metadata name="srcicon"/></gsf:link>
    </gsf:when>
    <gsf:when test='equals' test-value='Images'>
      <gsf:link type="source"><gsf:metadata name="thumbicon"/></gsf:link>
    </gsf:when>
    <gsf:when test='equals' test-value='Supplementary'>
      <gsf:link type="source"><gsf:metadata name="srcicon"/></gsf:link> <gsf:link type="document">
<gsf:icon type="document"/></gsf:link>
    </gsf:when>
    <gsf:otherwise>
      <gsf:link type="document"><gsf:icon type="document"/></gsf:link>
    </gsf:otherwise>
  </gsf:switch>
</td>

```

to this:

```

<td valign="top">
  <gsf:switch>
    <gsf:metadata name="dc.Format"/>
    <gsf:when test="equals" test-value="Lyrics">
      <gsf:link type="document">
        <gsf:icon file="lyrics.gif" select="collection" />
      </gsf:link>
    </gsf:when>
    <gsf:when test="equals" test-value="Discography">
      <gsf:link type="document">
        <gsf:icon file="disc.gif" select="collection" />
      </gsf:link>
    </gsf:when>
    <gsf:when test="equals" test-value="Tablature">
      <gsf:link type="document">
        <gsf:icon file="tab.gif" select="collection" />
      </gsf:link>
    </gsf:when>
    <gsf:when test="equals" test-value="MARC">
      <gsf:link type="document">
        <gsf:icon file="marc.gif" select="collection" />
      </gsf:link>
    </gsf:when>
    <gsf:when test="equals" test-value="Images">
      <gsf:link type="source">
        <gsf:metadata name="thumbicon"/>
      </gsf:link>
    </gsf:when>
    <gsf:when test="equals" test-value="Supplementary">
      <gsf:link type="source">
        <gsf:metadata name="srcicon"/>
      </gsf:link>
    </gsf:when>
    <gsf:when test="equals" test-value="Audio">
      <gsf:link type="source">
        <gsf:switch>
          <gsf:metadata name="FileFormat"/>
          <gsf:when test="equals" test-value="MIDI">
            <gsf:icon file="midi.gif" select="collection" />
          </gsf:when>
          <gsf:otherwise>
            <gsf:metadata name="srcicon"/>
          </gsf:otherwise>
        </gsf:switch>
      </gsf:link>
    </gsf:when>
  </gsf:switch>
</td>

```

23. **Preview** your collection as before. Now different icons are used for discography, lyrics, tablature, and MARC metadata. Even MP3 and MIDI audio file types are distinguished.

Building a full-size version of the collection

24. To finish, let's now build a larger version of the collection. To do this:

- Close the current collection (**File** → **Close**).
- Start a new collection called *large beatles* (**File** → **New...**).
- Base this new collection on *small beatles*.
- Copy the content of *sample_files* → *beatles* → *advbeat_large* → *import* into this newly formed collection. Since there are considerably more files in this set of documents the copy will take longer.
- **Build** the collection and **preview** the result. (If you want the collection to have an icon, you will have to add it from the **Format** panel.)

Scanned image collection

Sample files: [niupepa.zip](#)

Devised for Greenstone version: 2.60|3.06

Modified for Greenstone version: 2.87|3.11

Here we build a small replica of Niupepa, the Maori Newspaper collection, using five newspapers taken from two newspaper series. It allows full text searching and browsing by title and date. When a newspaper is viewed, a preview image and its corresponding plain text are presented side by side, with a "go to page" navigation feature at the top of the page.

*The collection involves a mixture of plugins, classifiers, and format statements. The bulk of the work is done by **PagedImagePlugin**, a plugin designed precisely for the kind of data we have in this example. For each document, an "item" file is prepared that specifies a list of image files that constitute the document, tagged with their page number and (optionally) accompanied by a text file containing the machine-readable version of the image, which is used for full text searching. Three newspapers in our collection (all from the series "Te Whetu o Te Tau") have text representations, and two (from "Te Waka o Te Iwi") have images only. Item files can also specify metadata. In our example the newspaper series is recorded as **ex.Title** and its date of publication as **ex.Date**. Issue **ex.Volume** and **ex.Number** metadata is also recorded, where appropriate. This metadata is extracted as part of the building process.*

1. Start a new collection called **Paged Images** and fill out the fields with appropriate information: it is a collection sourced from an excerpt of Niupepa documents.
2. In the **Gather** panel, open the *sample_files* → *niupepa* → *sample_items* folder and drag the two subfolders into your collection on the right-hand side. A popup window asks whether you want to add **PagedImagePlugin** to the collection: click <Add Plugin>, because this plugin will be needed to process the item files.

***PagedImagePlugin** will process the item files, creating a document for each one with a separate section for each page listed. Thumbnail and screen-resolution sized images of each page image will be generated.*

3. Go to the **Create** panel, **build** the collection and **preview** the result. Search for "waka" and view one of the titles listed (all three appear as *Te Whetu o Te Tau*). Browse by **titles** and view one of the *Te Waka o Te Iwi* newspapers. Note that only the *Te Whetu o Te Tau* newspapers have text; *Te Waka o Te Iwi* papers don't.

This collection was built with Greenstone's default settings. You can locate items of interest, but the information is less clearly and attractively presented than in the full Niupepa collection.

Grouping documents by series title and displaying dates within each group

*Under **titles**, documents from the same series are repeated without any distinguishing features such as date, volume or number. It would be better to group them by series title and display other information within each group. This can be accomplished using the **-bookshelf_type** option to the **List** classifier, and tuning the classifier's format statement.*

4. In the **Design** panel, under the **Browsing Classifiers** section, delete the **List** classifier for **ex.Source**. This classifier is not much use.
5. Select the classifier for **dc.Title;ex.Title** and click <Configure Classifier...>. Set **bookshelf_type** to **always**. This will create a bookshelf for each Title in the collection. Note, setting this option to **duplicate_only** will only create a bookshelf when more than one document shares a Title.
6. Setting this List classifier's **partition_type_within_level** to **none** will further allow you to browse the few documents in this collection all in one page, instead of having the additional level of

browsing by starting letter, presented horizontally at the top. Click **OK** to finish configuring the classifier.

- Build** the collection, and **preview** the **titles** list.
- Now we change the format statement for **titles** to display more information about the documents. In the **Format Features** section of the **Format** panel, select the **dc.Title;ex.Title** classifier (CL1) in the **Choose Feature** list. Click **<Add Format>** to add this format statement to your collection. Edit the contents of the **dc.Title;ex.Title** classifier format statement by removing the following in the **documentNode** template:

```
<td valign="top">
  <gsf:link type="source">
    <gsf:choose-metadata>
      <gsf:metadata name="thumbicon"/>
      <gsf:metadata name="srcicon"/>
    </gsf:choose-metadata>
  </gsf:link>
</td>
<td valign="top">
  <gsf:link type="document">
    <xsl:call-template name="choose-title"/>
  </gsf:link>
  <gsf:switch>
    <gsf:metadata name="Source"/>
    <gsf:when test="exists">
      <br/>
      <i>
        <gsf:metadata name="Source"/>
      </i>
    </gsf:when>
  </gsf:switch>
</td>
```

In its place, insert the following (which can be copied from *sample_files* → *niupepa* → *formats* → *titles_tweak_gs3.txt*):

```
<td valign="top">
  Volume: <gsf:metadata name="Volume"/> Number: <gsf:metadata name="Number"/> Date: <gsf:metadata
format="formatDate" name="Date"/>
</td>
```

Then, in the **classifierNode** template for **VLists**, replace the contents of the final `<td>...</td>` table cell element with the following which can also be copied from the file *titles_tweak_gs3.txt*:

```
<td valign="top">
  <xsl:call-template name="choose-title"/> (<gsf:metadata name="numleafdocs"/>)
</td>
```

- Preview** the new **titles** list.

As a consequence of using the **bookshelf_type** option of the **List** classifier, bookshelf icons appear when titles are browsed. This revised format statement has the effect of specifying in brackets how many items are contained within a bookshelf for classifier nodes. For document nodes, Title is not displayed. Instead, Volume, Number and Date information are displayed.

In the **Design** Pane, under **Browsing Classifiers**, configure the **titles List** classifier. Tick **sort_leaf_nodes_using** and set the metadata to `ex.Volume|ex.Number`. Rebuilding now will ensure the *ex.Volume* Number of each newspaper are listed in numeric order. This has the effect of also sorting the *ex.Number* value for each *ex.Volume*.

Browsing documents by Date.

- Back in the **Design** panel, under the **Browsing Classifiers** section, add a **DateList** classifier, leaving its **metadata** option set to **ex.Date**.

11. In the **Format Features** section of the **Format** panel, select **DateList** in the **Choose Feature** list, and click **<Add Format>** to add this format statement to your collection. In the **documentNode** template of the new **DateList** feature, replace:

```
<gsf:switch>
  <gsf:metadata name="Source"/>
  <gsf:when test="exists"/>
    <br/>
    <i>(<gsf:metadata name="Source">)</i>
  </gsf:when>
</gsf:switch>
```

with this, which can also be copied from the file *titles_tweak_gs3.txt*:

```
</td>
<td valign="top">
  <xsl:call-template name="choose-date"/>
```

12. The above makes reference to the "choose-date" template which we're about to create: select the **global** format statement in the **Format Features** and append the following definition for the "choose-date" template (which can be copied from *sample_files* → *niupepa* → *formats* → *global_tweak_gs3.txt*):

```
<gsf:template name="choose-date">
  <gsf:choose-metadata>
    <gsf:metadata format="formatDate" name="dc.Date"/>
    <gsf:metadata format="formatDate" name="exp.Date"/>
    <gsf:metadata format="formatDate" name="ex.dc.Date"/>
    <gsf:metadata format="formatDate" name="Date"/>
    <gsf:default>undated</gsf:default>
  </gsf:choose-metadata>
</gsf:template>
```

13. **Build** the collection, and **preview** the **dates** list.

14. The **dates** list groups documents by date. Greenstone's internal date format is YYYYMMDD, for example 18580601, and this is crucial for the **DateList** classifier to correctly parse date metadata and generate an ordered date list. However, the date has been made to look nice by adding a **format="formatDate"** attribute to Date metadata in the format statement.

15. Back in the **global** format statement, edit the display of the date metadata to remove the special date-formatting, so that it looks like:

```
<gsf:template name="choose-date">
  <gsf:choose-metadata>
    <gsf:metadata name="dc.Date"/>
    <gsf:metadata name="exp.Date"/>
    <gsf:metadata name="ex.dc.Date"/>
    <gsf:metadata name="Date"/>
    <gsf:default>undated</gsf:default>
  </gsf:choose-metadata>
</gsf:template>
```

Refresh in the web browser to view the new **dates** list. The dates are now shown in internal format.

16. Change the format statement back to reinstate the nicely formatted dates. This can be done by selecting **global** in assigned format statements panel and clicking **<Undo>** one or more times.

Searching at page level

17. The newspaper documents are split into sections, one per page. For large documents, it is useful to be able to search on sections rather than documents. This allows users to more easily locate the relevant information in the document.
18. Go to the **Search Indexes** section of the **Design** panel. Remove the **ex.Source** index and, if not already the case, check the **section** checkbox to build the indexes on section level as well as

document level. Make section level the default by selecting its **Default** radio button.

19. Set the display text used for the level drop-down menu by going to the **Search** section on the **Format** panel. Set the document level text to "newspaper", and the section level text to "page".
20. **Build** and **preview** the collection.

Choose **fielded search**. Compare searching at "newspaper" level with searching at "page" level. A useful search term for this collection is "aroaha".

You might notice that newspaper level search results only display the newspaper Title, and not any volume information, while page level search results only show the Title of the page (the page number), and not the Title of the newspaper. We'll modify the format statement to show Volume and Number information, and for page results, the newspaper title as well as the page number.

21. In the **Format Features** section of the **Format** panel, select **Search** in **Choose Feature** to adjust how search results are displayed.

The extracted Title for the current section is specified as `<gsf:metadata name="Title"/>` while the Title for the parent section is `<gsf:metadata name="Title" select="parent"/>` (if using metadata assigned at the document or root level, this would be `<gsf:metadata name="Title" select="root"/>`). Since the same **search** format statement is used when searching both whole newspapers and newspaper pages, we need to make sure it works in both cases.

Replace the lines comprising the final `<td>...</td>` table cell element with the following format statement (it can be copied and pasted from the file *sample_files* → *niupepa* → *formats* → *search_tweak_gs3.txt*):

```
<td>
  <gsf:switch>
    <gsf:metadata name="Title" select="parent"/>
    <gsf:when test="exists">
      <gsf:metadata name="Title" select="parent"/> Volume:<gsf:metadata name="Volume"
select="parent"/> Number:<gsf:metadata name="Number" select="parent"/> - Page:<gsf:metadata
name="Title"/>
    </gsf:when>
    <gsf:otherwise>
      <gsf:metadata name="Title"/> Volume:<gsf:metadata name="Volume"/> Number:<gsf:metadata
name="Number"/>
    </gsf:otherwise>
  </gsf:switch>
  <br/>
  <i>
    <gsf:choose-metadata>
      <gsf:metadata name="Date" format="formatDate" />
      <gsf:metadata name="Date" select="parent" format="formatDate" />
      <gsf:metadata name="Date" select="root" format="formatDate" />
      <gsf:default>undated</gsf:default>
    </gsf:choose-metadata>
  </i>
</td>
```

Preview the search results. Items display newspaper Title, Volume, Number and Date, and pages also display the page number.

*The collection you have just built involves a fairly complex document structure. There are two series of newspapers, **Te Waka** and **Te Whetu**.*

*In the **Te Waka** series there are two actual newspapers, Volume 1 Numbers 1 and 2. Number 1 has 4 pages, numbered 1, 2, 3, 4; Number 2 has 4 pages, numbered 5, 6, 7, 8. The page numbers increase consecutively through each volume, despite the fact that the volume is divided into different Numbers. Each page in the **Te Waka** series is represented by a single file, a GIF image of the page.*

*The **Te Whetu** series has three actual newspapers, Volume 1 Numbers 1, 2, and 3. Number 1 has 4 pages,*

numbered 1, 2, 3, 4; Number 2 has 5 pages, numbered 5, 6, 7, 8, 9; Number 3 has 5 pages, numbered 10, 11, 12, 13, 14. Again the page numbers increase consecutively through each volume. Each page in this series is represented by two files, a GIF image of the page and a text file containing the OCR'd text that appears on it.

The key to this structure is in the respective .item files. Here is a synopsis of the information they contain:

```
(9-1-1) Te Waka Volume 1 Number 1
  p.1 gif
  p.2 gif
  p.3 gif
  p.4 gif
(9-1-2) Te Waka Volume 1 Number 2
  p.5 gif
  p.6 gif
  p.7 gif
  p.8 gif
(10-1-1) Te Whetu Volume 1 Number 1
  p.1 gif text
  p.2 gif text
  p.3 gif text
  p.4 gif text
(10-1-2) Te Whetu Volume 1 Number 2
  p.5 gif text
  ...
  p.9 gif text
(10-1-3) Te Whetu Volume 1 Number 3
  p.10 gif text
  ...
  p.14 gif text
```

Advanced scanned image collection

Prerequisite: [Scanned image collection](#)

Sample files: [niupepa.zip](#)

Devised for Greenstone version: 2.70|3.06

Modified for Greenstone version: 2.87|3.11

In this exercise we build upon the collection created in the [Scanned image collection](#) exercise. We add a new newspaper by creating an item file for it, add a new newspaper using the extended XML item file format, and modify the formatting.

Adding another newspaper to the collection

Another newspaper has been scanned and OCREd, but has no item file. We will add this newspaper into the collection, and create an item file for it.

1. In the Librarian Interface, open up the Paged Image collection that was created in exercise [Scanned image collection](#) if it is not already open (**File** → **Open...**).
2. In the **Gather** panel, add the folder *sample_files* → *niupepa* → *new_papers* → *12* to your collection.

Inside the **12** folder you can see that there are 4 images and 4 text files.

3. Create an item file for the collection. Have a look at an existing item file to see the format. Start up a text editor (e.g. WordPad) to open a new document. Add some metadata. The **Title** for this newspaper is "Te Haeata 1859-1862". The **Volume** is 3, **Number** is 6, and the **Date** is "18610902". (Greenstone's date format is **yyyymmdd**.) Metadata must be added in the form:

```
<Metadata name>Metadata value
```

For this document, the metadata looks like:

```
<Title>Te Haeata 1859-1862
<Date>18610902
<Volume>3
<Number>6
```

4. For each page, add a line in the file in the following format:

```
pagenum:imagefile:textfile
```

For example, the first page entry would look like

```
1:images/12_3_6_1.gif:text/12_3_6_1.txt
```

Note that if there is no text file, you can leave that space blank. You need to add a line for each page in the document. Make sure you increment the page number as well as the image number for each line. (The full text for this file can be copied from *sample_files* → *niupepa* → *formats* → *12_3_6.item*.)

5. Save the file using **Filename** *12_3_6.item*, and save as a plain text document. (If you are using Windows, make sure the file doesn't accidentally end up getting saved as *12_3_6.item.txt*.) Back in the **Gather** panel of the Librarian Interface, locate the new file in the **Workspace** tree, and drag it into the collection, adding it into the **12** folder.
6. **Build** the collection and **preview**. Check that your new document has been added.

XML based item file

There are two styles of item files. The first, which was used in the previous section, uses a simple text based format, and consists of a list of metadata for the document, and a list of pages. This format allows specification of document level metadata, and a single list of pages.

The second style is an extended format, and uses XML. It allows a hierarchy of pages, and metadata specification at the page level as well as at the document level. In this section, we add in two newspapers which use XML-based item files.

7. In the **Gather** panel, add the folder *sample_files* → *niupepa* → *new_papers* → *xml* (you need to add the **xml** folder, not the **23** folder) to your collection.
8. Open up the file *xml* → *23* → *23_2.item* and have a look at the XML. This is **Number 2** of the newspaper titled *Matariki 1881*. The contents of this document have been grouped into two sections: **Supplementary Material**, which contains an **Abstract**, and **Newspaper Pages**, which contains the page images (and OCR text).
9. **Build** and **preview** the collection. The xml style items have been included.

Open Archives Initiative (OAI) collection

Sample files: [oai.zip](#)

Devised for Greenstone version: 2.60|3.06

Modified for Greenstone version: 2.87|3.11

This exercise explores service-level interoperability using the Open Archive Initiative Protocol for Metadata Harvesting (OAI-PMH). So that you can do this on a stand-alone computer, we do not actually connect to the external server that is acting as the data provider. Instead we have provided an appropriate set of files that take the form of XML records produced by the OAI-PMH protocol.

One of Greenstone's documented example collections is sourced over OAI. This exercise takes you through the steps necessary to reconstruct it. You may wish to take a look at the documented example collection OAI demo now to see what this exercise will build.

1. Start a new collection called **OAI Service Provider**. Fill out the fields with appropriate information.
2. In the **Gather** panel, locate the folder *sample_files* → *oai* → *sample_small* → *oai*. Drag this folder into the collection and drop it there.
3. If the **OAIPlugin** is not already in the **Document Plugins** list, then during the copy operation a popup window may appear asking whether to add **OAIPlugin** to the list of plug-ins used in the collection, because the Librarian Interface has not found an existing plug-in that can handle this file type. Press the **<Add Plugin>** button to include it.

The files for this collection consist of a set of images (in *JCDLPICS* → *srcdocs*) and a set of OAI records (in *JCDLPICS*) which contain metadata for the images.

When files are copied across like this, the Librarian Interface studies each one and uses its filename extension to check whether the collection contains a corresponding plug-in. No plug-in in the list is capable of processing the OAI file records that are copied across (they have the file extension .oai), so the Librarian Interface prompts you to add the appropriate plug-in.

*Sometimes there is more than one plug-in that could process a file—for example, the .xml extension is used for many different XML formats. The popup window, therefore, offers a choice of all possible plug-ins that matched. It is normally easy to determine the correct choice. If you wish, you can ignore the prompt (click **<Don't Add Plugin>**), because plug-ins can be added later, in the **Document Plugins** section of the **Design** panel.*

4. You will need to specify which document the OAI metadata values should be attached to. In the **Design** panel, select the **Document Plugins** section, then select the **OAIPlugin** and click **<Configure Plugin...>**. Locate the **document_field** option in the popup window and type **ex.dc.Identifier** (it may not be available in the drop-down list until after building). Click **<OK>**.
5. You also need to configure the image plug-in. Select the **ImagePlugin** line in the **Document Plugins** section and click **<Configure Plugin...>**. In the resulting popup window locate the **screenviewsize** option, switch it on, and type the number **300** in the box beside it to create a screen-view image of 300 pixels. Click **<OK>**.
6. Now switch to the **Create** panel and **build** and **preview** the collection.

OAIPlugin will process the OAI records, and assign metadata to the images, which are processed by **ImagePlugin**.

Like other collections we have built by relying on Greenstone defaults, the end result is passable but can

be improved. The next steps refine the collection using the metadata harvested by OAI-PMH into the .oai files.

7. In the **Browsing Classifiers** section of the **Design** panel, delete the two **List** classifiers (**dc.Title;ex.Title** and **ex.Source**).
8. Add an **AZCompactList** classifier based on **ex.dc.Subject** metadata. Configure it with **Subjects** as the **buttonname**.
9. Now add an **AZCompactList** classifier based on **ex.dc.Description** metadata. In its configuration panel set **mingroup** to **2**, **mincompact** to **1**, **maxcompact** to **10** and **buttonname** to **Captions**.

Setting **mingroup** to 2 will mean that two or more documents with the same description will be grouped into a bookshelf; the default **mingroup** of 1 means that every document will get a bookshelf. **mincompact** and **maxcompact** control how many documents are grouped into each section of the horizontal A-Z list. In this case, each group can have as few as one document, and no more than ten.

10. In the **Search Indexes** section of the **Design** panel, delete all indexes and add a new one based on **ex.dc.Description** metadata. Set the **Display text** for the **ex.dc.Description** index by going to the **Search** section in the **Format** panel and changing its label to "descriptions".
11. **Build** the collection and **preview** it.

Tweaking the presentation with format statements

12. In the **Format** panel, select **Format Features**. First, in the **browse** format statement, replace the templates for **documentNode** and **classifierNode** for **VLists** with the following (which can be copied from *sample_files* → *oai* → *format_tweaks* → *browse_tweak.txt*).

```
<gsf:template match="documentNode">
  <td valign="top">
    <gsf:link type="document">
      <gsf:metadata name="thumbicon"/>
    </gsf:link>
  </td>
  <td valign="middle">
    <i>
      <gsf:metadata name="ex.dc.Description"/>
    </i>
  </td>
</gsf:template>

<gsf:template match="classifierNode[@classifierStyle = 'VList']">
  <td valign="top">
    <gsf:link type="classifier">
      <gsf:icon type="classifier"/>
    </gsf:link>
  </td>
  <td valign="top">
    <xsl:call-template name="choose-title"/>
  </td>
</gsf:template>
```

Also replace the **documentNode** template of the **Search** format statement with the **documentNode** template above.

This format statement customizes the appearance of vertical lists such as the search results and captions lists to show a thumbnail icon followed by Description metadata.

13. Next, select the **display** format statement from the **Format Features** list and add the following to create a custom **documentHeading** format statement:

```
<gsf:template name="documentHeading">
```

```

<h3>
  <gsf:metadata name="ex.dc.Subject"/>
</h3>
</gsf:template>

```

By default **documentHeading** displays the document's **ex.Title** metadata. In this particular set of OAI exported records, titles are filenames of JPEG images, and the filenames are particularly uninformative (for example, 01dla14). You can see them in the **Enrich** panel if you select an image in *oai* → *JCDLPICS* → *srcdocs* and check its **ex.Source** and **ex.Title** metadata. The above format statement displays **ex.dc.Subject** metadata instead.

14. Still in the **display** format in the **Format Features** list, add the following (which can be copied from *sample_files* → *oai* → *format_tweaks* → *document_content.txt*) to create a custom **documentContent** format statement:

```

<gsf:template name="documentContent">
  <table>
    <tr>
      <td colspan="2" align="center">
        <a<xsl:attribute name="href">
          <gsf:metadata name="ex.dc.OrigURL"/>
        </xsl:attribute>
          <gsf:metadata name="screenicon"/>
        </a>
      </td>
    </tr>
    <tr>
      <td>Caption:</td>
      <td><i><gsf:metadata name="ex.dc.Description"/></i><br/>
        <gsf:link type="source">
          original <gsf:metadata name="ImageWidth"/><gsf:metadata name="ImageHeight"/> <gsf:metadata
name="ImageType"/> available
        </gsf:link>
      </td>
    </tr>
    <tr>
      <td>Subject:</td>
      <td><gsf:metadata name="ex.dc.Subject"/></td>
    </tr>
    <tr>
      <td>Publisher:</td>
      <td><gsf:metadata name="ex.dc.Publisher"/></td>
    </tr>
    <tr>
      <td>Rights:</td>
      <td><gsf:metadata name="ex.dc.Rights"/></td>
    </tr>
  </table>
</gsf:template>

```

This format statement alters how the document view is presented. It includes a screen-sized version of the image that hyperlinks back to the original larger version available on the web. (Unfortunately, the original versions of the images in this sample collection are no longer available on the web. If you want the link to lead to the local copy of the full size image, then use `<gsf:link type="source">anchor text goes here</gsf:link>` instead of creating an `a href` with the value `<gsf:metadata name="ex.dc.OrigURL"/>`. The use of both is demonstrated in the above format statement, where the smaller screen-size image is hyperlinked to the original image's (defunct, in this case) URL denoted by `ex.dc.OrigURL`, whereas the link in the *Caption* section leads to the local copy. Image property information extracted from the image, such as width, height and type, is also displayed as a consequence of using the above format statement. It uses XSLT to construct the hyperlink which makes use of the greenstone metadata containing the link to the original image.

15. Format statements are processed by the runtime system, so the collection does not need to be rebuilt for these changes to take effect. Click **<Preview Collection>** to see the changes.

Setting up your Greenstone OAI Server

Prerequisite: [A simple image collection](#)

Devised for Greenstone version: 2.85|3.06

Modified for Greenstone version: 2.87|3.11

Greenstone 3 collections are available over OAI by default. Their `collectionConfig.xml` files already specify that each collection is OAI enabled, through use of an `OAI:PMH serviceRack` element. If you want to disable a collection from being accessible over OAI, edit the `OAI:PMH serviceRack` element in that collection's `collectionConfig.xml`. This tutorial will look at how to make an existing collection available over OAI and testing its accessibility by getting it validated against the Open Archives validator.

1. Start up the Greenstone 3 Server by going to Windows *Start* → *All Programs* → *Greenstone-3* → *Greenstone3 Server*.

Press the **Enter Library** button and you will end up on your Digital Library home page as usual. Adjust the URL so that instead of the *library* suffix, it says *oaiserver*.

The page that loads now will contain an error message (*badVerb*) saying that you've provided an illegal OAI verb. This is because the OAI specification requires you to provide more instruction in the URL as to what you want. The specification defines verbs and possible arguments to them.

A basic verb is *Identify*, which requests the OAI server to return some information about the OAI repository that it's serving. Adjust the URL once more by suffixing *?verb=Identify*, so that your URL now looks like:

```
http://<domain:port>/greenstone3/oaiserver?verb=Identify
```

Visiting this page now gives some information about your Greenstone OAI repository.

2. Although the data transmitted over OAI is in the form of XML, Greenstone uses a stylesheet to transform that XML response into a user-friendly, structured web page that you see when you perform the **Identify** request (as happens when you visit the *verb=Identify* response page). This allows *Identify* and other verbs in the OAI specification to be shown in the main Greenstone OAI Server pages as a row of links. You can see these verbs represented in the main Greenstone *oaiserver* (or *oaiserver?verb=Identify*) page as a row of links starting with "Identify", displayed at the top and in the lower part of the page.

Clicking on the links will execute that verb as a request and return the response from your Greenstone OAI server as a structured web page. Try clicking on all the button links.

3. OAI defines a concept called a **Set**. In Greenstone, the OAI Set concept is mapped to the practical Greenstone collection. The link to the *ListSets* verb will therefore request the Greenstone OAI server to list all the collections that have been enabled for OAI.

Click on the **ListSets** link and have a look.

The response page for the *ListSets* verb will show you that your **backdrop** collection (created in the **Simple image collection** tutorial) is one of the collections available over OAI in your Greenstone repository.

4. You will see a couple of buttons next to each collection (or **Set**) listed here. The first is **Identifiers** and the second **Records**. Click on the **Identifiers** button for the **backdrop** Set. This will list all the OAI identifiers of the documents contained in your OAI collection.
5. Click the browser Back button to get back to the ListSets page and press the **Records** button located

next to the backdrop collection.

If you had specified some Dublin Core (dc) metadata for each of the images in the **backdrop** collection, then the page that loads will display this information for each document in the collection (Set).

Greenstone 3's OAI implementation uses the OAI standard for Dublin Core, *oai_dc*, metadata format. By default, it maps all Dublin Core metadata you may have assigned to your collections into *oai_dc*. This default mapping is specified in the configurable file *resources\oai\OAIConfig-oaiserver.xml.in*, from which *web\WEB-INF\classes\OAIConfig-oaiserver.xml* is generated. If all (or most) of your collections will be using a different metadata format, you can edit the *resources\oai\OAIConfig-oaiserver.xml.in* file's `elementList` section to create mappings from the metadata fields you're using to the metadata fields in *oai_dc*.

You can also specify mappings at collection level, overriding the mappings in *OAIConfig-oaiserver.xml.in* on a per-collection basis. So if a collection specifies metadata for a different metadata set format from the default mappings in *OAIConfig-oaiserver.xml.in*, adjust the collection's *web\sites\localsite\collect\\etc\collectionConfig.xml* file to tell Greenstone how to map the metadata fields of your chosen metadata set format into the *oai_dc* Dublin Core metadata set supported by the Greenstone OAI server.

For instance, look in the **demo** collection's *collectionConfig.xml* file (*web\sites\localsite\collect\lucene-jdbm-demo\etc\collectionConfig.xml*) and scroll down to the definition for the `OAIIPMH ServiceRack`. Look at its `ListMetadataFormats` section containing element mappings, which will explain and provide an example for how to specify such an oai mapping from the *DLS* metadata format that the **demo** collection uses, to the Dublin Core (*oai_dc*) metadata used by Greenstone's OAI server. Its **dls.Organization** metadata is mapped to **oai_dc.publisher** using the following line in the *collectionConfig.xml* configuration file (note the use of case):

```
<element name="dc:publisher">
  <mapping elements="dls.Organization" />
</element>
```

Because the **backdrop** collection uses DC metadata, no mapping is required, as the default mappings from DC metadata to *oai_dc* are already specified in *OAIConfig-oaiserver.xml*.

Validating the Greenstone OAI server

In this section, you'll be testing that you've set up your Greenstone OAI server correctly so that it's accessible over OAI. For this part of the exercise, you need to be on a networked computer and your host computer needs to be visible to the outside world. (That is, when you provide the full name of your computer, someone else in the world should be able to find that computer by typing its URL into their browser's address field.) You can consult the Greenstone Wiki page [Windows 10 instructions to make your GS3 tomcat web server public](#), which contains some instructions to configure the Windows 10 firewall and set up port mapping.

We'll be using an external OAI client to access our up-and-running Greenstone OAI server. It's not just any OAI client either, but an OAI Server validator.

6. We want the Greenstone library to be accessible to the Open Archives Validator, however URLs that use `localhost` can only be accessed locally. Therefore, if your Greenstone server runs on `localhost` (as it does by default), then you will need to stop the Greenstone server, edit the `tomcat.server` property of your Greenstone installation's top-level file *build.properties* and set this property to your domain name or your machine's IP address. Finally, restart the Greenstone server.
7. For this exercise, we will be visiting the **Open Archives Validator**, for which your OAIserver needs to provide a valid email address. In a text editor, open up your Greenstone installation's

resources\oai\OAIConfig-oaiserver.xml.in file. Set the value of the `adminEmail` element to the email address where the validation results are to be sent. Also set the `OAI repositoryIdentifier` element. The structure of its value is like a domain name and needs to be of the form of `word-dot-extension`, such as "greenstone.org". For more information on the structure of its value, see <http://www.openarchives.org/OAI/2.0/guidelines-oai-identifier.htm>. (If you wanted to additionally test the behaviour of the `ResumptionToken` feature against the OAI Validator, you would set the `resumeAfter` element to a low value like 5).

8. The modifications to *OAIConfig-oaiserver.xml.in* need to be brought into effect, so quit and relaunch the Greenstone 3 server application if it is already running. Otherwise, go to **Start** → **Greenstone** → **Greenstone3 Server** to start it up. When the library home page opens in your browser, change the library suffix in the URL to `oaiserver`, which is the baseURL of your OAI Server and would be of the form `http://domain/greenstone3/oaiserver`. Copy this URL and visit <http://www.openarchives.org/Register/ValidateSite>.
9. The Open Archives Validator page will request the URL to your Greenstone OAI server. Paste the URL you have in your copy buffer into the field provided for this, and press the **Validate baseURL** button to start running the tests. You will be told to check the `adminEmail` address you provided to continue the remaining tests and to get the validation report. Follow the instructions in their email in order to do so.

If the validator does not recognise the URL, make sure you have given the full domain of your host machine rather than just the host name. If that URL is still not accepted, visit the *oaiserver.cgi?verb=Identify* page again and check this works. If it doesn't, it may be that your machine is not set up to be accessible to outside networks. Check your proxy settings, make sure you've set up port forwarding and that your firewall is not interfering.

Downloading over OAI

Prerequisite: [Setting up your Greenstone OAI Server](#)

Devised for Greenstone version: 2.60|3.06

Modified for Greenstone version: 2.87|3.11

*GLI can serve as an OAI client application: it can connect to a remote OAI server and retrieve metadata, even download documents. The tutorial [Open Archives Initiative \(OAI\) collection](#) did not obtain the data from an external OAI-PMH server. This missing step is accomplished either by running a command-line program or by using the **Download** panel in the Librarian Interface. This exercise explains how you would do this using both methods. In the previous exercise, we set up the Greenstone server to serve your Greenstone 3 collections over OAI. In this tutorial, we will use GLI to connect to that OAI server and download OAI metadata for the **Simple image collection** and even download its documents. The principle is the same if you wish to connect to other OAI servers.*

Downloading using the Librarian Interface

1. Quit any running Greenstone applications. Launch GLI. This should launch the Greenstone server as well, so that the OAI server is also up and running.
2. In GLI, go to the **Download** panel. To the left, choose **OAI** as the **Download Setting**.
3. On the right, set the **Source URL** field to contain the URL to your Greenstone OAI server. It would be of the form

`http://<hostname:portnumber>/greenstone3/oaiserver`

(If you set up your Greenstone 3 server to operate over https, then adjust the above URL to have https as prefix and to contain the associated https port number instead.)

Make sure that you can generally access this URL from your browser.

Visit the library home page, as this will load the greenstone collections, so that any associated files like images or pdf documents become accessible for download. (Without visiting the library home page, the collections would not be loaded and the images from the Simple Images collection, that we will be downloading below alongside the oai files, will not be available for download.)

4. If the server is not running on localhost and your computer is behind a firewall or proxy server, you may need to edit the proxy settings in the Librarian Interface. Click the **<Configure Proxy...>** button. Switch on the **Use proxy connection?** checkbox. Enter the proxy server address and port number in the **HTTP Proxy Host:** and **Port:** boxes. Further, if you set up your Greenstone to run over https (or more generally, if you will be downloading from https URLs), tick the box labelled "No certificate checking for HTTPS downloads". Click **<OK>** to get back to the **OAI** section of the **Download** panel.
5. If at this stage you were to press the **<Server Information>** (in the central row of buttons), a dialog will pop up with basic details about the OAI server. At the end, it will display the names of the sets available via that OAI Server. A `setSpec` and a `setName` property will be defined for each available set. In our example, *backdrop* (the Simple Image collection) would be listed as one of the `setNames` with its `setSpec` as *backdrop*. Press the **<close>** to close the **Server Information** dialog.
6. Tick the **Metadata prefix** checkbox as well as the **Restrict to set** checkbox. For the latter, type the `setSpec` value of *backdrop*. Then tick **Get document**. Also tick **Only include file types** and include *jpg* in the list of comma separated values for it so that it becomes

`jpg,doc,pdf,ppt`

Next, tick **Max records** and set it to 10. There will be 9 images in the collection, so we don't really need to set the Max records value, but this is a helpful feature that you can use when downloading from an OAI server.

7. Finally, click **<Download>**, located beside the **Server Information** button. If you have set proxy information in **Preferences...**, a popup will ask for your user name and password. Once the download has started, a progress bar appears in the lower half of the panel that reports on how the downloading process is doing. GLI will download oai metadata and, because we have ticked the **Get document** checkbox, it will also be retrieving actual documents, but not more than 10, because of the limit of 10 that we've placed on the number of records to download.
8. After a while, it will have finished downloading. Change to the **Gather** panel, and on the left-hand side, open up the **Downloaded Files** folder. This is where Greenstone stores files you downloaded using the **Download** panel. In this case, it will contain a folder wherein the oai metadata files and images that you've just downloaded from your own Greenstone OAI server is stored. These files can then be added to a collection, as will be covered later in this tutorial.

Downloading using the command line

For command line downloading to work, your computer must have a direct connection to the Internet—being behind a firewall may interfere with the ability to download the information. You will need to use the Librarian Interface for downloading if you are behind a firewall.

9. Close the Librarian Interface.
10. Start up the Greenstone server application.

Visit the library home page to load the greenstone collections including any associated files, as explained above.

11. If you're on Windows, open a DOS window to access the command-line prompt. This facility should be located somewhere within your **Start** → **Programs** menu, but details vary between different Windows systems. If you cannot locate it, select **Start** → **Run**, enter *cmd* in the popup window that appears and hit *Enter*.

If you're on Linux or Mac, open a terminal.

12. Before you start, you must set up your Greenstone environment in the terminal. In the DOS window or terminal, move to the home directory where you installed Greenstone. This is accomplished by something like:

```
cd C:\Users\\Greenstone
```

13. Type:

```
gs3-setup.bat
```

to set up the ability to run Greenstone command-line programs. On Linux/Mac, you would run `source ./gs3-setup.sh`.

14. If you set up your Greenstone to run over `https` or intend to use the *command line* to download from any URLs that begin with `https` instead of `http` or if nothing downloads in the steps hereafter, then you will further need to edit your Greenstone 3 installation's `gs2build/bin/linux/wgetrc` (if on Linux), `gs2build/bin/linux/wgetrc` (if on Windows) or `gs2build/bin/linux/wgetrc` (if on Mac) file as follows. Open the file in a text editor and change the line that says:

```
#check_certificate = off
```


to:

```
check_certificate = off
```

Removing the hash sign at the start of this line changes it from being a mere comment to activating the line. Save the edited file and close it. The effect of this step will be that downloading from `https` URLs will now succeed even when download commands are run from the command line.

GLI uses a perl script, `downloadfrom.pl`, to do the downloading. This can be run on the command line, outside of GLI.

*The `downloadfrom.pl` script can download using several different protocols. These are specified using the `-download_mode` option. To see the available options for download mode, run `perl -S downloadfrom.pl -h`. This shows that the current options are: **Web, MediaWiki, OAI, Z3950, SRW**. For OAI downloading, use `-download_mode OAI`.*

To see the options for downloading using the OAI mode, you can run `perl -S downloadinfo.pl OAIDownload`. The options are the same as you can see in the GLI OAI download panel.

15. We'll use the `set` and `max_records` OAI Download options to limit the number of OAI records downloaded from the **backdrop** collection at your Greenstone's OAI server again:

```
perl -S downloadfrom.pl -download_mode OAI -url http://<hostname:portnumber>/greenstone3/oaiserver
-set backdrop -max_records 15
```

The OAI records will be downloaded into the folder where the `downloadfrom.pl` script is run from. To change this, use the `-cache_dir full-path-to-folder` option and set its value to the full path of the destination folder you choose. (If you wanted to download the documents along with the records, then you would additionally pass in the `-get_doc` flag to the above command as well as the `-get_doc_exts` flag followed by a comma-separated list of file extensions like "jpg,pdf".)

```
perl -S downloadfrom.pl -download_mode OAI -url http://<hostname:portnumber>/greenstone3/oaiserver
-set backdrop -max_records 15 -get_doc -get_doc_exts "jpg,pdf" -cache_dir "<type-full-path-to-a-download-folder>"
```

You can import the downloaded documents into a new Greenstone collection and build them in the usual manner.

Building the downloaded documents in GLI

16. If you used GLI to download documents over OAI, as seen in the first part of the tutorial, you can find the downloaded items in the **Downloaded Files** folder in the filesystem view on the left side of the **Gather** panel.

If you used the command line to download documents, the downloaded files will be stored wherever you ran the `downloadfrom.pl` script from.

17. Open GLI, locate the files you downloaded over OAI and drag and drop these into a new Greenstone collection called **OAI Collection**.
18. Go to the **Design** panel, and configure the **OAIPlugin** by ticking its **no_cover_image** option, then click OK to close the plugin configuration dialog. Generally, Greenstone will look for any images that have an identical name to the primary document being processed and will associate the image with the document as being the document's cover image. Because the OAI files and the image documents downloaded over OAI have matching names, each image would get treated as the cover image for its associated OAI file. We don't want that behaviour here, so we turn on the **no_cover_image** option. This allows the OAIPlugin to attach the metadata of each OAI file with its associated image (treating the image as the primary document, instead of as a cover image), just as intended.

Note that this time, we don't configure the **OAIPlugin's document_field** option to **ex.dc.Identifier**, because the OAI files that have been downloaded over OAI have the associated image's document identifier stored in the **(ex.)gi.Sourcedoc** metadata field. You can see this if you open up any of the downloaded OAI files in a text editor. The **(ex.)gi.Sourcedoc** field is consulted by default when the Greenstone building process tries to identify what source document to attach the metadata in each OAI file to.

19. Switch to the **Create** panel and press the **build** button. During this stage, the **OAIPlugin** will extract the metadata in the *oai* files and attach them to the associated *jpg* files of the downloaded **backdrop** collection. You can see this once the collection has been built by switching to the **Enrich** panel and clicking on an *oai* file, as no metadata is set for such files. However, if you then click on a *jpg* file and scroll down, there will be metadata names that start with *ex.dc*. This refers to Greenstone-extracted Dublin Core metadata. **ex.dc.Description** and **ex.dc.Title** will be set to the values you had assigned the images in the tutorial **A Simple Image Collection**. Greenstone will have added additional *ex.dc* metadata in the form of **ex.dc.Identifier**, which is the source URL for this image.
20. If you wish, you can now set up this collection in a manner similar to how the **backdrop** collection was set up in [A simple image collection](#). Don't forget to copy in any specific format statements, adjust them to use the *ex.dc* metadata instead of *dc* metadata, then **rebuild** and **preview** the collection.

Using the UnknownConverterPlugin to make unsupported document formats searchable

Devised for Greenstone version: 2.88|3.09

Modified for Greenstone version: 2.87|3.11

This is an advanced tutorial, in that it not only supposes you have familiarised yourself with most of what you've learned in preceding tutorials, but that you're also comfortable with downloading and installing software from the web, and have a little familiarity with using image editing software.

*The **UnknownConverterPlugin** builds on the idea of the **UnknownPlugin**, in that it can be configured to handle documents of unknown format and file extension. It can also be made to handle documents with known file extensions in a custom manner.*

*The **UnknownConverterPlugin** extends the **UnknownPlugin**'s abilities by letting you launch a tool you have installed on your own PC that can be run from the commandline to convert from the "unknown" file format to either text, html or gif/jpg/png images, or a folder of these. If you know how to launch this tool from the commandline to do the conversion, then you would configure the **UnknownConverterPlugin** by supplying the file format (file extension) of the documents it should process, the expected output file format (text, html or paged images), and the tool's conversion command that the **UnknownConverterPlugin** should launch to perform the conversion. In place of the input file and the output file or folder, you provide placeholders in the command to run. Once configured, the **UnknownConverterPlugin** will be used during building to process documents that match the specified file format. It will launch the commandline conversion tool with the command provided, and the expected output files as specified can then be processed by Greenstone in the usual manner.*

*An example scenario would be if your collection contained djvu files, for which Greenstone provides no custom plugin. However, there's a free commandline tool available that can convert from djvu to one of the text based formats that Greenstone can process, text or html. So in this case, you could try using the **UnknownConverterPlugin** with the commandline tool on djvu files that you've gathered. The result should be that the djvu files in your Greenstone collection are now searchable.*

Working with DjVu documents in Greenstone

DjVu (pronounced like the French phrase *déjà vu*) is a [document format](#) suited for archiving digital documents. [DjVuLibre](#), which provides open source tools for processing DjVu documents, describes DjVu as

"a web-centric format and software platform for distributing documents and images. DjVu can advantageously replace PDF, PS, TIFF, JPEG, and GIF for distributing scanned documents, digital documents, or high-resolution pictures. DjVu content downloads faster, displays and renders faster, looks nicer on a screen, and consume less client resources than competing formats. DjVu images display instantly and can be smoothly zoomed and panned with no lengthy re-rendering. DjVu is used by hundreds of academic, commercial, governmental, and non-commercial web sites around the world."

In this part of the tutorial we'll see how to get Greenstone to not just include a collection's DjVu documents, but make them searchable too. There are several tools out there to convert a DjVu document into text or HTML. For instance, Linux users can install the *ocrodjvu* package and use its *djvu2hocr* tool to extract the text content in HTML format. Janusz S. Bien, a Greenstone user on the mailing list, has recommended it as being of possible use to Greenstone users, as it's a front-end to OCR programs. In this tutorial, however, we'll look at using *djvutxt* which is part of the DjVuLibre suite of tools and which is also available for other operating systems like Windows.

Extracting the text from DjVu documents with DjVuLibre's djvutxt

1. Start up GLI and create a new collection called *DjVu Collection*.
2. Visit the ['DjVu-Digital vs. "Super Hero" PDF' page](#). The page compares a PDF sample document to its equivalent DjVu version and provides download links for both.

Download their [sample DjVu document](#) (originally [here](#)) into your *DjVu Collection*'s **import** folder at *Greenstone* → *web* → *sites* → *localsite* → *collect* → *djvucoll* → *import*. If you're offline, you can also get this file from *sample_files* → *djvu* → *superhero.djvu*.

3. Back in GLI, in the **Collection** view of the **Gather** pane, right click and select **Refresh folder view**. You should now see your new document "superhero.djvu" ready to be built.
4. Head over to the **Create** pane and build the collection. The document isn't recognised. You can press Preview to confirm that there's nothing much to look at in this collection.

If you were to search through the **Design** pane's **Document Plugins** for a "DjVuPlugin", you wouldn't find one, because Greenstone hasn't got one. Greenstone knows about a lot of common formats, but there's a great many formats that different people like to work with that Greenstone knows nothing about and which Greenstone developers have not created a custom plugin for.

*You've already learnt about the **UnknownPlugin** in the Multimedia tutorial and know that it can be configured to process document formats for which Greenstone has no custom plugin. However, UnknownPlugin cannot index textual document formats that are unknown to Greenstone to make them searchable upon building, because it doesn't know anything about their internal structure and consequently doesn't know how to extract their text content.*

*This is where the **UnknownConverterPlugin** comes in. It builds on the idea of the UnknownPlugin, allowing you to work with document formats unknown to Greenstone. But it offers the additional advantage of being able to extract the text of the unknown document, depending on an important proviso: that you have a software tool installed on your machine, one that can be run readily from the commandline, which can perform the process of converting the unknown document format into text or HTML (or a series of images). If the tool can convert the document to text or HTML, Greenstone can proceed as usual to index the content to make it searchable on previewing.*

5. So in order to process the "superhero.djvu" document in our collection, such that its text content gets indexed for searching, we need to do a number of things: find out if there's a free djvu to text conversion tool out there, work out how to run it from the commandline and finally configure the UnknownConverterPlugin to automatically run this commandline tool for us, so Greenstone can take care of the rest.

We're in luck, because among the DjVu related tools that [DjVuLibre](#) provides is one called "djvutxt" that can perform the text extraction for us. DjVuLibre is available for Windows, Mac and Linux:

- DjVuLibre provides binary installers for [Windows](#) and [Mac](#). Grab the one for your operating system and install it somewhere sensible: somewhere you have permissions to install and run it from. On Windows, running the installer in the regular manner requires you to have admin permissions. If you don't have admin rights, you can run the installer as follows (instructions taken from [this superuser exchange](#)) to install DjVuLibre in a non-admin location. Use a text editor to create a file called `nonadmin.bat` (beware the file doesn't end up with an additional `.txt` extension when saving it). Copy and paste, or carefully type, the following text into the file, then save and close it:

```
cmd /min /C "set __COMPAT_LAYER=RUNASINVOKER && start "" %1"
```

Next, open up a File Explorer and drag and drop the DjVuLibre setup executable icon onto the icon of the new `nonadmin.bat` file, to run setup in a way that bypasses the admin privileges

usually required for a successful installation. When installing, you'll now finally be allowed to choose a custom install directory, instead of the installer choosing an off-limits admin location like `C:\Program Files (x86)` for you. So make sure to choose a location in your User area as install directory. Upon successful installation, you're given the option to launch DjVuLibre's *DjView* tool, which will open the DjVuLibre manual (in djvu format). In the left pane of DjView, you can see a listing of the various tools DjVuLibre is comprised of, and read up on them. You can also read about *djvutxt* or the other DjVu tools that DjVuLibre provides in their [documentation page](#), but for this tutorial, we'll just be using their *djvutxt* tool.

- To install the DjVu binary on the Mac: double click on the downloaded dmg file, open the loaded dmg's contents in Finder, then copy the `DjView.app` into your Mac's Applications folder. You can achieve the same through the command line with a command similar to the following line, which instructs the OS to copy across the `DjView.app` in your opened dmg file (of the djvu installer) into the Mac Applications folder:

```
cp -r /Volumes/DjVuLibre-3.5.28+DjView-4.12-universal-2/DjView.app /Volumes/Macintosh\
HD/Applications/.
```

- Some [Linux machines may even come pre-installed with DjVuLibre](#). If not, you can use a package manager to install it for you, or compile it up easily from [source](#) in the usual Unix manner, as explained below.
- If you're on a Unix (Linux or Mac) system where you don't have the permissions needed to install DjVuLibre, yet you *do* have the requisite compiler installed (gcc, g++ on Linux, Xcode on Mac), as well as having "autoconf" installed on the Mac, then you can compile it up from source code as follows: download the [source tarball](#) and untar this in a user location. Open a terminal and change directory into the untarred DjVuLibre source folder. Then run the following three commands in sequence, adjusting the `prefix` flag to start with the full path to your Greenstone installation:

```
./configure --prefix=/PATH/TO/YOUR/GS/djvulibre
make
make install
```

If compiling was successful, djvulibre binaries would have been generated inside your Greenstone installation's new `djvulibre/bin` folder, at `/PATH/TO/YOUR/GS/djvulibre/bin`. For this tutorial, the most important of these djvulibre binaries is *djvuxt*, which will now be located at `/PATH/TO/YOUR/GS/djvulibre/bin/djvutxt` on your Unix system.

6. The next step is to find out how to run DjVuLibre's *djvutxt* conversion tool from the commandline.

The general format of the command is

```
djvutxt input.djvu output.txt
```

Open a DOS prompt on Windows (to do so: press the letter *r* while holding down the *Windows key*—the key located between the Alt and Ctrl keys on your keyboard— to launch the Windows *Run* popup dialog, then type `cmd` into it and hit Enter), or a terminal on Mac/Linux and experiment to see what it takes to convert your Greenstone installation's `web/sites/localsite/collect/DjVuColl/superhero.djvu` file.

You may have to invoke *djvutxt* using its full filepath, in which case on Windows the command would look like:

```
C:\PATH\TO\YOUR\djvutxt C:\PATH\TO\YOUR\GS\web\sites\localsite\collect\DjVuColl\import\superhero.djvu
C:\PATH\TO\YOUR\GS\superhero.txt
```

while on Unix systems the command would look like:

```
/PATH/TO/YOUR/djvutxt /PATH/TO/YOUR/GS/web/sites/localsite/collect/DjVuColl/import/superhero.djvu
/PATH/TO/YOUR/GS/superhero.txt
```

If you're on a Mac and had installed DjView.app into your Mac Applications folder, then the command you run in the Mac Terminal would look something like:

```
/Volumes/Macintosh\ HD/Applications/DjView.app/Contents/MacOS/djvutxt /PATH/TO/YOUR/GS/web/sites
/localsite/collect/DjVuColl/import/superhero.djvu /PATH/TO/YOUR/GS/superhero.txt
```

If you compiled up djvulibre from source, djvutxt will be in /PATH/TO/YOUR/djvulibre/bin/djvutxt.

Once you have the command working, inspect the output file. You should see mostly legible text in it. Only when you've been able to successfully complete this step should you proceed to the next steps.

Processing DjVu documents with the UnknownConverterPlugin

7. Now that you know how to run the *djvutxt* conversion tool from the commandline, open up the DjVu Collection in GLI. Go into the **Design** pane's **Document Plugins** section and add a new **UnknownConverterPlugin** instance. (There's already one in the Document Plugin pipeline, but it is not set up for processing djvu files.) Press **<Configure Plugin...>** and set up the plugin as follows:

- set its `convert_to` field to `text`
- set its `mime_type` field to `image/vnd.djvu`, which is one of the [mime types for the DjVu format](#)
- set its `process_extension` to `djvu`
- Finally, copy the full *djvutxt* command you ran from the commandline and paste it into the UnknownConverterPlugin Configuration dialog's `exec_cmd` field. Keep the full path to the *djvutxt* binary, but replace the entire input filepath with the literal string `%%INPUT_FILE` and replace the output filepath with the literal string `%%OUTPUT`. Doing so means that when you build the collection, Greenstone will replace `%%INPUT_FILE` with each DjVu document in your collection that it needs to process, and will replace `%%OUTPUT` with the expected text output file of each document upon conversion by *djvutxt*.

If you have any spaces in any filepaths in your `exec_cmd`, make sure to always nest that entire filepath in escaped double quotes (`\`), so Greenstone can preserve the spaces in it.

If any filepaths, other than `%%INPUT_FILE` and `%%OUTPUT` are within your Greenstone installation, you can use the `%%GSDLHOME`, `%%GSDL3SRCHOME` and `%%GSDL3HOME` (the latter for Greenstone 3's web folder) as placeholders and write out your filepaths relative to this. For instance, if your DjVuLibre is installed in your Greenstone's `ext` subfolder, then you would start the filepath to *djvutxt* with `%%GSDL3SRCHOME/ext`.

The value for your `exec_cmd` field may look something like the following, if you have DjVuLibre installed in `C:\Program Files`. Note the escaped double quotes bookending the path to *djvutxt*, to protect spaces in its filepath:

```
"C:\Program Files\DjVuLibre\djvutxt\" %%INPUT_FILE %%OUTPUT
```

On Unix systems, adjust the command you ran in the command line to now leave out any backslash protecting spaces in the command's filepaths, but ensure you have escaped double-quotes around such filepaths containing spaces. For example,

```
/Volumes/Macintosh\ HD/Applications/DjView.app/Contents/MacOS/djvutxt /PATH/TO/YOUR/GS/web/sites
/localsite/collect/DjVuColl/import/superhero.djvu /PATH/TO/YOUR/GS/superhero.txt
```

becomes

```
"/Volumes/Macintosh HD/Applications/DjView.app/Contents/MacOS/djvutxt\" %%INPUT_FILE %%OUTPUT
```

8. Having sufficiently configured the UnknownConverterPlugin, click on the OK button close the

plugin's Configuration dialog. Move to the **Create** pane and build the collection. Your document has now been recognised. What's more, if you preview it and search for the term "Interoperability", a term that occurs in our collection's *superhero.djvu* document, you should now get a search result linking to that document. So Greenstone has successfully indexed the document's text, thanks to DjVuLibre's *djvutxt* tool extracting the text which got fed into the rest of Greenstone's building pipeline.

Associating an icon with DjVu documents in Greenstone

9. When previewing the search result, you may notice that there's no proper icon for the document *superhero.djvu*. The Greenstone extracted text variant of the document has an icon, a plain text one. However, the *superhero.djvu* has the "unknown document format" icon, the one with the question mark on it. We can change this.
10. Go back to the **Design** pane to configure your **UnknownConverterPlugin** once more. This time, enable the `srcicon` field and set its value to `icondjvu`.

This is a macroname we're just inventing, though we're following existing Greenstone convention in naming document icon macros, in that it's of the form "`icon<file-extension>`".

Click OK to close the UnknownConverterPlugin configuration dialog. Quit GLI, since there's a little more work to do.

11. Greenstone doesn't have an icon for DjVu documents, since it doesn't know about the format. If you Google for the djvu icon, you'd probably find the [Wikipedia page for it](#).

Save one of their DjVu icon images. Then open the image in Windows Paint or GIMP or another image editor, and use the application's scaling feature to scale the image's height or the width (whichever is greater) to anywhere between 26 and 32 pixels. Save the scaled image as a GIF file with the name "`idjvu.gif`", storing it in your Greenstone installation's `web/interfaces/default/images` folder. You can also use free online image resizing websites to carry out this step.

If you're working offline, you can get a resized and ready copy of the `idjvu.gif` file from *sample_files* → *djvu* → *idjvu.gif*. Put it in your Greenstone 3 installation's `web/interfaces/default/images` folder.

12. Greenstone knows nothing about the `icondjvu` macro we defined as the value for UnknownConverterPlugin's `srcicon` field, so we have to teach Greenstone about this new macro. Use a text editor to open your Greenstone 3's `web/sites/localsite/siteConfig.xml` file.

Locate the line

```
<replace macro="_iconunknown_" scope="metadata" text="&lt;img src='interfaces/default/images/iunknown.gif' border='0'&gt;" resolve="false"/>
```

Add a similar line above or below it and adjust it to say:

```
<replace macro="_icondjvu_" scope="metadata" text="&lt;img src='interfaces/default/images/idjvu.gif' border='0'&gt;" resolve="false"/>
```

Save the file.

The above has now associated the icon image we want appearing for the djvu document with the macro we defined for the `srcicon` field in UnknownConverterPlugin's configuration.

13. Restart GLI, which will restart the Greenstone server, reloading the `siteConfig.xml` you have just edited. Rebuild the DjVu Collection again and preview it. This time, when you browse the collection, you should see the djvu icon appearing in place of the unknown icon for your DjVu

document.

14. Having designed your collection to handle DjVu documents, you can now add any other documents, including more DjVu documents. Greenstone should now be able to index the text content of DjVu documents in the collection to make them searchable, in all instances where text can be successfully extracted from them by `djvutxt`.

Make the search format statement look like below (you can copy it from *sample_files* → *djvu* → *formats* → *format_tweaks.txt*), then try searching:

```
<gsf:template match="documentNode">
  <td valign="top">
    <gsf:link type="document">
      <gsf:icon type="document"/>
    </gsf:link>
  </td>
  <td valign="top">
    <gsf:link type="source">
      <gsf:choose-metadata>
        <gsf:metadata name="thumbicon"/>
        <gsf:metadata name="srcicon"/>
      </gsf:choose-metadata>
    </gsf:link>
  </td>
  <td>
    <gsf:link type="document">
      <xsl:call-template name="choose-title"/>
    </gsf:link>
    <gsf:switch>
      <gsf:metadata name="equivDocLink"/>
      <gsf:when test="exists">
        Also available as: <gsf:metadata name="equivDocLink"/><gsf:metadata name="equivDocIcon"/>
      </gsf:when>
    </gsf:switch>
  </td>
</gsf:template>
```


Use METS as Greenstone's Internal Representation

Devised for Greenstone version: 2.60

Modified for Greenstone version: 2.87

1. In the Greenstone Librarian Interface, open up one of your existing collections, for example the **Small HTML Collection** collection.

*To be able to substitute **GreenstoneMETSPlugin** for **GreenstoneXMLPlugin** you need to be in **Expert mode**.*

2. Click **File** → **Preferences...** → **Mode** and change to **Expert** mode.
3. Switch to the **Design** panel and select **Document Plugins**. Remove **GreenstoneXMLPlugin** from the list of plug-ins and add **GreenstoneMETSPlugin**, with the default configuration options. Move this plugin to where **GreenstoneXMLPlugin** was (just below **ZipPlugin**).
4. Now change to the **Create** panel, locate the options for the import process to the left (labelled **Import Options**) and set **saveas** to **GreenstoneMETS**. Import options are not available unless you are in **Expert** mode.
5. **Rebuild** the collection. If you choose to preview it, things should look as before.
6. In your file browser, locate the *archives* folder for the collection you are working with (in *Greenstone3* → *web* → *sites* → *localsite* → *collect* → *<collname>* → *archives*). For each document in the collection, Greenstone has generated two files: *docmets.xml*, the core METS description, and *doctxt.xml*, a supporting file. (Note: unless you are connected to the Internet you may be unable to view *doctxt.xml* in your web browser, because it refers to a remote resource.) Depending on the source documents there may be additional files, such as the images used within a web page. One of METS' many features is the ability to reference information in external XML files. Greenstone uses this to tie the content of the document, which is stored in the external XML file *doctxt.xml*, to its hierarchical structure, which is described in the core METS file *docmets.xml*.

Moving a collection from DSpace to Greenstone

Sample files: [dspace.zip](#)

Devised for Greenstone version: 2.60

Modified for Greenstone version: 2.87

1. Start a **new collection** called **StoneD** and fill out its fields appropriately.
2. In the **Design** panel add **DSpacePlugin**. Leave the plugin options at their defaults and press **<OK>**.
3. Using the up arrow, **move** the position of **DSpacePlugin** to the top of the list (above **GreenstoneXMLPlugin**).
4. In the **Gather** panel, locate the folder *sample_files* → *dspace*. It contains five example items exported from a DSpace institutional repository. Copy them into your collection by dragging them over to the right-hand side of the panel. Cancel out of any dialog offering to add plugins.
5. **Build** the collection and **preview** it to see the basic defaults exhibited by a DSpace collection.

*If you browse by **titles**, you will find 7 documents listed, though only 5 items were exported from DSpace. Two of the original items had alternative forms in their directory folder. The DSpace plug-in options control what happens in such situations: the default is to treat them as separate Greenstone documents.*

*Below we use a plug-in option (**first_inorder_ext**) to fuse the alternative forms together. This option has the effect of treating documents with the same filename but different extensions as a single entity within a collection. One of the files is viewed as the primary document—it is indexed, and metadata is extracted from it if possible—while the others are handled as "associated files."*

*The **first_inorder_ext** option takes as its argument a list of file extensions (separated by commas): the first one in the list that matches becomes the primary document.*

6. Back in the **Design** panel's **Document Plugins** section, select **DSpacePlugin** and click **<Configure Plugin...>**. Switch on its configuration option **first_inorder_ext**. Set its value to "pdf,doc,rtf".
7. **Build** and **preview** the collection.

There are now only 5 documents, because only one version of each document has been included—the primary version.

Adding indexing and browsing capabilities to match DSpace's

The DSpace exported files contain Dublin Core metadata for title and author (amongst other things).

8. In the **Design** panel, select **Search Indexes**. Delete the **ex.Source** index, and add one for **ex.dc.Contributor**. Rename the **ex.dc.Contributor** index by going to the **Search** section in the **Format** panel. Select this index and change its value to *contributors*.
9. Go back to the **Design** panel, select **Browsing Classifiers**. Select the **ex.Source List** classifier and click **<Configure Classifier...>**. Change the **metadata** option to **ex.dc.Contributor**. Activate the **bookshelf_type** option and set its value to **always**. If not already active, activate the **partition_type_within_level** option. Then set it to **none**. Finally, activate **buttonname** and set this to **Contributors**. Click **<OK>** to close the dialog.
10. Now select the **Format Features** section of the **Format** panel, and select the **browse** format statement in the list of assigned format statements. Add the following text before the final **</td>** of the **documentNode** template:

```

<gsf:switch>
  <gsf:metadata name="equivDocLink"/>
  <gsf:when test="exists">
    <br/>Also available as:
    <gsf:metadata name="equivDocLink"/>
    <gsf:metadata name="equivDocIcon"/>
    <gsf:metadata name="/equivDocLink"/>
  </gsf:when>
</gsf:switch>

```

11. Also, let's add a format statement for the classifier based on **ex.dc.Contributor** metadata. In the **Choose Feature** menu (under **Format Features** on the **Format** panel), select the item that starts with:

CL2: List -metadata ex.dc.Contributor

12. Click **<Add Format>**. Replace

```
<xsl:call-template name="choose-title"/>
```

with

```
<gsf:metadata name="ex.dc.Title"/>
```

Then scroll down to the **classifierNode** template for **VLists**. Here, replace:

```
<gsf:metadata name="Title"/>
```

with

```
<gsf:metadata name="Title"/> (<gsf:metadata name="numleafdocs"/>)
```

This will display the number of documents for each bookshelf in the *Contributors* classifier. And for individual documents within each bookshelf, it will display the **ex.dc.Title**.

13. Make the same change to **CL2's documentNode** template as you did above for the **browse** format statement's **documentNode** template, by adding the following text before the final `</td>` of **CL2's documentNode** template:

```

<gsf:switch>
  <gsf:metadata name="equivDocLink"/>
  <gsf:when test="exists">
    <br/>Also available as:
    <gsf:metadata name="equivDocLink"/>
    <gsf:metadata name="equivDocIcon"/>
    <gsf:metadata name="/equivDocLink"/>
  </gsf:when>
</gsf:switch>

```

14. Repeat the exact same step for the **search** format feature's **documentNode** template, so that it now also makes reference to `<gsf:equivDocLink/>`.

15. **Build** the collection once again and **preview** it.

There are still only 5 documents, but against some of the entries appears the line "Also available as:" followed by icons that link to the alternative representations.

Moving a collection from Greenstone to DSpace

Prerequisite: [Moving a collection from DSpace to Greenstone](#)

Devised for Greenstone version: 2.60|3.06

Modified for Greenstone version: 2.87|3.11

*In this exercise you export a Greenstone collection in a form suitable for DSpace. It is possible to do this from the Librarian Interface's **File** menu, which contains an item called **Export...**, that allows you to export collections in different forms. However, to gain a deeper understanding of Greenstone, we perform the work by invoking a program from the Windows command-line prompt. This requires some technical skill; if you are not used to working in the command-line environment we recommend that you skip this exercise.*

Using Greenstone from the command line

1. If you're on Windows, open a DOS window to access the command-line prompt. This facility should be located somewhere within your **Start** → **Programs** menu, but details vary between different Windows systems. If you cannot locate it and you are running **Windows XP**, select **Start** → **Run** and enter `cmd` in the popup window that appears. In **Windows Vista** or **Windows 7**, click the Start button and type `cmd` in the search box at the bottom of the Start menu. On **Windows 10**, hold down the Windows key (located between the Ctrl and Alt keys) and press `s` to bring up the **Search Windows** box and type `cmd` in it.

If you're on a Unix system, Linux or Mac, open a terminal.

2. In the DOS window or unix terminal, move to the home directory where you installed Greenstone. On Windows, this is accomplished by something like:

```
cd C:\Users\\Greenstone
```

3. Type:

```
gs3-setup
```

to set up the ability to run Greenstone command-line programs.

On a Linux or Mac machine, you would similarly open a terminal, change directory into your Greenstone installation's top-level folder and type:

```
source ./gs3-setup.sh
```

4. Change directory into the folder containing the Stoned collection you built in the last exercise.

```
cd web\sites\localsite\collect\stoned
```

5. Run the following command to export the collection using the DSpace import/export format:

```
perl -S export.pl -saveas DSpace -site localsite -removeold stoned
```

Exporting in Greenstone is an additive process. If you ran the `export.pl` command once again, the new files exported would be added—with different folder names—to those already in the export folder. For the kind of explorations we are conducting we might re-run the command several times. The `-removeold` option deletes files that have previously been exported.

6. This command has created a new subfolder, `web` → `sites` → `localsite` → `collect` → `stoned` → `export`. Use the file browser to explore it. In it are the files needed to ingest this set of documents into DSpace.

You could just as well run the `export.pl` command on a different Greenstone collection and transfer the output to a DSpace installation by using DSpace's batch-import facility.

Editing metadata sets

Devised for Greenstone version: 2.70w|3.06

Modified for Greenstone version: 2.87|3.11

GEMS (Greenstone Editor for Metadata Sets) can be used to modify existing metadata sets or create new ones. GEMS is launched from the Librarian Interface when you want to create a new metadata set, or edit an existing one. In this exercise, we run GEMS outside of the Librarian Interface.

Running GEMS

1. Start the Greenstone Editor for Metadata Sets (GEMS)

Start → **All Programs** → **Greenstone-3.11** → **Greenstone Editor for Metadata Sets (GEMS)**

(If you're on a Unix system, use a terminal to run the `gli/gems.sh` start-up script. On a Mac, you can also double-click on the `green_gems` application icon that's located in your Greenstone installation folder in order to launch GEMS.)

2. GEMS starts up with no metadata set loaded. You can start a new set, or open an existing one, from the **File** menu.

Creating a new metadata set

3. In this exercise, we will create a new metadata set. In order to save time, we will base it on an existing one: Development Library Subset. From the **File** menu, select **File** → **New...** A popup window appears: **New Metadata Set**. Fill in the fields. Use "My Metadata Set" for the **Metadata set title**:, "my" for the **Metadata set namespace**:, and select "Development Library Subset Example Metadata" from the **Base this metadata set on**: drop down list. Click **<OK>**.
4. The new metadata set will be displayed. The left hand side lists the elements (and sub-elements, if any) for the set, and the right hand side displays the set or element attributes. Since the new set was based on the Development Library Subset metadata set, it already contains all the elements from that set.

Adding a new element to a metadata set

5. Right click on **My Metadata Set** in the left hand tree (or in the blank space in the left hand side) and choose **Add Element** from the menu that appears. In the popup window, type "Category" for the new element name, and click **<OK>**. The new element will appear in the list.
6. In the right hand side, the default attributes will appear for the new element. "Label" and "definition" are used in the Librarian Interface when displaying metadata elements and their descriptions (the "definition" is shown as additional text for the element). These attributes can be set in multiple languages.
7. Save the new metadata set by **File** → **Save**, then close the GEMS by **File** → **Exit**.

Building and searching with different indexers

Devised for Greenstone version: 2.70w|3.06

Modified for Greenstone version: 2.87|3.11

Greenstone supports three indexers **MG**, **MGPP** and **Lucene**.

MG is the original indexer used by Greenstone which is described in the book "**Managing Gigabytes**". It does section level indexing and compression of the source documents. **MG** is implemented in C.

MGPP is a re-implementation of **MG** that provides word-level indexes and enables proximity, phrase and field searching. **MGPP** is implemented in C++.

Lucene (<http://lucene.apache.org/>) is a java-based, full-featured text indexing and searching system developed by Apache. It provides a similar range of search functionality to **MGPP** with the addition of single-character wildcards and range searching. It was added to Greenstone to facilitate incremental collection building, which **MG** and **MGPP** can't provide, and is the default indexer for new collections.

Build with Lucene

1. Start a new collection (**File** → **New...**) called **Demo Lucene** and base it on the **Demo Collection (Lucene) [lucene-jdbm-demo]** collection, fill out its fields appropriately.
2. In the **Gather** panel, select **Documents in Greenstone Collections**, then select and open up *localsite* → *Demo Collection (Lucene) [lucene-jdbm-demo]*. This will display the documents in the **Greenstone demo** collection. Drag all 11 folders in the demo folder into the new collection.
3. Go to the **Enrich** panel, look at the metadata that is associated with each directory. Go to the **Search Indexes** section in the **Design** panel. Look at the top right area of the panel, where you will see that the **Lucene indexer** is already in use. This is because the **Demo Collection (Lucene) [lucene-jdbm-demo]** collection, which this collection is based on, uses the **Lucene indexer**.
4. **Build** and **preview** the collection.

Search with Lucene

5. Lucene provides single letter and multiple letter wildcards and range searching. The query syntax could be quite complicated (for more information, refer to <http://www.lucenetutorial.com/lucene-query-syntax.html>). Here we will learn how to use the wildcards while constructing queries.
6. * is a multiple letter wildcard. To perform a multiple letter wildcard search, append * to the end of the query term. For example, *econom** will search for words like *econometrics*, *economist*, *economical*, *economy*, which have the common part *econom* but different word endings.
7. To perform a single letter wildcard search, use ? instead. For example, search for *economi??* will only match words that have two and only two letters left after *economi*, such as *economist*, *economics*, and *economies*.
8. Please note that stopwords are used by default with Lucene indexer, so searching for words like *the* will match 0 documents. This is explained in a message on the search page, which states that such words are too common and were ignored.

Build with MGPP

9. Start a new collection called **Greenstone Demo MGPP** and also base it on the **Demo Collection (Lucene) [lucene-jdbm-demo]**.

10. In the **Gather** panel, drag all 11 folders from **Documents in Greenstone Collections** → *localsite* → *Demo Collection (Lucene) [lucene-jdbm-demo]* into the new collection.
11. In the **Search Indexes** section of the **Design** panel, you will notice that the active indexer is **Lucene**. Click the **Change...** button at the right top corner of the panel. A new window will pop up for selecting the Indexers. After selecting an indexer, a brief description will appear in the box below. Select MGPP and click **OK**. Please note that the **Assigned Indexes** section may have changed accordingly.
12. There are four options at the bottom of the panel, the first 3 of which are **Stem**, **Casefold** and **Accent fold**. Notice these three are enabled.
13. In the **Indexing Levels** section, also select **section**, if it isn't already, but make document the default.
14. **Build** and **preview** the collection.

Search with MGPP

15. MGPP supports stemming, casefolding and accentfolding. By default, searching in collections built with the MGPP indexer is set to **whole word must match** and **ignore case differences**. So searching *econom* will return 0 documents. Searching for *fao* and *FAO* return the same result — 89 word counts and 11 matched documents.

Go to the **text search** page by clicking the **text search** button at the top right corner. You can see that **stem** is off, which means the **word endings** option is set to **whole word must match**. And **case** (folding) is set to **ignore case differences**.

16. Sometimes we may want to ignore word endings while searching so as to match different variations of the term. Change the **stem** option from **off** to **on**. This will change the search settings from the default, which is that the **whole word must match**, to **ignore word endings**. Now try searching for *econom* again. This time, 9 documents are found.

Please note that word endings are determined according to the third-party stemming tables incorporated in Greenstone, not by the user. Thus the searches may not do precisely what is expected, especially when cultural variations or dialects are concerned. In addition, not all languages support stemming; only English and French have stemming at the moment.

Go to the **fielded search** page next, and once there, change the **stem** option back to **off** (**whole word must match**) to avoid confusion later on.

17. Sometimes we may want to search for the exact term, that is, differentiate the upper cases from lower cases. Now you're on the **fielded search** page, switch **case** folding to **off** (**upper/lower case must match**). Now try searching for *fao* and *FAO* respectively. Notice the search results are different this time, with *fao* not returning any results.

Change the **case** folding option back to **on** (**ignore case differences**) to avoid confusion later on.

Use search mode hotkeys with query term

*MGPP has several hotkeys for setting the search modes for a query term. These hotkeys explicitly set the **stem** option and the **case** option for the query being constructed. Use them in the plain **text search** or **form search**.*

18. **#s** and **#u** are hotkeys for the **stem** option. Appending **#s** to a query term will specifically enable the **ignore word endings** function. For example, click on the **Fielded search** button and try searching for *econom#s*. 9 documents are found, which is the same as in the previous section.

19. Appending **#u** to a query term will explicitly set the current search to **whole word must match**.

Note that using hotkeys will only affect that query term. That is, hotkeys are used per term. For example, if a query expression contains more than one term, some terms can have hotkeys and others not, and the hotkeys can be different for different terms. This provides a fine-grained control of the query, whereas changing the controls for a search field on the **advanced search** page will apply to all the query terms in that field.

20. Hotkeys **#i** and **#c** control the case sensitivity. Appending **#i** to a query term will explicitly set the search to **ignore case differences** (i.e. case insensitive). For example, a search for *fao#i* returns 11 documents.

21. In contrast, appending **#c** will specifically turn off the casefolding, that is, **upper/lower case must match** then.

22. Finally, the hotkeys can also be used in combination. For example, you can append *#uc* to a query term so as to match the whole term (without stemming) and in its exact form (differentiate upper cases and lower cases).

For example, try searching for *econom#si* and compare against the results when searching for *econom#sc* and for *Econom#sc*. The first search is case insensitive and the last two searches are both case sensitive. The number of results for the last two searches should add up to the number of search results for the first search.

A quick reference of the search mode hotkeys in MGPP

Word endings:

#s *ignore word endings*

#u *whole word must match*

Case differences:

#i *ignore case differences*

#c *upper/lower case must match*

Incrementally building a collection using the command line

Prerequisite: [Building and searching with different indexers](#)

Sample files: [incr_build.zip](#)

Devised for Greenstone version: 3.08

Modified for Greenstone version: 3.11

To allow you to quickly try out and experiment with our tutorial exercises, we tend to keep the number of sample files small. Every time you rebuild these collections, for simplicity, the default settings used in Greenstone mean that the previous version built is removed in its entirety. We refer to this as a *full-rebuild*. When building larger collections, this is inefficient.

Greenstone also has the ability to rebuild collections *incrementally*: this means the previous version of the collection is retained and only the changes detected need to be incorporated. There are, however, quite a few aspects to incremental building to control. This is the focus of this tutorial exercise.

To gain the best level of understanding, this tutorial builds collections using the command line.

1. In GLI, create a new collection called *Incremental With Manifests* and base it on the *Demo Collection*. The short name of this collection will become *incremen*, and this will be the name of the collection's folder on the file system.
2. Use GLI's Workspace view to navigate to this tutorial's sample files folder, *incr_build*. It will contain a folder named *import*. Open this. In GLI's **Gather** panel, drag and drop the 3 subfolders into your new collection. (You can also carry out this step using a file browser to copy the contents of the *incr_build/import* sample files folder into `web\sites\localsite\collect\incremen\import`.)
3. Go to the **Design** panel > **Search Indexes** and look for **Indexing Levels**. Make **document** level searching the **default**.
4. Do not build the collection in GLI. We'll be building and rebuilding manually, from the command-line terminal. So close GLI once the files and folders have finished copying into your collection. You can choose to run the Greenstone server at any stage, however.
5. Since this is the first time we're building our collection, we're going to do a complete build. And we'll use the command line to do so. Open a terminal. To open a terminal in Windows, press Ctrl+R and type `cmd` in the **Run** dialog that displays. To open a terminal on a Mac machine, click on menu *Go* → *Utilities* → *Terminal*. Use the terminal to `cd` into your Greenstone installation folder. For instance, if you have your Greenstone installed on Windows as "*Greenstone*" within your account folder at `c:\Users\me`, then type the following in your terminal and hit Enter:

```
cd C:\Users\me\Greenstone
```

If there are any spaces in the filepath, put double quotes on either side of the filepath.

On Linux or Macs, the general command is the same, but the installed location would be different and the slashes go the other way. For example, if installed in `/Users/me/Greenstone3`, you'd type the following and hit Enter:

```
cd /Users/me/Greenstone3
```

Now you're ready to set up the Greenstone environment in your terminal. On Windows, type the following into your terminal and hit Enter again:

```
gs3-setup.bat
```

On Linux and Mac:

```
source ./gs3-setup.sh
```

When using a terminal, you'll need to hit Enter after each command in order to execute the command you just finished typing. We won't repeat this instruction any more. Just remember to hit Enter after every complete command entered into a terminal.

With the terminal now operating within your Greenstone installation folder, and with the Greenstone environment now set up and ready, type the following command to do a complete build of your new collection. Although the command contains the word "rebuild" in it, since this is the first time the collection's being built, it will just build it.

```
perl -S full-rebuild.pl -site localsite incremen
```

For the rest of this tutorial exercise, leave open this terminal in which you have set up your Greenstone's environment. We'll be using it throughout.

6. If the Greenstone server is not running (as would happen if you had closed GLI and didn't start the standalone Greenstone Server Interface application), then run it from the Start Menu on Windows now. You could also run the Greenstone server by running the `gs3-server.bat` script in the terminal if you're trying this on a Windows machine, or by running the `gs3-server.sh` script from a Linux/Mac terminal. Press the Enter Library button.

7. Preview the *incremen (Incremental With Manifests)* collection.

Throughout this tutorial, when previewing an (incrementally) rebuilt collection, make sure to reload any web page in the collection in order to ensure you're seeing any changes you've made. A "[force reload](#)", also referred to as a "hard refresh", is better: either hold down Ctrl while clicking the reload/refresh button, or press Ctrl+F5 in some browsers or Ctrl+Shift+R in others to make the browser do a force reload.

When previewing, try searching for "kouprey" and you should get results, as this term occurs in the document *bl8ase*.

Next, try searching for "groundnuts" and no documents should match.

Incrementally adding some additional new documents to a collection

8. If you want, you can use GLI to drag and drop the *fb33fe*, *fb34fe* and *wb34te* folders, located in the *incr_build/more-files* subfolder of sample files, into your collection.

Alternatively, you can use a File Browser to copy the folders *fb33fe*, *fb34fe* and *wb34te*, located in the *incr_build/more-files* sample files subfolder, into your collection's `import` folder at `web\sites\localsite\collect\incremen\import`.

The above step will only have gathered 3 new documents into your collection. However, since the changes have not been built, previewing at this stage will make no difference.

9. We want to build just the newly added documents into the collection if possible, instead of rebuilding everything. This time, instead of running `full-rebuild`, we'll be running the `incremental-import` and `incremental-buildcol` scripts to perform the two phases of a Greenstone build operation incrementally, these being the *import* and *buildcol* phases. Incremental building allows us to (re)build just what is necessary, rather than everything.

Since we know exactly which files have been added and thus which files need to be built, we can write a manifest file specifying this. The manifest files used by the Greenstone incremental building process are just XML files that can be created and edited in a plain text editor, and which indicate which files need to be (re)processed by a Greenstone incremental build operation.

We've already prepared the manifest files we'll be using in this tutorial exercise for you. Use a File Browser to copy the *manifests* subfolder from the *incr_build* sample files into your `increment` collection folder that's located inside your Greenstone installation directory (at `web\sites\localsite\collect\increment`).

In a text editor, open the *add-new-files.xml* manifest file found in the newly copied *manifests* subfolder. Inspect the contents of this manifest file. It should contain:

```
<?xml version="1.0" encoding="UTF-8"?>
<Manifest>
  <Index>
    <Filename>fb33fe/fb33fe.htm</Filename>
    <Filename>fb34fe/fb34fe.htm</Filename>
    <Filename>wb34te/wb34te.htm</Filename>
  </Index>
</Manifest>
```

The above lists the 3 main documents to be added/indexed by Greenstone (hence the keyword `<Index>`). Since these documents are located inside their own subfolders when copied into the *import* folder, the manifest file also indicates the relative folder structure of these documents (relative to the collection), e.g. *"fb33fe/fb33fe.htm"* shows that the *fb33fe.htm* HTML document is located in the folder *fb33fe*. Only the main documents to be added are listed, not the associated image files also found at the same folder level, as Greenstone will track down all the image files referred to by the main html documents to be indexed and will process them as files associated with the html.

- Return to the terminal you had left open. We can finally run the commands for the incremental build operation.

Use the terminal to first run the incremental import stage:

```
perl -S incremental-import.pl -incremental -manifest manifests/add-new-files.xml -site localsite
increment
```

The build log output will end with the messages

```
* 3 documents were considered for processing
* 3 documents were processed and included in the collection"
```

This means just the 3 newly added documents were imported, just as specified by our manifest file *add-new-files.xml*.

Once that `incremental-import` command has finished running, start off the incremental `buildcol` stage of the build process:

```
perl -S incremental-buildcol.pl -activate -site localsite increment
```

The incremental import command specifies the manifest file that Greenstone is to consult in order to work out which files should be processed and how (whether each is to be Indexed, Deleted or Reindexed). By the `buildcol` stage, the specific files would then be ready for further incremental processing by the `buildcol` script. The `-activate` flag to the incremental `buildcol` script tells Greenstone to (re-)activate the updated collection if the Greenstone server is running.

- Preview the collection either by running the Greenstone Server Interface application, if it isn't already running, or by starting the Greenstone server from the command line with the command:

```
ant start
```

(To stop the Greenstone server at any point, use the command `ant stop`. To stop-then-start it, you'd use `ant restart`.)

When the server is running, preview your library home page, located by default at <http://localhost:8383/greenstone3/library>. Visit the *Incremental with Manifests* collection and click on the Titles browser. There should be 3 additional documents now, and you should be able to search for terms that occur in them. For example, searching for "groundnuts" again should return a result this time, since this term occurs in the newly added document *fb33fe*.

Incrementally deleting some documents from a collection

12. Inspect the *delete-some-files.xml* manifest file (located in your `increment` collection folder's *manifests* subfolder). It contains:

```
<?xml version="1.0" encoding="UTF-8"?>
<Manifest>
  <Delete>
    <OID>b18ase</OID>
    <OID>fb33fe</OID>
  </Delete>
</Manifest>
```

As per the above manifest file, the operation to be performed by an incremental build is a `<Delete>` operation on two documents. For the delete operation, the documents are not indicated by the `<Filename>` XML element, but by the `<OID>` element which specifies the object identifier. We need to use the OID here because we're telling Greenstone precisely what the identifiers of the documents are that we wish to have removed from our collection. The identifiers of every built document in a Greenstone collection are specified in the Identifier field of the document's *doc.xml* file located in the collection's `archives` folder. The *doc.xml* file is the Greenstone-specific XML format in which Greenstone stores documents already imported.

For instance, to find the identifier of the *b18ase.htm* document in your built collection, open up `web\sites\localsite\collect\inremen\archives\b18ase.dir\doc.xml` in a text editor. Then scroll down, looking for a piece of Greenstone extracted metadata labelled *Identifier*, which is the OID for this document:

```
<Metadata name="Identifier">b18ase</Metadata>
```

The above value for the document identifier is what's used in the *delete-some-files.xml* manifest file to refer to this document. This document is one of two that are to be deleted as per the manifest file. Make sure to close the *doc.xml* file if you have it open.

13. Finally, let's incrementally rebuild the collection, specifying the manifest file that Greenstone should use this time to carry out the incremental build operation. As before, there are two steps.

First run the modified incremental import command:

```
perl -S incremental-import.pl -incremental -manifest manifests/delete-some-files.xml -site localsite
inremen
```

From the build output you will notice that 0 documents would have been considered for importing, because documents are only being deleted this time around, and none being newly added.

When the `incremental-import` has finished running, run the same `incremental buildcol` command as before (it doesn't change):

```
perl -S incremental-buildcol.pl -activate -site localsite inremen
```

If you were to scroll through the `buildcol` output in the terminal this time, you would see the following:

```
GreenstoneXMLPlugin: processing fb33fe.dir\doc.xml
GreenstoneXMLPlugin: processing b18ase.dir\doc.xml
```

Only these 2 files were actually processed by `buildcol`, and that's because the manifest specified they were being deleted.

- When it has finished, preview the collection once more and check that the 2 documents have been removed. They should not turn up in the browse classifiers, nor in search results. For example, search for "kouprey" again. Check that when you search for the term this time, that no documents matched the query. (Since it only occurred in document *b18ase*, which has now been removed from the collection.) Likewise, searching for "groundnuts" should not return results either, because document *fb33fe* wherein it occurred has also been removed.

Editing a document's text and metadata, and then incrementally rebuilding the collection

- Inspect the *mod-text-and-meta.xml* manifest file (located in `inremen/manifests`) in a text editor. It should contain:

```
<?xml version="1.0" encoding="UTF-8"?>
<Manifest>
  <Reindex>
    <Filename>fb34fe/fb34fe.htm</Filename>
    <Filename>b20cre/b20cre.htm</Filename>
  </Reindex>
</Manifest>
```

Note the `<Reindex>` used this time. It indicates which documents that are **already** in the collection are to be re-processed when the collection is incrementally rebuilt as per this manifest file.

- Open up the file *fb34fe/fb34fe.htm* of your `inremen` collection's `import` folder in a text editor and add, remove or change some text nested anywhere in between the HTML tags within the `<BODY>` tag. Be careful not to partially modify HTML element names or HTML entities (entities start with an ampersand, `&`, and end with a semi-colon, `;`), as doing so can make your text contents invalid HTML.

Save and close the edited file.

- Start up GLI. Open the `inremen` collection and go to the Enrich panel. Add or modify *dc.Title* metadata for the *b20cre* document. Do not accidentally build the collection using GLI.
- Quit GLI. Optionally run the Greenstone server application if it isn't already running.

In the above two steps, we've modified the text contents of document *fb34fe* and the metadata associated with *b20cre*. Our *mod-text-and-meta.xml* manifest file already indicates that these two files are to be reindexed, so we can go ahead and incrementally rebuild the collection with this manifest file.

- Run the incremental rebuild operation to re-process just these two files. To do so, pass the *mod-text-and-meta.xml* manifest file this time.

First run:

```
perl -S incremental-import.pl -incremental -manifest manifests/mod-text-and-meta.xml -site localsite
inremen
```

At the end of importing, you'd see the following messages displayed in the terminal, because only the 2 documents marked to be reindexed as per the new manifest are processed:

```
* 2 documents were considered for processing
* 2 were processed and included in the collection"
```

Now run:

```
perl -S incremental-buildcol.pl -activate -site localsite inremen
```

20. Preview the collection once more. Check that the 2 documents contain your edits: try searching for any additional words you added and confirm that document fb34fe turns up in the results. Also check the dc.Title metadata that you had modified can now be searched and appears as the title for the *b20cre* document in the Titles browsing classifier.

In this tutorial, we looked at cutting down the amount of time spent on rebuilding a collection by manually controlling the rebuild operation so that it processes only what has changed. We do so by means of a manifest that specifies exactly which files need to be rebuilt and how (whether any need to be Indexed, Deleted or Reindexed). Greenstone also has an automatic incremental rebuild feature, sparing you the need to specify a manifest file in the import phase. Omitting the manifest argument in the above exercises activates this behaviour. However, this is typically slower, because Greenstone now needs to scan the entire `import` folder and compare this with the information in the `archives` folder to determine what has changed.

21. Now repeat all the above exercises in the same sequence once again, but with a new collection called *autoincr* also based on the *Demo* collection. Remember to make document level for searching the default. And build the collection the first time around with `perl -S full-rebuild.pl -site localsite autoincr`, also largely as before. However, this time *don't* pass in any manifest file as an argument to the subsequent rebuild commands which use the `incremental-import.pl` script. And *before* running rebuild commands for the delete operation this time, *manually delete* the following physical folders from the import directory: `web/sites/localsite/collect/incremen/import/b18ase` and `web/sites/localsite/collect/incremen/import/fb33fe`, as now there is no manifest file letting greenstone now which documents are "deleted" (so now they need to be actually deleted for greenstone to automatically detect that they should not be included in the rebuilt collection). So you'd be running these commands after each change this time:

```
perl -S incremental-import.pl -incremental -site localsite autoincr
perl -S incremental-buildcol.pl -activate -site localsite autoincr
```

After each incremental build, preview your *autoincr* collection to check that the browsing classifiers contain the expected documents and that searching returns the expected results.

Automatic incremental indexing

Just as there is the command `full-rebuild.pl` to completely build a collection from scratch, there is also the command `incremental-rebuild.pl`. The final exercise you have just completed could equally have been achieved by running the following after each change:

```
perl -S incremental-rebuild.pl -site localsite autoincr
```

For every collection, the *import* phase can be run incrementally (either using a manifest file or automatically), however, the ability for the *buildcol* phase to be incremental depends on the indexer in use. Lucene and Solr indexers support incremental indexing, but the MG and MGPP indexers do not. A warning is issued if you attempt to run the *buildcol* phase incrementally when the chosen indexer does not support this.

Customization: Themes

Sample files: [custom.zip](#)

Devised for Greenstone version: 3.05

Modified for Greenstone version: 3.11

This exercise looks at changing the appearance of your entire Greenstone Library using Themes created with jQuery ThemeRoller. First, we will look at how to use premade themes. Then, we will make a custom theme using ThemeRoller.

Using Greenstone Visual Themes

1. Enter your Greenstone Library. (Go to *Start* → *All Programs* → *Greenstone-3.11* → *Greenstone3 Server* and click **Enter Library**.)
2. In order to make changes to the library's theme, you must be logged in as an administrator. Click **Login** in the top right-hand corner. The default login is:

```
Username: admin
Password: admin
```

Click **Login**. It is advisable to change the default password, if you haven't already. To do this, click the **admin** button in the top right-hand corner, and select **Account Settings**. Click **Change Password**. Enter the **Old password** (admin); then enter a new password in the **New password** and **Retype password** text boxes. Click **Submit** and the password will be changed. Click on the *My Greenstone Library* link at the top left to go back to the home page. There you'll find a link to the **Administration Page** further down (you may need to scroll down if there are many collections). Click on it and you'll be taken to the **Administration Page**, where you can view a list of the current users.

3. Go to **Preferences**. Now that we are signed in as an administrator, we can see the **Visual theme:** option, where we can quickly change the theme of our collections. These themes were created using JQuery UI's ThemeRoller (<http://jqueryui.com/themeroller/>). The first three options (Greenstone Default and Greenstone Custom 1 and 2) are included as part of the Greenstone 3 installation. The remaining themes are retrieved from the web via the jQuery UI API.

Select **Theme: Greenstone Custom 2** from the drop-down menu, and click **Set preferences**. Now, all of the collections in your Greenstone library should have the Greenstone Custom 2 theme.

Return to the library homepage (click *My Greenstone Library* in the upper left-hand corner) and explore several of your collections. They should all display the new theme.

Creating a custom theme using ThemeRoller (advanced)

4. Go to the jQuery UI ThemeRoller webpage (<http://jqueryui.com/themeroller/>). The **Roll Your Own** menu of the **ThemeRoller** bar on the left can be used to design a custom theme. As changes are made in this menu, the corresponding items on the right reflect these changes. The most important sections for Greenstone customization are:

Section	HTML class	Greenstone 3 usage
Header/Toolbar	ui-widget-header	This sets the appearance of the large, main banner and the footer

Content	ui-widget-content	Sets the appearance for the main area of the page, where the content (e.g. a list of documents) is displayed
Clickable: default state	ui-state-default	Sets the appearance for the small bar along the top, most buttons (Login, Help, Preferences, Test Search, Form Search, Advanced Search), and all unselected browse tabs
Clickable: hover state	ui-state-hover	This sets the appearance for the Login, Help, and Preferences buttons and browse tabs when the mouse hovers over them
Clickable: active state	ui-state-active	This sets the appearance of the selected browse tab

ThemeRoller will not affect some aspects of your Greenstone library's appearance, e.g. the actual design layout, the background color, the font, the search button.

5. Change the Theme to the following values:

Header/Toolbar	Background color	#a23336
	Texture	glass
		50%
	Border	#000000
	Text	#000000
Icon	#000000	
Content	Background color	#000000
	Texture	inset-soft
		25%
	Border	#000000
	Text	#c2bcbc
Icon	#c2bcbc	
Clickable: default state	Background color	#7e7676
	Texture	glass
		55%
	Border	#000000
	Text	#ffffff
Icon	#ffffff	
Clickable: hover state	Background Color	#303030
	Texture	glass
		75%
	Border	#000000
	Text	#ffffff
Icon	#ffffff	

Clickable: active state	Background Color	#000000
	Texture	glass
		65%
	Border	#000000
	Text	#ffffff
Icon	#ffffff	

- Click the **Download theme** button. Select "1.13.2 (Stable, for jQuery1.8+)". Under **Components** uncheck *Toggle All*, which will uncheck all of the boxes. Under **UI Core** check *jQuery 1.8+ Support* and *Keycode*, and under **Widgets** check *Datepicker*. At the bottom of the page, under **Theme**, *Custom Theme* should be selected. (If there's an option to provide a **Theme Folder Name**, enter *TutorialTheme*.) Leave **CSS Scope** blank. Click **Download**. In the pop-up window, select **Save File** and click **OK** to download the zip file.

If you're unable to download the theme you've created, then we have prepared a zip file in advance containing the same ThemeRoller theme: *sample_files* → *downloads* → *TutorialTheme2019.zip*. After extracting, in the following instructions, work with the folder "TutorialTheme" *inside* any TutorialTheme2019 folder.

- Go to the folder where the zip file was downloaded, and unzip the folder. On Windows, unzip as follows: right-click and select **Extract All...** Remove the folder name "jquery-ui-1.11.4.custom" from the end of the suggested zip extraction path. (For example, if it suggests *C:\Users\me\Downloads\jquery-ui-1.11.4.custom*, then change it to *C:\Users\me\Downloads\.*) Then click **Extract**.

The extracted folder will be called *jquery-ui-1.11.1.custom*, or something similar. Rename it to **TutorialTheme**. It should contain css files, an index.html file, a couple of js files and 2 subfolders (called images and external). We largely only need the folder's structure, the css files (particularly jquery-ui.theme.min.css) and the images subfolder, but it's easiest just to copy the entire folder into *Greenstone3* → *web* → *interfaces* → *default* → *style* → *themes*.

- Go to *Greenstone3* → *web* → *interfaces* → *default* → *transform* → *pages* and open **pref.xml** in a text editor. Find the following section and add in the highlighted text, which can be copied from *gs3-theme.tweak.txt*. Take note of the initial comma.

```
<xsl:text disable-output-escaping="yes">
  $(document).ready(function(){
    $("#switcher").themeswitcher({
      imgpath: "interfaces/" + gs.xsltParams.interface_name + "/style/images/",
      additionalThemes: [
        icon:"theme_90_greenstone.png", url:"interfaces/" + gs.xsltParams.interface_name + "/style
/themes/main/jquery-ui-1.8.16.custom.css"},
        {title:"Greenstone Custom 1", name:"custom-theme1", icon:"theme_90_start_menu.png",
url:"interfaces/" + gs.xsltParams.interface_name + "/style/themes/alt_theme_1/jquery-
ui-1.8.16.custom.css"},
        {title:"Greenstone Custom 2", name:"custom-theme2", icon:"theme_90_mint_choco.png",
url:"interfaces/" + gs.xsltParams.interface_name + "/style/themes/alt_theme_2/jquery-
ui-1.8.16.custom.css"},
        {title:"Greenstone Custom 3", name:"custom-theme3", icon:"theme_90_trontastic.png",
url:"interfaces/" + gs.xsltParams.interface_name + "
/style/themes/alt_theme_3/jquery-ui-1.8.16.custom.css"},
        {title:"Tutorial Theme", name:"TutorialTheme", icon:"TutorialTheme.png", url:"interfaces/"
+ gs.xsltParams.interface_name + "/style/themes/TutorialTheme/jquery-ui.theme.min.css"}
      ]
    });
  });
</xsl:text>
```

Save and close **pref.xml**.

- Return to the Library homepage. If not still logged in, login again. Go to the **Preferences** page and

click on the drop-down menu for **Visual theme:**. (If the **Visual theme:** option has disappeared, ensure that you copied the highlighted section in its entirety into the correct place in **pref.xml**, and check especially that you did not miss the initial comma.) The fifth item in the list should now be "Tutorial Theme". It will not have a calendar icon like the other themes, because the image referenced in the above statement - TutorialTheme.png - does not exist. We'll create an icon to match the theme below. For now, select Tutorial Theme, and click **Set preferences**. All of your Greenstone collections will be in this new, custom theme.

*The **Visual theme:** drop-down menu in **Preferences** includes an image for each theme, along with the theme title. If no image is available, only the title is displayed. If you would like to include an image for your custom theme similar to those visible for the other theme options, follow the instructions below.*

10. Return to the **TutorialTheme** folder (in *Greenstone3* → *web* → *interfaces* → *default* → *style* → *themes*). Open *index.html* in a web browser. Scroll down so that the Datepicker calendar is completely visible on your screen. Take a screenshot: either by using your browser's screenshot feature, first selecting the outline of the Datepicker image, or else use your PC's ability to take the screen shot. (On Windows, you can do this by pressing the print screen - **PrtScn** - button.)
11. Open a picture editor (like Paint) and Paste. Select and crop the image around the calendar under Datepicker. Resize the image to 90px by 80 px. (In Paint, click **Resize**. Under **Resize**, choose **Pixels**. Unselect **Maintain aspect ratio**. Still under **Resize** - not the **Skew** section - change **Horizontal** to 90 and **Vertical** to 80 and click **<OK>**.) Save the image in *Greenstone3* → *web* → *interfaces* → *default* → *style* → *images* as TutorialTheme.png.
12. Return to the **Preferences** page, and the Tutorial Theme in the **Visual theme:** drop-down menu will now have a corresponding calendar icon.

Collection-Specific Themes

Prerequisite: [Customization: Themes](#)

Sample files: [custom.zip](#)

Devised for Greenstone version: 3.05

Modified for Greenstone version: 3.11

The **Visual theme**: menu on the **Preferences** page sets the theme for the entire Greenstone library. This tutorial explains how to use themes on a collection level, by changing the theme of the **backdrop** collection.

Creating a custom theme

1. Go to the jQuery UI ThemeRoller webpage (<http://jqueryui.com/themeroller/>). Change the Theme to the following values:

Header/Toolbar	Background color	#141a38
	Texture	highlight_soft
		45%
	Border	#ffffff
	Text	#ffffff
Icon	#ffffff	
Content	Background color	#ffffff
	Texture	flat
		75%
	Border	#ffffff
	Text	#222222
Icon	#222222	
Clickable: default state	Background color	#e9c416
	Texture	glass
		55%
	Border	#ffffff
	Text	#555555
Icon	#888888	
Clickable: hover state	Background Color	#e9d47b
	Texture	glass
		75%
	Border	#ffffff
	Text	#212121
Icon	#454545	
Clickable: active state	Background Color	#ffffff
	Texture	glass
		65%
	Border	#ffffff
Text	#212121	

	Icon	#454545
--	------	---------

- Click the **Download theme** button. Select "*1.13.2 (Stable, for jQuery 1.8+)*". Under **Components** uncheck *Toggle All*, which will uncheck all of the boxes. Under **UI Core** check *jQuery 1.8+ Support* and *Keycode*, and under **Widgets** check *DatePicker*. At the bottom of the page, under **Theme**, *Custom Theme* should be selected. (If there's an option to provide a **Theme Folder Name**, enter *CollectionTheme*.) Leave **CSS Scope** blank. Click **Download**. In the pop-up window, select **Save File** and click **OK** to download the zip file.

Once again, if you're unable to download the theme you've created, then we have also prepared a zip file containing the *CollectionTheme*. Get it from *sample_files* → *downloads* → *CollectionTheme2019.zip*. After extracting, in the following instructions, work with the folder "CollectionTheme" *inside* any *CollectionTheme2019* folder.

- Go to the folder where the zip file was downloaded, and unzip the folder: on Windows, right-click, select **Extract All...** If you downloaded the zip file from JQuery's ThemeRoller site, at this stage you may need to remove the folder name *jquery-ui-1.11.4.custom* from the end of the suggested zip extraction path and then click **Extract**. The extracted folder will then be called *jquery-ui-1.11.1.custom*, or something similar. Rename it to **CollectionTheme**. A rename will not be necessary if you are using the *CollectionTheme2019.zip*.

The extracted folder should contain *css* files, an *index.html* file, a couple of *js* files and 2 subfolders (called *images* and *external*). Copy the entire folder into *Greenstone3* → *web* → *sites* → *localsite* → *collect* → *backdrop* → *style*.

Setting a collection's theme

- Open the Greenstone Librarian Interface (from the Windows *Start* menu) and open the **backdrop** collection (use the **File** menu).
- In the **Format** panel, select **Format Features**. Add the following template to the bottom of the **global** format features (this can be copied from *gs3-collect-theme.txt*):

```
<xsl:template name="additionalHeaderContent">
  <xsl:variable name="httpCollection">
    <xsl:value-of select="/page/pageResponse/collection/metadataList/metadata[@name='httpPath']"/>
  </xsl:variable>
  <link href="{httpCollection}/style/CollectionTheme/jquery-ui.theme.css" rel="stylesheet"
type="text/css" />
</xsl:template>
```

You could also make it refer to **jquery-ui.theme.min.css** instead, which is another version of the same *css* style file, one with a reduced file size.

- Click **Preview Collection** to see the **backdrop** collection with the new, custom theme.

Customizing your home page

Sample files: [custom.zip](#)

Devised for Greenstone version: 3.06

Modified for Greenstone version: 3.11

This tutorial demonstrates how to change the home page of your Greenstone3 installation, including how to use XSL to add certain features (like an up-to-date list of collections and the cross-collection search box).

1. Start up your Greenstone server (*Start* → *All Programs* → *Greenstone3.11* → *Greenstone3 Server*) and click the **Enter Library** button. This will take you to your library home page, which we will be replacing with our custom home page.

Changing the library's home page

2. By default, the library's home page is dictated by the *home.xsl* file in *Greenstone3* → *web* → *interfaces* → *default* → *transform* → *pages*. We want to be able to easily revert back to the default home page, so we won't make changes to this file directly. Instead, we're going to tell the interface to use a new, different file to create the home page.

To do this, open the **interfaceConfig.xml** file found in *Greenstone3* → *web* → *interfaces* → *default* in a text editor, and change:

```
<subaction name="home" xslt="pages/home.xsl"/>
```

To this:

```
<subaction name="home" xslt="pages/home-tutorial.xsl"/>
```

3. Restart the server by clicking the **Restart Library** button in the small Greenstone Server window. (Any time you make changes to an *interfaceConfig.xml* file, you must restart the server for the changes to take effect.) When the server restarts, look at your home page again. Because *home-tutorial.xsl* does not yet exist, you should see:

```
Couldn't find and/or load the stylesheet "pages/home-tutorial.xsl" for a=p, sa=home, in collection.
```

4. Copy the folder *tutorialbliss* from *sample_files* → *custom* to *Greenstone3* → *web* → *interfaces* → *default* → *style* → *themes*. This is a free CSS template that our new home page will use.
5. Copy *home-tutorial.xsl* from *sample_files* → *custom* to *Greenstone3* → *web* → *interfaces* → *default* → *transform* → *pages*. It contains the following text:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:java="http://xml.apache.org/xslt/java"
  xmlns:util="xalan://org.greenstone.gsd13.util.XSLTUtil"
  xmlns:gslib="http://www.greenstone.org/skinning"
  extension-element-prefixes="java util"
  exclude-result-prefixes="java util">

  <xsl:template match="/">
    <html>
      <head>
        <meta name="keywords" content="" />
        <meta name="description" content="" />
        <title>My Greenstone Library</title>

        ...

        <div id="footer">
          <p>2013 Sitename.com. | Photo by <a href="http://www.leagoon.com/">Leagoon</a> | Design by
          <a href="http://www.freecsstemplates.org/" rel="nofollow">FreeCSSTemplates.org</a>.</p>
```

```

        </div>
        <!-- end #footer -->
    </body>
</html>
</xsl:template>

</xsl:stylesheet>

```

6. Refresh you library home page. You should now have a nice, new home page. Basically, the entire contents of the file *index.html* in *sample_files* → *custom* → *tutorialbliss* were copied (after some modifications) in between the `<xsl:template match="/">` and `</xsl:template>` tags of *home-tutorial.xsl*. The following modifications were made:

Changed the text to be Greenstone-related
 Removed escape characters (, ©, etc.) that are not currently supported in GS3
 Ensured every tag had a closing tag (or was self-closing)
 Corrected the paths for the CSS stylesheet and image
 Added in links to certain Greenstone pages (like the *Login*, *Help*, and *Preferences* pages).
 Added a page icon.

7. Adding links to specific pages is relatively straightforward. If you want to link to the backdrop collection, for instance, you could simply add in `backdrop` (`{ $library_name }` is a variable that, in this case, will become "library" -- you will learn more about library names in later tutorials). However, if you have many collections, this is time consuming, and the home page would have to be modified every time you added or removed a collection. Instead, we can use XSL to insert appropriate HTML into our page for each collection in our library. First, we will **define** a template that does this, and then we will **call** this template in the right spot in our HTML.

Adding the list of collections

8. Open *home-tutorial.xsl* in a text editor. Copy the the highlighted text into the area indicated (this, and all other text excerpts can be copied from *gs3-hompage.txt* in *sample_files* → *custom*):

```

        <!-- end #footer -->
    </body>
</html>
</xsl:template>

<xsl:template name="collectionsList">
  <xsl:for-each select="./page/pageResponse/collectionList/collection">
    <xsl:variable name="collectionName" select="@name"/>
    <li>
      <a href="{ $library_name }/collection/{ $collectionName }/page/about">
        <xsl:value-of select="displayItemList/displayItem[@name='name']"/>
      </a>
    </li>
  </xsl:for-each>
</xsl:template>

</xsl:stylesheet>

```

9. Then, find the section below, and change this:

```

<h2>Select a Collection:</h2>
<ul>
  <li><a href="#">Collection 1</a></li>
  <li><a href="#">Collection 2</a></li>
  <li><a href="#">Collection 3</a></li>
  <li><a href="#">Collection 4</a></li>
  <li><a href="#">Collection 5</a></li>
</ul>

```

To this:

```

<h2>Select a Collection:</h2>
<ul>

```

```

    <xsl:call-template name="collectionsList"/>
  </ul>

```

10. Save this file, and refresh your library homepage. The right side bar should now contain a list of all of your collections. Click on one to make sure the link works correctly, and then return to the home page.

Adding a cross-collection search box

11. The current search box does not actually work. We are going to replace this with a search box that will search all collections in our library.

First, add the following template before the final `</xsl:stylesheet>`:

```

<xsl:template name="searchBox">
  <xsl:for-each select="//page/pageResponse/serviceList/service[@name='TextQuery']">
    <form name="QuickSearch" method="get" action="{ $library_name }">
      <input type="hidden" name="a" value="q"/>
      <input type="hidden" name="rt" value="rd"/>
      <input type="hidden" name="s" value="{@name}"/>
      <input type="hidden" name="s1.collection" value="all"/>
      <input type="text" name="s1.query" size="20" id="search-text" value="" />
      <input type="submit" id="search-submit">
        <xsl:attribute name="value">
          <xsl:value-of select="util:getInterfaceText($interface_name, /page/@lang,
'home.quick_search')"/>
        </xsl:attribute>
      </input>
    </form>
  </xsl:for-each>
</xsl:template>

```

12. Then, find the following section:

```

<form method="get" action="#">
  <div>
    <input type="text" name="s" id="search-text" value="" />
    <input type="submit" id="search-submit" value="" />
  </div>
</form>

```

And replace it with this:

```

<xsl:call-template name="searchBox"/>

```

13. Save *home-tutorial.xml*, and refresh your home page. You now have a working cross-collection search box. Search for "economy" to see that it works, and then return to the home page.

Login Links

14. Under the heading "Library Links" in the left sidebar of your home page, you will notice that there are several links. However, we don't want all of these links to appear all of the time. "Login" should only appear when the user is *not* logged in, while "admin" (Account Settings), "Register as a new user", "Administration", and "Logout" should only appear when a user *is* logged in.
15. We will define and call a template that will display the correct links depending on whether a user is logged in or not.

Add the following template before the final `</xsl:stylesheet>`:

```

<xsl:template name="loginButton">
  <xsl:variable name="username" select="/page/pageRequest/userInformation/@username"/>
  <xsl:variable name="groups" select="/page/pageRequest/userInformation/@groups"/>

  <xsl:choose>
    <xsl:when test="$username">
      <xsl:if test="contains($groups,'admin')">
        <li class="login"><a href="{ $library_name }/admin/AddUser">Add user</a></li>

```



```

        <li class="login"><a href="{ $library_name }/admin/ListUsers">Administration</a></li>
    </xsl:if>
    <li class="login"><a href="{ $library_name }/admin/AccountSettings?s1.username=
{ $username }">Logged in as: <xsl:value-of select="$username"/></a></li>
    <li class="login"><a href="{ $library_name }?logout=">Logout</a></li>
</xsl:when>
<xsl:otherwise>
    <li class="login">
        <a href="{ $library_name }?a=p&sa=login&redirectURL=
{ $library_name }%3Fa=p%26sa=home">Login
        <xsl:attribute name="title">
            <xsl:value-of select="util:getInterfaceText($interface_name, /page/@lang,
'login_tip')"/>
        </xsl:attribute>
        </a>
    </li>
</xsl:otherwise>
</xsl:choose>
</xsl:template>

```

16. Then, find the following section:

```

<li><a href="?a=p&sa=login&redirectURL={ $library_name }%3Fa=p%26sa=home">Login</a></li>
<li><a href="{ $library_name }/admin/AccountSettings?s1.username=">Account Settings</a></li>
<li><a href="{ $library_name }/admin/AddUser">Register a new user</a></li>
<li><a href="{ $library_name }/admin/ListUsers">Administration</a></li>
<li><a href="{ $library_name }?logout=">Logout</a></li>

```

And replace it with this:

```
<xsl:call-template name="loginButton"/>
```

17. Save *home-tutorial.xml*, and refresh your home page. If you are logged in, click "**Logout**" to view the homepage in logged out mode. If you are not logged in, click "Login" and enter your username and password to view the homepage in logged in mode.

Adding your library's site name

18. In *home-tutorial.xml*, replace the first two occurrences of "**A New Home Page**" with `<xsl:call-template name="siteName"/>`.

Save *home-tutorial.xml*, and refresh your home page. The name of the browser window and the page's main header should now be "My Greenstone Library". By using this template, you can ensure your library's name is consistent across your entire library. If you decide to change your library name, you only need to make the change in one place. The following section demonstrates how to do this.

Changing your library's site name

19. To change the name of your library, open the *siteConfig.xml* file located in *Greenstone3* → *web* → *sites* → *localsite* in a text editor. Near the start of the file, find the line:

```
<displayItem name="siteName" lang="en">My Greenstone Library</displayItem>
```

And replace "My Greenstone Library" with another name, like "The Best Digital Library". Then save and close the file.

20. For changes to *siteConfig.xml* to take effect, the site must be reconfigured. To do this, you can either restart the server, or, in a browser window, navigate to **http://localhost:8383/greenstone3/library?a=s&sa=c** (replace 8383 if you are running the Greenstone3 server on another port, like 8080 or 8088). Wait for the page to reload, which may take a while if there are many collections in your digital library. Navigate back to your home page (by clicking the link in the upper left corner). The page title and header should now be your new library name. Enter one of your collections and you should see that your library name (in the top left corner) has changed here, as well.

Defining Libraries

Sample files: [libraries.zip](#)

Devised for Greenstone version: 3.06

Modified for Greenstone version: 3.11

*Greenstone3 is built on the idea of sites and interfaces. **Sites** mainly contain content, i.e. they contain collections, which, in turn, contain documents and associated metadata. Sites are a convenient way to group collections.*

***Interfaces** dictate the presentation -- e.g. the HTML, CSS, Javascript, etc. used to present the content. You can have multiple sites and interfaces.*

***Libraries** combine content and presentation. To define a library, you select the site and the interface you want it to use. And, of course, you can have multiple libraries, and each site and interface can be used multiple times.*

*In this tutorial, we will become familiar with **servlets.xml.in** (where libraries are defined), create a new site, add an interface, and define a new library.*

***Note:** The active version of this file is called **servlets.xml** and is located in Greenstone3 → web → WEB-INF → **servlets.xml**, but it is not to be manually edited. Each time the Greenstone 3 server is run, this file is automatically re-generated from its template file at Greenstone3 → resources → web → **servlets.xml.in**, which can be edited as we do in this tutorial.*

Exploring libraries

1. Start up your Greenstone server (Start → All Programs → Greenstone3.11 → Greenstone3 Server) and click the **Enter Library** button. This will take you to the default library's home page. This library uses the site called **localsite** and the interface called **default**.

Change the url from `http://localhost:8383/greenstone3/library` to `http://localhost:8383/greenstone3/halftone-library` and reload the page. This is another library that is defined by Greenstone3. Again, the site it uses is **localsite**, but this library (called **halftone-library**) uses the **halftone** interface, which is based on the Greenstone2 layout. Let's take a look at where these libraries are defined.

2. In a Windows Explorer, navigate to the `Greenstone3 → resources → web` folder and open the template file **servlets.xml.in** in a text editor. Find the following section:

```
<servlet>
  <servlet-name>library</servlet-name>
  <description>The standard gsd13 library program</description>
  <servlet-class>org.greenstone.gsd13.LibraryServlet</servlet-class>
  <init-param>
    <param-name>library_name</param-name>
    <param-value>library</param-value>
  ...
  <init-param>
    <param-name>default_lang</param-name>
    <param-value>en</param-value>
  </init-param>
</servlet>
```

This *library servlet* defines the default Greenstone3 library (called **library**). As you can see, the **library_name** is *library*, the **site_name** is *localsite*, and the **interface_name** is *default*. If you look at the next servlet section, you will see that it has a **library_name** of *halftone-library*, uses *localsite* and the interface *halftone*. If you continue to scroll down, you will see several other library servlet definitions, albeit inactive as they're commented out. There are also other kinds of de-activated

servlets, such as *gateway*. These are beyond the scope of this tutorial.

- At the end of each library servlet definition, you will see a corresponding **servlet mapping** for it, looking something like this:

```
<servlet-mapping>
  <servlet-name>library</servlet-name>
  <url-pattern>/library/*</url-pattern>
</servlet-mapping>
```

Every servlet -- and, therefore, every library we define -- needs a servlet mapping, which tells the server at what URL the servlet should be located. (Notice that the servlet-name here is the same as above, and the url-pattern matches the library_name; this is required. The servlet-name does not have to match the library_name/url-pattern, but it is good practice to make them all the same.)

Now that we've seen how libraries are defined in Greenstone3, let's take a look at where sites are located, how to create a new site, and how to define a new library that uses this site.

Creating a new site

- Sites are located in *Greenstone3* → *web* → *sites*. In this folder, you will see two sites: *gateway* and *localsite*. (*gateway* is a bit different, and it won't be discussed in this tutorial.) Create a new folder in *sites* called **mysite**. This will be the name of your new site. In **mysite**, create a folder called **collect**. This is where all of this site's collections will be located. Finally, copy the **siteConfig.xml** file from the **localsite** folder into the **mysite** folder. (You can, of course, make modifications to the siteConfig.xml to configure your site, but, for simplicity's sake, we'll just use an exact copy.)
- You have now created a site called **mysite**. However, it has no content yet, and it would be nice to have at least one collection in your new site. From *localsite* → *collect*, copy the folder **lucene-jdbm-demo**, and paste it into *mysite* → *collect*.

Defining a new library

- Now, let's define a library that uses this new site. In **servlets.xml.in**, add the following (which can be copied from *sample_files* → *libraries* → *def_libs.txt*); The exact location doesn't matter, just put it with the other servlets:

```
<servlet>
  <servlet-name>library2</servlet-name>
  <description>A new library with my new site!</description>
  <servlet-class>org.greenstone.gsdl3.LibraryServlet</servlet-class>
  <init-param>
    <param-name>library_name</param-name>
    <param-value>library2</param-value>
  </init-param>
  <init-param>
    <param-name>site_name</param-name>
    <param-value>mysite</param-value>
  </init-param>
  <init-param>
    <param-name>interface_name</param-name>
    <param-value>default</param-value>
  </init-param>
  <init-param>
    <param-name>receptionist_class</param-name>
    <param-value>DefaultReceptionist</param-value>
  </init-param>
  <init-param>
    <param-name>default_lang</param-name>
    <param-value>en</param-value>
  </init-param>
</servlet>
```

This defines a new library called **library2** that uses **mysite** and the **default** interface. Immediately after that, or else at the bottom of the file, add in a servlet mapping to tell the server where our new

library should be located:

```
<servlet-mapping>
  <servlet-name>library2</servlet-name>
  <url-pattern>/library2/*</url-pattern>
</servlet-mapping>
```

7. Save **servlets.xml.in**. For the changes to take effect, we must first restart the server. Click the **Restart Library** button in the Greenstone Server window. When the new browser window opens, navigate to `http://localhost:8383/greenstone3/library2` to see your newly defined library. Notice that only one collection appears -- the Lucene Demo Collection that we copied into **mysite**.

Adding and using a new interface

8. Now that we know how to create and use a new site, let's add a new interface and define a library that uses it. Creating a new interface takes time (and is a tutorial all its own), so, for this tutorial, we are just going to use a pre-made interface. From the *sample_files* → *libraries* folder, copy the folder **althor** into the *Greenstone3* → *web* → *interfaces* folder. In **servlets.xml.in**, define a new library that uses the althor interface:

```
<servlet>
  <servlet-name>rand</servlet-name>
  <description>A new library with my new interface!</description>
  <servlet-class>org.greenstone.gsd13.LibraryServlet</servlet-class>
  <init-param>
    <param-name>library_name</param-name>
    <param-value>rand</param-value>
  </init-param>
  <init-param>
    <param-name>site_name</param-name>
    <param-value>mysite</param-value>
  </init-param>
  <init-param>
    <param-name>interface_name</param-name>
    <param-value>althor</param-value>
  </init-param>
  <init-param>
    <param-name>receptionist_class</param-name>
    <param-value>DefaultReceptionist</param-value>
  </init-param>
  <init-param>
    <param-name>default_lang</param-name>
    <param-value>en</param-value>
  </init-param>
</servlet>
```

and provide the servlet mapping:

```
<servlet-mapping>
  <servlet-name>rand</servlet-name>
  <url-pattern>/rand/*</url-pattern>
</servlet-mapping>
```

9. Save **servlets.xml.in**, restart the server again and navigate to `http://localhost:8383/greenstone3/rand`, where you will see the **rand** library, which displays **mysite** using the **althor** interface.

*The **althor** interface was created using a free CSS template created by luiszuno.com. If any of your collections have a home page image, an image slider will appear on the home page of your library. (You can add a home page image to a collection in the **General** section of the **Format** panel in the **GLI**.)*

Changing default settings for the Greenstone server and GLI

10. When we restarted the server throughout this tutorial, Greenstone launched a browser window to `http://localhost:8383/greenstone3/library`, and we had to navigate to our new library's URL. When you are working in a particular library, you may want to go directly to its URL when the server starts. To do this, in the **Greenstone Server** window, go to *File* → *Settings...* and select your library

(for example, **rand**) from the Servlet dropdown menu. Now, when you click **Restart Library**, a window will open to *http://localhost:8383/greenstone3/rand*.

11. In the GLI, you can change the current library under *File* → *Preferences* → *Connection*. First, select the **Site** you want to work with (for example, *mysite*). The **Servlet** dropdown menu will then include all of the libraries that use *mysite*. Select one. Be sure to also change the **Collect Directory** to the *mysite/collect* folder. Click **OK** for the changes to take effect. Now, you will be adding and editing collections in the **mysite** site, and, when you click the Preview button, the selected library will be launched.
12. As you begin using multiple libraries while using GLI, you should be aware of a couple of things. First, any changes you make in the GLI are **collection-level**, which means they will be consistent for this collection no matter what library you are using. However, changes may only take effect *immediately* in the current library. The server may need to be restarted for the changes to appear in other libraries. Second, switching between servlets -- or viewing a collection in a library other than the one currently being used by the GLI -- can create a lock on files in a collection, which will cause an error during collection building. To avoid this, if you use multiple libraries after launching the GLI, **restart the server** before rebuilding a collection.

Designing a new interface: Part 1

Sample files: [interfaces.zip](#)

Devised for Greenstone version: 3.06

Modified for Greenstone version: 3.11

This tutorial series demonstrates how to create a new interface using the default interface as a base. In this first tutorial, we will set the stage by creating a barebones interface, defining a library that uses the interface, and gathering the various Javascript, CSS, and image files that will be used by the new interface.

Creating a new interface

1. In a file browser, navigate to the **interfaces** folder of your Greenstone3 installation (*C:* → *Users* → *<user-name>* → *Greenstone3* → *web* → *interfaces*).

Create a new folder (in the right-click menu, select New → Folder) and call it **perrin**. The name of this folder (perrin) determines the name of your new interface.

2. From *sample_files* → *interfaces* → *aybara* folder, copy the **interfaceConfig.xml** file into the **perrin** folder. Open the file *perrin* → *interfaceConfig.xml* in a text editor. The interface configuration file is used to specify the languages the interface is available in, certain options, and what "actions" are available. **Actions** create the web pages for the library and specify what XSL file should be used for each. For example, if you completed the Home Page Tutorial, you changed the *home* subaction to point to *home-tutorial.xsl* instead of *home.xsl*.

We are going to make one important change to this file. Our new interface is going to borrow heavily from the default interface. We will be reusing many of the XSL files, Javascript, and images. We could copy all of these files into our new interface, however, there are a few drawbacks to this solution:

- we will end up with duplicates of a large number of files
- there are many files in the default interface that we won't need
- it will be difficult to keep track of which files are different/unique to our new interface

A better solution is to base our new interface on the default interface. A base interface is used by Greenstone as a kind of backup. If Greenstone can't find a file in our interface, it will then try to find it in the base interface. To assign default as base interface, in *perrin* → *interfaceConfig.xml*, change the second line from this:

```
<interfaceConfig>
```

To this:

```
<interfaceConfig baseInterface="default">
```

3. Save and close **interfaceConfig.xml**.

Defining a new library

4. Now, let's define a library that uses the new interface. In *Greenstone3* → *resources* → *web* open **servlets.xml.in**, and add the following (the exact location doesn't matter, just put it with the other servlets). You can copy the following from *sample_files* → *interfaces* → *aybara* → *interface.txt*.

```
<servlet>
  <servlet-name>golden</servlet-name>
  <description>A new library with my new interface!</description>
  <servlet-class>org.greenstone.gsdl3.LibraryServlet</servlet-class>
```

```

<init-param>
  <param-name>library_name</param-name>
  <param-value>golden</param-value>
</init-param>
<init-param>
  <param-name>site_name</param-name>
  <param-value>localsite</param-value>
</init-param>
<init-param>
  <param-name>interface_name</param-name>
  <param-value>perrin</param-value>
</init-param>
<init-param>
  <param-name>receptionist_class</param-name>
  <param-value>DefaultReceptionist</param-value>
</init-param>
<init-param>
  <param-name>default_lang</param-name>
  <param-value>en</param-value>
</init-param>
</servlet>

```

This defines a new library called "golden" that uses "localsite" and our new "perrin" interface. Now, immediately after, add in a servlet mapping to tell the server where our new library should be located:

```

<servlet-mapping>
  <servlet-name>golden</servlet-name>
  <url-pattern>/golden/*</url-pattern>
</servlet-mapping>

```

5. Save `servlets.xml.in`. Start up your Greenstone server (*Start* → *All Programs* → *Greenstone3* → *Greenstone3 Server*) and click the **Enter Library** button. This will take you to the default library's home page.

Navigate to `http://localhost:8383/greenstone3/golden`. This library is using the *perrin* interface. However, since *perrin* contains no files except its configuration file, Greenstone is borrowing everything from *default*, *perrin*'s base interface.

Gathering files

6. To change how the *perrin* interface looks, we will be adding some key XSL files (*home.xsl*, *header.xsl*, and *main.xsl*). However, before we do that, we want to gather all of the new CSS, Javascript, and image files we will be using with our new interface.

Since designing web pages is quite challenging in itself, we will be taking advantage of free CSS templates that can be downloaded and modified. There are many websites that offer high quality web page designs, often for free. (*Always be sure to read the license information fully to ensure you do not violate the usage agreement in any way.*)

7. Visit <http://www.os-templates.com/free-website-templates/news-magazine> and click the **Download This Template** button. In the popup window, select **Save File** and click **OK**. This will download a file with a *7z* (*7zip*) extension. If you have *7zip* installed, then you can extract the file in place. If not, you can either download the free utility, *7zip*, from the web, or alternatively use an online service to convert it to a zip file that Windows can extract.

To do the latter, visit <https://convertio.co/uz-zip/> or <https://onlineconvertfree.com/convert-format/7z-to-zip/>. (If conversions don't work on Microsoft Edge, try another web browser like Firefox, if you have it installed. Or you can search the web for alternative free online zip conversion services.) The general process is the same: Upload your *.7z* file. Make sure the output format is set to zip and, once your file has been uploaded, you'll be presented with a convert button which you need to press. Once the conversion process has finished, a Download button will appear. Press it to download the *.zip* file onto your machine. (Such sites may also furnish you with a button by means of which you can clear your uploaded files from their servers.) On Windows, your converted *zip*

version of the template will have been saved in your **Downloads** folder.

In an Explorer window, navigate to your Downloads folder (*C:\Users\<user-name>\Downloads*). Right-click on the downloaded zip file containing the template, select **Extract All...** and, in the popup window, click **Extract**.

*This template is free to use and modify, but the Copyright and link information **must remain intact** in files and in the footer of every page. The template **cannot be distributed** (modified or otherwise) without express permission from OS-Templates. (The full license is provided in the **license.txt** file included with the template).*

- In the newly extracted folder, open the **layout** subfolder and copy the **styles**, **scripts**, and **images** folders into *Greenstone3 → web → interfaces → perrin*. Also copy the file **license.txt** from the extracted folder into the same location. Move the javascript files (files that end with a js extension) that are in the new *perrin → scripts → galleryviewthemes* folder into *perrin → scripts*. Move the *perrin → scripts → galleryviewthemes → themes* into *perrin → images*, and rename this *themes* folder to *galleryviewthemes*. If you wish, you can now delete the *perrin → scripts → galleryviewthemes* folder.

In the new *perrin → images* folder, delete all the images, because we will not be using any of these.

- In the *perrin → styles* folder, open *layout.css* in a text editor and delete the following line:

```
@import url("tables.css");
```

Save and close *layout.css*.

- Now, from the *sample_files → interfaces → aybara* folder, copy the **styles**, **images**, and **transform** folders, as well as the **index-GS3.html** file into *Greenstone3 → web → interfaces → perrin*. (If a popup window asks whether you want to merge the images and styles folders, select "Yes".)

In *perrin → styles*, open *featured_slide.css* in a text editor and change the following line:

```
.loader{background:url("../scripts/galleryviewthemes/loader.gif") center center no-repeat;}
```

to the following, where just the word "scripts" is replaced with the word "images":

```
.loader{background:url("../images/galleryviewthemes/loader.gif") center center no-repeat;}
```

Save and close *featured_slide.css*.

- Next, copy the two javascript files in *sample_files → interfaces* (*jquery.galleryview-2.1.1.js* and especially *jquery.galleryview.2.1.1.min.js*) to *Greenstone3 → web → interfaces → perrin → scripts*, replacing any existing files of identical name. These two javascript files are modified versions to cope with an update to jquery. The modified files are necessary for the next tutorial's image slider to show up and work.

We have now gathered all of the Javascript, CSS, and images our new interface will use.

Double click on *perrin → index-GS3.html* to open it in a browser window. This file isn't used by the new interface, but provides a preview of how the interface will look.

In the next tutorial, we will begin changing how our interface looks, starting with the home page.

Designing a new interface: Part 2

Prerequisite: [Designing a new interface: Part 1](#)

Sample files: [interfaces.zip](#)

Devised for Greenstone version: 3.06

Modified for Greenstone version: 3.11

*In this tutorial, we will be adding **home.xsl** and **header.xsl** files to the **perrin** interface. Modifying XSL files must be done with care. If the XSL is ill-formed or invalid, the page will not display properly. This tutorial attempts to highlight some of the common mistakes that might result in errors. Whenever you are editing XSL files, remember to **save very often**, and then refresh your browser to see if there is an error.*

1. Start up your Greenstone server (*Start* → *All Programs* → *Greenstone3* → *Greenstone3 Server*) and click the **Enter Library** button and navigate to `http://localhost:8383/greenstone3/golden` in the web browser, which still looks identical to the home page of the default interface.

Changing the home page content

2. From the *sample_files* → *interfaces* → *aybara* folder, copy **home.xsl** into *Greenstone3* → *web* → *interfaces* → *perrin* → *transform* → *pages*. In the browser, refresh the library home page. The content of the page should now be only "HOME PAGE CONTENT GOES HERE". Open **home.xsl** in a text editor and take a look at it.
3. In a previous tutorial, we created a home page that was completely different than the rest of our library's pages. This time, however, the home page will share a large portion of its layout and style with the rest of the library's pages. So, we're only going to change the home page's *content* by modifying the **/page/pageResponse** template. We will be adding three sections to our home page: an image slider, a section that can contain featured content, and a list of all of the collections in the library, along with their descriptions.

Replace HOME PAGE CONTENT GOES HERE with the following (these and other code snippets used in this tutorial can be copied from the subsection of *sample_files* → *interfaces* → *aybara* → *interface.txt* named after this tutorial):

```
<div class="content">
  <div id="featured_slide">
    <ul id="featurednews">
      <xsl:call-template name="collSlider"/>
    </ul>
  </div>
</div>
```

4. Save **home.xsl** and refresh your web browser. **DON'T PANIC!** You should now see an *Apache Tomcat error*. This is because we are calling a template (*collSlider*) that doesn't exist. Now add the following before the final `</xsl:stylesheet>`:

```
<xsl:template name="collSlider">
  <xsl:for-each select="./collectionList/collection">
    <xsl:variable name="collectionFolder" select="@name"/>
    <xsl:variable name="collectionName" select="displayItemList/displayItem[@name='name']"/>
    <xsl:variable name="homeImage">
      <xsl:choose>
        <xsl:when test="displayItem[@name='smallicon']">
          sites/<xsl:value-of select="$site_name"/>/collect/<xsl:value-of
select="$collectionFolder"/>/images/<xsl:value-of select="displayItem[@name='smallicon']"/>
        </xsl:when>
        <xsl:otherwise>
          interfaces/<xsl:value-of select="$interface_name"/>/images/default.jpg
        </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>

    <li>
```

```

    <div class="panel-overlay">
      <a href="{ $library_name }/collection/{ $collectionFolder }/page/about">
        <h2><xsl:value-of select="$collectionName"/></h2>
      </a>
    </div>
  </li>
</xsl:for-each>
</xsl:template>

```

5. Save **home.xml** and refresh your web browser. You should no longer have an error. For every collection in your library, you should now see an image followed by the collection's name. In a moment, we will add in the Javascript that will turn these into a slider, but first, let's consider what the collSlider template is doing.

Basically, for every collection in your library (`<xsl:for-each select="./collectionList/collection">`), it is creating a list item that includes an image and the collection's name, which links to the collection's about page. If the collection has a Home page image (the "smallicon"), this will be displayed. Otherwise a backup image (*default.jpg*, which is located in *perrin* → *images*) will be displayed.

6. Now, add Javascript files that will turn these list items into an image slider. Before the final `</xsl:stylesheet>`, add the following:

```

<xsl:template name="additionalHeaderContent">
  <script type="text/javascript" src="interfaces/{ $interface_name }/scripts/jquery.min.js">
<xsl:text> </xsl:text></script>
  <script type="text/javascript" src="interfaces/{ $interface_name }/scripts/jquery.easing.1.3.js">
<xsl:text> </xsl:text></script>
  <script type="text/javascript" src="interfaces/{ $interface_name }/scripts/jquery.timers.1.2.js">
<xsl:text> </xsl:text></script>
  <script type="text/javascript" src="interfaces/{ $interface_name }/scripts
/jquery.galleryview.2.1.1.min.js"><xsl:text> </xsl:text></script>
  <script type="text/javascript" src="interfaces/{ $interface_name }/scripts
/jquery.galleryview.setup.js"><xsl:text> </xsl:text></script>
</xsl:template>

```

Save **home.xml** and refresh your web browser. All of the images and collection names should now be condensed into an image slider. (It is still missing some style, because we haven't added links to our CSS files yet.) This **additionalHeaderContent** template is "called" in the **header.xml** file, which we will see later in this tutorial.

7. Add the next section of the home page's content. In the `/page/pageResponse` template, insert the highlighted section where indicated:

```

    <xsl:call-template name="collSlider"/>
  </ul>
</div>
</div>
<div class="column">
  <ul class="latestnews">
    <li>
      <p><strong><a href="#">Highlighted Item 1</a></strong><br/>This is a place where you can
put information about an item you would like to highlight in your collection, with or without an
accompanying image.</p>
    </li>
    <li>
      <p><strong><a href="#">Highlighted Item 2</a></strong><br/>This is a place where you can
put information about an item you would like to highlight in your collection,with or without an
accompanying image.</p>
    </li>
    <li class="last">
      <p><strong><a href="#">Highlighted Item 3</a></strong><br/>This is a place where you can
put information about an item you would like to highlight in your collection, with or without an
accompanying image.</p>
    </li>
  </ul>
</div>
<br class="clear" />
</xsl:template>

```

Save **home.xml** and refresh your web browser. This section will be an area where you can add in

your own content. Notice the file paths for the image files begin with **interfaces/{*Sinterface_name*}**. As its name suggests, **{*Sinterface_name*}** becomes the name of the interface, in this case "perrin".

8. Finally, add in the highlighted section:

```
<br class="clear" />
<div id="hpage_cats">
  <xsl:call-template name="collectionsList"/>
</div>
</xsl:template>
```

And add the following templates before the final `</xsl:stylesheet>`:

```
<xsl:template name="collectionsList">
  <xsl:for-each select="./collectionList/collection">
    <xsl:choose>
      <xsl:when test="position() mod 2 = 1">
        <div class="fl_left">
          <xsl:call-template name="collDescription"/>
        </div>
      </xsl:when>
      <xsl:otherwise>
        <div class="fl_right">
          <xsl:call-template name="collDescription"/>
        </div>
        <br class="clear" />
      </xsl:otherwise>
    </xsl:choose>
  </xsl:for-each>
</xsl:template>

<xsl:template name="collDescription">
  <xsl:variable name="collectionFolder" select="@name"/>
  <xsl:variable name="collectionName" select="displayItemList/displayItem[@name='name']"/>
  <xsl:variable name="homeImage" select="displayItemList/displayItem[@name='icon']"/>
  <xsl:variable name="aboutImage" select="displayItemList/displayItem[@name='smallicon']"/>
  <xsl:variable name="collDesc" select="displayItemList/displayItem[@name='description']"/>
  <xsl:variable name="numDocs" select="metadataList/metadata[@name='numDocs']"/>

  <h2><a href="{library_name}/collection/{collectionFolder}/page/about"><xsl:value-of
select="$collectionName"/></a></h2>
  <xsl:if test="$aboutImage">
    
  </xsl:if>

  <xsl:choose>
    <xsl:when test="$collDesc">
      <p class="justify"><xsl:apply-templates
select="displayItemList/displayItem[@name='description']"/></p>
    </xsl:when>
    <xsl:otherwise>
      <p class="justify">Welcome to the <xsl:value-of select="$collectionName"/> collection. This
collection contains <xsl:value-of select="$numDocs"/> documents.</p>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

These templates are more complicated, but a basic explanation follows (don't worry if you don't understand it). You can skip ahead to Step 9 if you'd like.

First look at the `collDescription` template. It creates a header that links to the collection's about page (`<h2><xsl:value-of select="$collectionName"/></h2>`). Then, if there is an About page image for the collection, this is displayed. Finally, if the collection has a description (`<xsl:when test="$collDesc">`), it will display this description. Otherwise, it will display a generic sentence stating the collection's name and how many documents it contains.

Now, let's look at the `collectionsList` template. Basically, for every collection in the library (`<xsl:for-each select="./collectionList/collection">`), it is calling the `collDescription` template, i.e. doing

everything explained in the previous paragraph. However, the CSS we will be using requires that our div's alternate between class "fl_left" and class "fl_right" or the page won't display correctly. So, if the collection is an odd number in the list (`<xsl:when test="position() mod 2 = 1">`), the collection description will be in a `<div class="fl_left">`, otherwise (i.e. when it is an even number in the list) the collection description will be in a `<div class="fl_right">`. Once we link our CSS, this will make our descriptions fall into two nice columns.

9. Save and close **home.xml** and refresh your web browser, which should now include the names and descriptions (and, accompanying images if any of your collections have an About page image) for each collection in your library.

We have now made all of our changes to **home.xml**. Next, we will add a new **header.xml** file, where the HTML header content is located.

Changing the HTML header content

10. From the *sample_files* → *interfaces* → *aybara* folder, copy **header.xml** into *Greenstone3* → *web* → *interfaces* → *perrin* → *transform* → *layouts*. (Be aware that the destination now is the *layouts* folder and not the *pages* folder whereto you had previously copied **home.xml**) Refresh your web browser: nothing should have changed.
11. Now open **header.xml** in a text editor. It contains only one template called "create-html-header", which was copied directly from the default interface's **header.xml**. We are going to remove the links to the CSS files used by the default interface, and add in the links to our new CSS files. Scroll down and replace the following lines:

```
<xsl:choose>
  <xsl:when test="/page/pageResponse/interfaceOptions/option[@name = 'cssTheme']/@value">
    <!-- Get the theme from the interfaceConfig.xml file -->
    <link rel="stylesheet" href="{/page/pageResponse/interfaceOptions/option[@name =
'cssTheme']/@value}" type="text/css"/>
  </xsl:when>
  <xsl:otherwise>
    <link rel="stylesheet" href="interfaces/{\$interface_name}/style/themes/main/jquery-
ui-1.8.16.custom.css" type="text/css"/>
  </xsl:otherwise>
</xsl:choose>
<link rel="stylesheet" href="interfaces/{\$interface_name}/style/core.css" type="text/css"/>
```

With this:

```
<link rel="stylesheet" href="interfaces/{\$interface_name}/styles/layout.css" type="text/css" />
<link rel="stylesheet" href="interfaces/{\$interface_name}/styles/g3-core-min.css"
type="text/css" />
```

12. Save **header.xml** and refresh your web browser. The page should have changed drastically. We removed the links to the default CSS, which removed all of the old style. But, since we added in links to our new CSS, all of the content we just added to the home page should now be styled much better.

Near the bottom of **header.xml**, you will see this line: `<xsl:call-template name="additionalHeaderContent"/>`. For the home page, this is where the four `<script>` elements from Step 6 above will be placed.

Close **header.xml**. In the next and final tutorial on designing an interface, we will add and modify **main.xml**.

Designing a new interface: Part 3

Prerequisite: [Designing a new interface: Part 2](#)

Sample files: [interfaces.zip](#)

Devised for Greenstone version: 3.06

Modified for Greenstone version: 3.11

In this tutorial, we will change the layout for all of the pages in our library at once by adding and modifying the **main.xsl** file for our interface.

Examining main.xsl

1. From the *sample_files* → *interfaces* → *aybara* folder, copy **main.xsl** into *Greenstone3* → *web* → *interfaces* → *perrin* → *transform* → *layouts*. Refresh your web browser -- you should now only see the new content we added to **home.xsl**.

Open **main.xsl** in a text editor. It contains two templates: **mainTemplate** and **gs_footer**. The **mainTemplate** looks like this:

```
<xsl:template name="mainTemplate">
  <html>
    <head>
      <!-- ***** in header.xsl ***** -->
      <xsl:call-template name="create-html-header"/>
    </head>
    <body>
      <div class="container" id="gs_content">
        <xsl:apply-templates select="/page"/>
      </div>
      <xsl:call-template name="gs_footer"/>
    </body>
  </html>
</xsl:template>
```

Every page in our library uses this template. As you can see, it contains the basic structure of our HTML page. Inside the HTML header, the **create-html-header** template is called (`<xsl:call-template name="create-html-header"/>`). This is the template in **header.xsl** we edited in the previous tutorial. Inside the body of the HTML, we use two templates: **/page** and **gs_footer**. (The **/page** template is "applied" instead of called. The reason for this is complex and not relevant to this tutorial.)

The **/page** template is where the content of each individual page is. For the home page, it is the content we just added to **home.xsl**. Since the **perrin** interface does not have XSL files for any of the other page types, the content of all of the other pages depends on the default interface: on the collection's about page, it's the collection description; on search pages, it is the query boxes and search results; on document pages it is the document content, etc.

The **gs_footer** template is where we will put our page's footer.

2. Now we will add the HTML that all of the pages of our collection will share. (You might want to take another look at **index-GS3.html** to remind yourself what the layout will look like.)

Let's start by adding a footer to our library. In the **gs_footer** template, immediately after the line `<!-- Put footer in here. -->` insert the following (which, along with other code to be inserted for this tutorial, can be copied from this tutorial's subsection of *sample_files* → *interfaces* → *aybara* → *interface.txt*):

```
<!-- *****DO NOT REMOVE THE LINK TO OS TEMPLATES: Template is free to use/modify, but this link
MUST remain on all pages. Do not remove Copyright information. Replace "Your Webpage Here" (and it's
link) with your own information.-->
<div class="wrapper col8">
  <div id="copyright">
    <p class="fl_left">Copyright © 2013 - All Rights Reserved - <a href="#">Your Webpage
```

```

Here</a></p>
  <p class="fl_right">Template by <a href="http://www.os-templates.com/" title="Free Website
Templates">OS Templates</a></p>
  <br class="clear" />
</div>
</div>

```

3. Save **main.xml** and refresh the web browser. The footer will now appear at the bottom of every page. You can click into a collection to see that it is also on these pages. (If your collection does not have a description, the footer will be the only thing that appears on the collection's about page.)
4. You'll notice that the image slider and the highlighted items are right up against the edge of the page. In **main.xml**, replace

```

<div class="container" id="gs_content">
  <xsl:apply-templates select="/page"/>
</div>

```

with:

```

<div class="wrapper">
  <div class="container" id="gs_content">
    <xsl:apply-templates select="/page"/>
    <br class="clear" />
  </div>
</div>

```

Important note: in any interface's **main.xml** file, the attribute `id="gs_content"` **must** be set on the `<div>` element containing the `<xsl:apply-templates select="/page"/>` element. Much of the proper functioning of Greenstone3 is dependent on this. So be aware of this if you modify or create your own `main.xml` for a custom Greenstone 3 interface.

5. Save **main.xml** and refresh the web browser. The slider and highlighted items should now line up with the collection descriptions below them. There is also a grey line above the footer.

Adding a navigation bar

6. Next, we're going to add the navigation bar and quick search box. Since the templates for this are pretty complex, we're going to start by adding only plain HTML to see the basic structure. Insert the following immediately after the `<body>` tag:

```

<div class="wrapper col2">
  <div id="topbar">
    <div id="topnav">
      <ul>
        <li class="active"><a href="index.html">Home</a></li>
        <li><a href="style-demo.html">Browse</a>
          <ul>
            <li><a href="#">Link 1</a></li>
            <li><a href="#">Link 2</a></li>
            <li><a href="#">Link 3</a></li>
          </ul>
        </li>
        <li><a href="#">Search</a>
          <ul>
            <li><a href="#">Link 1</a></li>
            <li><a href="#">Link 2</a></li>
            <li><a href="#">Link 3</a></li>
          </ul>
        </li>
      </ul>
    </div>
  </div>
<!--*****_-->
<div id="search">
  <form action="#" method="post">
    <fieldset>
      <legend>Site Search</legend>
      <input type="text" value="Search Our Website..." onfocus="this.value=
(this.value=='Search Our Website...')? '' : this.value ;" />
      <input type="submit" name="go" id="go" value="Search" />
    </fieldset>
  </form>
</div>

```

```

        </fieldset>
    </form>
</div>
<br class="clear" />
<div id="advanced"><a href="#">advanced search</a></div>
<!--*****-->
</div>
</div>

```

7. Save **main.xml** and refresh the web browser. The navigation bar should now appear at the top of the screen.

As you can see in the code above, the navigation links are simply HTML list items, and the dropdown menus are simply lists inside of lists. This is a very common way to make navigation bars.

8. However, this navigation bar is static. It will be exactly the same for every page in the library. Instead, and we want it to change depending on which page we are on and which collection we are in. To do this, we will be using a couple of pretty complicated looking templates. Just remember, the end result will be HTML structured in the same way as the lists above. First, call the **navBar** template by replacing this:

```

<li class="active"><a href="index.html">Home</a></li>
<li><a href="style-demo.html">Browse</a>
  <ul>
    <li><a href="#">Link 1</a></li>
    <li><a href="#">Link 2</a></li>
    <li><a href="#">Link 3</a></li>
  </ul>
</li>
<li><a href="#">Search</a>
  <ul>
    <li><a href="#">Link 1</a></li>
    <li><a href="#">Link 2</a></li>
    <li><a href="#">Link 3</a></li>
  </ul>
</li>

```

with this:

```
<xsl:call-template name="navBar"/>
```

Then add the following templates before the final `</xsl:stylesheet>`:

```

<xsl:template name="navBar">
  <xsl:choose>
    <xsl:when test="page/pageResponse/collection">
      <xsl:variable name="count" select="count(/page/pageResponse/collection/serviceList
/service[@name='ClassifierBrowse']/classifierList/classifier)"/>
      <xsl:variable name="currentPage" select="page/pageRequest/@fullURL"/>

      <li><a href="{\$library_name}">Home</a></li>
      <li>
        <xsl:if test="page/pageRequest/@subaction='about'"><xsl:attribute
name="class">active</xsl:attribute></xsl:if>
        <a href="{\$library_name}/collection/{\$collNameChecked}/page/about">About</a>
      </li>

      <xsl:choose>
        <xsl:when test="\$count > 3">
          <li><a href="{\$currentPage}">Browse</a>
            <ul>
              <xsl:call-template name="Browsing"/>
            </ul>
          </li>
        </xsl:when>
        <xsl:otherwise>
          <xsl:call-template name="Browsing"/>
        </xsl:otherwise>
      </xsl:choose>

      <xsl:if test="/page/pageResponse/collection/serviceList/service/@type='query'">
        <li><a href="{\$currentPage}">Search</a>

```



```

        <ul>
          <xsl:for-each select="/page/pageResponse/collection/serviceList
/service[@type='query']">
            <xsl:variable name="search" select="@name"/>
            <xsl:variable name="search_name" select="displayItem[@name='name']"/>
            <li><a href="{ $library_name }/collection/{ $collNameChecked }/search/{ $search }">
<xsl:value-of select="$search_name"/></a></li>
          </xsl:for-each>
        </ul>
      </li>
    </xsl:if>

    </xsl:when>
    <xsl:otherwise> </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template name="Browsing">
  <xsl:for-each select="/page/pageResponse/collection/serviceList/service[@name='ClassifierBrowse']
/classifierList/classifier">
    <li>
      <xsl:choose>
        <!-- If this tab is selected then colour it differently -->
        <xsl:when test="util:contains(/page/pageRequest/paramList/param[@name = 'cl' and
/page/pageRequest/@action = 'b']/@value, @name)">
          <xsl:attribute name='class'>active</xsl:attribute>
        </xsl:when>
        <xsl:otherwise> </xsl:otherwise>
      </xsl:choose>

      <a>
        <!-- Add a title element to the <a> tag if a description exists for this classifier -->
        <xsl:if test="displayItem[@name='description']">
          <xsl:attribute name='title'><xsl:value-of select="displayItem[@name='description']"/>
        </xsl:attribute>
        </xsl:if>

        <!-- Add the href element to the <a> tag -->
        <xsl:choose>
          <xsl:when test="@name">
            <xsl:attribute name="href"><xsl:value-of select="$library_name"/>/collection
/<xsl:value-of select="/page/pageResponse/collection[@name=$collNameChecked]/@name"/>/browse
/<xsl:value-of select="@name"/></xsl:attribute>
          </xsl:when>
          <xsl:otherwise>
            <xsl:attribute name="href"><xsl:value-of select="$library_name"/>/collection
/<xsl:value-of select="/page/pageResponse/collection[@name=$collNameChecked]/@name"/>/browse
/1</xsl:attribute>
          </xsl:otherwise>
        </xsl:choose>

        <!-- Add the actual text of the <a> tag -->
        <xsl:value-of select="displayItem[@name='name']"/>
      </a>
    </li>
  </xsl:for-each>
</xsl:template>

```

A basic explanation of these templates follows (don't worry if you don't understand it). You can skip ahead to Step 9 if you'd like.

Let's take a look at the **navBar** template and try to understand what it's doing. First, it's checking to see if the page is a collection page (`<xsl:when test="page/pageResponse/collection">`). If so, it will create a link back to the home page (`Home`) and to the current collection's about page.

Then, if there are 3 or fewer browsing classifiers, they will each become a link on the navigation bar. If there are more than 3 browsing classifiers, then they will be made into a dropdown menu under the heading "Browse". (If there are too many links in the navigation bar, they will start to overlap.) Finally, if there are any search types, they will be in a dropdown menu under the heading "Search". In addition, it adds the attribute "class=active" to the link for the current page so that it will be colored differently.

9. Save **main.xml** and refresh the browser. The home page should have no links in the navigation bar at all. Enter a few different collections and see what the navigation bar looks like in each of those.

Adding functionality to the quick search box

10. Currently, the quick search box doesn't actually work. To make the quick search work, replace this:

```
<div id="search">
  <form action="#" method="post">
    <fieldset>
      <legend>Site Search</legend>
      <input type="text" value="Search Our Website..." onfocus="this.value=
(this.value=='Search Our Website...')? '' : this.value ;" />
      <input type="submit" name="go" id="go" value="Search" />
    </fieldset>
  </form>
</div>
<br class="clear" />
<div id="advanced"><a href="#">advanced search</a></div>
```

With this:

```
<xsl:choose>
  <xsl:when test="page/pageRequest/@subaction='home'">
    <xsl:call-template name="crossCollSearch"/>
  </xsl:when>
  <xsl:when test="page/pageRequest/paramList/param[@name='c' and /page/pageResponse
/collection[@name=$collNameChecked]/serviceList/service[@name='TextQuery']">
    <xsl:call-template name="CollectionSearch"/>
  </xsl:when>
  <xsl:otherwise/>
</xsl:choose>
```

This says when you are on the home page, use the **crossCollSearch** template; when you are on a collection page (and that collection has **plain**, i.e. "TextQuery", search enabled), use the **CollectionSearch** template; otherwise, don't put anything at all.

11. Now add in the **crossCollSearch** and **CollectionSearch** templates before the final `</xsl:stylesheet>`:

```
<xsl:template name="crossCollSearch">
  <div id="search">
    <xsl:for-each select="/page/pageResponse/serviceList/service[@name='TextQuery']">
      <form name="QuickSearch" method="get" action="{ $library_name }">
        <input type="hidden" name="a" value="q"/>
        <input type="hidden" name="rt" value="rd"/>
        <input type="hidden" name="s" value="{ @name }"/>
        <input type="hidden" name="s1.collection" value="all"/>
        <input type="text" name="s1.query" id="search-text" value="Search all collections..."
onfocus="this.value=(this.value=='Search all collections...')? '' : this.value ;" />
        <input type="submit" name="go" id="go" value="Search" />
      </form>
    </xsl:for-each>
  </div>
  <br class="clear" />
</xsl:template>

<xsl:template name="CollectionSearch">
  <div id="search">
    <xsl:variable name="subaction" select="/page/pageRequest/@subaction"/>
    <form action="{ $library_name }/collection/{ $collNameChecked }/search/TextQuery">
      <!-- This parameter says that we have come from the quick search area -->
      <input type="hidden" name="qs" value="1"/>
      <input type="hidden" name="rt" value="rd"/>
      <input type="hidden" name="s1.level">
        <xsl:attribute name="value">
          <xsl:choose>
            <xsl:when test="/page/pageRequest/paramList/param[@name = 's1.level']">
              <xsl:value-of select="/page/pageRequest/paramList/param[@name =
's1.level']/@value"/>
            </xsl:when>
            <xsl:otherwise>
              <xsl:value-of select="/page/pageResponse/collection/serviceList
/service[@name='TextQuery']/paramList/param[@name = 'level']/@default"/>
            </xsl:otherwise>
          </xsl:choose>
        </xsl:attribute>
      </input>
    </form>
  </div>
</xsl:template>
```

```

        </xsl:choose>
      </xsl:attribute>
    </input>
  <xsl:choose>
    <xsl:when test="/page/pageResponse/service[@name = 'TextQuery']/paramList/param[@name =
'startPage']">
      <input type="hidden" name="s1.startPage" value="1"/>
    </xsl:when>
    <xsl:otherwise>
      <input type="hidden" name="startPage" value="1"/>
    </xsl:otherwise>
  </xsl:choose>
  <xsl:if test="not(/page/pageRequest/paramList/param[@name = 's1.hitsPerPage'])">
    <input type="hidden" name="s1.hitsPerPage" value="20"/>
  </xsl:if>
  <xsl:if test="not(/page/pageRequest/paramList/param[@name = 's1.maxDocs'])">
    <input type="hidden" name="s1.maxDocs" value="100"/>
  </xsl:if>
  <!-- The query text box -->
  <span class="querybox">
    <xsl:variable name="qs">
      <xsl:apply-templates select="/page/pageResponse/collection[@name=$collNameChecked]
/serviceList/service[@name='TextQuery']/paramList/param[@name='query']" mode="calculate-default"/>
    </xsl:variable>
    <nobr>
      <xsl:apply-templates select="/page/pageResponse/collection[@name=$collNameChecked]
/serviceList/service[@name='TextQuery']/paramList/param[@name='query']">
        <xsl:with-param name="default"
select="java:org.greenstone.gsd13.util.XSLTUtil.tidyWhitespace($qs, /page/@lang)"/>
      </xsl:apply-templates>
    </nobr>
  </span>
  <!-- The submit button (for TextQuery) -->
  <xsl:if test="/page/pageResponse/collection[@name=$collNameChecked]/serviceList
/service[@name='TextQuery']">
    <input type="submit" name="go" id="go" value="Search" > </input>
  <br/>
  </xsl:if>
</form>
</div>
<br class="clear" />
<div id="advanced"><a href="{ $library_name }/collection/{ $collNameChecked }/search
/TextQuery">advanced search</a></div>
</xsl:template>

```

12. Save **main.xml** and refresh the web browser. Try searching for a word on the home page (like "economy"), and then enter a collection or two and try searching from there, as well.

Adding the library name and login links

13. Next, we'll add a header to our page that displays the library's name (with a link to the home page), and, if we're inside a collection, displays the collection's name (with a link to the collection's about page). After the `<body>` tag in **mainTemplate**, add the following:

```

<div class="wrapper">
  <div id="header">
    <div class="fl_left">
      <h1><a href="{ $library_name }"><xsl:call-template name="siteName"/></a></h1>
      <p>&#160;
        <xsl:if test="page/pageResponse/collection">
          <a href="{ $library_name }/collection/{ $collNameChecked }/page/about">
            <xsl:value-of select="page/pageResponse/collection/displayItemList
/displayItem[@name='name']"/>
          </a>
        </xsl:if>
      </p>
    </div>
  <br class="clear"/></div>
</div>

```

Save **main.xml** and refresh the web browser to see the library name display at the top of the page. Enter a collection to see the collection name displayed.

14. Finally, we will add the top bar which includes the links for logging in, the help page, and the

preferences page. In the **mainTemplate**, replace the `<body>` tag with:

```
<body id="top">
  <div class="wrapper col0">
    <div id="topline">
      <ul>
        <xsl:call-template name="loginLinks"/>
        <li><a href="{ $library_name }/collection/{ $collNameChecked }
/page/pref">Preferences</a></li>
        <li><a href="{ $library_name }/collection/{ $collNameChecked }/page/help">Help</a></li>
      </ul>
      <br class="clear" />
    </div>
  </div>
</div>
```

and add the following template before the final `</xsl:stylesheet>`:

```
<xsl:template name="loginLinks">
  <xsl:variable name="username" select="/page/pageRequest/userInformation/@username"/>
  <xsl:variable name="groups" select="/page/pageRequest/userInformation/@groups"/>

  <xsl:choose>
    <xsl:when test="$username">
      <xsl:if test="contains($groups, 'admin')">
        <li class="login"><a href="{ $library_name }/admin/AddUser">Add user</a></li>
        <li class="login"><a href="{ $library_name }/admin/ListUsers">Administration</a></li>
        <li class="login"><gslib:depositorTitleAndLink/></li>
      </xsl:if>
      <li class="login"><a href="{ $library_name }/admin/AccountSettings?s1.username=
{ $username }">Logged in as: <xsl:value-of select="$username"/></a></li>
      <li class="login"><a href="{ $library_name }?logout=">Logout</a></li>
    </xsl:when>
    <xsl:otherwise>
      <li class="login">
        <a href="{ $library_name }?a=p&sa=login&redirectURL=
{ $library_name }%3Fa=p%26sa=home">Login
        <xsl:attribute name="title">
          <xsl:value-of select="util:getInterfaceText($interface_name, /page/@lang,
'login_tip')"/>
        </xsl:attribute>
      </a>
    </li>
  </xsl:otherwise>
</xsl:choose>
</xsl:template>
```

15. Save and close **main.xml** and refresh the web browser to see the top bar. Click the **Login** button and log in as **admin** (if you hadn't already changed the password, use the default password of "admin") to see all of the links appear along the top, including links to the **Add User** page, to the **Administration** page, and to Logout.

Using WebSwing GLI (Web GLI)

Devised for Greenstone version: 2.88|3.11

Modified for Greenstone version: 2.87|3.11

When a librarian or other collection designer wants to create/edit a collection on a Greenstone installation that's running on a remote server machine (such as on a cloud), there are several options to connect to the remote machine.

There is the new WebSwing GLI, which allows you to use the remote Greenstone installation's GLI through your local web browser; the specifics of which are described in this tutorial.

There is also client-gli, which requires you to first set up the remotely running Greenstone server to support client-GLI applications connecting to it. It also requires you to have a Greenstone installed in order to run a client-GLI.

(In the past, there was also the GLI applet, but support for Java applets has been deprecated by browsers over the past years.)

WebSwing GLI is the easiest route, as it is already set up and ready to use once you have your Greenstone server up and running. For this reason, going forward, WebSwing GLI will be maintained as the most up-to-date version of GLI that allows creating and editing collections on a remotely running Greenstone 3.

The following assumes you have a Greenstone 3 server that's already running, whether on a cloud or locally such as at the default location of <http://localhost:8383/greenstone3/Library>.

Creating a user account

*To connect with GLI to you need a user account that has collection-editing permissions or you can use the default "admin" user account that is already set up with the permissions of all-collections-editor. Refer to the tutorial **Customization: Themes** as to the password for the admin user, unless you've already changed its default password.*

To create a new user account that is allowed to create collections and edit the ones it creates:

1. Visit your running Greenstone server's home page in the browser. Scroll down to the link labelled *Administration Page* and click it. As it says, it "Allows you to manage users". Before proceeding, you'll be asked to log in as the *admin* user (or as any custom users you created with administration privileges) in order to add new users.
2. Press the button **Add a New User**.
3. Fill out the fields. As a bare minimum, you will need to enter a username for the new user, enter a password for the account twice and assign what *Groups* the new user belongs to. Groups controls both access permissions and content creation permissions: whether the new user is in the right group to view restricted Greenstone collections and documents, and whether the user is in the right group to edit certain (or any) collection.
4. If you wish to give your new user the all-mighty rights of an administrator, you can add "administrator" to their Groups field. Usually, you will not want to go around giving every new user administrator rights. There are 3 kinds of pre-existing Groups that determine a user's content creation permissions: if they belong to *all-collections-editor* (like the default *admin* user does), they can create any Greenstone collection and edit any collection. If their Groups field contains *personal-collections-editor*, they have the right to create any collection they want, but only to edit the collections they created and not those created by others. Setting a user's Groups field to *COLLNAME-collection-editor*, where you have to edit COLLNAME to a specific collection's

shortname, will allow the user to edit that collection. For instance, if a user has *lucene-jdbm-demo-collection-editor* in its list of groups, they have the right to edit the default Greenstone 3 collection "lucene-jdbm-demo".

5. For now, you could add "personal-collections-editor" to your new user's group to give them the right to create any collection and edit those. This will be sufficient to cover most Greenstone 3 tutorials. But for your user to have the additionally ability to edit the default "lucene-jdbm-demo" demonstration collection, you can append *lucene-jdbm-demo-collection-editor* to the groups field (separated by comma). Alternatively, you can just set the user's Groups field to *all-collections-editor*, if you're going to go through all the tutorials with WebSwing GLI using this new user account.

Accessing WebSwing GLI: a Greenstone Librarian Interface (GLI) application accessible over your browser

6. First run the Greenstone server:

You need the GS3 server to be running in order to access WebSwing GLI. To launch the GS3 server, on Windows you can select the *Greenstone 3* shortcut or double-click your GS3 installation's top-level `gs3-server.bat` script to launch the Greenstone Server Application dialog. A small dialog should appear where you can click the central *Enter Library* button. Then wait for this application to open a browser and display the Greenstone server home page for you. Note that if you're on linux or a Mac, you can use the equivalent `./gs3-server.sh` script to launch the same server application. For Mac binaries, you also have the option of double-clicking the `gs3-server` shortcut to launch this application.

Alternatively, if you choose to run the Greenstone server from the command line, then first run `gs3-setup.bat` to set up the Greenstone environment needed for WebSwing GLI to properly function, before running `ant start`. The linux and Mac equivalent is to run `source ./gs3-setup.sh` before similarly running `ant start` to start up the GS3 server.

7. Start the WebSwing Greenstone Librarian Interface:

Visit your running Greenstone server's home page in the browser. By default, for a locally running Greenstone, this would be at `http://localhost:8383/greenstone3/library`.

8. Scroll down to the link *Greenstone Librarian Interface (GLI)* and click it.
9. After some time a popup dialog will request a username and password. Enter the details for either the default *admin* account, or any user created earlier.
10. WebSwing GLI will load in your browser. Once it's finished loading up, you can start using GLI *almost* as usual.

WebSwing GLI is mostly the same as regular GLI except for a few notable differences:

- In the **Gather** panel, the **Workspace** tree view will not give you access to the file system (of either your machine or the remote machine running Greenstone) for web security reasons. With WebSwing GLI, you won't be gathering your documents by dragging and dropping them but by using a file browser to upload them into your collection:
When you have a new or existing collection open in WebSwing GLI, you will find an **Upload icon** at the bottom right of WebSwing GLI's **Gather** panel. You can **click** this Upload icon to upload either individual files or a zipped file from your local machine onto the remote Greenstone collection.
Often, you will find you need to upload many documents or folders of them, not individual files. In that case, first **zip up** your files and/or folders locally, then use the **Upload icon** to select the zip to

upload it.

(If GLI offers to add the ZipPlugin to your collection, you can choose to cancel out of it, as you generally mean to unzip them yourself in the gathering documents phase when using WebSwing GLI.)

Once the uploaded zip appears in the **Collection** tree view, **right click** on it and select **Unzip**.

- Regular GLI allows you to double-click on any document you've gathered to view it whenever you have a default application set up to view that type of file. WebSwing GLI can only allow you to view documents that your browser is able to display. Double-clicking on any other type of document will download it onto your local machine, where you can then view it if you have the appropriate application installed for its file type.
- Similar to the Upload icon in the Gather panel, in the **General** section of the **Format** panel you will find that the two browse buttons (the **URL to 'about page' image:** and **URL to 'home page' image:**), which allow you to select custom images to use for your collection on the home page and about page, now actually has the net effect of uploading image files you select from your local file system onto your collection on the remote Greenstone installation.

*For the rest, WebSwing GLI will work like regular GLI in the tutorials. So you are now set to, for instance, go through the **A simple image collection** tutorial using WebSwing GLI this time, bearing in mind the above differences. Try it out to familiarise yourself with WebSwing GLI.*

Copyright © 2005-2019 by the [New Zealand Digital Library Project](#) at [the University of Waikato](#), New Zealand

Permission is granted to copy, distribute and/or modify this document under the terms of the [GNU Free Documentation License](#), Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled [“GNU Free Documentation License.”](#)