7

Automating the Maintenance of Greenstone Collections

Wendy Osborn[1], Steve Fox[1], David Bainbridge[2] and Ian H.Witten[2]
[1]*University of Lethbridge*
[2]*University of Waikato*
[1]*Canada*
[2]*New Zealand*

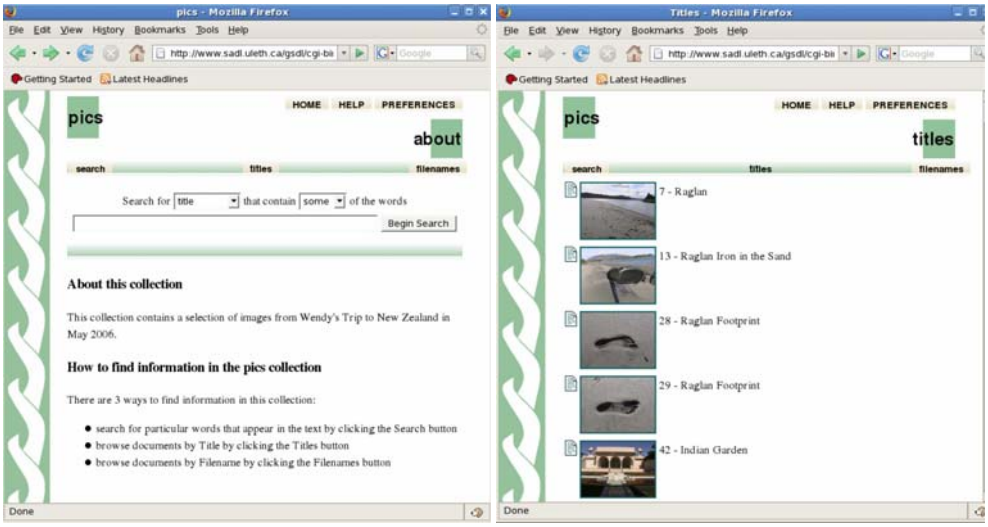
1. Introduction

Many applications generate multimedia documents, such as images and video, on a daily basis. For example, many municipalities have a photo-radar system to catch vehicles that violate traffic laws such as speeding or ignoring red lights. Many pictures of vehicle license plates are created every day. If these images are organized into a digital library, the collection would need to be updated regularly to incorporate new images. Another example is a traveller who wants to update a digital library of trip photos with new pictures of her travels while traveling around the world.

When documents and metadata are added to a digital library collection on a regular basis, such as hourly, daily or weekly, an automated and scheduled approach to collection maintenance is preferred over having to manually update the collection. In addition, an automated approach should be simple to use, therefore making time available for other important tasks.

Digital library software such as Greenstone (Witten et al., 2009), DSpace (Tansley et al., 2006) and Fedora (Lagoze et al., 2006) require that items be added manually to the collection. In Fedora, data is retrieved at the time of viewing. However, a location needs to be manually configured. Further, although Fedora and DSpace do provide application programming interfaces (APIs) to extend functionality, programming knowledge is required for using an API and for setting up tools based on it.

We present a solution for automating and scheduling updates that occur on a regular basis. Our solution is implemented in the Greenstone digital library software system (Witten et al., 2009), and comes in two parts. The first part of our approach is a command-line scheduling module. The Scheduler both automates the construction and modification of a collection, and schedules the construction to occur at specific intervals, such as hourly, daily or weekly. In addition, the owner of a collection can update the collection manually, without affecting the scheduled collection builds. Further, the Scheduler interacts with the existing task scheduling mechanism on the host system, which keeps the Scheduler minimal, yet powerful. The second part of our approach involves incorporating the Scheduler into the Greenstone Librarian Interface (GLI) (Witten, 2004). This will allow users who are more comfortable with managing collections through a graphical user interface to take advantage



(a) Front Page of pics Collection (b) Viewing pics Collection

Fig. 1. pics Collection in Greenstone

of the functionality of the Scheduler. In addition, this allows us to handle certain tasks associated with scheduling in a uniform and flexible manner.

This chapter proceeds as follows. Sections 2, 3 and 4 present background information on Greenstone, the Librarian Interface, and Cron. Section 5 presents the Scheduler, the command-line tool for scheduling automatic builds of Greenstone collections. Section 6 presents an evaluation of the Scheduler. Section 7 presents the extensions to the Librarian Interface required to support the Scheduler. Section 8 presents a scenario that overviews the use of the Scheduler from the Librarian Interface. Finally, Section 9 concludes the chapter and provides some future directions of research.

2. Greenstone

Greenstone (Witten et al., 2009) is a suite of software for creating digital library collections and making them available locally or via the Internet. A collection can contain documents of different formats, including images, PostScript and PDF files, audio, formatted and unformatted text, and many others. A collection built using Greenstone can be customized in many ways. For example, a collection owner can customize the types of documents that can appear in the collection, the appearance of the interface to the collection, and how the collection will be accessed by other users. In addition, Greenstone is extensible. For example, functionality to support other data formats that are not provided with Greenstone can easily be added to a collection.

There are two types of accessors in Greenstone. The first is an index that provides support for searching. The second is a classifier that provides support for browsing. A collection can be configured for browsing by any metadata that is specified by the collection owner or extracted by Greenstone. Furthermore, a collection can be configured for searching on the same metadata fields, as well as full text.

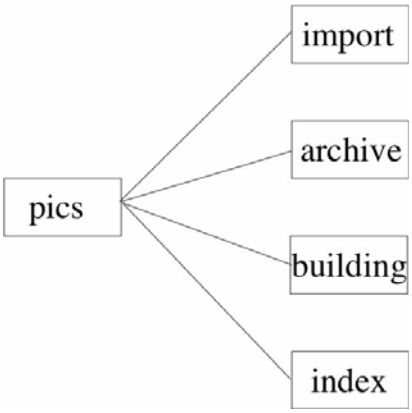


Fig. 2. Internal Representation of a Greenstone Collection

Figures 1(a) and 1(b) depict a Greenstone collection called *pics*. The *pics* collection is configured to accept images only. Figure 1(a) shows the front page of the collection, which provides general information on the collection. In addition, the collection is configured for both browsing and searching by either Title or Filename. Figure 1(b) shows a display of some images from the pics collection. In this view, the collection is being browsed by Title. The user can click on any image to obtain a larger image for viewing.

Figure 2 depicts the internal representation of a Greenstone collection. Here, the four main directories of a collection — *import, archives, building, and index* — are displayed. The purpose of each is described below:

- *import*. The *import* directory contains all documents that are to be added to the collection. If desired, the documents can be organized in a hierarchical directory structure within the *import* directory.
- *archives*. The *archives* directory stores all documents that have been processed from the import folder and added to the collection. All documents in the *archives* directory are represented in a canonical XML format.
- *building*. The *building* directory is a working directory for creating all indices and classifiers that are specified for the collection.
- *index*. The *index* directory contains the indices and classifiers after they are created.

A Greenstone collection is created or modified by the following four steps (Witten et al., 2009): document addition, document importation, accessor creation, and collection activation. Each step is described in detail below:

1. *document addition*. New documents that will be added to a collection are placed into the *import* directory for the collection.
2. *document importation*. Each document in the import directory is processed for inclusion by creating a canonical XML representation for it. This is accomplished by specifying an *import* command. Different importing options can be specified by the user. For example, documents in the *import* directory can be added to an existing collection by using a –*keepold* option. Alternatively, all documents in the *import* directory—new and existing—can be used to create a new instance of the collection by using a –*removeold* option. All imported documents are placed in the *archives* directory.

3. *accessor creation*. After the documents in the *import* directory are added to the collection, indices and classifiers are set up by processing the canonical XML representations of all documents. This is accomplished by specifying a *build* command. New indices and classifiers can be built by specifying a *–removeold* option. Alternatively, existing indices and classifiers can be modified by specifying a *–keepold* or *–incremental* option. The resulting indices and classifiers are located in the *building* directory.
4. *collection activation*. Finally, the collection is activated by moving the indices and classifiers from the *building* directory to the *index* directory. The collection is now viewable online.

The command for each step can be executed in a terminal or command window, or via the Greenstone Librarian Interface.

3. Greenstone Librarian Interface

The Greenstone Librarian Interface (GLI) (Witten, 2004) is a graphical user interface that provides a user-friendly method to build and configure Greenstone collections. It incorporates the four steps above. The only steps that are required by the user are to place documents to the import directory via the Gather panel, and to add the documents to the collection via the Create panel. In addition, the Librarian Interface allows a user to create new collections, select metadata sets, configure which document types to allow, and select *import* and *build* command options. Figure 3 shows the Librarian Interface.

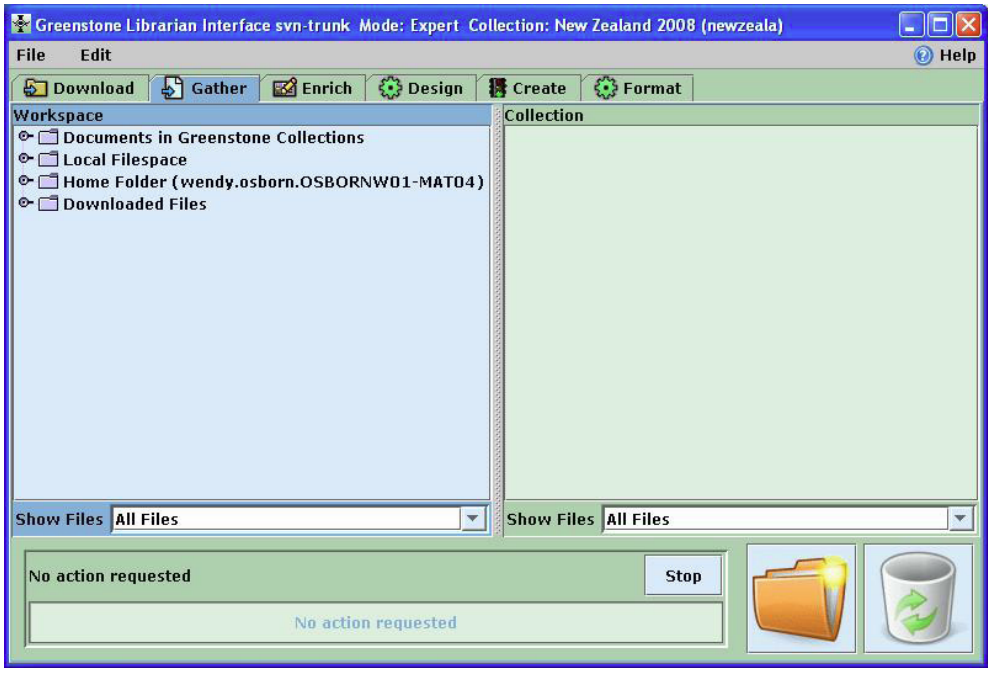


Fig. 3. The Greenstone Librarian Interface

4. Cron

Cron (Nemeth et al., 2007) is a program for users to schedule tasks that will run automatically at a specified time. A task can be one command, or a script containing several commands that are executed in sequence. Initially, Cron was implemented for Unix and Linux platforms, with most systems running Vixie Cron (Vixie, 1994). Mac OS X also runs Vixie Cron. In addition, versions of Cron now exist for Windows platforms, such as Pycron (Schapira, 2004). We summarize the general ideas behind all implementations of Cron that are applicable to our work.

Cron runs continuously in the background of the operating system. Every minute, Cron reads several task configuration files. Each such file is called a crontab file. A crontab file contains a record for every task that is scheduled for execution. Cron locates and runs all tasks that are scheduled at the current time.

The format of a crontab record is (*min hr dom moy dow user task*), where *min*, *hr*, *dom*, *moy*, and *dow* are the minute, hour, day of month, month of year and day of week, respectively, *user* is the username that the command will run under, and *task* is the command or script that is executed at the specified time. A task can be scheduled to run hourly, daily, weekly, monthly, or yearly:

- *hourly*. An hourly scheduled task executes every hour at a specified minute.
- *daily*. A task that is scheduled daily executes at a specific hour, which is required, and a specific minute. If no minute is specified, execution will occur every minute during the specified hour.
- *weekly*. A weekly scheduled task runs on a specified day, which is required, and at a specified hour and minute. If no hour is specified, execution will occur every hour during the specified day.
- *monthly*. A monthly scheduled task runs on a specified day (between the 1st and last day of the month).
- *yearly*. A yearly scheduled task runs on a specified month.

Any unspecified values are replaced with an asterisk in the crontab record.

Systems that use Vixie Cron support two types of crontab files – system crontab and user crontab. The system crontab files are primarily for system administration and maintenance tasks. Also, even if all tasks have a specified low-level username, root privileges are required for modifying a system crontab file. User crontab files, on the other hand, can be set up by any user on the system to execute tasks under their own username. Assuming the user has permission to execute the task, no root permissions are required. In addition, Pycron only supports user crontab files. Therefore, the Scheduler will employ a user crontab file.

Figure 4 displays a sample user crontab file. It contains 4 tasks that are scheduled for specific times. The first task, /collect/pics/gsdl.pl, is scheduled to be run at 30 minutes past every hour. The second task, /usr/bin/cleanup.bash, is scheduled for execution daily at 11:59pm.

```
30  *  *  *  * /collect/pics/gsdl.pl
59 23  *  *  * /usr/bin/cleanup.bash
00  6  *  *  7 /home/someuser/alarm
00  0  1  1  * echo "Happy New Year!"
```

Fig. 4. Sample Crontab File

The third task, home/someuser/alarm, is scheduled every Sunday at 6:00am. Finally, the fourth task, which echoes "Happy New Year!", is schedule for execution every January first at 12:00 AM.

5. The Scheduler

We first present our design for the Scheduler command-line module in Greenstone. The Scheduler is written in Perl and runs on Linux, Windows and Mac OS X. It also utilizes the Cron scheduling service (Nemeth et al., 2007) for scheduling collection re-building. We chose Perl because a Perl script can be executed across different platforms without the need to recompile. Similarly, we chose Cron because it is available for Linux and Mac OS X (Vixie, 1994), as well as Windows (Schapira, 2004). Therefore, we can maintain the the cross-platform requirement of Greenstone.
The Scheduler requires the following parameters from the user as input:

1. the collection to be rebuilt,
2. the full *import* command that is required to import documents into the collection,
3. the full *build* command required to construct the indices and classifiers for the collection, and
4. a specification of either an hourly, daily, or weekly build.

For example, if the user wants the collection *pics* to be scheduled for construction on a daily basis, the parameters would look something like this:

schedule.pl pics "import.pl –removeold pics"
"buildcol.pl –removeold pics" daily

where *pics* is the name of the collection, "*import.pl –removeold pics*" is the Greenstone command for importing documents into the collection *pics*, "*buildcol.pl –removeold pics*" is the Greenstone command for creating the required indices and classifiers for *pics*, and *daily* indicates that the collection will be scheduled for construction on a daily basis.
Using the arguments provided by the user, the Scheduler performs two main tasks: 1) create a script that automates the building of the collection (i.e. build script), and 2) create a crontab record that schedules the execution of the build script at specified intervals.

5.1 Automation script generation

The Scheduler generates a build script for any *import* and *build* command, and for the Linux, Windows or Mac OS X platforms. The build script contains all instructions that are required for building the specified collection. The commands include those for setting the Greenstone environment variables required for the collection building process, the *import* command and the *build* command.
Figure 5 shows a sample build script for Linux that contains the instructions for building the collection *pics*, as specified in the *schedule.pl* command above. The first four instructions set the environment variable that are required for the *import.pl* and *buildcol.pl* commands. The next two instructions are the specified *import* and *build* commands. The remaining instructions handle cleanup in order to activate the collection. Similarly, Figure 6 shows a sample build script for Windows that is generated for the same *schedule.pl* command above.

```
#!/usr/bin/perl

$ENV{'GSDLHOME'}="/gsdl";
$ENV{'GSDLOS'}="linux";
$ENV{'GSDLLANG'}="EN";
$ENV{'PATH'}="/usr/local/gsdl/bin/script:/usr/local/gsdl/bin/linux";
system("import.pl -removeold pics");
system("buildcol.pl -removeold pics");
system("\\rm -r /gsdl/collect/pics/index/*");
system("mv /gsdl/collect/pics/building/*
            /gsdl/collect/pics/index/");
system("chmod -R 755 /gsdl/collect/pics/index/*");
```

Fig. 5. Sample Automation Script for Linux

```
#!/usr/bin/perl

$ENV{'GSDLHOME'}="C:\\gsdl";
$ENV{'GSDLOS'}="windows";
$ENV{'GSDLLANG'}="en";
$ENV{'PATH'}="C:\\gsdl\\bin\\windows\\perl\\bin;C:\\gsdl\\bin\\windows;
C:\\gsdl\\bin\\script;C:\\Perl\\bin\\;
C:\\WINDOWS\\system32;C:\\WINDOWS;
system("import.pl pics");
system("buildcol.pl pics");
system("rd \/S \/Q \"C:\\gsdl\\collect\\pics\\index\"");
system("md \"C:\\gsdl\\collect\\pics\\index\"");
system("xcopy \/E \/Y \"C:\\gsdl\\collect\\pics\\building\\*\"
\"C:\\gsdl\\collect\\pics\\index\\\"");
```

Fig. 6. Sample Automation Script for Windows

5.2 Crontab generation

A crontab record is created to execute the automation script at a specified interval. The user has the option of specifying an hourly, daily or weekly build. An hourly build is scheduled for the beginning of the hour, a daily build is scheduled for midnight, and a weekly build is scheduled for Sunday morning at midnight. After creating the crontab record, it is added to the crontab file or replaces an existing crontab record. Figure 7 depicts the daily crontab record for the automation script in Figure 5, while Figure 8 depicts the crontab record corresponding to the automation script in Figure 6.

```
00 0 * * * /gsdl/collect/pics/gsdl.pl
```

Fig. 7. Crontab Record for Linux

```
00 0 * * * c:\gsdl\collect\pics\gsdl.pl
```

Fig. 8. Crontab Record for Windows

6. Evaluation

In this section, we discuss the performance of the Scheduler. The focus of our experiments is to evaluate the Scheduler for correct execution in specific situations. For correct execution, we looked at the following:

- *Crontab*. Does the Scheduler generate a correct crontab record? A crontab record is correct if: 1) it is accepted by the crontab program (Linux and Mac OS X only), and it causes Cron to execute the automation script at the proper time.
- *Automation Script*. In addition, does the scheduler generate the correct automation script to build a collection. An automation script is generated correctly if the collection it is created for is rebuilt correctly given the parameters that are specified when the script is created.

We conducted three experiments. The first is an hourly build of one collection. The second is an hourly build of two collections. The third is a daily build of one collection. All three experiments were performed on both Linux and Windows. These tests were not performed extensively on Mac OS X. However, the scheduler was executed many times on this platform to ensure that it does work.

6.1 Hourly build of one collection

The focus of this experiment is to ensure that the Scheduler produced a correct crontab record and automation script to re-build a collection of images every hour for 24 hours. To determine that each execution of the automation script was successful, we looked at the following. For Linux, Cron was configured to send an email message containing the output of the automation script. The email message for each of the 24 builds contains an output for a successful Greenstone build. For Windows, Pycron maintains a log that contains two records for each automation script execution – one record indicating the start of execution, and one record indicating the end of execution and a return code. The terminating records for all 24 builds have a return code of zero, which indicates a successful execution of the automation script.

In both cases, a visual inspection of the collection was also performed periodically to verify the success of the Scheduler.

6.2 Hourly build of two collections

The focus of this experiment is to ensure that if multiple collections are scheduled to be rebuilt at the same time, they are successfully rebuilt. We scheduled the building of two collections on hourly intervals for 24 hours. A perusal of email messages (for Linux) and the log (for Windows) verify that both collections were rebuilt successfully.

6.3 Daily build of one collection

The focus of this experiment is twofold. The first is to ensure correct daily execution of an automation script generated by the Scheduler. The second is to simulate a situation of a collection that has images added to it on a daily basis. We schedule one collection to be built daily for seven days. The collection starts with 10 images, and 10 images are added daily. In addition to the successful daily build, the collection did reflect the addition of the new images that arrived daily.

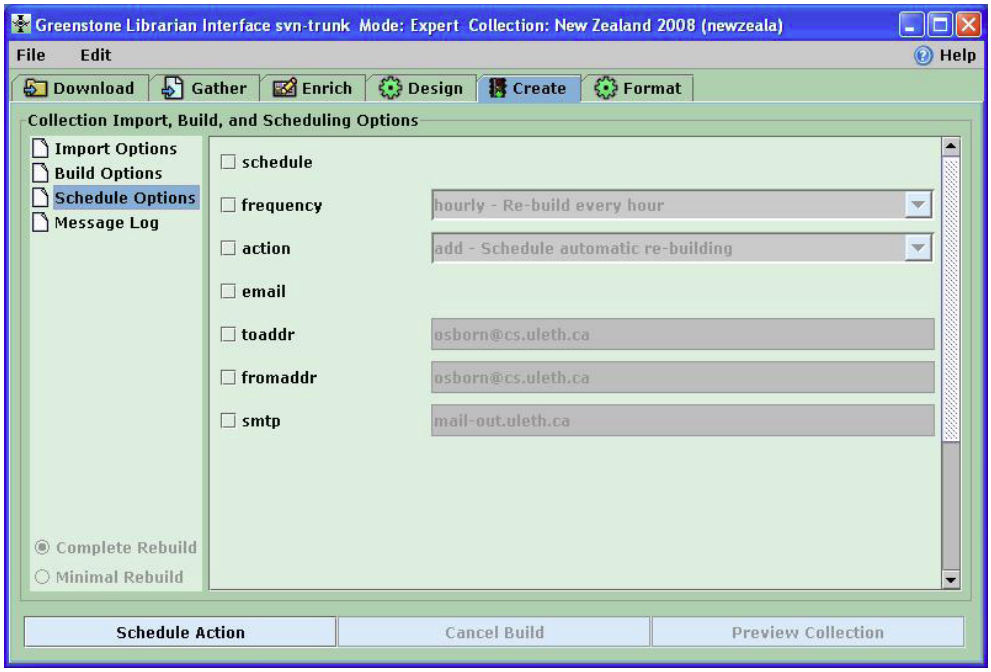


Fig. 9. The Schedule Options Panel

7. Scheduling in the Librarian Interface

The Scheduler is a minimal, yet powerful tool for maintaining Greenstone collections. It hides the details concerning the collection building process, and also the details for scheduling a Cron task. However, the Scheduler has two main limitations. The first is that the user is still required to know the syntax and command-line parameters for both the *import* and *build* commands in order to perform scheduling. The second is that, currently, notification of collection building success—or failure—is still dependent on the version of Cron (and indirectly, the operating system it is running on) that is used.

The Librarian Interface provides a user-friendly tool for building collections, including configuring the *import* and *build* commands. Therefore, it is ideal to extend the Librarian Interface to allow the configuration of the Scheduler as well. Figure 9 depicts the Librarian Interface extension to support the scheduling of collection builds. Currently, the Create panel is displaying most of the parameters (i.e. Schedule Options) for the Scheduler.

Notice that no explicit options exist for the *import* and *build* commands. This is because the *import* and *build* commands are created based on the arguments selected from the Import Options and Build Options panels. Therefore, the user no longer needs to know the exact syntax of the *import* and *build* commands.

7.1 User modes

The Librarian Interface has four modes of use— Library Assistant, Librarian, Library Systems Specialist, and Expert—stepwise increasing the functionality available to the user

(Witten, 2004). It was important to determine which modes would have access to the Scheduling Options, and for those modes that were granted access, how much access each would be granted. It was decided that Library Systems Specialists and Experts would be provided access to the Scheduling Options. In addition to determining which options to make available to each user mode, it is also necessary to determine how the Scheduler would interact with the existing *import* and *build* functionality in the Librarian Interface.

7.1.1 Mode level for options

The Scheduler was first modified so that the passing of command-line arguments conformed with that of other Greenstone commands, such as *import* and *build*. This also allowed us to easily specify which user mode could have access to each argument when it is added as a Schedule Option in the Librarian Interface. The Expert user mode is granted full access to all Schedule Options, while Librarian Systems Specialist is only granted limited access to options. This user mode can only specify whether or not to schedule a collection build with the default values—a frequency of hourly, and no sending of email.

7.1.2 Scheduling and building

Another important decision was how the Scheduler would interact with the collection building functionality of the Librarian Interface. The question asked was, should scheduling be done at the same time as collection building, or be a completely separate task?

For Expert mode, the functionality for Scheduling is separate from that of collection building. This is because an expert user may want to configure and manually re-build their collection before scheduling an automatic re-build of it. For Library System Specialist mode, the Scheduling functionality is done at the same time as collection building. If the specialist chooses to schedule, a new scheduled build is created. If the specialist chooses not to schedule, any existing scheduled builds are deleted.

7.2 Cron event logs

It is important to maintain logs that keep track of the outcome of a scheduled collection build. Both Vixie Cron and Pycron maintain a log that keeps track of the attempted execution of all scheduled tasks. Neither scheduling service keeps track of the success or failure of a scheduled task, nor do they keep track of the output of a task. In addition, to view logs created by Vixie Cron, the user must have root access.

Another desirable feature of maintaining logs is the ability to be able to only record output that is considered important, and to disregard all other task output. For example, if the output from the input and build commands of Greenstone is required, but the output from moving indices and classifiers is not considered important, the log should reflect this.

Therefore, the Scheduler is modified in two ways to handle the logging of building script output. The first is to create a custom log for each execution of the automation script. The automation script creates a unique filename every time it is executed by using a timestamp. The second is to specify in the automation script which actions will have its output redirected to the logfile. The actions whose output is to be ignored will have its output redirected to the 'bit bucket' (e.g. /dev/null/ in Linux).

7.3 Email notification

It is also important than an email notification service be provided, which will inform users of the outcome of their scheduled collection build. We handle email notification from the Scheduler and Librarian Interface for the following reasons:

1. *User notification*. Whether Cron notifies users about the outcome of a scheduled task depends on the its implementation. For Vixie Cron, the outcome of a task (i.e. output from either successful task completion, or error output) is emailed to the owner of the task. Pycron does not send email notification.
2. *Flexibility of Notification*. In Vixie Cron it is possible to suppress email notification, either by setting an environment variable to null, or by redirecting all output to a file or the 'bit bucket'. However, this is normally an all-or-nothing event—either all output, or none, is sent by email. Similar to logging, a desirable feature would be to send email that contains only the most important parts of the building process, and ignores other parts of the building process.
3. *Greenstone Email Support*. Greenstone comes with a Perl email script, that is a wrapper for the Perl sendmail command. The email script is platform-independent. Therefore, it is ideal to use it to provide a uniform way to send email concerning the execution of a scheduled task. In addition, the script does not require the piping of build output directly to it, but instead can send the contents of a file.

Therefore, both the Librarian Interface and the Scheduler are modified so that email notification is handled in a uniform and user-friendly manner across all operating systems.

First, options have been added to the Scheduler that are required for the Perl email script—specifically, a flag to specify that email will be sent (*–email*), the sender (*–fromaddr*), receiver (*–toaddr*) and the email server that will be used to send the email (*–smtp*). Also, the corresponding fields exist in the Schedule Options pane of the Librarian Interface. In order to assist users in using the email features of scheduling, the Librarian Interface attempts to populate the fields *–toaddr*, *–fromaddr*, and *–smtp* in the Schedule Options pane by consulting the configurations for the Librarian Interface and Greenstone. If suitable values are available from these sources, they are assigned to the appropriate fields.

Second, the output from the building script must be captured and re-directed to the Perl email script. The capturing of output already takes place, in the event log. This serves as the file that the email script will send to the user. Also, since it contains only the output that is considered important, this will be reflected in the email message as well.

Finally, the Scheduler is modified so that, if specified, the automation script will send an email message containing the contents of the log to the specified recipient. An added bonus is that if email is not specified, the log can still be consulted by the user if required.

7.4 Scheduled building in isolation

An important modification to the Scheduler is to ensure that a scheduled build is completed in its entirety without interference from another scheduled build. A build may take a significant amount of time depending on the size of the collection—from seconds for a small one to 33 hours for a collection containing 20 GB of raw text and 50 GB of metadata (Boddie et al., 2008).

To handle this, the automation script for the collection checks for a lock file, which indicates that a collection build is underway. If the file exists, the collection owner is notified via email and information is placed in the event log. Otherwise, a lock file is created before the

scheduled build begins, and is removed when the build finishes. This ensures that multiples builds do not occur concurrently.

8. Example: collecting pictures while traveling

In this section, we present a simple application of scheduling from the Librarian Interface. In this scenario, we have a traveler who wants to post pictures of their trip in a Greenstone collection for her friends to view. Instead of waiting until the end of the trip, the traveler wants to post her pictures from each day, incrementally adding to the collection on a daily basis. The traveler does not want to worry about obtaining the Librarian Interface to rebuild the collection while traveling. Instead, she simply wants to upload the pictures to the *import* directory of her collection, and have her collection rebuilt automatically and on a daily basis. This can be accomplished by setting up a scheduled, automatic rebuild of the collection of travel photos from the Librarian Interface.

First, before departing, the traveler runs the Librarian Interface and creates a new collection. Then, the user selects the Create tab to display the collection creation pane. From here, clicking on Schedule Options will display the available options for setting up a scheduled, automatic collection build of the collection of travel pictures. Figure 10(a) depicts the available scheduling options, which are displayed with default and derived values as appropriate. Here, the traveler selects schedule, which indicates that she wants to set up a scheduled, automatic build. Also, she selects a frequency of hourly and an action of add (or, to create a new scheduled build).

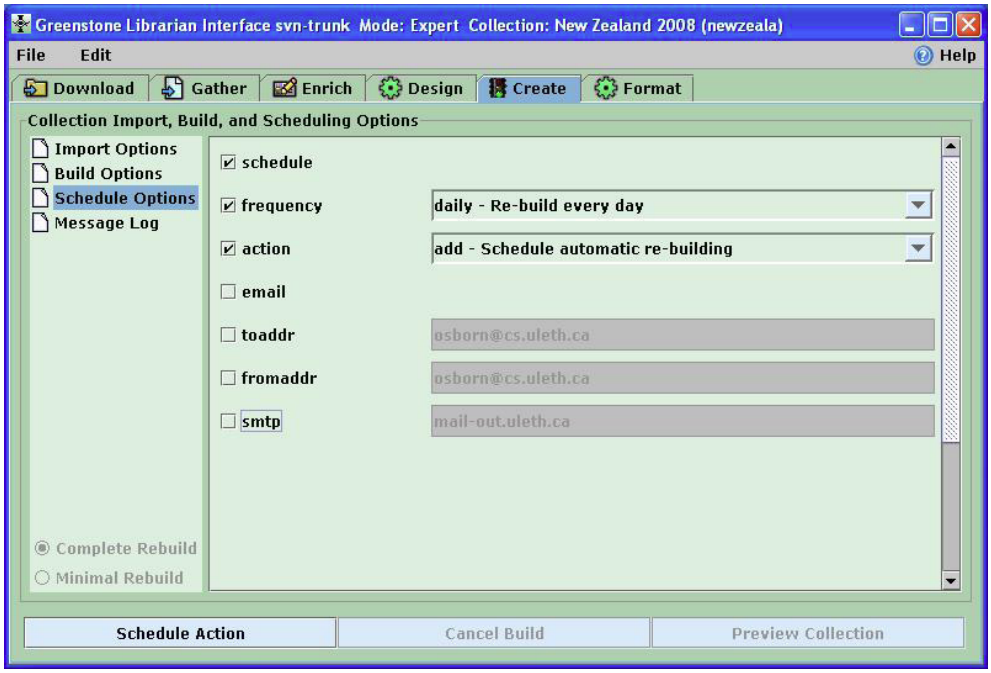
Next, the traveler clicks on Schedule Build. This will set up the building script for the collection of travel pics, as well as the *crontab* record that will indicate to Cron that the collection is to be rebuilt daily. The collection is now ready to be re-built while the traveler is away.

At the end of the first day of travel, she uploads three pictures, which are added to the collection when the collection is re-built automatically overnight. The updated collection is depicted in Figure 10(b). The next day, the traveler uploads three more pictures. When the collection is re-built overnight, these pictures are added to the existing collection. Figure 10(c) depicts the updated collection with the new pictures.

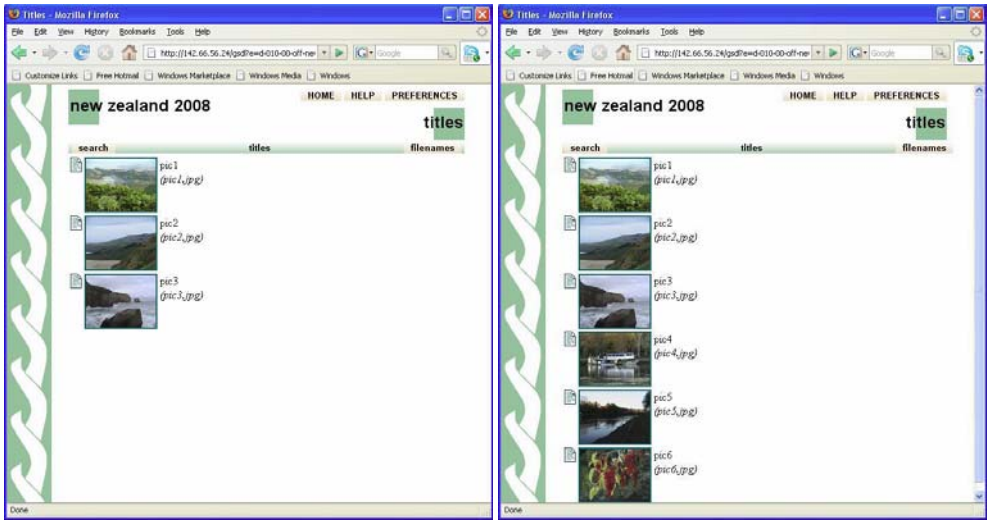
Although not shown here, the user can switch to the Import Options and Build Options and select any options that are required for collection importing and building. These options are incorporated into the automatic building script that is created for the collection. In addition, the user can manually build and configure their collection as many times as necessary to confirm the right sequence is being performed, before setting up a scheduled, automatic build.

9. Conclusion and future work

In this paper, we present our solution to automated and scheduled collection maintenance in Greenstone. First, we propose the Scheduler, which provides support for an automatic rebuild of a collection at specific intervals by specifying a few simple parameters. The Scheduler interacts with a resident task scheduler on the local operating system, which results in a minimal but powerful tool. Several experiments are performed to show the correct execution of the Scheduler for different build times and for different numbers of collections.



(a) Schedule Options Panel



(b) Build - First Day

(c) Build - Second Day

Fig. 10. Collection Build Scheduling

In addition, we propose the incorporation of the Greenstone Scheduler into the Librarian Interface. This overcomes two limitations of the Schedule—the requirement to know the syntax and command-line arguments for both the *import* and *build* commands, and the inconsistency of notification of collection building success or failure. Providing an interface to the Scheduler improves its usability and provide further abstraction of the scheduling process from the user. Some future directions of work include the following. The first is to allow the user to select a specific period of time (e.g. 20 minutes after the hour) for their collection to be re-built. Currently, collection building occurs at the top of the hour (hourly), at midnight (daily) and on Sunday at midnight (weekly). The second is support for dependencies between fields in the Librarian Interface. For example, if a user selects the email option, it requires *–toaddr*, *–fromaddr*, and *–smtp*. Currently, the user must ensure that these are also selected, as it is not done automatically. A final research direction is to provide support for an automatic re-build of a collection when certain events are triggered. For example, the arrival of a new document can trigger an automatic rebuild of the collection.

10. References

Boddie, S., Thompson, J., Bainbridge, D. &Witten, I. H. (2008). Coping with very large digital collections using greenstone, *Proc. of the ECDLWorkshop on Very Large Digital Libraries*.

Lagoze, C., Payette, S., Shin, E. &Wilper, C. (2006). Fedora: an architecture for complex objects and their relationships, *International Journal on Digital Libraries* 6(2): 124–138.

Nemeth, E., Snyder, G. & Hein, T. R. (2007). *Linux Administration Handbook*, Prentice-Hall.

Schapira, E. (2004). Python cron - great cron for windows. Website. Last visited June 2010. http://sourceforge.net/projects/pycron.

Tansley, R., Bass, M. & Smith, M. (2006). Dspace as an open archival information system: Status and future directions, *Proc. of the 10th European Conference on Digital Libraries*.

Vixie, P. (1994). Vixie cron for FreeBSD. Website. Last visited June 2010. http://www.freebsd .org/cgi/cvsweb.cgi/src/usr.sbin/cron/.

Witten, I. H. (2004). Creating and customizing collections with the Greenstone Librarian Interface, *Proc. of the Int'l Symp. on Digital Libraries and Knowledge Communities in Networked Information Society*.

Witten, I. H., Bainbridge, D. & Nichols, D. (2009). *How to Build a Digital Library*, Morgan Kaufmann.