How to build your own digital library with Greenstone

The Greenstone Digital Library Software from the New Zealand Digital Library project provides a new way of organizing information and making it available over the Internet. It offers many unique features.

- Accessible over the internet via a standard Web browser
- Provides full-text searching for all collections
- Provides flexible browsing facilities tailored to individual collections
- Facilitates maintainability by creating all searching and browsing structures automatically
- Takes advantage of any metadata that is available
- Uses a scheme of “plugins” to facilitate extension to different document types
- Is designed for large (multi-gigabyte) document collections
- Works with documents of any language (including Chinese, Arabic, Maori)
- Easy to tailor the presentation interface to different languages
- Can accommodate text, pictures, and audio
- Makes use of advanced compression techniques to reduce storage
- Keeps logs of user activity
- Collections can be made available in precisely the same form on CD-ROM
- CD-ROMs are easy to install operate on Windows 3.X, 95, 98, and NT
- Operates under Unix and Windows NT, and works with standard Web servers
- Is open-source software, distributed under the Gnu public licence.

This manual provides a comprehensive introduction to the software and its capabilities. It is divided into four main parts, corresponding to different kinds of use.

The first part gives an overview of the capabilities of the software, and includes a comprehensive glossary of terms used throughout the manual.

The second part is targeted at end users of Greenstone collections. It begins by describing the very simple installation procedure for CD-ROMs. Following that there is a complete account of how to find information in Greenstone collections. In fact, the interface is fairly self-explanatory—the best way to learn is by doing—and this section essentially comprises the on-line help information for a typical collection.

The third part is for those who want to build their own library collections using Greenstone. To do this, you need some understanding of the files and file structure used within the system, and the overall process of creating a collection—which involves assembling the source documents, “importing” them into the Greenstone system, and then “building” the full-text indexes and browsing structures. We also describe the file format of imported documents, although this information is not usually necessary for building collections.

Next we show how to update an existing collection by adding new material and rebuilding it. This is easy to do: the software is designed for ease of maintenance of digital library collections.

Building a new collection from scratch can be more challenging. It is easy if there is an existing collection that can be used as an exact model, with the same source file structure and the same kind and format of metadata—for then you can just copy the relevant specifications for the existing collection and rebuild. For genuinely new collections, we describe the "plugins" that are available to accommodate different document formats, and the "classifiers" that are available to build browsing structures—like sorted lists of names or dates, or hierarchical browsers for classification hierarchies. We also describe how you can control the format of what is presented on the screen.

The third part ends with a description of how to put a collection onto a standalone Greenstone CD-ROM that can be distributed for others to use, and discusses how to translate the interface into other languages that are not yet supported.

The fourth part describes how to install the full Greenstone software on Unix systems and on Windows 95/98/NT. Most users will not need to install the software, and can skip this section.

The fifth part is for those who want to modify the software to add new capabilities. We begin by describing the directory structure in which the software resides. The remainder of this section is still being written.

The Greenstone software is developing rapidly, and documenting a moving target is difficult. At this stage there are quite a number of small things left to do. In some cases these are described here as though they were complete; these sections have been highlighted. Most of the highlighting will be removed within the next few weeks.

A. Overview of Greenstone

The Greenstone Digital Library is organized in as a number of independent collections. A *collection* of information comprises several (typically several thousand, or even several million) *documents*, and a uniform interface is provided to all documents in a collection. The collections in a library are each organized in a different way—though there is a strong family resemblance in the way collections are presented.

Collections are accessed through a standard Web browser (for instance, Netscape or Internet Explorer) and combines easy-to-use browsing with powerful search facilities. There are several ways to find information in most Greenstone collections. For example, you can *search for particular words* that appear in the text, or within a section of a document. You can *browse documents by title*: just click on a book to read it. You can *browse documents by subject*. Subjects are represented by bookshelves: just click on a bookshelf to look at the books. Where appropriate, documents come complete with a table of contents: you can click on a chapter or subsection to open it, expand the full table of contents, or expand the full document into your browser window (useful for printing). The international Unicode character set is used internally, so documents can be written in any language. The interface is available in English, French, Spanish, German, Maori, Chinese, and Arabic. The New Zealand Digital Library website (www.nzdl.org) provides numerous example collections.

Along with each collection is a statement of its purpose and coverage, and an explanation of how the collection is organized. Most collections can be accessed by both *searching* and *browsing*. Searching is full-text, and the user can choose between indexes built from different parts of the documents. Some collections have an index of full documents, an index of sections, an index of paragraphs, an index of titles, and an index of section headings, each of which can be searched for particular words or phrases. Browsing involves lists that the user can examine: lists of authors, lists of titles, lists of dates, hierarchical classification structures, and so on. Different collections offer different browsing facilities. Indexes for both browsing and searching are constructed during a process called *building* according to instructions in a collection configuration file.

Rich Web browsing facilities can be provided by manually linking parts of documents together and building explicit tables of contents and indexes. However, manually-created structures are difficult to maintain, and inevitably fall into disrepair when the collection expands. The Greenstone software takes a different tack: it facilitates *maintainability* by creating all searching and browsing structures automatically from the documents themselves. No links are inserted by hand. This means that if new documents in the same format become available, they can be merged into the collection automatically. Indeed, for some collections this is done by processes that wake up regularly, scout for new material, and rebuild the indexes—all without manual intervention.

Collections comprise many documents: usually thousands, tens of thousands, or even millions. Each document may be hierarchically organized into *sections* (and subsections, and sub-subsections). Each section comprises one or more *paragraphs*. *Metadata*, which is descriptive information such as author, title, date, keywords, and so on, may be associated with each document, or with individual sections of documents. Metadata is the raw material for browsing indexes. It must either be provided explicitly (for example, in a spreadsheet) or be derivable automatically from the source documents. Metadata is converted to a standard form (using the Dublin Core metadata scheme) and stored with the document for internal use.

In order to accommodate different kinds of source documents, the software is organized in such a way that “plugins” can be written for new document types. Plugins currently exist for plain text documents, html documents, and for recursively traversing directory structures containing such documents. A collection may have source documents in different forms: it is just a matter of specifying all the necessary plugins. In order to build browsing indexes from metadata, an analogous scheme of “classifiers” is used: classifiers create indexes of various kinds based on metadata. Source documents are brought into the Greenstone system through a process called *importing*, which uses appropriate plugins and classifiers as specified in the collection configuration file.

Greenstone collections can contain text, pictures, and even audio clips. Compression technology is used to ensure best use of storage. (Throughout the Greenstone software, the MG software is used for compression and indexing: see Witten, I.H., Moffat, A. and Bell, T. *Managing Gigabytes: compressing and indexing documents and images*, Morgan Kaufmann, second edition, 1999.) Most non-textual material is either linked to the textual documents or accompanied by textual descriptions (such as captions for photos) to allow full-text searching and browsing.

The Greenstone system includes an “administrative” function whereby specified users can examine the composition of all collections, protect documents so that they can only be accessed by registered users on presentation of a password, and so on. Logs of user activity are kept that record all queries made to every Greenstone collection (though this facility can be disabled).

Although primarily designed for Internet access over the World-Wide Web, Greenstone collections can be made available, in precisely the same form, on CD-ROM. A Greenstone CD-ROM can be used on a standalone PC to access the information on the disk. The user interface is through a standard Web browser (Netscape is provided on each CD), and the interaction is identical to accessing the collection on the Web—except that response times are faster and more predictable. If the PC is connected to a network (intranet or Internet), the Greenstone software acts as a network server, using a custom-built Web server provided on each CD, to make exactly the same information available to others who need only use their standard Internet browser software. The software works under all Windows 3.X, 95, 98, and NT operating systems; it can be installed straight off the CD-ROM in a matter of seconds. The use of compression technology ensures that the greatest possible volume of information can be packed on to a CD-ROM.

The collection-serving software operates under Unix (currently tested under Linux and SunOS) and Windows NT, and works with all standard Web servers. A flexible process structure allows different collections to be served by different computers, yet be presented to the user in the same way, on the same Web page, as part of the same digital library. Collections can be updated and new ones brought on-line at any time, without bringing the system down; the process responsible for the user interface will notice (through periodic polling) when new collections appear and add them to the list presented to the user. The software architecture allows for different parts of a collection to be distributed over several computers, and for different collections to be searched together, and for a single collection to be replicated on several computers and the user to receive service, transparently, from the one that responds quickest—though these are not yet implemented.

The Greenstone Digital Library Software is open-source software, available from the New Zealand Digital Library (www.nzdl.org) under the terms of the Gnu public licence. The software includes everything described above: Web serving, CD-ROM creation, collection building, multi-lingual capability, plugins and classifiers for a variety of different source document types. It includes an autoconfigure script to allow easy installation on Unix systems. In the spirit of open-source software, users are encouraged to contribute modifications and enhancements.

Glossary

Term	Meaning
autoconf	Unix program used to configure the Greenstone software installation package to suit your system
Autorun	Windows feature that starts a program automatically whenever a CD-ROM is inserted
Boolean query	Query to an information retrieval system that may contain AND, OR, NOT
Browsing	Accessing a collection by scanning an organized list of metadata values associated with the documents (such as author, title, date, keywords)
buildcol.pl	Program used to build collections
Building	Process of creating the indexing and browsing structures that are used to access a collection
C++	Programming language in which the majority of the Greenstone software is written
Casefolding	Making uppercase and lowercase words look the same, for searching purposes
CGI	Common Gateway Interface, a scheme that allows users to activate programs on the host computer by clicking on Web pages
CGI script	Code associated with a button, menu, or link on a Web page that specifies what the host computer is to do when it is clicked
cgi-bin	Directory in which CGI scripts are stored (differs from system to system)
Classifier	Greenstone code module that examines document metadata to form an index for browsing
Collection	Set of documents that are brought together under a uniform searching and

	browsing interface
Collection configuration file	File that specifies how a collection is to be imported and built, what indexes and language interfaces are to be provided, etc
Collection server	Program responsible for providing access to a collection when it is being used
Configuration file	See collection configuration file, main configuration file, site configuration file
CVS	Concurrent Versioning System, a scheme for maintaining source code used throughout Greenstone
db2txt	Tool for viewing a GDBM database as text (see GDBM)
Demo collection	A subset of the Humanities Development Library, distributed with the Greenstone software and used for illustration in this tutorial
Digital library	Collection of digital objects (text, audio, video), along with methods for access and retrieval, and for selection, organization, and maintenance
Document	Basic unit from which digital library collections are constructed; it may include text, graphics, sound, video, etc.
Dublin core	A standard way of describing metadata
Fast CGI	Facility that allows CGI scripts to remain continuously active so that they do not have to be restarted from scratch every time they are called for
Filter program	That part of a collection server that implements querying and browsing operations
Format string	A string that specifies how the Greenstone software should display document and other istings
GB-encoding	Standard way of encoding Chinese and other oriental languages
GDBM	Gnu database manager, a program used within the Greenstone software to store metadata for each document
Gimp	Image-manipulation program used (on Unix) to create icons in Greenstone
GML	Greenstone Markup Language, a file format used for storing documents internally
Gnu licence	Software licence that permits users to copy and distribute computer programs freely, and modify them—so long as all modifications are made publicly available
Greenstone	The name of this digital library software
GSDL	Abbreviation for Greenstone Digital Library
$GSDLHOME	Operating system variable that represents the top-level directory in which all Greenstone programs and collections are stored (*%GSDLHOME%* on Windows systems)
$GSDLOS	Operating system variable that represents the operating system currently being used (*%GSDLOS%* on Windows systems)
hashfile	Program used at import or build time to generate the OID of each document
HDL	Humanities Development Library, a Greenstone collection of humanitarian information for developing countries
HTML	HyperText Markup Language, the language in which Web documents are written
import.pl	Program used to import documents

Importing	Process of bringing collections of documents into the Greenstone system
Index	Information structure that is used for searching or browsing a collection
InstallShield	Windows program that allows a system to be installed from a CD-ROM, used by Greenstone CD-ROMs
Main configuration file	File that contains specifications common to all collections served by this site
Metadata	Descriptive data such as author, title, date, keywords, and so on, that is associated with a document (or document collection)
MG	Managing Gigabytes, a program used by the Greenstone system for full-text indexing, that incorporates compression techniques (see Witten, I.H., Moffat, A. and Bell, T. *Managing Gigabytes: compressing and indexing documents and images*, Morgan Kaufmann, second edition, 1999)
mgbuild	MG program for building a compressed full-text index
mgquery	MG program for querying a compressed full-text index
mkcol.pl	Program that creates and initializes the directory structure for a new collection
New Zealand Digital Library	Research project in the Computer Science Department at the University of Waikato, New Zealand, that created the Greenstone software (www.nzdl.org)
OID	Object Identifier, a unique identification code associated with a document
Perl	Programming language used for many of the text-processing operations that occur during the building process
Ping	Message sent to a system to determine whether it is running or not
Plugin	Code modules for handling documents of different formats, used during the importing and building processes
Protocol	Set of conventions by which a receptionist communicates with a collection server
Ranked query	Natural-language query to an information retrieval system, for which the documents that match the query are sorted in order of relevance
Receptionist	Program that organizes the Greenstone user interface
RTF	Rich Text Format, a standard format for interchange of text documents
Searching	Accessing a collection through a full-text search of its contents (or parts of contents, such as section titles)
Server	See Collection server, Web server
setup.sh, *setup.csh*, *setup.bat*	Command used to set up your environment to recognize the Greenstone software
Site configuration file	File that contains specifications used to configure the Greenstone software for the site in which it is installed
Stemming	Stripping endings off a query term to make it more general
STL	Standard template library, a widely-available library of C++ code
txt2db	Program used at build time to create the GDBM database
Unicode	Standard scheme for representing the character sets used in the world's languages
UNU	The United Nations University; also used to refer to a Greenstone collection created for that organization
Web server	Standard program that computers use to make information accessible over the

B. Using Greenstone collections

The Greenstone software is designed to be easy to use. Web-based and CD-ROM collections have interfaces that are identical. Installing a CD-ROM collection on any Windows computer is very easy indeed; a standard installation setup program is used. A CD-ROM collection can be used locally on the computer where it is installed; also, if this computer is connected to a network, the software automatically and transparently gives network access to the same collection to any other computers on the network.

If the software is to be used on a Unix server, or if it will be put to serious large-scale use as a Web server, or if more than one collection is to be offered, it will be necessary to install the complete Greenstone software on the server. This is a rather more complex procedure because it requires auxiliary software to be installed and may need special configuration options to be set. Most Greenstone users will never have to undertake the installation of the software on their computer; however, for those who do, the procedure is described in section B.2 below. Most readers should skip this section; it is of no relevance to everyday Greenstone use.

In Section B.3 we describe the searching and browsing facilities offered by a typical Greenstone collection, the “Demo” collection that is supplied with the Greenstone software. Other collections offer similar facilities; if you can use one, you can use them all.

B.1 Using a Greenstone CD-ROM

To use a Greenstone CD-ROM, just put it into the CD-ROM drive on any Windows PC. Most likely (if “autorun” is enabled on your PC), a window will appear inviting you to install the Greenstone software. If not, find the CD-ROM disk drive (on current Windows systems you can get this by clicking on the *My Computer* icon on the desktop) and double-click it, ir the *install.exe* file inside it. In either case the Greenstone Setup program will be entered, which guides you through the setup procedure. Most people respond *yes* to all the questions except for the one which offers to install the Netscape browser; if you already have a browser you probably don’t need to install a new one.

When the installation procedure has finished, you’ll find the library in the *Programs* submenu of the Windows *Start* menu, under the name of the collection (for example, “Humanity Libraries” or “United Nations University”). There will be a *Standalone Version* as well as a *Server or Network* version, and an *Uninstall* option which will completely remove the library and associated software from your system should you so desire. You should use the *Server or Network* version whether or not you are connected to a network; the *Standalone Version* is only supplied in case the system experiences difficulty in ascertaining your network setup.

Once the software has been installed, the library will be entered automatically every time you re-insert the CD-ROM if autorun is enabled.

B.2 Finding information in a Greenstone collection

The easiest way to learn to use a Greenstone collection is to try it out. Don't worry—you can't break anything. Click liberally: most images that appear on the screen are clickable. If you hold the mouse stationary over an image, most browsers will soon pop up a message that tells you what will happen if you click.

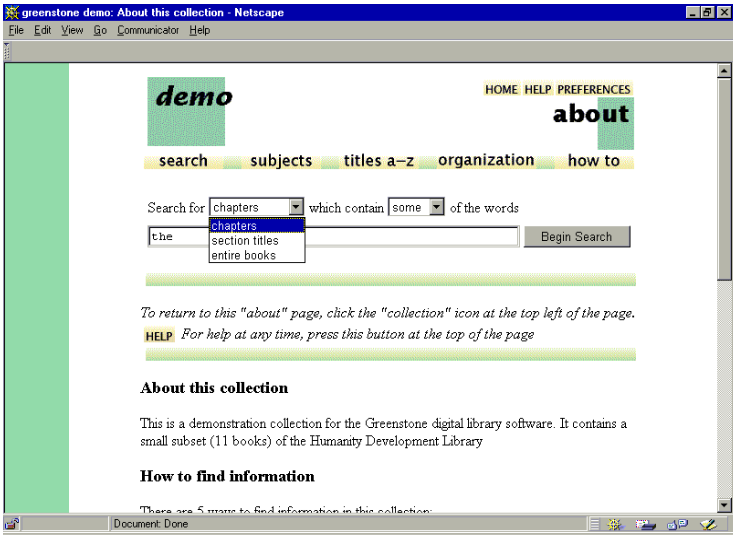


Figure 1 Using the Demo collection

Experiment! Choose common words like "the" and "and" to search for—that should evoke some hits, and nothing will break.

Greenstone digital library systems often comprise several separate collections—for example, computer science technical reports, literary works, internet FAQs, magazines. There will be a home page for the digital library system which allows you to access any collection; in addition, each collection has its own "about" page that gives you information about how the collection is organized and the principles governing what is included in it. To get back to the "about" page at any time, just click on the "collection" icon that appears at the top left side of all searching and browsing pages.

Figure 1 shows a screenshot of the "Demo" collection supplied with the Greenstone software, which is a very small subset of the Humanity Development Library collection; we will use it as an example to describe the different ways of finding information. (If you can't find the Demo collection, use the Humanity Development Library instead; it looks just the same.) First, almost all icons are clickable. Several icons appear at the top of almost every page; Table 1 shows you what they mean.

Table 1 What the icons at the top of each page mean

demo	This takes you to the "about" page
HOME	This takes you to the Digital Library's home page, from which you can select another collection
HELP	This provides help text similar to what you are reading now
PREFERENCES	This allows you to set some user interface and searching options that will then be used henceforth

The "*search … subjects … titles a-z … organization … how to*" bar underneath gives access to the searching and browsing facilities. The leftmost button is for searching, and the ones to the right of it—four, in this collection—evoke different browsing facilities. These may differ from one collection to another.

How to find information

Table 2 shows the five ways to find information in the Demo collection.

You can *search for particular words* that appear in the text from the "search" page. (This is just like the "about" page that comes up when you begin, except that it doesn't contain the *about this collection* text.) It can be reached from other pages by pressing the *search* button. You can *access publications by subject* by pressing the *subjects* button. This brings up a list of subjects, represented by bookshelves. You can *access publications by title* by pressing the *titles a-z* button. This brings up a list of books in alphabetic order. You can *access publications by organization* by pressing the *organization* button. This brings up a list of organizations. You can *access publications by how to listing* by pressing the *how to* button. This brings up a list of "how to" hints.

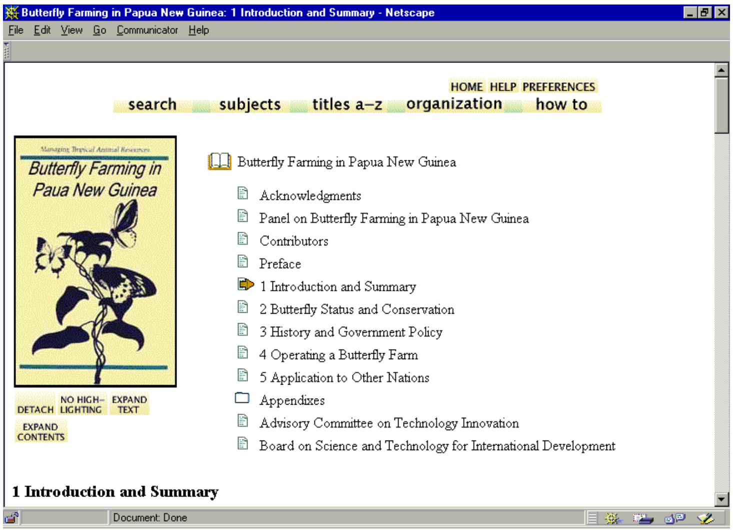


Figure 2 A book in the Demo collection

Table 2 What the icons on the search/browse bar mean

search	Search for particular words
subjects	Access publications by subject
titles a–z	Access publications by title
organization	Access publications by organization
how to	Access publications by "how to" listing

How to read the books

In the Demo collection, you can tell when you have arrived at an individual book because there is a photograph of its front cover (Figure 2). Beside the photograph is a table of contents with an arrow marking where you are. This table is expandable: click on the folders to open them or close them. Click on the open book at the top to close it.

Underneath is the text of the current section (*Introduction and Summary* in the example, beginning at the very bottom of the illustration). When you have read through it, there are arrows at the end to take you on to the next section or back to the previous one.

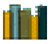
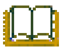
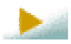
Below the photograph are four buttons. Click on *detach* to make a new browser window for this book. (This is useful if you want to compare books, or read two at once.) If you have reached this book through a search, the search terms will be highlighted: the *no highlighting* button turns this off. Click on *expand text* to expand out the whole text of the current section, or book. Click on *expand contents* to expand out the whole table of contents so that you can see the titles of all chapters and subsections.

In some collections, the documents do not have this kind of hierarchical structure. In this case, no table of contents is displayed when you get to an individual document—just the document text. In some cases, the document is split into pages, and you can read sequentially or jump about from one page to another.

What the icons mean

When you are browsing around the collection, you will encounter the items shown in Table 3.

Table 3 Icons that you will encounter when browsing

	Click on a book icon to read the corresponding book
	Click on a bookshelf icon to look at books on that subject
	View this section of the text
	Open this folder and view contents
	Click on this icon to close the book
	Click on this icon to close the folder
	Click on the arrow to go on to the next section ...
	... or back to the previous section
DETACH	Open this page in a new window
EXPAND CONTENTS	Expand table of contents
EXPAND TEXT	Display all text
HIGH-LIGHTING	Highlight search terms

How to search for particular words

From the search page, follow these simple steps to make a query:

- Specify what items you want to search: in the Demo collection you can search books, chapters, or section titles.
- Say whether you want to search for all or just some of the words
- Type in the words you want to search for into the query box
- Click the *Begin Search* button

When you make a query, the titles of up to twenty matching documents will be shown. There is a button at the end to take you on to the next twenty. From there you will find buttons to take you on to the third twenty or back to the first twenty, and so on. However, for efficiency reasons a maximum of fifty is imposed on the number of documents returned. You can change these numbers by clicking the *preferences* button at the top of the page.

Click the title of any document, or the little icon beside it, to open it. The icon may show a book, or a folder, or a page: it will be a book icon if you are searching books; otherwise if you are searching sections it will be a folder or page icon depending on whether or not the section found has subsections.

SEARCH TERMS

The stuff that you type into the query box is interpreted as a list of words called "search terms." Each search term contains nothing but alphabetic characters and digits. Terms are separated by white space. If any other characters such as punctuation appear, they serve to separate terms just as though they were spaces. And then they are ignored. You can't search for words that include punctuation.

For example, the query

```
Agro-forestry in the Pacific Islands: Systems for
    Sustainability (1993)
```

will be treated the same as

```
Agro forestry in the Pacific Islands Systems for Sustainability
    1993
```

QUERY TYPE

There are two different kinds of query.

- Queries for *all* the words. These look for documents (or chapters, or titles) that contain all the words you have specified. Books that satisfy the query are displayed, in alphabetical order.
- Queries for *some* of the words. Just list some terms that are likely to appear in the documents you are looking for. Books are displayed in order of how closely they match the query. When determining the degree of match,
    - the more search terms a document contains, the closer it matches;
    - rare terms are more important than common ones;
    - short documents match better than long ones.

Use as many search terms as you like—a whole sentence, or even a whole paragraph. If you specify only one search term, it doesn't much matter whether you use an *all* or a *some* query, except that in the second case documents will be ordered by the search term's frequency of occurrence.

Advanced search features

While the above is enough to meet most searching needs, some more advanced search features are provided. These are activated from the *preferences* page, which is reached by clicking the *preferences* button at the top of the page. When you change your *preferences*, do not click your browser's *Back* button—that would undo the changes. Instead, click any of the buttons on the search/browse bar.

CASE SENSITIVITY AND STEMMING

When you specify search terms, you can choose whether upper and lower case must match between the query and the document: this is called "case sensitivity." You can also choose whether to ignore word endings or not: this is called "stemming."

Under *Search options* on the *Preferences* page you will see a pair of buttons labeled *ignore case differences* and *upper/lower case must match*; these control the case sensitivity of your queries. Below is a pair of buttons labeled *ignore word endings* and *whole word must match*: these control stemming.

For example, if the buttons *ignore case differences* and *ignore word endings* are selected, the query

```
African building
```

will be treated the same as

```
africa builds
```

because the uppercase letter in "African" will be transformed to lowercase, and the suffixes "n" and "ing" will be removed from "African" and "building" respectively (also, "s" would be removed from "builds").

Generally case differences and word endings should be ignored unless you are querying for particular names or acronyms.

PHRASE SEARCHING

If your query includes a phrase in quotation marks (" and "), only documents containing that phrase, exactly as typed, will be returned.

If you want to use phrase searching, you need to learn a little about how it works. Phrases are processed by a post-retrieval scan. First the query is issued in the normal way—all the words in the phrase are included as search terms—and then the documents returned are scanned to eliminate those in which that phrase does not appear.

During the post-retrieval scan, phrases are checked just as they are, including any punctuation. For example, the query

```
what's a "post-retrieval scan?"
```

will first retrieve all documents that match all of the words

```
what s a post retrieval scan
```

and then the documents returned will be checked for the phrase

```
post-retrieval scan?
```

There is a restriction to the phrase searching mechanism: you cannot include more than one phrase in a query.

Finally, for computational reasons, the smallest unit of granularity is automatically enforced whenever phrase searching is used. In most collections, this is the paragraph level.

ADVANCED QUERY MODE

In *advanced query mode*, which can be selected on the *Preferences* page, the queries for *all* of the words, described above, are actually Boolean queries. They consist of a list of terms joined by logical operators & (and), | (or), and ! (not). Absent operators are interpreted as & (and): thus a query without any operators returns documents that match *all* the terms.

If the words AND, OR, and NOT appear in your query they are treated as ordinary search terms, not operators. For operators you must use &, |, and !. In addition, parentheses can be used for grouping.

Setting your preferences

The preferences page, which is reached by clicking the *preferences* button at the top of each page, controls several different things. When you change your preferences, do not click your browser's *Back* button—that would undo the changes. Instead, click one of the buttons on the search/browse bar.

Search preferences allow you to control casefolding and stemming as described above, determine how many hits should be returned and how many should be displayed on each page, and switch between simple and advanced query mode (in the latter, Boolean operators are allowed in *all* queries).

Language preferences allow you to specify the interface language, and the coding scheme (which is particularly important for the Chinese language).

The interface format can be graphical, which is normal, or textual, which is useful if you are visually impaired and use a large-font interface or a speech synthesizer.

C. Building collections

We now look at the process of building collections. This is for librarians!—but Greenstone empowers anyone to become a digital librarian. The next section describes the computer files that are used in a collection, including the original source files that contain the material in the collection, and the file structure that is used internally. Each document can be hierarchically structured, with sections and subsections.

Section C.2 describes how to update an existing collection by adding new material and rebuilding it. This is easy to do: the software is designed for ease of maintenance of digital library collections.

Then we look at the more challenging process of building a new collection from scratch. Building simple collections is easy—to build a collection with exactly the same structure as an existing one, with source documents in exactly the same format, is extremely easy. The Greenstone software is very flexible, allowing different kinds of source document and different sorts of browsing indexes, and this flexibility comes at a price: if you want to take advantage of it, you must learn to control it. Section C.3 describes all the issues that are involved in building new collections.

Finally, we look at how to put a collection onto a standalone Greenstone CD-ROM that can then be distributed for others to use, and how to translate the interface into other languages that are not yet supported.

C.1 The files in a Greenstone collection

When you create a new collection or add material to an existing one, you need to put the original source documents in a place from which they can be brought into the system, a process known as "importing" the documents into Greenstone. When they are imported, documents are converted into a simple HTML-like format known as GML for Greenstone Markup Language, which includes any "metadata," or descriptive data such as author, title, date, keywords, and so on, that is associated with the document. Documents are assumed to be in Unicode, expressed in the UTF-8 coding scheme (of which the ASCII characters form a subset).

Files and directories

The Greenstone system is placed in a directory structure rooted at *$GSDLHOME* (on the New Zealand Digital Library project's computers, it's rooted in */home/nzdl/gsdl*). In order to use the software, you should set this environment variable accordingly.

There are some other initialization functions that need to be done in order to make best use of the software; for example, augmenting your search path so that Greenstone utility programs are found automatically. These functions can be accomplished automatically by a setup script in *$GSDLHOME/setup.sh* (or *setup.csh* if you are using the C-shell; for other shells consult your local Unix expert). On Windows, the directory structure is rooted at *%GSDLHOME%*, and the setup script is in the file *%GSDLHOME%\setup.bat*. Table 4 gives Unix and Windows experts more information on what is required.

We will generally omit the *$GSDLHOME* prefix from filenames in this documentation.

Table 4 Contents of the file *setup.sh*

$GSDLHOME/setup.sh

```
export GSDLHOME=/home/nzdl/gsdl                    set by autoconf
export GSDLOS=`uname -s | tr A-Z a-z`
export PATH=$PATH:$GSDLHOME/bin/script:$GSDLHOME/bin/$GSDLOS
```

%GSDLHOME%\setup.bat

```
set GSDLHOME=c:\gsdl
set GSDLOS=windows
set PATH=%PATH%;%GSDLHOME%\bin\script;%GSDLHOME%\bin\windows;
```

The directory *collect* (i.e. *$GSDLHOME/collect*) contains a directory for each collection; to see what collections there are, both public and private, you should look here. Collection names usually bear some resemblance to what the collection is called on the New Zealand Digital Library Web page, although that page, of course, only shows public collections. For each collection, there will be several subdirectories, shown in Table 5a. The contents of these directories for the Demo collection are show in Table 5b, for reference. The *perllib* directory contains PERL programs that are specific to this collection.

Table 5 (a) Subdirectories for an individual collection ; (b) Subdirectories for the Demo collection

(a)	*archives*	Source files from which the collection is built, usually in GML format
	building	Stores the collection’s indexes while the collection is being built
	etc	Collection-specific configuration file *collect.cfg*, and any necessary data files
	import	Original raw material for the collection, in its original format
	index	Stores the indexes for the live collection, as it is served to users

(b) *collect/ demo*

```
archives:
    HASH0105.dir  HASH017d.dir  HASH63e6.dir  HASHaad6.dir
    HASH0144.dir  HASH026b.dir  HASH7df3.dir  HASHe52a.dir
    HASH0173.dir  HASH54cf.dir  HASHa0a5.dir  archives.inf

building:

etc:
    collect.cfg   mags.txt      sub.txt       org.txt

import:
    bostid        ecourier      faobetf       index.txt   wb

index:
    build.cfg     dtx           stt           stx         text

perllib:
    classify
```

Creating a collection

The process of creating a collection goes like this. The original raw material is placed in the *import* directory, in whatever form it is available—plain text files, Word documents, HTML files, depending on where it came from. Then the *import* process is invoked. Controlled by the configuration file in *etc/collect.cfg*, the files in *import* are converted into GML format and placed in the *archives* directory. The original raw material remains in *import*. Then the *build* process is invoked. Again controlled by the configuration file, the requisite indexes for the collection are built and placed in the *building* directory. Finally, the contents of the *building* directory are moved into the *index* directory, and the new version of the collection automatically becomes live.

This procedure may seem cumbersome. But all the steps are necessary for efficient operation with large collections. First, the *import* process could be performed on the fly during the building operation—but because building indexes is a multipass operation, the often lengthy importing would be repeated several times. Second, the *build* process can take considerable time—some days, for very large collections. It puts the result into the separate *building* directory so that, if the collection already exists, it can continue to be served to users in its old form from the directory *index* throughout the building operation.

If desired, the separate step of importing can be avoided by putting the original raw material in the *archives* directory. The building process uses filename extensions to recognize the type of files being dealt with. On finding files other than *.gml* files in the archives directory, it will invoke the import procedure automatically for each pass of index creation. This saves the extra space required to store the collection twice, in both original and imported form, at the expense of the additional time required to perform the import conversion for each pass of the building process.

The configuration file that controls the collection creation procedure resides in the collection's *etc* directory, along with any collection-specific data files that are required. (For example, building a hierarchy of classifications requires a data file of sub-classifications, which is placed here.)

The imported documents

In order to identify documents in the Greenstone system, a unique object identifier or “OID” is assignment to each original source document and stored as metadata within that document. It is important that no two documents have the same OID, and that a document’s OID persists throughout the index-building process—so that a user’s search history is unaffected by rebuilding the collection. OIDs are assigned by hashing the contents of the original source document during the *import* process; an example OID is “HASHa723e7e164df07c833bfc4”.

Once it has been imported, each document is stored in its own subdirectory of the *archives* directory, along with any other files—for example, image files—that are associated with the document. The name of the subdirectory is generated from the document’s OID, and the document is stored in GML format in a file called *doc.gml*.

To ensure compatibility with Windows 3.X systems, only eight characters are used in directory and file names. The much longer OIDs are mapped into multiple directory names, but the directory structure is kept as shallow as possible by using only as much of the OID as required. For example, just the first eight characters of the documents OID are used, except that if a directory with this name already exists, a subdirectory of it is created using the next eight, and so on (in fact, this seldom happens).

Inside the documents

Within a single document, the GML format imposes a limited amount of structure. (We will give examples of the GML format shortly, in Table 8. Documents are divided into paragraphs. Moreover, they can be split hierarchically into sections and subsections; these may be nested to any depth. OIDs are extended to identify sections and subsections by appending section and subsection numbers, separated by periods, to a document’s OID. For example, subsection 3 of section 2 of document HASHa7 is referred to as HASHa7.2.3. When you read a book in a Greenstone collection, the section hierarchy is manifested in the table of contents of the book. For example, books in the Demo collection have a hierarchical table of contents showing chapters, sections, and subsections. Documents in the Computer Science Technical Reports do not have a hierarchical subsection structure, but each document is split into pages and you can browse around the pages of a retrieved document. Chapters, sections, subsections, and pages are all implemented simply as “sections” within the document.

The document structure is also used for searchable indexes. There are three levels of index: *documents*, *sections*, and *paragraphs*. A document index contains the full document. When a section index is created, each section stretches from a *gsdlsection* tag to the next-occurring *gsdlsection* tag—thus a chapter that immediately begins with a new section will produce an empty document in the index. Sections and subsections are treated alike: the hierarchical document structure is flattened for the purposes of creating searchable indexes. As well as indexes of text, indexes of any kind of metadata can also be created. For example, most collections offer searchable indexes of section titles.

It has already been mentioned that metadata is stored with documents. The Dublin Core metadata standard is used, which defines the metadata types in Table 6 (starred ones are used in current collections). However, metadata types that are not in the Dublin Core may be used too (for example, the Demo collection contains “how to” and “Magazine” metadata).

Table 6 The Dublin Core metadata standard. Starred items are used in current collections

Name	**Metadata subtag**	**Definition**
*Title	`Title`	A name given to the resource
*Creator	`Creator`	An entity primarily responsible for making the content of the resource
*Subject and keywords	`Subject`	The topic of the content of the resource
*Description	`Description`	An account of the content of the resource
*Publisher	`Publisher`	An entity responsible for making the resource available
Contributor	`Contributor`	An entity responsible for making contributions to the content of the resource
*Date	`Date`	A date associated with an event in the life cycle of the resource
Resource type	`Type`	The nature or genre of the content of the resource
Format	`Format`	The physical or digital manifestation of the resource
*Resource identifier	`Identifier`	An unambiguous reference to the resource within a given context: this is the object identifier or OID
Source	`Source`	A Reference to a resource from which the present resource is derived
*Language	`Language`	A language of the intellectual content of the resource
Relation	`Relation`	A reference to a related resource
Coverage	`Coverage`	The extent or scope of the content of the resource
Rights management	`Rights`	Information about rights held in and over the resource

The GML format

All text documents are converted to a common format, GML, when they are imported into the Greenstone system. The HTML convention of tags enclosed in angle brackets is adopted for markup; any <, >, or “ characters within the text are stored as their HTML equivalent *(< >* and *")*.

A *<gsdlsection>* tag denotes the start of each document section, and the corresponding *</gsdlsection>* closing tag marks the end of that section. Each *<gsdlsection>* tag can contain any number of subtags shown in Table 7. Metadata is indicated within the *gsdlsection* tag (for example, *<gsdlsection Title="Food and Nutrition Bulletin">*); thus different metadata can be associated with individual sections of a document. Any subtags other than those in Table 7 are considered to be metadata that is attached to that section.

Table 7 Subtags of <*gsdlsection*>

gsdlsourcefilename	Original file from which the GML was generated
gsdldoctype	Type of document (currently the only recognized type is indexed_doc)
gsdlassocfile	File associated with the document (e.g. an image file)
gsdlnum	Subsection number (this does not exist at the document's top level)

Table 8a is a GML file that contains a simple document comprising a single section with title, and three associated images. Table 8b shows a more complex document: a book with two sections called *Preface* and *Conclusions*, the second of which has two subsections. Note that a chapter is simply treated as a top-level section. In some collections documents are split into individual pages. These are treated as sections, though they don't usually have titles. Table 8c shows a book with two sections (corresponding to chapters), the first of which has two pages and the second one page.

Table 8 Three examples of documents in GML format

(a)

```
<gsdlsection
  gsdlsourcefilename = "uu02fe.txt"
  gsdldoctype = "indexed_doc"
  Identifier="HASHa723e7e164df07c833bfc4"
  Title = "Freshwater Resources in Arid Lands"
  gsdlassocfile="cover.jpg:image/jpeg"
  gsdlassocfile="p21.jpg:image/jpeg"
  gsdlassocfile="p22.jpg:image/jpeg">
   This is the text of the document
</gsdlsection>
```

(b)

```
<gsdlsection
  gsdlsourcefilename = "uu02fe.txt"
  gsdldoctype = "indexed_doc"
  Identifier="HASHa723e7e164df07c833bfc4"
  Title = "Freshwater Resources in Arid Lands">
  <gsdlsection gsdlnum="1" Title="Preface">
    This is the text of the preface
  </gsdlsection>
  <gsdlsection gsdlnum="2" Title="Conclusions">
    <gsdlsection gsdlnum="1" Title="Part 1">
      This is the first part of the conclusions
      </gsdlsection>
    <gsdlsection gsdlnum="2" Title="Part 2">
      This is the second part of the conclusions
      </gsdlsection>
</gsdlsection>
```

(c)

```
<gsdlsection
  gsdlsourcefilename = "uu02fe.txt"
  gsdldoctype = "indexed_doc"
  Identifier="HASHa723e7e164df07c833bfc4"
  Title = "Freshwater Resources in Arid Lands">
  <gsdlsection gsdlnum="1" Title="Chapter 1">
    <gsdlsection gsdlnum="1">
      This is the text of the first page of chapter 1
    </gsdlsection>
    <gsdlsection gsdlnum="2">
```

```
      This is the text of the second page of chapter 1
    </gsdlsection>
  </gsdlsection>
  <gsdlsection gsdlnum="2" Title="Chapter 2">
    <gsdlsection gsdlnum="1">
      This is the text of the first and only page of chapter 2
    </gsdlsection>
  </gsdlsection>
</gsdlsection>
```

C.2 Updating existing collections

Updating an existing collection is easy. Since the collection already exists, its directory (a subdirectory of *collect*) will exist, with subdirectories *import*, *archives*, *building*, *index*, and *etc*. The *etc* directory will already contain a collection configuration file *collect.cfg*.

The collection is to be updated with new files. For example, in the Demo collection the raw material consists of marked up HTML files containing <<TOC>> tags to split books into sections and subsections, and <<I>> tags to indicate where an image is to be inserted. For each book in the library there is a directory that contains a single HTML file representing the book, and separate files containing the associated images. We assume that the new material is in exactly the same format as the existing files that have previously been used to create the collection.

What you have to do

You need to import the new material into the Greenstone system, and then rebuild the collection. The procedure is as follows. Put the new material into the *import* directory, and then import it by executing *import.pl*:

```
import.pl [options] collection-name
```

The program *import.pl* processes the files and directories in the *import* directory and converts the documents to GML format, placing the result in the *archives* directory. Any other files (e.g. image files) that are associated with a document are also copied to *archives*.

The options for *import.pl* are given in the table below. You can override the default directory names *import* and *archives*, and specify that just a few documents are to be converted—which is useful for testing purposes when dealing with large collections. By default, *import.pl* will not re-import old material in the *import* directory for which GML files have previously been created in *archives*. However, we can re-import the lot by specifying the *removeold* option.

The purpose of the *import* step is to put all documents into a standard format before building the collection. Otherwise each document would have to be converted several times, because building is a multi-pass operation. However, it is possible to simply omit this step. In this case the original source material is put straight into the *archives* directory, and the program *import.pl* is not executed at all. The building step processes all files in the *archives* directory. Noticing from the filename extensions that they need to be converted first, the building process silently invokes this step for each pass.

Once the new material has been imported, the collection should be rebuilt:

```
buildcol.pl [options] collection-name
```

The final step is to remove the old contents of the *index* directory, if any exist, and move the contents of the *building* directory into the *index* directory, from which the collection is served to users. On Unix systems,

```
rm -r index/*
mv building/* index
```

The *building* directory is merely used as a holding-place during the potentially lengthy building process. It is quite possible that there are active users of the collection at the time that *building* is moved into *index*. Since documents are referred to internally by their OID, which persists through the rebuilding process, users who are examining the results of a query or browse operation will still access the expected documents. If a query is actually in progress at the time the contents of *index* are changed, it may encounter an internal MG error which will cause it to be transparently re-executed, this time on the new version of the collection.

Some useful debugging features are incorporated into *import.pl* and *buildcol.pl*. You can perform a *build* operation but print (to *stdout*) the information that is generated instead of passing it to MG or *txt2gdbm*. Suppose, for example, you are writing a plugin to handle a particular collection, and the Greenstone interface shows that the text is coming out wrong. Running *buildcol.pl* with the *debug* switch and the appropriate *mode* option will show exactly what the plugin is sending to MG

In summary, once you have put the raw material into the *import* directory these three commands are all you need to rebuild a collection:

```
import.pl collection-name
buildcol.pl collection-name
rm -r index/*
mv building/* index
```

Table 9 Options for *import.pl*, *buildcol.pl* and *mkcol.pl*

	Option	**Meaning**
import.pl	-verbosity <number>	0=none, 3=too much
	-importdir <directory>	Where the original material comes from (default *import*)
	-archivedir <directory>	Where the converted material goes to (default *archives*)
	-cachedir <directory>	Not yet implemented
	-maxdocs <number>	Maximum number of documents to convert
	-keepold	Do not delete the current contents of the *archives* directory (the default)
	-removeold	Delete everything in the *archives* directory first
	-gzip	Compress the resulting GML files using *gzip*
	-debug	Print out the GML documents instead of saving them to GML files

buildcol.pl	-verbosity <number>	0=none, 3=too much
	-archivedir <directory>	Where the building material comes from (default *archives*)
	-builddir <directory>	Where the indexes that are built go to (default *building*)
	-cachedir <directory>	Cache the archives to this directory (used to specify a local disk when archives are on a remote disk)
	-maxdocs <number>	Maximum number of documents to include
	-allclassifications	Normally, information for empty classifications created by a classifier is suppressed. If this flag is set, it is included (this may create empty folders).
	-debug	Print the output that would normally be piped to MG (when compressing text or building an index) or *text2db* (when writing the GDBM file)
	-mode all -mode compress_text -mode build_index -mode infodb	Just compress the text, or just build the index(es), or just write the GDBM database (*all*, the default, is a full build)—for use in conjunction with -debug
	-keepold	Do not delete the current contents of the *building* directory before starting (defaults to *false*)—for use in conjunction with -mode to continue after a failure of the build process
	-index	If not set, all indexes in *collect.cfg* are built; this switch allows selective indexing for debugging (e.g. -index document:Title)

mkcol.pl		
	`-creator`	Email address of the collection's creator
	`-maintainer`	Email address of the collection's maintainer
	`-public`	Whether collection is to be made public or not
	`-beta`	Whether collection is beta version or not
	`-index`	List of indexes to include
	`-defaultindex`	The default index
	`-title`	Title of the collection
	`-about`	"About this collection" text
	`-plugins`	List of plugins used by the collection

Example

Consider as an example the Demo collection. There are two ways to build the collection. The first is to install the raw material in the directory *collect/demo/import*, run *import.pl* to convert to GML and copy to the *archives* directory, then run *buildcol.pl* on the resulting archives. The second is to install the raw material in the directory *collect/demo/archives* and run *buildcol.pl* directly on this raw material.

The first method is much faster than the second, because the conversion process is done once only, instead of being done for each pass of the build operation (there are two passes to compress the text, two for each index, and a further pass to construct the GDBM file). However, the second method uses less disk space, because no GML files are stored, and retains the original directory structure.

Once the collection has been built, it will be discovered automatically by the receptionist, which polls the *collect* directory regularly to update its list of collections.

How it works

The original material in the *import* directory may be in any format, and a "plugin" is required to process each format type (described below). The plugins that a collection uses must be specified in the collection configuration file. The *import* program reads the list of plugins and passes each document to each plugin in order until it finds one that can process it. Since we are working with an existing collection, we have assumed that all plugins necessary to process new material have already been specified in the configuration file.

The *buildcol.pl* step actually creates the indexes for both searching and browsing. The MG software is generally used to do the searching. The *buildcol.pl* process invokes the MG module *mgbuild* to create each of the indexes that is required. For example, the Humanities Development Library has three indexes, one for entire books, one for chapters, and one for section titles. (Subdirectories of the *index* directory are created for each of these indexes.)

MG also compresses the text of the collection (the compressed version is placed in a subdirectory *index/text*). In addition, the image files are linked into the *index* subdirectory. Now none of the material in the *import* and *archives* directories is needed to run the collection (though they would be needed to rebuild it).

Associated with each collection is a database stored in GDBM (Gnu database manager) format. This contains an entry for every section of every document document, giving its OID, its internal document number used by MG (which, unlike the OID, may change from one build to the next), and metadata such as its title. Information for each of the browsing indexes, which appear as buttons on the Greenstone search/browse bar, is also extracted during the building process and stored in the GDBM database. A "classifier" program (see below) is required for each browsing index to extract the appropriate information from GML documents. Like plugins, classifiers are written on an *ad hoc* basis for the particular information required, and where possible are reused from one collection to another.

The program *buildcol.pl* builds the indexes based on whatever appears in the *archives* directory. The first plugin specified by all collections is one that processes GML files (called *GMLPlug*), and so if *archives* contains files created by *import.pl* they will be processed correctly. If the *archives* directory contains material in the original format, it will be converted using the appropriate plugin for that format. This is why the *import* process is an optional one (although usually it is highly desirable because it speeds up the operation substantially).

C.3 Creating new collections

The first step in creating a new collection is to run the program *mkcol.pl*.

```
mkcol.pl [options] collection-name
```

This sets up the directory structure and generates a skeleton collection configuration file. The options are shown below. Only the creator is required, all others have defaults. For example, you might type

```
mkcol.pl –creator … collection-name
```

The original source material should be placed in the *import* directory. The next step is to run *import.pl* to import the documents into the *archives* directory, and then *buildcol.pl* to create the indexes and compressed text in the *building* directory as described above. Both these operations are controlled by the collection's configuration file, *etc/collect.cfg*. You will need to edit this file to specify appropriate plugins to decipher the imported file format and convert it to the GML language; without these, the collection that is built may be empty because files for which there is no appropriate plugin are simply ignored (and a warning message is printed to the error channel). You will need to edit the configuration file to specify which full-text search indexes are to be built: the default is a single index for the entire text of the collection. You will need to specify appropriate classifiers to create the information necessary for browsing interfaces. In the absence of these specifications, the configuration file created by *mkcol.pl* will make a bland collection with just one searchable index and no browsing indexes—and it may be empty because no documents can be imported.

Thus the entire sequence needed to create a new collection is

`mkcol.pl –creator … collection-name`
place the source material in the import directory

edit the configuration file etc/collect.cfg appropriately

```
import.pl collection-name
buildcol.pl collection-name
mv building/* index
```

The collection configuration file

All you have to do now is to come up with an appropriate configuration file and re-run the import and build processes. Look at the one you have in *etc/collect.cfg*: you will find that it looks something like Table 10a. Lines are numbered for reference: they can appear in any order, each line being an independent command chosen from the list in Table 10c followed by a number of fields separated by white space.

Table 10 (a) Plain collection configuration file, as created by *mkcol.pl*; (b) Collection configuration file for the Demo collection; (c) Summary of configuration file commands

(a)	creator	sjboddie@cs.waikato.ac.nz	a1
	maintainer	sjboddie@cs.waikato.ac.nz	a2
	public	true	a3
	beta	true	a4
	indexes	document:text	a5
	defaultindex	document:text	a6
	plugins	GMLPlug TEXTPlug ArcPlug RecPlug	a7
	classify	AZList metadata=Title	a8
	collectionmeta	collectionname "my_new_collection"	a9
	collectionmeta	.document:text "documents"	a10

(b)

```
creator          sjboddie@cs.waikato.ac.nz                                          b1
maintainer       sjboddie@cs.waikato.ac.nz                                          b2
public           true                                                               b3
beta             true                                                               b4

indexes          section:text section:Title document:text                           b5
defaultindex     section:text                                                       b6

plugins          GMLPlug HBPlug ArcPlug IndexPlug RecPlug                           b7

classify         Hierarchy hfile=sub.txt metadata=Subject sort=Title                b8
classify         HDLList Title                                                      b9
classify         Hierarchy hfile=org.txt metadata=Organization sort=Title           b10
classify         List metadata=Howto                                                b11

format           QueryResults "<td valign=top>[link][icon][/link]</td>              b12
                  <td>{If}{[parent(All': '):Title],[parent(All': '):Title]: }
                  [link][Title][/link]</td>"
format           CL4Vlist "<br>[link][Howto][/link]"                                b13
format           DocumentImages true                                                b14
format           DocumentText "<h3>[Title]</h3>\\n\\n<p>[Text]"                     b15

collectionmeta   collectionname "greenstone demo"                                   b16
collectionmeta   collectionextra "This is a demonstration collection for the        b17
                  Greenstone digital library software.\nIt contains a small
                  subset (11 books) of the Humanity Development Library"
collectionmeta   iconcollection                                                     b18
                  "http://www.nzdl.org/gsdl/collect/demo/images/demo.gif"
collectionmeta   .section:Title "section titles"                                    B19
collectionmeta   .document:text "entire books"                                      b20
collectionmeta   .section:text "chapters"                                           b21
```

(c)

`creator`	Email address of the collection's creator
`maintainer`	Email address of the collection's maintainer
`public`	Whether collection is to be made public or not
`beta`	Whether collection is beta version or not
`indexes`	List of indexes to build
`defaultindex`	The default index
`subcollection`	Define a subcollection based on metadata
`indexsubcollections`	Specify which subcollections to index
`defaultsubcollection`	The default indexsubcollection
`languages`	List of languages to build indexes in
`defaultlanguage`	Default index language
`status`	Status page enabled/disabled
`collectionmeta`	Defines collection-level metadata
`plugins`	List of plugins to use at build time
`format`	A format string (explained below)
`classify`	Specifies a classifier to use at build time

The collection configuration file for the Demo collection is shown in Table 10b. For each collection, the bland configuration file that *mkcol* has produced automatically must be edited into something that looks like this. The most formidable part, the *format* and *collectionmeta* statements, are cosmetic: the basic collection structure is created by the *indexes* statement, which determines what searchable indexes there are, the *plugin* statement, which determines how the raw input files are read and parsed, and the four *classify* statements, each of which produces one of the four browsing access buttons.

The first block, lines a1–a4 and b1–b4, specifies the creator and maintainer of the collection, whether it is supposed to be made publicly available or not, and whether it is a beta version or not.

The next line specifies the full-text searching indexes that are to be built. Line a5 specifies that there is to be just one index. Indexes can be constructed at the *document*, *section*, and *paragraph* levels. They can contain material from *text* or any metadata—most commonly *Title*. These two elements are separated by a colon in the *indexes* line. Line b5 calls for three indexes to be created: the first for sections, the second for section titles, and the third for complete documents. In addition, text and multiple metadata types can be included in a single index using commas as separators, e.g. *sectiontext,Title,Creator*.

Collection-level metadata such as the long form of the collection's name, its icon, the names of indexes, and the "about" text can be defined using collectionmeta entries in the configuration file. Lines a9–a10 specify very rudimentary metadata; this has been expanded to lines b17–b22 by adding entries that are largely self-explanatory.

The plugins (line b7), classifiers (lines b8–b11), and format strings (lines b12–b16) are described in separate sections below. Plugins and classifiers make use of several auxiliary files that will be described in due course: they are collected together in Table 11 for easy reference. Briefly, the *index.txt* file is used by the *IndexPlug* plugin to assign metadata to the documents; *mags.txt* is a file that is special to this collection that determines which of the books in it are actually magazines, for special treatment in the *titles* browsing list; *sub.txt* specifies the subject hierarchy that appears when you press the *subject* browsing button; and *org.txt* defines the organization hierarchy that appears when you press the *organization* browsing button. These are subsets of the corresponding files for the Humanities Development Library, which accounts for the fact that the numberings do not start from 1. You can see *index.txt* in the *import* directory in Table 5b; the other three files reside in the *etc* directory. (For maximum software reuse, the same format used for configuration files is used for the files that are read in by plugins and classifiers.)

Finally we will describe the subcollection and language-specific index facilities; these are not used in the Demo collection.

Table 11 Auxiliary files used in building the Demo collection

(a) *index.txt* (used by *IndexPlug*)

```
key:            Subject Organization Howto Magazine
bostid/b22bue   16.11  bostid  "start a butterfly farm"
faobetf/fb33fe  14.12  faobfs  <Subject>16.11
faobetf/fb34fe  14.12  faobfs  "farm snails"  <Subject>16.11
bostid/b18ase   16.11  bostid  "introduce little-known Asian …"
bostid/b20cre   16.11  bostid
bostid/b17mie   16.11  bostid  "introduce small animals and …"
bostid/b21wae   16.5   bostid  "utilize the …"  <Subject>16.11
ecourier/ec158e 23.15 ecc <Subject>8.1  "<Magazine>The Courier"
ecourier/ec159e 23.15 ecc <Subject>6.1  "<Magazine>The Courier"
ecourier/ec160e 23.15 ecc <Subject>21.1 "<Magazine>The Courier"
wb/wb34te        6.4   wb      "achieve gender equality"
```

(b) *mags.txt* (used by *HDLList*)

```
"The Courier"                    3  "The Courier"
```

(c) *sub.txt* (used by *Hierarchy*)

```
6      6     "Society, Culture, Community, Woman, Youth, …"
6.1    6.1   "Social sciences, sociology (works comprising …"
6.4    6.4   "Women, gender and development, women's …"
8      8     "Communication, Information and Documentation"
8.1    8.1   "Communication, telecommunication, mass …"
14     14    "Agriculture and Food Processing"
14.12  14.12 "Better Farming series of FAO and INADES"
16     16    "Animal Husbandry and Animal Product Processing"
16.5   16.5  "Cattle"
16.11  16.11 "Other animals (micro-livestock, little known …)"
21     21    "Settlements, Housing, Building - …"
21.1   21.1  "Settlements and housing: general works incl. …"
23     23    "Development Periodicals and Magazines"
23.15  23.15 "The Courier ACP 1990 - 1996 Africa-Caribbean-…"
```

(d) *org.txt* (used by *Hierarchy*)

```
bostid  4    BOSTID
1echo   10   E.C.H.O
ecc     11   "EC Courier"
faobfs  15   "FAO Better Farming series"
```

Plugins

Plugins are used by the collection-building software to accomplish all the format-specific parsing of each document. The configuration file gives a list of all the plugins to be used in building that collection. For example, the Demo collection (line b7) specifies five plugins:

```
plugins GMLPlug HBPlug ArcPlug IndexPlug RecPlug
```

During the import operation, each file or directory is passed to each plugin in turn until one is found that can process it—thus earlier plugins take priority over later ones. If no plugin is found that can process the file, a warning is printed (to the error channel) and processing passes to the next file. The same procedure is used during building, this time with the *archives* directory.

Recursion is necessary to traverse directory hierarchies. Although the importing (and building) programs do not perform recursion explicitly, some plugins cause indirect recursion. A plugin may itself pass files or directory names into the plugin pipeline, allowing a directory hierarchy to be traversed. The standard way of doing this is with *RecPlug*, which recurses through a directory structure. If this is present it should be the last element in the pipeline.

General plugins are kept in *perllib/plugins*, while collection-specific ones are in *collect/collection-name/perllib/plugins*. In case of a name conflict, collection-specific plugins override general ones. The plugins that are currently available are listed in Table 12. Only the first three are recursive.

Table 12 Currently-implemented plugins

	Plugin	Purpose
General	`ArcPlug`	Processes files named in index file *archives.inf*
	`RecPlug`	Recurses through a directory structure
	`IndexPlug`	Assigns metadata from *index.txt* file
	`GMLPlug`	Processes GML files generated by *import.pl*
	`TEXTPlug`	Processes plain text.
	`HTMLPlug`	Processes html, taking care to replace hyperlinks
	`EMAILPlug`	Processes email messages, recognizing author, subject, date, etc.
	`HBPlug`	Processes HTML marked up for UN collections
	`FOXPlug`	Processes Foxbase dbt files
Specific	`PrePlug`	Processes HTML output by PRESCRIPT, splitting documents into pages
	`GBPlug`	Processes Project Gutenberg etext
	`TCCPlug`	Processes email documents from Computists' Communique

We implied earlier that communication between the *import* and *building* processes is achieved simply by the former placing files in the *archives* directory, and the latter picking them up. In fact, to make the operation more efficient, and to allow the files to be sorted in different ways before building commences, *import* writes a file called *archives.inf* into the top-level directory of the *archives* file structure containing the names of the GML files it has imported. (If *archives.inf* already exists, *import* simply adds to it.) This also allows non-GML files (such as image files) to be ignored, rather than being processed by the plugin pipeline and causing an error message. The function of *ArcPlug* is to check whether it is being passed an *archives.inf*, and, if so, to read the list of files it contains and pass each one in turn into the plugin pipeline. *ArcPlug* should always be included unless *import.pl* will never be used.

As mentioned above, *RecPlug* checks to see if the filename it receives is a directory. If so, each file in that directory is passed into the plugin pipeline. If any of these is itself a directory, it will be picked up by *RecPlug* in the next pass. *RecPlug* is used merely to traverse the directory structure when importing or building directly from original material.

IndexPlug is used to assign metadata to documents from a separate metadata file called *index.txt*, which is generally created manually. This plugin checks the filename it receives to see if it is *index.txt*, and if so, reads the list of files, assigns metadata to them as specified, and passes each one in turn into the plugin pipeline. The *index.txt* file associated with the Humanities Development Library (from which the Demo collection is excerpted) contains extra metadata to be associated with each book (Subject classification, Organization, "how to" classification, and Magazine title), and was generated manually from Excel files supplied with the collection. Creating this file was the most time-consuming part of making the collection.

The content of this metadata file is shown in Table 11a. The first line is a key to what metadata fields are included, in this case *Subject*, *Organization*, *Howto*, and *Magazine* metadata. Subsequent lines give the filename of a document followed by its metadata values. For example, the second line of Table 11a assigns to the document that originated in file *bostid/b22bue* the Subject *16.11*, the Organization *bostid*, the "how to" *start a butterfly farm*, and nothing for *Magazine*. A mechanism is provided to allow for multiple field values and missing fields. The next line not only assigns *14.12* to Subject and *faobfs* to Organization by placing field names in angle brackets, but also assigns *16.11* to Subject as well; it makes no assignment to "how to". All metadata specifications may be repeated in a document file. While this may seem clunky, it combines a mechanism to utilize metadata provided in spreadsheet files with a way of allowing more flexible non-column-oriented specifications.

IndexPlug resembles *ArcPlug* in that only those files that appear in the list will be included. It cannot be used to add metadata to just a few files, for the other files will not be included in the collection. If importing and building take place in separate stages, *IndexPlug* will operate during the importing stage, and the files it produces will be picked up by *ArcPlug* during building.

GMLPlug processes documents in GML format. It should always be included unless *import.pl* will never be used.

TEXTPlug is a plugin for processing plain text. It does not alter the input but places it between *<pre>* and *</pre>* tags for display as preformatted text in an HTML page. It is being extended to allow the input to be GB-encoded, to allow Chinese files to be read.

HTMLPlug processes plain html. It replaces hyperlinks: if the target is a document within the collection it makes the appropriate linkage; if not it calls a CGI script that warns the user that they are leaving the collection. (Also being extended with GB-encoding).

EMAILPlug processes email messages in the standard format. It extracts metadata such as *to*, *from*, *subject*, and *date*. It preserves embedded hyperlinks, and turns email addresses into hyperlinks.

HBPlug processes marked-up HTML files supplied for United Nations collections. *HBPlug* is specially written for these collections. It reads the input file, splits it into sections at <<TOC>> tags and replaces <<I>> tags with corresponding HTML *<img src … >* tags. By default, the receptionist displays each document as a sequence of pages, with *next page* and *last page* arrows and a *go to page* widget; however, the documents in the United Nations collections are hierarchically structured.

FOXPlug processes a Foxbase DBT file. It provides the basic functionality to read DBT and DBF files and process each record. It should be overridden for a particular database to process the appropriate fields in the file.

The suite of collection-specific plugins is growing rapidly. *PrePlug*, used by the Computer Science Technical Reports collection, processes the simplified HTML output by PRESCRIPT. It splits documents into pages on *<!–End Of Page–>* or *<!–Page No–>* tags if present, otherwise at paragraph breaks every 50 lines or so. It also checks whether the document has a corresponding *.info* file, from which it obtains metadata. *GBPlug*, used by the Gutenberg collection, separates out introductory boilerplate and splits documents into pages of 80 or so lines. It also extracts title and author metadata from the text. *TCCPlug* processes the email documents received from the Computists' Communique. It splits documents into separate articles and extracts title and date metadata. The Niupepa collection is unusual in that each document contains multiple files: there is one file for each page. It uses an overridden version of *IndexPlug* which processes each document itself as well as reading in *index.txt*.

Classifiers

Classifiers are used to create a collection's browsing indexes, for example, the Demo collection's Subject, and Titles A-Z, organization, and "how to" indexes. This information is stored in the GDBM database, and the classifiers are called during the final phase of *buildcol.pl* when this database is being built.

Like plugins, classifiers are specified in a collection's configuration file; unlike plugins there is one line for each classifier. This line contains the keyword "classify" followed by the name of the classifier and any options it takes. For example, line a8 of Table 10a makes an alphabetic list of the contents of the *Title* metadata field: it takes all documents with this metadata, sorts them by title, and splits them into alphabetical subsections.

A similar command appears as line b9 of Table 10b, and you can see the result by clicking the *Titles A-Z* button of the Demo collection. The *HDLList* that it specifies is a classifier written especially for the Humanities Development Library, from which the Demo collection was derived. It is identical to the standard *AZList* classifier described below, except that it adds a hierarchy classification for the "Magazines" section, and obtains this information by reading the file *mags.txt* shown in Table 11b. In this case only one document (*The Courier*) is declared as a magazine; in the full Humanity Develoment Library there are several such documents. This classifier has no options: since it is only used by this collection, the functionality is hard-coded.

The other *classify* lines in the demonstration collection configuration file specify a hierarchical browsing index for the *Subject* metadata (line b8), a hierarchical browsing index for the *Organization* metadata (b10), and a plain list for the "how to" metadata (b11).

Table 13 Currently-implemented classifiers

Classifier	Argument	Meaning
`List`		A list of documents
	`metadata`	Include documents containing this metadata element
	`buttonname`	Name of the button used to access this classifier (defaults to value of metadata argument)
`SectionList`		A list of sections in documents
`AZList`		A list of documents split into alphabetical ranges
	`metadata`	Include all documents containing this metadata element
	`buttonname`	Name of the button used to access this classifier (defaults to value of metadata argument)
`AZSectionList`		Like AZList but includes every section of the document
`DateList`		Like AZList but sorted by date, as used by Niupepas 'Dates' section
`Hierarchy`		Hierarchical classification
	`hfile`	A classification file
	`metadata`	Metadata element to test against hfile identifier
	`buttonname`	Name of the button used to access this classifier (defaults to value of metadata argument)
	`sort`	Metadata element to sort documents by (optional)

Table 13 lists the classifiers that are available. Each one has a "buttonname" argument which allows you to specify how the classifier will be accessed from the search/browse bar in the interface. Buttons are provided for each Dublin Core metadata type, and for some other types of metadata. (Soon they will be generated on demand by Gimp, a scriptable image-manipulation program.)

The *List* classifier creates a simple single-level list of documents, like the Demo collection's "how to" list which shows a simple paged list containing all the books with a "how to" metadata field. If the *metadata* argument is specified, only documents containing this metadata element are included, and the list is sorted by this metadata. If it is omitted, all documents are included, in build order.

The *SectionList* classifier is like *list* but includes all sections of a document except the top level, rather than just the top level. This is used to create lists of sections (articles, chapters or whatever) as in the Computists' Communique collection, where each issue is a single document and comprises several independent news items, each in its own section.

The *AZList* classifier sorts documents alphabetically by metadata and splits the resulting list into alphabetical ranges. If the metadata is *Creator* it is treated as a name when sorting, otherwise it is treated as a title (so that leading "a" and "the" are ignored in the sort).

The *AZSectionList* is similar but an entry is given for each section (except top-level sections).

DateList is similar to *AZList* except that documents are sorted by their *Date* metadata. This is used, for example, in the Niupepa collection's "Dates" classification. The date is expected to be in the form *yyyymmdd* (this is being extended in future to handle other formats). At present the top-level list is by year (also being extended to work by month/week/day).

The *Hierarchy* classifier implements a hierarchical browser, as used by the Demo collection's *subject* and *organization* browsers. The file specified by *hfile* gives a textual description of the hierarchy. In the case of the Demo collection, line b8 of Table 10b specifies the file *sub.txt*, which is shown in Table 11c (and was generated from an Excel file supplied with the Humanities Development Library collection). There is one line for each section, comprising three fields:

- identifier, which matches the specified metadata value for each section to be included;
- position-in-hierarchy identifier, in multi-part numeric form—e.g. 1, 1.2, 2.6.1;
- title of section.

This example is a slightly confusing one because the number representing the hierarchy appears twice on each line. The metadata type "Hierarchy" stores its values in hierarchical numeric form, which accounts for the first occurrence. It is the second occurrence that is used to determine the hierarchy that the browser implements.

A second example is the organization hierarchy of the Demo collection, shown in Table 11d as file *org.txt*—which is smaller and simpler than *sub.txt* because this metadata has a simpler structure than the subject hierarchy. Here the metadata type *Organization* contains acronyms such as *accu*. The hierarchy has just one level, so the position-in-hierarchy identifier is just an integer. The title does not need to be quoted if it does not contain spaces.

The data at each leaf of the hierarchy is sorted. If *sort* is specified, the list will be sorted by that metadata element; *sort=nosort* will prevent any sorting (i.e. the list will be produced in build order). Otherwise, documents will be sorted by section according to the *Title* metadata.

Classifiers, like plugins, are often designed specifically for particular collections. They are implemented in such a way that they can be nested. For example, an *AZList* classifier like *Titles A-Z* might contain hierarchical elements, or sections of a hierarchical classification might be split up by date using *DateList*. Because of the huge potential number of combinations, a specialized classifier must be written for the particular collection under consideration. This is not difficult: an example can be found in the Computists Communique collection in *tcc/perllib/classify/TCCList.pm*.

Format strings

The *format* statement in the collection configuration file controls the look of the Web pages presented to the user. Format statements are created in conjunction with *classify* options (or in one of the special forms discussed below).

Format strings have changed; we now use CL1, CL2, … which correspond to the first, second, etc classify line in the collection configuration file. For example, *CL4* on line b14 corresponds to the fourth classify line, b11, which is for the "how to" strings.

Browsing displays are considered to be vertical or horizontal lists, or lists of dates—Vlist, Hlist, or DateList.

All the text below needs modifying.

For example, given a *List* classifier using *Title* metadata, the words "List" and "Title" are concatenated together, and the line

```
format ListTitle formatstring
```

formats the resulting list. The *formatstring* can contain text, HTML, and metadata tags to be interpreted at runtime. For example, a simple "title by author" listing would be generated by this command:

```
format ListTitle
  "<td>[link][icon][/link]</td><td>[Title] by [Creator]</td>"
```

Within square brackets may appear any metadata—*Title* and *Creator*, in this example—or one of the items shown in Table 14. The format string generates a small text icon that links to the document text, followed by the title, the string "by", and the author. In lists of result and classification listings, each line is a table row, and thus the above string appears in the context

```
<tr> formatstring </tr>
```

Table 14 Items that may appear in format strings

`[Text]`	The document's text
`[link] … [/link]`	The HTML to link to the document itself
`[icon]`	An appropriate icon (usually the little text icon when in results string)
`[num]`	The document result number (useful for debugging).

Line b12 of Table 10b gives a similar example: the *AZListTitle* classifier generates a small icon linked to the document text followed by the *Title* metadata; you can see the result by clicking *titles A–Z* in the Demo collection. A simpler example is line b13, which generates a list that you can see by clicking *how to* containing the text of the "how to" metadata formatted as a link to the relevant document.

The above format statements are linked to *classify* options; but there is a second kind of format statement that does not involve a classifier but instead takes one of the forms shown in Table 15. This allows the results of queries, the document text, and other aspects of the interface to be presented in appropriate ways.

For example, line b15 of Table 10b formats the actual document text, with the title of the relevant chapter or section preceding the text itself. The last format statement in Table 10b, line b16, puts the cover image at the top left of each document page.

Table 15 shows other kinds of format statements. In some collections links to document pages should be disabled unless that page contains useful information—for example, in bibliography collections where "documents" are abstracts, the link should be disabled when no abstract exists. This is accomplished by the *QueryLinks* format option. The *DocumentButtons* option controls what buttons are to be displayed on a document page. Here, *string* is a list of buttons to be displayed (separated by |), possible values being *Detach*, *Highlight*, *Expand Text*, and *Expand Contents*. Reordering the list reorders the buttons.

Table 15 The format options

`format QueryResults formatstring`	Format of the query results list
`format QueryLinks metadata`	Links from query results list will be disabled unless metadata exists for that document
`format DocumentIcon string/true/false`	If true, put cover image at top left of document page; if a format string, this is the heading to go at the top left
`format DocumentContents true/false`	Display table of contents (if document is hierarchical), or next/previous section arrows and "page k of n" text (if not).
`format DocumentButtons string`	Controls the buttons that are displayed on a document page (default *Detach*|*Highlight*|*Expand Text*|*Expand Contents*)
`format DocumentText formatstring`	Format of the text to be displayed on a document page (default *[Text]*)
`format DocumentArrowsBottom true/false`	Display next/previous section arrows at bottom of document page (default true).

Line b14 of Table 10 contains a rather complicated specification that formats the query result list returned by a search. A simplified version of the format string is

```
"<td valign=top>[link][icon][/link]</td>
 [link][Title][/link]</td>"
```

This is again designed to appear as a table row, which is how the query results list is formatted. It gives a small icon linked to the text, as usual, and the document title, hyperlinked to the document itself.

In this collection, documents are hierarchical. In fact the above hyperlink anchor will be the the title of the section that the query returns. However, it would be better to augment it with the title of the enclosing section, the enclosing chapter, and the book in which it occurs. There exists a special metadata item, *parent*, which is not stored in documents but is implicit in any hierarchical document, that can produce such a list. This will either return the parent document, or if used with the qualifier *All*, the list of hierarchically enclosing parents, separated by a character string that can be given after the *All* qualifier. Thus

```
"<td valign=top>[link][icon][/link]</td>
 <td>{[parent(All': '):Title]: }[link][Title][/link]</td>"
```

will have the desired effect of producing a list of the book title, chapter title, etc that enclose the target section, separated by colons, with a further colon followed by a hyperlink to the target section's title.

Unfortunately, if the target is itself a book, there is no parent and so an empty string will appear followed by a colon. To remove this requires a conditional statement. For this reason, conditional *if* and *or … else* statements can be placed in format strings:

```
{If}{[metadata], action-if-non-null, action-if-null}
{Or}{action, else another-action, else another-action, etc}
```

In both cases, curly brackets are used to signal that the statements should be interpreted and not just printed out as text. The *If* tests whether the metadata is empty and takes the first clause if not, otherwise the second one (if it exists). Any metadata item can be used, including the special metadata *parent*. The *Or* statement evaluates each action in turn until one is found that is non-null. That one is sent to the output and the remaining actions are skipped. Currently, *If* and *Or* statements cannot be nested.

Returning to line b14 of Table 10b, the full format string is

```
"<td valign=top>[link][icon][/link]</td>
 <td>{If}{[parent(All': '):Title],
         [parent(All': '):Title]: }[link][Title][/link]</td>"
```

This precedes the *parent* specification with a conditional that checks whether the result will be empty, and only outputs the parent string if it is present. Incidentally, the *parent* can be qualified by *Top* instead of *All*, which gives the top-level document name that encloses a section—in this case, the book name. No separating string is necessary with *Top*.

Subcollections and language-specific indexes

Finally we sketch the subcollection and language-specific index facilities, which are not used in the Demo collection. Subcollections can be defined, and separate indexes built for each one. For example, in one collection there is a large subset of documents entitled *Food and Nutrition Bulletin*. Three indexes are created, all at the section level: one for the entire collection, another for the *Food and Nutrition Bulletin* documents, and the third for the remaining documents. The configuration file contains the following lines:

```
indexes             section:text
subcollection       fn "Title/^Food and Nutrition Bulletin/i"
subcollection       other "!Title/^Food and Nutrition Bulletin/i"
indexsubcollections fn  other  fn,other
```

The second and third lines define subcollections called *fn*, which contains the *Food and Nutrition Bulletin* documents, and *other*, which contains the remaining documents. The third field of these definitions is a Perl regular expression that identifies these subsets using the *Title* metadata: we seek titles that begin with *Food and Nutrition Bulletin* in the first case and ones that do not in the second case (note the "!"). The final *i* makes the pattern-matching case-insensitive. The metadata field, in this case *Title*, can be any valid field, or *Filename* to match against the document's original filename. The fourth line, *indexsubcollections*, specifies three indexes: one for the *fn* subcollection, one for the *other* subcollection, and the third for both subcollections (i.e. all the documents). Note that if two entries had been specified on the *indexes* line, the total number of indexes generated would have been six rather than three.

If a collection contains documents in different languages, separate indexes can be built for each language. Language is a metadata statement; values are specified using the ISO 639 standard two-letter codes for representing the names of languages—for example, *en* is English, *zh* is Chinese, and *mi* is Maori. Since metadata values can be specified at the section level, parts of a document can be in different languages.

For example, if the configuration file contained

```
indexes section:text section:Title document:text paragraph:text
languages en zh mi
```

section text, section title, document text, and paragraph text indexes would be created for English, Chinese, and Maori—twelve indexes altogether. Adding a couple of subcollections multiplies the number of indexes again. Care is necessary to guard against index bloat.

(This index specification could be defined using the *subcollection* facility rather than the *languages* facility. However, since the current syntax precludes creating subcollections of subcollections, it would then be impossible to index each language in the subcollections separately.)

C.4 Making a Greenstone CD-ROM

C.5 Translating the interface into other languages

The Greenstone software is designed to make it easy to present collections in a variety of different languages. At present, three languages are supported for the user interface—English, Maori, and Chinese. There is a Maori newspaper collection and a Chinese-language collection available from the NZDL home page, but in fact any collection can be viewed in all three languages: it is just a matter of making a choice on the *Preferences* page. This will shortly be altered to make it possible to specify which presentation languages are appropriate for each particular collection. When any particular text or icon is not available in the chosen language, the software is designed to default to the English-language equivalent.

New languages can be added by translating the relevant items in the macro files in *macros*. By looking for the text "Maori", for example, you will see that entries in files *about.dm*, *base.dm*, *document.dm*, *help.dm*, *home.dm*, *pref.dm*, and *query.dm* need to be translated. ???How would you find out about the icons needed?

The fact that the text and icons needed are scattered around different places needs tidying up, and this will be done soon. ???Stefan, it would be good to be able to find out what icons a collection uses, so that they could be translated???

D. Installing the Greenstone software

To make your computer into a Greenstone server, you need to install the Greenstone software. We assume that you already have Web server software installed. Greenstone uses three packages that you will have to obtain from elsewhere:

- STL, the C++ standard template library (www.sgi.com/Technology/STL)
- PERL 5, the PERL programming language (www.gnu.org/software/perl) (the Windows version is included with Greenstone in *packages/wingdbm*)
- GDBM, the GNU database system (www.gnu.org/software/gdbm)

MG, the Managing Gigabytes full-text indexing program, is contained within the Greenstone software and will be installed as part of the installation procedure.

Greenstone can use the Fast CGI system to speed up operation. If this is not installed, it will work as a regular CGI script.

D.1 Installing on Unix

Here is the procedure for installing the software on Unix.

1. Download gsdl-2.1.tar.gz frrom www.nzdl.org/technology

2. Unzip the distribution and cd to the gsdl directory it creates

   type `cd gsdl`

3. Run the configure script using *./configure*. No parameters are needed. This script will check the resources on your system and configure the Greenstone software accordingly.

   type `./configure`

   Possible problems at this stage include:

   - Bad STL header files. The configuration script terminates with the message *STL Broken—Obtain newer version of GNU C Compiler*. This is caused by faults in the header files supplied with GCC; known affected versions are 2.7.2 to 2.7.2.3.
   - Perl missing or broken. The configuration script terminates with the message: *Perl 5 not available—cannot install*. This may be because Perl is not installed at all, or is installed but not executable, or the test script *configtest.pl* is missing. (Further error messages, such as *configtest.pl not found*, will appear; these vary from system to system.) Alternatively Perl may be installed, but is an earlier verson than 5.0 (in which case no further message is generated).
   - GDBM is not installed. The configuration script terminates with the message: *GDBM Library not available—cannot install*.

   Fastcgi is included with the package, and it compiles as fastcgi by default.

At present the library is installed to *$GSDLHOME*: soon it will be changed to ask where the cgi-bin or fastcgi-bin directory is located

4. Compile the Greenstone software by running *Make*. Again, no parameters are needed.

   type `make`

   Possible problems at this stage include:

   - Compilation fails. Various errors may be reported. Problems generally certainly stem from faulty header files supplied with GCC; common problem headers include the *vector* template.

5. Install the software by running *make install*. This installs the newly-compiled binaries to all the right places.

   type `make install`

6. Copy file *etc/site.cfg.example* to *etc/site.cfg* and edit it to suit your site. Typically you'll need to edit the *httpprefix* and *httpimg* fields.

7. Using a Web browser, look at the URL *…/cgi-bin/library*. This will be the home page of your Greenstone system. A small test collection, *demo*, is provided and this will be the only collection shown at this stage.

8. Download other collections from www.nzdl.org. The collections available are shown in Table 16, along with approximate sizes (tar'd and gzipped).

Table 16 Collections and their sizes

collection	abbrev	built size (Mb)	download size (Mb)
Computer science bibliography	csbib	866	?
Computer science technical reports	cstr	2010	~1800
Project Gutenberg	gutenberg	432	457
HCI bibliography	hcibib	36	5
Humanity Development Library	hdl	199	387
Indigenous Peoples	ipc	7.5	4
Maori newspapers	niupepa	12	?
Oral history	ohist	2.5	?
The computists weekly	tcc	21	8
Tidbits magazine	tidbits	10	5
United Nations University collection	unu	52	?
Women's history	whist	6	?
		3654	4947

D.2 Installing on Windows

The Greenstone software works equally well under Windows and Unix (although the remainder of this documentation is written with Unix in mind, to avoid circumlocution).

Installing binaries

Precompiled binaries can be installed along with pre-built collections, just like we currently provide on cd-rom. We'd also provide a library executable for running through web servers, this would need to be installed to the cgi-bin directory along with a configuration file (installshield could ask the user where the cgi-bin directory is).

There are three different types of library executable for windows:

1. A single-user version for collections on CD-ROM or hard drive. This cannot be accessed from other machines.

2. A server version of the above which can be accessed from other machines. This contains its own self-contained Web server.

3. A cgi application for running on an existing eb server (Apache, IIS or similar).

The source code for these versions is very similar. The only difference between the first two is that one line of code is commented out. The difference with the third is that the makefile is edited to replace a couple of *.cpp* and *.h* files.

The third version provides exactly the same facilities as the Unix system. To provide a degree of system-independence, the value of GSDLHOME is not compiled in but is read from the *site.cfg* configuration file.

We are currently implementing an InstallShield package to allow you to easily install the Greenstone binaries on Windows systems; then the value of the GSDLHOME field in site.cfg will be set from InstallShield (???George to do)

Installing the source code

If you want to work on the Greenstone software itself, you will need to install the source code. This gives you the ability to

(c) build collections
(d) modify the building or library (interface) code to suit your needs
(e) compile the library to run through a web server
(f) compile the library to run as a stand-alone program off cd-rom or hard disk.

WIndows installation is very similar to the Unix procedure. First, download the zipped package and unzip it (eventually, this step will be done with Installshield). To compile the library, use *make* (*nmake* on some systems) in the $GSDLHOME directory. Compilation is known to work with the Microsoft C++ compiler; other compilers may require some modifications to be made.

GDBM is included in the *packages/wingdbm* directory and is compiled by the top-level *make* command. To build collections, the PERL programming language must be downloaded and installed. STL, the Standard Template Library, will also need to be downloaded and installed; there are several

Windows ports (e.g. ???Stefan, URLs). Finally, the *crypt* package is used by the authentication mechanism; a Windows port can be downloaded from ???Stefan. (Ultimately we hope to include GNU-licensed versions of STL and crypt along with the package.)

D.3 Configuration files

The operation of the software is controlled by several configuration files. We will encounter the collection-specific configuration file *collect.cfg* in Part C when learning how to build collections, and the format of these files will be described there. The Greenstone system includes other configuration files, all of which are read using the same software and so have the same format.

The receptionist can run in a "general" mode or in a "collection-specific" model. In the former model it reads two configuration files, in this order:

```
etc/site.cfg
etc/main.cfg
```

Site configuration files are used to configure the Greenstone software for the site where it is installed. Typical entries in these files are shown in Table 17; again, to fully understand what these mean you will have to read Part C first. The system site configuration file is useful for keeping configuration options that are particular to a given site, like *httpimg* and *gwcgi*. It is also useful for work in progress, where new options would conflict with older versions of the collection.

The main configuration file *main.cfg* contains information that is common to all collections served by the receptionist. It resides in the top level directory, *$GSDLHOME*.

In "collection-specific" mode the receptionist will read four configuration files:

```
etc/site.cfg
collect/collection-name/etc/site.cfg
etc/main.cfg
collect/collection-name/etc/collect.cfg
```

As their name implies, collection configuration files are collection-specific, and may override settings in the main configuration files.

Table 17 Entries in the system configuration files

site.cfg	`httpprefix`	HTTP address of *$GSDLHOME*
	`httpimg`	HTTP address of the directory containing the images used in the interface
	`gwcgi`	HTTP address of the CGI script being run. (If the HTTP server sets the environment variable *$SCRIPT_NAME*, this overrides it)
	`maxrequests`	Maximum number of requests a Fast CGI process will serve before it exits
main.cfg	`maintainer`	Email address of the receptionist's maintainer
	`macrofiles`	Display macro files that are to be loaded by the receptionist
	`status`	Status page enabled/disabled

E. Modifying the software

[introductory text here]

E.1 Working with Greenstone

Where to find the software

The Greenstone software is stored in the directories shown in Table 18. In the *src* directory, the *receptionist* is the program that organizes the Greenstone user interface and presents it to the user over the Web interface, while one *collection server* process is created for each collection to manage interaction with it. The other programs are used at built time: *txt2db* to create the GDBM database and *hashfile* to generate the OID of a document. Each collection's GDBM file is stored as a *.bdb* file in the collection's *index/text* directory, and the *db2txt* program is a useful tool for viewing a GDBM database as text. The executable versions of these utilities are stored in *bin/$GSDLOS*.

Moving to the *bin* directory, *$GSDLOS* is a shell variable that identifies the operating system being used: this is necessary when a file system is shared between machines with different operating systems. Possible values are *linux*, *sunos*, and *windows*, and the appropriate value is set in the *setup* script. The Perl scripts are stored in files whose names end in *.pl*. Although this is unnecessary for Unix systems, Windows systems need the extension in order to execute these scripts properly.

Table 18 The directories that contain the Greenstone software system

Directory			Contents
$GSDLHOME	/src		C++ source code
		/recpt	Program that organizes the Greenstone user interface (approx 11,500 lines of C++)
		/colservr	Program responsible for providing access to a collection when it is being used (approx 5,500 lines)
		/txt2db	Program used at build time to create the GDBM database
		/db2txt	Tool for viewing a GDBM database as text
		/hashfile	Program used at import or build time to generate document OIDs
	/bin		Executable code
		/$GSDLOS	Binaries
		/script	Perl scripts
	/perllib		Perl modules used at build time
		/plugins	The plugins for handling documents of different formats. The build software will look here first when searching for plugins to build this collection. This allows for collection specific plugins to be written, existing plugins to be overridden and changed, etc.
		/classify	The classifiers for sorting documents into different classification types (e.g. a list sorted by title, a hierarchical classification etc). Collection specific classifiers (same as for plugins).
	/lib		C++ code used by both receptionist and collection server (approx 4,000 lines)
	/packages	/fcgi	Source code for Fast CGI
	/packages	/mg-1.3d	Source code for MG
	/unicode		Unicode translation tables for the GB Chinese character set
	/etc		Configuration files, initialization and error logs, authorization databases
	/macros		Display macro files
	/images		Images used by interface

At present (because as yet only the null protocol is implemented) the receptionist and collection server code are compiled together to generate a single executable file called *library* for the receptionist and collection servers together. This is installed in the *cgi-bin* directory, a system-dependent location which is determined when the Greenstone system is installed and written into the relevant makefiles.

Turning to the *perllib* and *lib* directories, the Perl and C++ source code both make substantial use of libraries. Examples of C++ code used by both receptionist and collection server include *text_t.cpp* and *text_t.h*, which implement the *text_t* string class used throughout all the code, and *display.cpp* and *display.h*, which implement the *display* class for manipulating and displaying the macro language that is used to specify the contents of Web pages.

Some packages are also used by the system. Although it is really an external package, the Fast CGI source code needs to be located with the Greenstone software because references to it are compiled in with the code. MG is the

Managing Gigabytes scheme for full-text indexing and search; the binaries are installed to *bin/$GSDLOS*.

Other subdirectories contain miscellaneous information used in the Greenstone system. *etc* contains the top-level configuration files *main.cfg* and *site.cfg*, the authorization databases *key.db* and *users.db* (both in GDBM format), and error logs. *macros* contains macro files that are used for text display. *images* contains all the images used by the interface.

The *admin* facility

The source code control system

E.2 Using the collection server protocol

CGI arguments

Description of a receptionist

E.3 Macro files

Syntax of macro files

Macro packages

Macro options