

Morgan Kaufmann Publishers is an imprint of Elsevier.
30 Corporate Drive, Suite 400, Burlington, MA 01803, USA

Copyright © 2003, 2010 by Elsevier Inc. All rights reserved.

Designations used by companies to distinguish their products are often claimed as trademarks or registered trademarks. In all instances in which Morgan Kaufmann Publishers is aware of a claim, the product names appear in initial capital or all capital letters. All trademarks that appear or are otherwise referred to in this work belong to their respective owners. Neither Morgan Kaufmann Publishers nor the authors and other contributors of this work have any relationship or affiliation with such trademark owners nor do such trademark owners confirm, endorse, or approve the contents of this work. Readers, however, should contact the appropriate companies for more information regarding trademarks and any related registrations.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopying, scanning, or otherwise—without prior written permission of the publisher.

Permissions may be sought directly from Elsevier's Science & Technology Rights Department in Oxford, UK: phone: (+44) 1865 843830, fax: (+44) 1865 853333, E-mail: permissions@elsevier.com. You may also complete your request online via the Elsevier homepage (<http://www.elsevier.com>), by selecting "Support & Contact" then "Copyright and Permission" and then "Obtaining Permissions."

Library of Congress Cataloging-in-Publication Data

A catalog record for this book is available from the Library of Congress.

British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library.

ISBN: 978-0-12-374857-7

For information on all Morgan Kaufmann publications,
visit our Web site at www.mkp.com or www.elsevierdirect.com

Printed in the United States of America
09 10 11 12 13 5 4 3 2 1

Working together to grow libraries in developing countries		
www.elsevier.com www.bookaid.org www.sabre.org		
ELSEVIER	BOOK AID International	Sabre Foundation

Preface

On the top floor of the Tate Modern Art Gallery in London is a meeting room with a magnificent view over the River Thames and down into the open circle of the reconstructed Globe Theatre nearby. Here, at a gathering of senior administrators who fund digital library projects internationally, one of the authors stood up to introduce himself and ended by announcing that he was writing a book entitled *How to Build a Digital Library*. As he sat down, his neighbor nudged him and asked with a grin, “A work of fiction, eh?” A few weeks earlier and half a world away, the same author was giving a presentation about a digital library software system at an international digital library conference in Virginia, when a colleague in the audience noticed someone in the next row who, instead of paying attention to the talk, downloaded that very software over a wireless link, installed it on his laptop, checked the documentation, and built a digital library collection of his e-mail files—all within the presentation’s 20-minute time slot.

These little cameos illustrate the extremes. Digital libraries?—colossal investments, which like today’s national libraries will grow over decades and centuries, daunting in complexity. Conversely: digital libraries?—off-the-shelf technology; just add documents and stir. Of course, we are talking about very different things: a personal library of ephemeral notes hardly compares with a national treasure-house of information. But don’t sneer at the “library” of e-mail: this collection gives its user valued searching and browsing facilities, and with half a week’s (rather than half an hour’s) work, one could create a prototype document-management system that organizes documents for a large multinational corporation.

Digital libraries are organized collections of information. Our experience of the World Wide Web—vibrant yet haphazard, uncontrolled and uncontrollable—daily reinforces the impotence of information without organization. Likewise, experience of using online public-access library catalogs from the desktop—impeccably but stiffly organized, and distressingly remote from the actual documents themselves—reinforces the frustrations engendered by organizations without fingertip-accessible information. Can we not have it both ways? Enter digital libraries.

Whereas physical libraries have been around for 25 centuries, digital libraries span only 15 years. Yet, in today’s information society, with its Siamese twin the knowledge economy, digital libraries will surely figure among the most important and influential institutions of the new century. The information revolution not only supplies the technological horsepower that drives digital libraries, but also fuels an unprecedented demand for storing, organizing, and accessing information. If information is the currency of the knowledge economy, digital libraries are the banks where it is invested.

We do not believe that digital libraries are supplanting existing bricks-and-mortar libraries—not in the near- and medium-term future that this book is about. And we certainly don’t think you should be burning your books in favor of flat-panel displays! Digital libraries are new tools for achieving human goals by changing the way that information is used in the world. We are talking about new ways of dealing with knowledge, not about replacing existing institutions.

What is a digital library? What does it look like? Where does the information come from? How do you put it together? Where to start? The aim of this book is to answer these questions in a plain and straightforward manner, with a strong practical “how to” flavor.

We define digital libraries as

focused collections of digital objects, including text, video, and audio, along with methods for access and retrieval, and for selection, organization, and maintenance.

To keep things concrete, we show examples of digital library collections in an eclectic range of areas, with an emphasis on cultural, historical, humanitarian, and musical applications, as well as technical ones. These collections are formed from different kinds of material, organized in different ways, presented in different media and different languages. We think they will help you see how digital libraries can be applied to real problems. Then we show you how to build your own.

The Greenstone Software

The Greenstone Digital Library Software is a comprehensive software resource that illustrates the ideas in the book and could form a basis for your own digital library. It is freely available as open source software on the World Wide Web (at www.greenstone.org) and comes precompiled for all popular platforms. A fully operational, flexible, extensible system for constructing easy-to-use digital libraries, Greenstone is widely deployed internationally and is being used (for example) by United Nations agencies and related organizations to deliver humanitarian information in developing countries. The ability to build new digital library collections, particularly in developing countries, is promoted by UNESCO’s *Information for All Programme*. Through this initiative, the intergovernmental agency provides support for Greenstone in the form of testing, translating, and distributing the software.

In this second edition of *How to Build a Digital Library* we have decoupled Greenstone from the more general material in the book by packing everything related to the software into a separate Part II, which now serves as a comprehensive tutorial guide to Greenstone. All the material in Part I has broad application and is not tied to any particular software infrastructure for digital libraries.

Since the first edition was published, Greenstone has acquired a new interactive interface for librarians that makes it far easier to build and serve collections. In addition, the software has grown enormously and now includes an unparalleled range of facilities for textual and multimedia documents, handling and extracting metadata in various formats, configuring collections, and interoperation with other standards and protocols. In fact, almost everything described in Part I can be accomplished within the Greenstone software: it is a complete industrial-strength implementation of essentially all the techniques covered in this book.

Updated and Revised Content

We finished writing the first edition of this book in late 2002 and now, in August 2009, are just polishing this second edition. The field of digital libraries has changed radically in the intervening years. Although much of the core material remains the same, we have made the most of our opportunity to update it to reflect the changes that have taken place over seven years. We have thoroughly revised and edited everything; in addition, we have a new co-author, David Nichols. The most enjoyable part has been adding new material—here are the highlights.

Responding to popular demand, we include a new chapter on people in digital libraries. We entered the digital library field with a perspective that was strongly colored by our technical background in computer science. However, from our experience in giving scores of Greenstone courses internationally and interacting with the user base on the mailing list, we now have extensive experience of the way digital libraries operate. Chapter 2 introduces the roles that people play in digital libraries and discusses issues of identity and anonymity, help and support services, individual usage and group collaboration, and the growing area of user contributions to digital collections.

Chapter 5 on multimedia is also completely new. The pace and scope of change in global multimedia are staggering. YouTube, iTunes, and Flickr are just a few examples of how people around the world create and share material in newly connected and digitally complex ways. Low-cost portable devices are helping to shape a dynamic multimedia-driven approach to communications, arts, business, and research. Multimedia content management presents cross-disciplinary challenges in integrating and organizing data from a plethora of media types that, to date, have remained stratified (e.g., images of various kinds, music and songs, commentary, video, and text in various guises and formats). Few technologies today allow high-quality, user-friendly, and graceful mixing of multiple media types into rich collections of accessible information. This is a gap that digital libraries must fill.

Chapter 6 on metadata includes extensive new material on metadata for audio, video, multimedia, and compound objects, as well as a new discussion of metadata quality.

Internationalization is another burgeoning area. The first edition of *How to Build a Digital Library* contained plenty of information on the subject, but recognizing that some readers with an English-only perspective may find such material unnecessary and distracting, we have consolidated it into a single new chapter, Chapter 8. Greenstone itself is fully internationalized, with interfaces in fifty different languages.

There are countless other changes in the book that reflect the way the field is developing. We have updated many of the examples to reflect today's larger and more comprehensive digital collections. We include more material on planning a digital library, document surrogates, and faceted browsing. Many new formats are described, including the Open Document Format for Office Applications and Microsoft's Office Open XML, as well as other document types, such as e-mail, spreadsheets, and presentations. We have focused our metadata chapter on external metadata and separated the issue of markup (internal metadata); we also reduced the level of detail in which the HTML format is described. This edition includes much new information on interoperability, and separates it from the issue of Greenstone support—which, as stated above, is consigned to Part II.

How the Book Is Organized

The gulf between the general and the particular has presented interesting challenges in organizing this book. As the title says, our aim is to show you how to build a digital library, and we really do want you to build your own collections (it doesn't have to take long, as the conference attendee mentioned in the first paragraph discovered). But to work within a proper context, you need to learn something about libraries and information organization in general. And if your practical work is to proceed beyond a proof-of-concept prototype, you will need to come to grips with countless nitty-gritty details.

We have tried to present what you need to know in a logical sequence, introducing new ideas where they belong and developing them fully at that point. However, we also want the chapters to function as independent entities that can be read in different ways. We are well aware that books like this are seldom read through from cover to cover! The result is, inevitably, that some topics are scattered throughout the book.

We cover three different themes: the intellectual challenges of libraries and digital libraries, the practical standards involved in representing documents digitally, and how to use Greenstone to build your own collections. Many academic readers will want a textbook, some a general text on digital libraries, others a book with a strong practical component that supports student projects.

For a general introduction to digital libraries, read Chapters 1 and 2 to learn about libraries and library organization, then Chapter 3 to find out about what digital libraries look like from a user's point of view, and then skip straight to Chapter 9 to see what the future holds.

To learn about the ways that documents are represented digitally, skim Chapter 1, read Chapters 4, 6, and 7 to learn about the standards, and then look at Chapter 3 to see how they can be used to support interfaces for searching and browsing. If you are interested in multimedia, read Chapter 5 as well—or instead, because it largely stands alone.

To learn how to build a digital library as quickly as possible, skim Chapter 1 (but check Sections 1.6 and 1.7) and turn straight to Part II. If you run into things you need to know about library organization, different kinds of interfaces, document formats, or metadata formats, you can return to the intervening material.

For a textbook on digital libraries without any commitment to specific software, use Part I of the book in sequence. For a course with a strong practical component, read all chapters—and, in parallel, turn your students loose on Part II!

What the Book Covers

We open with four scenarios intended to dispel any ideas that digital libraries are no more than a routine development of traditional libraries with bytes instead of books. Then we discuss the concept of a *digital library* and set it in the historical context of library evolution over the ages. We go on to exemplify features of a large-scale, real-world digital library by looking at a usage scenario. One thread that runs through the book is internationalization and the role of digital libraries in developing

countries—for whom we believe that digital libraries represent a “killer app” of computer technology. There follows a discussion of issues involved in copyright and “harvesting” material from the Web. Finally, we close by discussing the planning of a digital library and by briefly introducing the Greenstone software, which is fully described in Part II.

Chapter 2 is about the people in digital libraries. As noted above, it covers the many roles played by people and discusses issues of identity and anonymity, help and support services, individual usage and group collaboration, and the growing area of user contributions to digital collections—and what all this implies in terms of software support.

As the definition of *digital library* given earlier implies, digital libraries involve two communities: end users who are interested in access and retrieval, and librarians who select, organize, and maintain information collections. Chapter 3 takes the user’s point of view. Of course, digital libraries would be a complete failure if you had to study a book in order to learn how to use them—they are supposed to be easy to use!—and this book is really directed at the library builder, not the library user. Nevertheless, it is useful to survey what different digital libraries look like. Examples are taken from domains ranging from human development to culture, with audiences ranging from children to library professionals. Document contents range from text to newspaper images, and multimedia material ranges from musical query-by-humming to browsing pictures according to their content. (International examples are reserved for Chapter 8.) We show many examples of browsing structures, from simple lists to hierarchies, date displays, and faceted structures, and close with a description of the use of a popular institutional repository system.

Next we turn to documents, the digital library’s raw material. Chapter 4 begins with character representation, in particular Unicode, which is a way of representing all the characters used in all the world’s languages (although again international aspects are covered in Chapter 8). Plain text formats introduce some issues that you need to know about. Here we take the opportunity to describe full-text indexing, the basic technology for searching text, and we also introduce the issue of word segmentation. We then describe the process of optical character recognition (OCR), including typical costs and an example OCR project. Next we look at documents on the Web: HTML and XML, including style sheets and the presentation of Web documents. Next we study popular formats for document representation, beginning with the page description languages PostScript and PDF (Portable Document Format) and continuing with the word-processor formats RTF (Rich Text Format), the Open Document Format for Office Applications, and LaTeX, which is commonly used for mathematical and scientific documents. Finally, we look at other document types: e-mail, spreadsheets, and presentation files.

Chapter 5 gives a comprehensive account of multimedia, beginning with a brief introduction to compression and transforms. Then we describe audio, image, and video formats; we go on to introduce so-called “rich media,” a term designed to emphasize interaction with multimedia composed of potentially different types of media; and we end with a discussion of music and digital music libraries. A plethora of coding techniques and formats are covered: PCM, WAV, AIFF, AU, MP3, AAC, Ogg Vorbis, and FLAC for audio; GIF, PNG, JPEG, JPEG 2000, and TIFF for images; MPEG-1, MPEG-2, MPEG-4, and Ogg Theora for video, along with a discussion of streaming and proprietary formats; SMIL and Adobe Flash for rich media; and MIDI for music.

Besides textual and multimedia documents, there is another kind of raw material for digital libraries: metadata. Often characterized as “data about data,” metadata figures prominently in this book because it forms the basis for organizing both digital and traditional libraries. Chapter 6 covers metadata and explains how it is expressed in traditional library catalogs and in digital libraries. Like the previous two chapters, Chapter 6 covers many different formats, including MARC, MARCXML, Dublin Core, qualified Dublin Core, MODS, BibTeX, and EndNote for documents; TIFF, EXIF, XMP, IPTC, and MIX for images; MPEG-7 and MPEG-21 for multimedia; RDF, METS, OAI-ORE, LOM, and SCORM for compound objects. We introduce issues of metadata quality, and the idea of extracting metadata from the raw text of the documents themselves, giving examples of what can be extracted.

Chapter 7 reaches out to look at other standards and protocols that allow digital libraries to interoperate with one another and with related technologies. We describe the Z39.50 protocol used by current library automation systems, the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH), methods for persistent object identification, including various kinds of digital object identifier (Handles, DOIs, and OpenURLs), Web services, and the Search/Retrieve URL Service (SRU). We also discuss different standard ways of authenticating users: LDAP, OpenID, and Shibboleth. We close by looking at two prominent open source digital library software systems: DSpace and Fedora.

Chapter 8 also reaches out, this time internationally. We begin with some examples of multilingual interfaces to digital libraries and examples of collections of documents in different languages. Then we give a comprehensive account of the Unicode standard for representing the characters used in all the world’s languages. Hindi and Indic scripts present interesting problems of character coding that have not yet been entirely solved in Unicode-compliant applications, and we give a glimpse of the complexities involved. Chinese and some other Asian languages involve other issues because they are written without word spaces. Also, “alphabetical order” is moot in non-alphabetic ideographic languages, and this has implications for browsing digital libraries.

Part I closes with visions of the future of digital libraries and mentions some important related topics that we have not been able to develop fully. We hope that this book will help you learn the strengths and pitfalls of digital libraries, gain an understanding of the principles behind the practical organization of information, and come to grips with the tradeoffs that arise when implementing digital libraries.

Part II is about the Greenstone Digital Library Software. Switching to a tutorial style of presentation, we present exercises that develop collections that demonstrate particular capabilities discussed in Part I. Chapter 10 commences by building a diverse range of digital library collections using the Librarian Interface, collections that include a representative selection of document and multimedia types from Chapters 4 and 5: documents in Word, HTML, and other formats; photographic and facsimile images; audio and video. They also utilize several bibliographic formats from Chapter 6. In some cases, bibliography files constitute digital library documents in their own right; in others, they are combined with the source documents that the bibliographic entries describe.

Chapter 11 describes the internal structure of Greenstone—the files and folders that support its operation—and shows how collections can be built outside the Librarian Interface using command-

line scripts. This provides greater control and is often appropriate for very large collections. The chapter also presents practical examples of interoperability, again mirroring the topics covered in Part I's Chapter 7, and it closes with a discussion of very-large-scale usage of Greenstone. Finally, Chapter 12 explores the possibilities afforded by Greenstone's presentation layer in more depth to illustrate advanced user interface techniques.

We hate acronyms and shun them wherever possible—but in this area you just can't escape them. A glossary of terms is included near the end of the book to help you through the swamp.

Having worked your way through Parts I and II, you will be well prepared to develop large-scale production-level digital library systems like those illustrated in the book. The rest is up to you. Our aim will have been achieved if you actually *build a digital library*!

About the Web Site

A great deal of supplementary material is available on the *How to Build a Digital Library* Web site at www.greenstone.org/howto. There you will find a novel full-text index of the book at the sentence level in which you can locate sentences containing any word combination and find their page numbers in the printed book. You can view all the figures in full color and browse a list of acronyms, a hierarchical structure of phrases, and a collage of the images. You can also download machine-readable versions of all the XML examples, and sample files for the exercises in Part II. The Web site also contains an appendix that gives more information on markup and XML.

Acknowledgments

The best part of writing a book is reflecting on all the help you have had from your friends. This book is the outcome of a long-term research and development effort at the University of Waikato—the New Zealand Digital Library Project. Without the Greenstone software, the book would not exist, and we begin by thanking Rodger McNab, who charted our course years ago by making the major design decisions that underlie Greenstone. Rodger has long departed from our group, but the influence of his foresight remains—a legacy that this book exploits. Next comes Stefan Boddie, the man who kept Greenstone going for many years, navigating the shoals with a calm and steady hand on the tiller. Craig Nevill-Manning had the original inspiration for the expedition: he showed us what could be done, and left us to it. Today the ship is capably steered by Kathy Don, Oran Fry, Anna Huang, Anu Krishnan, and Max Rouast.

Every crew member, past and present, has helped with this book, and we thank them all. Most will have to remain anonymous, but we must mention a few striking contributions (in no particular order). Te Taka Keegan and Mark Apperley undertook the Māori Newspaper project described at the end of Section 4.2 and illustrated in Figure 8.1. Through Te Taka's efforts we receive inspiration every day from the magnificent *toki* that resides in our laboratory and can be seen in Figure 1.11, a gift from the Māori people of New Zealand that symbolizes our practical approach to building digital libraries. Lloyd Smith (along with Rodger and Craig) did the groundwork for the music collections that are illustrated here. Steve Jones builds many novel user interfaces, especially ones involving phrase

browsing. Sally Jo Cunningham is the resident expert on library organization and related matters. Stuart Yeates designed and built the acronym extraction module, while Dana McKay worked on such things as extracting date metadata, as well as drafting the Greenstone manuals. YingYing Wen was our chief source of information on the Chinese language and culture, while Malika Mahoui took care of the Arabic side. Matt Jones from time to time provided us with sage and well-founded advice. Kathy Don helped us get many technical details straight, and Anna Huang helped enormously with the figures.

Many others in the digital library lab at Waikato have made heroic technical contributions to Greenstone. John Thompson designed and implemented the original Librarian Interface. Michael Dewsnip made countless improvements to the software. Gordon Paynter built the phrase browsing interface, helped design the Greenstone communication protocol, and improved many aspects of metadata handling. Hong Chen, Trent Mankelow, John McPherson, David Milne, Todd Reed, Shaoqun Wu, and Xiaofeng Yu have all worked to improve the software. Geoff Holmes and Bill Rogers helped us with some very nasty low-level Windows problems. Eibe Frank and Olena Medelyan worked on keyphrase extraction, while Veronica Liesaputra built the realistic book illustrated in Figure 3.5. Annette Falconer worked on a Women's History collection that opened up new avenues of research. Rob Akscyn has been a continual source of inspiration, and his wonderful metaphors enliven this book. There are many others: we thank them all.

Tucked away as we are in a remote (but very pretty) corner of the Southern Hemisphere, visitors to our department play a crucial role: they act as sounding boards and help us develop our thinking in diverse ways. Some deserve special mention. George Buchanan has been a frequent visitor from the United Kingdom; he helped develop the OAI-PMH server and built the CD-ROM writing module, and he continues to work with our team. Elke Duncker, also from the United Kingdom, advised us on cultural and ethical issues, while Stefan Rüger helped with multimedia digital libraries. The influence of Carl Gutwin from Saskatchewan is particularly visible in the phrase browsing and keyphrase extraction areas; Wendy Osborn from Alberta developed the scheduled rebuilding capability. Gary Marsden from Cape Town also made significant contributions. Dan Camarzan, Manuel Ursu, and their team of collaborators in Brasov, Romania, have worked hard to improve Greenstone and put it into the field. Alistair Moffat from Melbourne, Australia, along with many of his associates, was responsible for MG, the full-text searching component, and he and Tim Bell of Christchurch, New Zealand, have been instrumental in helping us develop the ideas expressed in this book.

Special thanks are due to Michel Loots of Human Info in Antwerp and John Rose, formerly of UNESCO in Paris. Informed by their great wealth of experience, both have encouraged, cajoled (and occasionally bullied) us into making our software available in a form designed to be most useful to people in developing countries. We are particularly grateful to them for opening up this new world to us; it has given us immense personal satisfaction and the knowledge that our technological efforts are materially helping people in need. John now works for our project and is instrumental in our outreach activities in developing countries.

We acknowledge the support of Maria Trujillo of Colombia, and Chico Fernandez-Perez of the FAO in Rome. Until he was so sadly and unexpectedly snatched away from us, we derived great benefit from the boundless enthusiasm of Ferrers Clark at the Canadian national science and technology

library. We have learned much from conversations with Dieter Fellner of Braunschweig, particularly with respect to generalized documents, and from Richard Wright at the BBC in London. Last but by no means least, Harold Thimbleby in the United Kingdom has been a constant source of moral support.

We would like to acknowledge all who have translated the Greenstone interface—at the time of writing we have interfaces in 50 different languages. We are grateful to Jojan Varghese and his team from Vergis Electronic Publishing, Mumbai, India, for taking the time to explain the intricacies of Hindi and related scripts. We also thank everyone who has contributed to the packages included in the Greenstone distribution. And we thank all our Greenstone users, from whom we have learned so much.

The Department of Computer Science at the University of Waikato has supported us generously in all sorts of ways, and we owe a particular debt of gratitude to Mark Apperley for his enlightened leadership, warm encouragement, and financial help. In the early days we were funded by the New Zealand Lotteries Board and the New Zealand Foundation for Research, Science and Technology, which got the project off the ground. We have also received support from the Ministry of Education, while the Royal Society of New Zealand Marsden Fund has supported closely related work on text mining and computer music. The Alexander Turnbull Library has given us access to source material for the Māori Niupepa project, along with highly valued encouragement.

We thank Rick Adams and Heather Scherer of Morgan Kaufmann, and we gratefully acknowledge the efforts of the reviewers Bonnie Wilson and J. Stephen Downie, who carefully read the manuscript and made pertinent and constructive comments that helped us improve this book significantly.

Much of this book was written in people's homes while the authors were traveling around the world. We visited an extraordinary variety of delightful little villages—Killinchy in Ireland, Great Bookham and Welwyn North in England, Pampelonne in France, Mascherode in Germany, Canmore in Canada—as well as cities such as London, Siena, Heidelberg, Paris, Tokyo, Calgary, New Orleans, Austin, and San Francisco. To our hosts: you all know who you are—thanks! Numerous institutions helped with facilities, including Middlesex University in London, Braunschweig Technical University and the European Media Lab in Germany, the University of Calgary in Canada, the University of Texas at Austin, the University of North Carolina at Chapel Hill, the Payson Center for International Development and Technology Transfer in New Orleans, and the University of Siena in Italy. The generous hospitality of Google during a two-month stay is gratefully acknowledged: this proved to be a very stimulating environment in which to think about large-scale digital libraries.

All our traveling has helped spin the threads of internationalization and human development that are woven into the pages that follow. Our families—Annette, Pam, Anna, Elizabeth, Natasha, and Nikki—have supported us in countless ways, sometimes journeying with us, sometimes keeping the fire burning at home in New Zealand. They have had to live with this book, and we are deeply grateful for their sustained support, encouragement, and love.

Orientation: The world of digital libraries

Digital libraries are changing the world.

■ Example One: Supporting Human Development

Kataayi is a grassroots cooperative organization based in the village of Kakunyu in rural Uganda. In recent years its enterprising members have built ferro-cement rainwater catchment tanks, utilized renewable energy technologies such as solar, wind, and biogas, and established a local industry making clay roofing tiles—among many other projects. But amid such human resourcefulness, information resources are scarce. The nearest public phone, fax, library, newspapers, and periodicals are found in the district town, Masaka, 20 km distant over rough roads. Masaka boasts no e-mail or Internet access. The difficulty of getting there effectively discourages local inhabitants from taking advantage of the information and communication technologies that we take for granted in developed countries.

The Kataayi community believe that an information and communication center will have a major development impact in their area. They laid the groundwork by acquiring a computer and solar power generation equipment. They established an e-mail connection via cellular phone and set up a computer training program. They constructed a brick building to house the center (Figure 1.1). And they gathered several books. But they need more information resources—lots more. They were looking for books covering such topics as practical technology, fair-trade marketing, agriculture, environmental conservation, spirituality, and social justice issues.

Then they discovered digital libraries. The Humanity Development Library is a compendium of some 1,200 authoritative books and periodicals on just such topics, produced by many disparate organizations—UN agencies and other international organizations. In print, these books would weigh 340 kg, cost \$20,000, and occupy a small library bookstack. Instead, the collection takes the form of a digital library and is distributed on a single CD-ROM throughout the developing world at essentially no cost. Related digital library collections cover topics including disaster relief, agriculture, the environment, medicine and health, food and nutrition; more are coming. These digital libraries will increase Kataayi's information resources immeasurably, at a minuscule fraction of the cost of paper books.

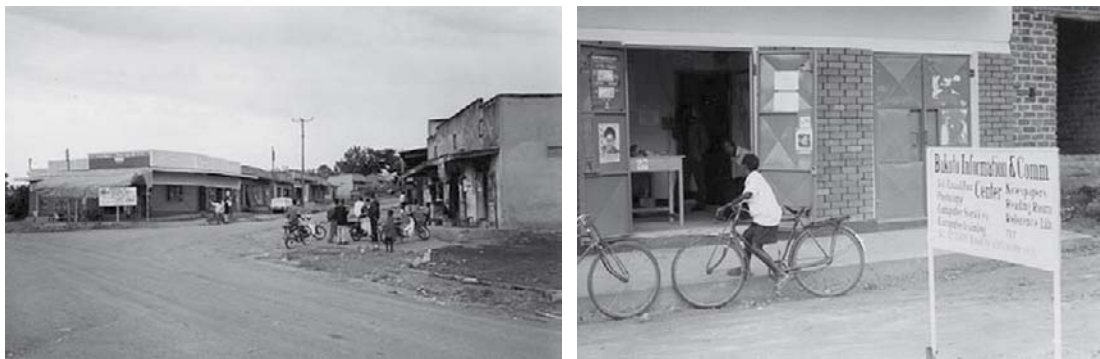


Figure 1.1: Kataayi's information and communication center

■ Example Two: Pushing on the Frontiers of Science

Leave this local community and enter a very different one that operates internationally and on a far larger scale. For over a decade, physicists have been using automated archives to disseminate the results of their research. The first archive, in high-energy physics, began in the early 1990s. It targeted a tiny group of fewer than 200 physicists working on a particular set of research problems, who wanted to communicate their progress. Within months the clientele had grown fivefold. Numerous other physics databases sprang into existence. Within a few years these archives served tens of thousands of researchers; by the year 2000 they processed 150,000 requests per day and the number of papers passed the half-million mark in 2008.

The physics archival digital libraries are entirely automated. To submit a research paper, contributors fill out title, author, and abstract on an electronic form and transmit the full text of the paper. Upon receipt, which is instantaneous, the paper immediately and automatically becomes part of the archive, permanently accessible to others. The contributions are not reviewed or moderated in any way, except for a quick scan to ensure that they are relevant to the discipline. The chaff is simply ignored by the community. The upshot is that research results are communicated on a dramatically accelerated timescale, and the expense and waste of hard-copy distribution is eliminated.

For some areas of physics, online archives have already become the dominant means of communicating research progress. Many people believe that the scheme has effectively replaced commercial publication as a way of conveying both topical and archival research information. Why don't researchers in every discipline follow suit? Soon, perhaps, they will. Proponents of these online archives forecast the imminent demise of commercially published research journals and believe that communicating research results using "chemicals adsorbed onto sliced processed dead trees" will rapidly become a quaint anachronism. On the other hand, many disagree: they argue that peer review is still highly valued in most scientific disciplines, and that even in the small, specialized communities where they are used, online archives augment rather than replace peer-reviewed journals.

■ Example Three: Preserving a Traditional Culture

The physics archive is centered at the Los Alamos National Laboratory in New Mexico. Only 58 km away as the crow flies, but light-years distant in other respects, is the Zia Pueblo, home of one of a score of Native American tribes in New Mexico (Figure 1.2). By 1900 the population had fallen to less than 100, and the tribe was expected to die out during the 20th century. With the return of some land, and medicine and education provided by U.S. government programs, fortunes have improved and the people now number 600. But today the Zia Pueblo face a major problem: the loss of their language and traditional culture. Young people are not learning the Zia Pueblo traditions or Keresan, its language. This is a common complaint in traditional societies, overexposed as we all are to the deafening voice of popular commercial culture blaring everywhere from television, radio, and advertising billboards.

To preserve the Zia language and traditions, a digital library has been proposed. It will include an oral history compilation, with interviews of tribal elders conducted in their native language. It will include an anthology of traditional songs, with audio recordings, musical scores transcribed from them, and lyrics translated by a native speaker. It will include video recordings of tribal members performing Pueblo dances and ceremonies, along with a synopsis describing each ceremony and a transcription and translation of the recorded audio. The goal is to produce a multimedia artifact, the purpose of which is not so much to archive the material as to make it publicly available and to involve members of the tribe in collecting and disseminating it.



Figure 1.2: The Zia Pueblo village

■ Example Four: Exploring Popular Music

Turn from this small, esoteric group in New Mexico to the wide-ranging, disorganized, eclectic panoply of music that is played in the Western world today. In all human societies, music is an expression of popular culture. Different generations identify strongly with different musical styles and artists. People's taste in music reflects their personality and sense of identity: teenagers, in particular, regard their musical preferences as being strongly bound up with who they are. Music is a medium that is both popular and international. Pop music culture transcends social boundaries, be they national borders or socioeconomic groupings. Yet music also exhibits strong cultural diversity: folk music is specific to a particular country or region, and different styles characterize local ethnic groupings.

Imagine a digital music library that reflects popular taste, a library that people from all walks of life will want to use. From an immense music collection you can retrieve tunes in many ways: by humming a theme, by recalling words from the title or lyrics, by giving the composer's name, or you can specify any combination of these. Flexible browsing facilities allow you to meander through the collection, listening to tunes rendered by a synthesizer, or indeed to live recordings. Almost any song you can think of is there, often in dozens of different versions.

Experimental versions of such libraries already exist. A huge volume of musical material is already on the Web in the form of MIDI files, the musical representation used by synthesizers. It is easy to locate and download hundreds of thousands of files covering a wide range of styles, from classical symphonies to current pop songs, from jazz classics to ethnic folk songs. In a very real sense, these reflect popular taste, comprising whatever people have decided to spend their time entering. You will find a score of versions of The Beatles' *Yellow Submarine* and Bach's *Air on a G-string*. All this music can be indexed by automatically identifying melodic themes and extracting text containing the title, artist, composer, and lyrics. Contentious copyright issues can be avoided by leaving all source material on its home site: what the library provides is not a repository but a catalog and means of access. And the Web is a prolific source of other musical resources, from record stores to guitar tablatures for popular tunes. Having found a tune, you can listen to samples of recordings by different artists, obtain a CD, watch a video, or buy sheet music.

The scope of digital libraries

These four examples, at different stages of development and deployment, hint at the immense range of digital libraries. From the perspective of ordinary people, libraries often seem scholarly and esoteric. But they are not necessarily so. Practical topics are of interest to practical people like Kataayi's members. Academic libraries have as their purpose research and education: high-energy physicists already base their research activity on electronic document collections. Digital libraries offer unique ways of recording, preserving, and propagating culture in multimedia form. Collections that reflect popular taste in music (or film, or TV) have already become mass-market consumer products, with delivery to teenagers on miniature, mobile, Web-capable, pocket devices.

An application that makes a sustained market for a promising but underutilized technology is often called a *killer app*. The term was coined in the mid-1980s for the Lotus spreadsheet, then the major

driving force behind the business market for IBM PCs. (VisiCalc had previously played a similar role in the success of the Apple II.) The World Wide Web is often described as the Internet's killer app. The killer app for digital libraries may well be music collections; in turn, [Section 1.4](#) shows that as far as the developing world is concerned, digital libraries themselves may be killer apps for computer technology.

1.1 Libraries and Digital Libraries

Is a digital library an institution or a piece of technology? The term *digital library*, like the word *library*, means different things to different people. Many people think of libraries as bricks and mortar, a quiet place where books are kept. To professional librarians, they are institutions that arrange for the preservation, collection, and organization of material, as well as for access to it. And a library's material is not just books: there are libraries of art, film, sound recordings, botanical specimens, and cultural objects. To researchers, libraries are networks that provide ready access to the world's recorded knowledge, wherever it is held. Today's university students of science and technology, sadly, increasingly think of libraries as the World Wide Web—that is, they misguidedly regard the Web as the ultimate library.

A digital library is not simply a “digitized library.” We hope that you are reading *How to Build a Digital Library* because you are thinking of building one. However, we do not imagine that you are the director of the New York Public Library, contemplating replacing that magnificent edifice with a computer ([Figure 1.3](#)). Nor do we want you to think, even for a moment, of burning your books at home and sitting by the fireside on winter evenings absorbed in a flat-panel computer display. (Some say that, had books been invented after computers were, they would have been hailed as a great advance.) Rather, we hope that you are inspired by a vision, perhaps something like the examples we began with, of achieving new human goals by changing the way that information is used in the world. Digital libraries are about new ways of dealing with knowledge—preserving, collecting, organizing, propagating, and accessing it—not about deconstructing existing institutions and putting them in an electronic box.

In this book, a digital library is defined as

a focused collection of digital objects, including text, video, and audio, along with methods for access and retrieval, and for selection, organization, and maintenance of the collection.

This broad interpretation of “digital objects” (not just text) is reflected in the examples above. Beyond audio and video, we want to include 3D objects, simulations, dynamic visualizations, and virtual reality. The second and third parts of the definition of a digital library deliberately accord equal weight to user (access and retrieval) and librarian (selection, organization, and maintenance). The librarian's functions are often overlooked by digital library proponents, who generally have a background in technology and approach their work from this perspective rather than from the viewpoint of library or information science.

But selection, organization, and maintenance are central to the notion of a library. If *data* is characterized as recorded facts, then *information* is the set of patterns, or expectations, that underlie the



Figure 1.3: The New York Public Library

data. You could go on to define *knowledge* as the accumulation of your set of expectations, and *wisdom* as the value attached to knowledge. Not all information is created equal, and wisdom is what librarians add to the library by making decisions about what to include in a collection—difficult decisions—and by following up with appropriate ways of organizing and maintaining the information. Indeed, it is exactly these features that distinguish digital libraries from the anarchic mess that we call the World Wide Web.

Digital libraries do tend to blur what has traditionally been a sharp distinction between user and librarian. (The collections in the examples above were not, in the main, created by professional librarians.) Nevertheless, it is important to keep in mind the distinction between the two roles. Digital library software supports users as they search and browse the collection; equally, it supports librarians as they strive to provide appropriate organizational structures and to maintain them effectively.

Digital libraries are libraries without walls. But they do need boundaries. The very notion of a collection implies a boundary: the fact that some things are in the collection means that others must lie outside it. And collections need a kind of presence, a conceptual integrity, that gives them cohesion and identity: that is where the wisdom comes in. Every collection should have a well-articulated *purpose*, which states the objectives it is intended to achieve, and a set of *principles*, which are the

directives that will guide decisions on what should be included and—equally important—what should be excluded. These decisions are difficult.

Digital collections often present an appearance that is opaque: a screen (typically, a Web page) with no indication of what, or how much, lies beyond. Is it a carefully selected treasure or a morass of worthless ephemera? Are there half a dozen documents or many millions? At least physical libraries occupy physical space, present a physical appearance, and exhibit tangible physical organization. When standing on the threshold of a large bricks-and-mortar library, you gain a sense of presence and permanence that reflects the care taken in building and maintaining the collection inside. No one could confuse it with a dung heap! Yet in the virtual world the difference is not so palpable.

We draw a clear distinction between a digital library and the World Wide Web: the Web lacks the essential features of selection and organization. We also want to distinguish a digital library from a Web site—even one that offers a focused collection of well-organized material. Existing digital libraries invariably manifest themselves in this way. But a Web site that provides a wealth of digital objects, along with appropriate methods of access and retrieval, should not necessarily be considered a “library.” Libraries are storehouses to which new material can easily be added. Many well-organized Web sites are created manually through hand-crafted hypertext linkage structures. But just as adding new acquisitions to a physical library does not involve delving into the books and rewriting parts of them, it should be possible for new material to become a first-class member of a digital library without any need for manual updating of the structures used for access and retrieval.

What establishes a new acquisition in tangible format in the collection structure of a physical library is partly where it is placed on the shelves, but more important is the information about it that is included in the library catalog. In digital libraries, we call the equivalent “cataloging” information *metadata* (data about data) and it figures prominently in the digital libraries described in this book.

1.2 The Changing Face of Libraries

Libraries are society’s repositories for knowledge: temples, if you like, of culture and wisdom. Born in an era when agriculture was humankind’s greatest preoccupation, libraries experienced a resurgence with the invention of printing in the Renaissance, and really began to flourish when the industrial revolution prompted a series of inventions that mechanized the printing process—the steam press, for example.

Libraries have been around for more than 25 centuries, although only one individual library has survived more than 5 centuries, and most are far younger. The exception is a collection of more than 2,000 engraved stone slabs, or *steles*, situated in Xi’an, an ancient walled city in central China with a long and distinguished history. The collection was established in the Song dynasty (ca. 1100 A.D.) and has been gradually expanded with new work since that time. Each stele stands 2 or 3 meters high and is engraved with a poem, story, or historical record (Figure 1.4). For example, Confucius’s works are here, as is much classic poetry, and an account of how a Christian sect spread eastward to China along the Silk Road. Chinese writing is an art form, and this library gathers together the works of many outstanding calligraphers over a period of two millennia. It also contains the heaviest books in the world.

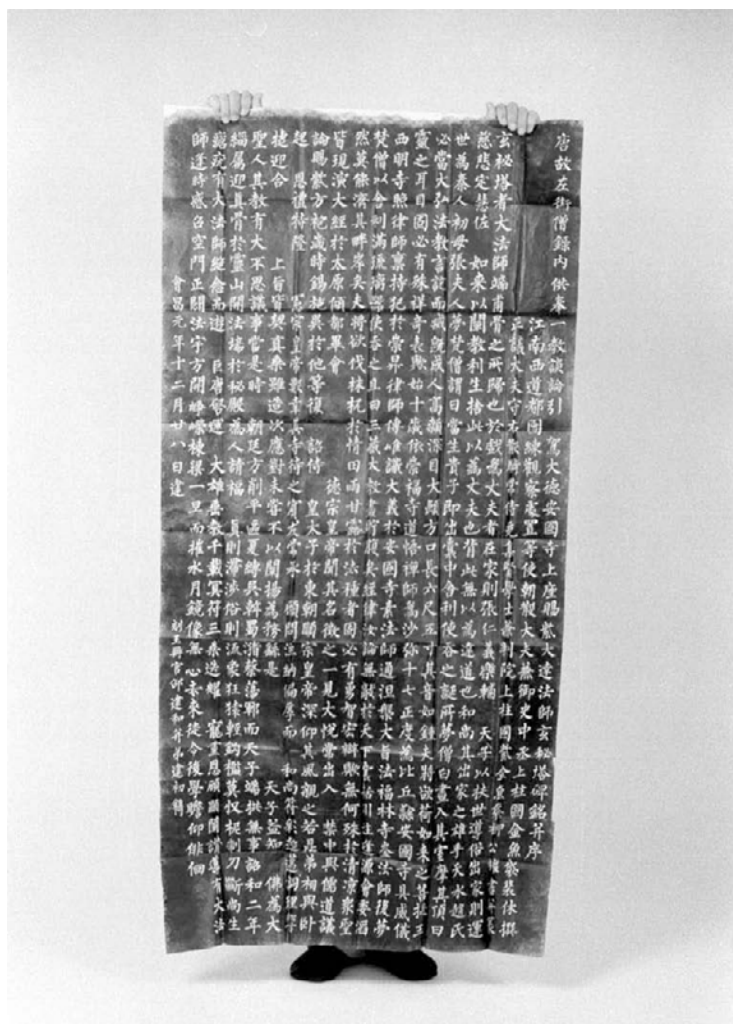


Figure 1.4: Rubbing from a stele in Xi'an

We think of the library as the epitome of a stable, solid, unchanging institution, and indeed the silent looming presence of 2,000 enormous stone slabs (often called the “forest of steles”) certainly projects a sense of permanence. But this is an exception. Over the years, libraries have continuously evolved. Originally intended for storage and preservation, libraries have refocused to place users at the center, with increased emphasis on information exchange.

Ancient libraries were useful to only the small minority of people who could read, and they were accessible within stringent limitations imposed by social conditions. Medieval monastic and university libraries held chained copies of books in public reading areas. Other copies were available for loan, although substantial security was demanded for each volume borrowed.

The public library movement took hold in the 19th century. Still, the libraries of the day had book-stacks that were closed to the public: patrons perused the catalog and chose their books, which were then handed out over the counter. In continental Europe, most libraries still operate this way. However, progressive 20th-century librarians came to realize the advantage of allowing readers to browse among the shelves and to make their own selections, and the idea of open-access libraries became widely adopted in English-speaking countries, marking the fulfillment of the principle of free access to the contents of libraries by all—the symbolic snapping of the links of the chained book.

Today we stand on the threshold of the digital library. The information revolution not only supplies the technological horsepower that drives digital libraries, but fuels an unprecedented demand for storing, organizing, and accessing information—a demand that, for better or worse, is economically driven, rather than curiosity-driven as in days gone by. If information is the currency of the knowledge economy, digital libraries will be the banks where it is invested. Indeed, Goethe once said that visiting a library was like entering the presence of great wealth that was silently paying untold dividends.

In the beginning

The fabled library of Alexandria is widely recognized as the world's first great library—although long before it, Assyrian king Assurbanipal (668–626 B.C.) established a comprehensive, well-organized collection of tens of thousands of clay tablets; and long before that, Chinese written records began, having a history extending at least as far back as the 18th century B.C. Created around 300 B.C., the Alexandrian library grew at a phenomenal rate and, according to legend, contained some 200,000 volumes within 10 years.

The work of the acquisitions department was rather more dramatic than in the libraries of today. During a famine, for example, the king refused to sell grain to the Athenians unless he received in pledge the original manuscripts of some leading authors. The manuscripts were diligently copied and the copies returned to the owners, while the originals went into the library. By far the largest single acquisition occurred when Mark Antony stole the rival library of Pergamum and gave it lock, stock, and barrel (200,000 volumes) to Cleopatra as a love token; she passed it over to Alexandria for safe-keeping. These acquisitions increased the collection to 700,000 volumes. More than 2,000 years would pass before any other library would attain this size, notwithstanding technological innovations such as the printing press.

Tragically, the Alexandrian library was destroyed. Julius Caesar set fire to the ships in the harbor of Alexandria in 47 B.C. and the fire spread to the shore. Much of the library survived this, but it was willfully laid waste (according to the Moslems) by Christians in 391 A.D. or (according to the Christians) by Moslems in 641 A.D. In the Arab conquest, Amru, the captain of Caliph Omar's army, would apparently have been willing to spare the library, but the fanatical Omar is said to have disposed of the problem of information explosion with the immortal words, "If these writings of the Greeks agree with the Koran, they are useless and need not be preserved; if they disagree, they are pernicious and ought to be destroyed."

The information explosion

Moving ahead a thousand years, let us peek at what was happening in a library at a major university near the center of European civilization a century or two after Gutenberg's introduction of the movable-type printing press around 1450. (The printing press was invented in China much earlier, around five centuries before Gutenberg.) Trinity College, Dublin, one of the oldest universities in Western Europe, was founded in 1592 by Queen Elizabeth I. In 1600 the library contained a meager collection of 30 printed books and 10 handwritten manuscripts. This grew rapidly, by several thousand, when two Trinity College Fellows mounted a shopping expedition to England, and by a further 10,000 when the library received the personal collection of Archbishop Ussher, a renowned Irish man of letters, on his death in 1661.

At the time, however, even this collection was dwarfed by that of Duke August of Wolfenbüttel, Germany, whose collection had reached 135,000 imprints by his death in 1666 and was the largest contemporary library in Europe, acclaimed as the eighth wonder of the world. These imprints were purchased in quires (i.e., unbound) and shipped to the duke in barrels. The duke had them bound in 31,000 volumes with pale parchment bindings that you can still see today. Incidentally, this collection inspired Casanova, after spending seven days visiting the library in 1764, to declare that "I have sometimes thought that the life of those in heaven may be somewhat similar to [this visit]." Coming from the world's most renowned lover, this is high praise indeed.

Returning to Ireland, another great event in the development of Trinity College occurred in 1801, when an act was passed by the British Parliament decreeing that a copy of every book printed in the British Isles was to be donated to the Trinity College Library. This privilege extends to this day and is shared by five other libraries—the British Library, the University Libraries of Oxford and Cambridge, and the National Libraries of Scotland and Wales. This "legal deposit" law had a much earlier precedent in France, where King François I decreed in 1537 that a copy of every book published was to be placed in the Bibliothèque du Roi (long since incorporated into the French National Library). Similarly, the Library of Congress receives copies of all books published in the United States. But we digress.

There were no journals in Archbishop Ussher's collection. The first scholarly journals appeared just after his death: the *Journal des Sçavans* began in January 1665 in France, and the *Philosophical Transactions of the Royal Society* began in March 1665 in England. These two have grown, hydra-like, into hundreds of thousands of scientific journals today, although some are now being threatened with replacement by electronic archives.

In the 18th century, printing really took hold. For example, more than 30,000 titles were published in France during a 60-year period in the mid-1700s. Some 300 years after Gutenberg developed the printing press in order to make the Bible more widely available, the press became the vehicle for disseminating the European Enlightenment—an emancipation of human thinking from the weight of authority of the church.

In the United States, President John Adams created a reference library for Congress when Washington became the new national capital in 1800. He began by providing \$5,000 "for the pur-

chase of such books as may be necessary for the use of Congress, and for putting up a suitable apartment for containing them therein.” The first books were ordered from England and shipped across the Atlantic in 11 hair trunks and a map case. The library was housed in the new Capitol until August 1814, when—in a miniature replay of Julius Caesar’s exploits in Alexandria—British troops invaded Washington and burned the building. Some 3,000 volumes were lost in that fire. Another fire destroyed two-thirds of the rebuilt collection in 1851. Unlike Alexandria, however, the Library of Congress has regrown—indeed its rotunda is a copy of the one built in Wolfenbüttel two centuries earlier. Today the Library of Congress contains approximately 22 million volumes.

The information explosion began to hit home in Ireland in the middle of the 19th century. In 1835, work started on the production of a printed catalog for the Trinity College Library (Figure 1.5), but by 1851 only the first volume, covering letters A and B, had been completed. The catalog was finally finished in 1887, but only by restricting the books that appeared in it to those published up to the end of 1872. Other libraries, however, were wrestling with much larger volumes of information. By the turn of the century, the Trinity College Library had around a quarter of a million books, while the Library of Congress had nearly three times that number. Both were dwarfed by the British Museum (now part of the British Library), which at the time had nearly 2 million books, and the French National Library in Paris, which had over 2.5 million.

The Alexandrian principle

In an early statement of library policy, an Alexandrian librarian was reported as being “anxious to collect, if he could, all the books in the inhabited world, and, if he heard of, or saw, any book worthy of study, he would buy it”—and two millennia later this was formulated as a self-evident principle of librarianship: *it is a librarian’s duty to increase the stock of his library*. When asked how large a library should be, librarians answered, “Bigger. And with provision for further expansion.”

Only recently has the Alexandrian principle begun to be questioned. In 1974, following a 10-year building boom then unprecedented in library history, the *Encyclopedia Britannica* noted that “even the largest national libraries are ... doubling in size every 16 to 20 years” and gently warned that “such an increase can hardly be supported indefinitely.” And the struggle continues. In the 20th century’s last decade the national libraries of the United Kingdom, France, Germany, and Denmark all opened new buildings. The ones in London and Paris are monumental in scale. Standing on the bank of the Seine River, the Bibliothèque Nationale de France consists of four huge towers designed to look like open books, surrounding a sunken garden plaza (Figure 1.6). The reading rooms occupy two levels around the garden, with bookshelves encircling them on the outer side.

Sustained exponential growth cannot continue. A collection of essays published in 1976 entitled *Farewell to Alexandria: Solutions to Space, Growth, and Performance Problems of Libraries* dwells on the problems that arise when growth must end. Sheer limitation of space has forced librarians to rethink their principles. Now they talk about “aggressive weeding” and “culling,” “no-growth libraries,” the “optimum size for collections,” and some even ask, “Could smaller be better?” In a striking example of aggressive weeding, the library world was rocked in 1996 by allegations that the San Francisco Public Library had surreptitiously dumped 200,000 books, or 20 percent of its collection,

- Resolutie van de staten generael der Vereenighde Nederlanden, dienende tot antwoord op de memorie by de ambassadeurs van sijne majesteyt van Vranckrijck.
's Graven-hage, 1678. 4°. Fag. H. 2. 80. N°. 20.
Fag. H. 2. 85. N°. 17. Fag. H. 3. 42. N°. 4.
- Tractaet van vrede gemaect tot Nimwegen op den 10 Augusty, 1678, tusschen de ambassadeurs van [LOUIS XIV.] ende de ambassadeurs vande staten generael der Vereenighde Nederlanden.
Fag. H. 2. 85. N°. 21.
- Nederlantsche absolutie op de Fransche belydenis.
Amsterdam, 1684. 4°. Fag. H. 2. 50. N°. 22.
- Redenen dienende om aan te wijzen dat haar ho. mog. [niet] kunnen verhindert werden een vredige afkomst te maken op de conditien by memorien van den grave d' Avaux van de 5 en 7 Juny, 1684, aangeboden.
[s. l.] 1684. 4°. Fag. H. 2. 86. N°. 3.
Fag. H. 2. 96. N°. 8. Fag. H. 3. 44. N°. 52.
- Redenen om aan te wijzen dat de bewuste werving van 16000 man niet kan gesustineert werden te zullen hebben kunnen strekken tot het bevorderen van een accommodement tusschen Vrankrijk en Spaigne.
[s. l.] 1684. 4°. Fag. H. 2. 86. N°. 4.
Fag. H. 2. 96. N°. 2.
- D' oude mode van den nieuwen staat van oorlogh.
[s. l. 1684]. 4°. Fag. H. 2. 86. N°. 12.
Fag. H. 2. 96. N°. 3.
- Aenmerkingen over de althans swevende verschillen onder de leden van den staat van ons vaderlant.
[s. l.] 1684. 4°. Fag. H. 2. 92. N°. 1.
Fag. H. 2. 98. N°. 16. Fag. H. 3. 1. N°. 18.
- Missive van de staten generael der Vereenighde Nederlanden, . . . 14 Maert, 1684.
's Graven-hage, 1684. 4°. Fag. H. 2. 92. N°. 10.
- Missive van de staaten generael der Vereenigde Nederlanden, . . . 11 July, 1684.
[sin. tit. 1684]. 4°. Fag. H. 2. 96. N°. 13.
Fag. H. 3. 44. N°. 69.
- Resolutie vande staten generael der Vereenighde Nederlanden, . . . 2 Maert, 1684.
's Gravenhage, 1684. 4°. Fag. H. 2. 92. N°. 11.
Fag. H. 3. 44. N°. 9.
- Extract uyt de resolutien van de staten generael, . . . 31 Maert, 1684.
[s. l.] 1684. 4°. Fag. H. 2. 92. N°. 13.
Fag. H. 2. 96. N°. 25. Fag. H. 3. 44. N°. 11.
Fag. H. 3. 44. N°. 15.
- Antwoort van de staten generael der Vereenighde Nederlanden op de propositie van wegen sijne churf. doorl. van Ceulen, Maert 23, 1684, gedaen.
's Gravenhage, 1684. 4°. Fag. H. 2. 92. N°. 12.

Figure 1.5: A page of the original Trinity College Library catalog



Figure 1.6: The Bibliothèque Nationale de France. Dominique Perrault, architect; © Alain Goustard, photographer

into landfills, because its new building, although lavishly praised by architecture critics, was too small for all the books.

The notion of focused collections is replacing the Alexandrian model of an ideal library that is vast and eternally growing. The notion of service to library users is replacing the idea of a library as a storehouse of all the world's knowledge. These movements will surely be reinforced by the experience of the World Wide Web, which amply illustrates the anarchy and chaos that inevitably result from sustained exponential growth. The events of the last quarter century have even shaken librarians' confidence in the continued existence of the traditional library. Defensive tracts with titles like *Future Libraries: Dreams, Madness and Reality* deride "technolust" and the empty promises of the technophiles.

Early technodreams

Let us, for a moment at least, consider the technophiles. Around 1936, science fiction writer H. G. Wells promoted the concept of a "world brain" based on a permanent world encyclopedia that

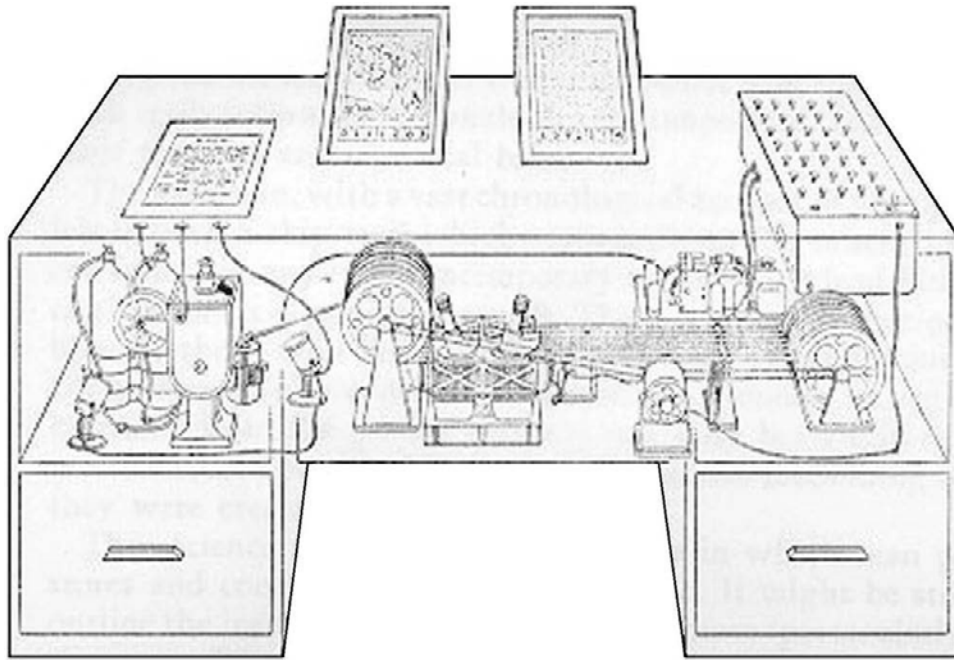


Figure 1.7: Artist's conception of the Memex, Bush's automated library. Courtesy of Mary and Alfred Crimi Estate

"would be the mental background of every intelligent [person] in the world. It would be alive and growing and changing continually under revision, extension and replacement from the original thinkers in the world everywhere," and he added sardonically that "even journalists would deign to use it."

In 1945, Vannevar Bush, the highest-ranking scientific advisor in the U.S. war effort, urged us to "consider a future device for individual use, which is a sort of mechanized private file and library ... a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility" (Figure 1.7).

In 1960 J. C. R. Licklider, head of the U.S. Department of Defense's Information Processing Techniques Office, envisioned that human brains and computing machines would be tightly coupled together and supported by a "network of 'thinking centers' that will incorporate the functions of present-day libraries together with anticipated advances in information storage and retrieval."

Toward the end of the 20th century we became accustomed to hearing similar pronouncements from the U.S. Presidential Office, rising above the road noise of the information superhighway.

The library catalog

Wells, Bush, Licklider, and other visionary thinkers were advocating something very close to what we might now call a *virtual library*. To paraphrase the dictionary definition, something is *virtual* if it

exists in essence or effect though not in actual fact, form, or name. A virtual library is a library for all practical purposes, but a library without walls—or physical books.

In truth, a virtual representation of books has been at the core of libraries right from the beginning: the catalog. Even before Alexandria, libraries were arranged by subject and had catalogs that gave the title of each work, the number of lines, the contents, and the opening words. In 240 B.C. an index was produced to provide access to the books in the Alexandrian library that was a classified subject catalog, a bibliography, and a biographical dictionary all in one.

A library catalog is a complete model that represents, in a predictable manner, the universe of books in the library. Catalogs provide a summary of library contents. Today we call this *metadata*. And it is highly valuable in its own right. As a late 19th-century librarian wrote, “Librarians classify and catalog the records of ascertained knowledge, the literature of the whole past. In this busy generation, the librarian makes time for his fellow mortals by saving it. And this function of organizing, of indexing, of time-saving and thought-saving, is associated peculiarly with the librarian of the 19th century.”

Other essential aids to information-seeking in libraries are published bibliographies and indexes. Like catalogs, these are virtual representations—metadata—and they provide the traditional means of gaining access to journal articles, government documents, microfiche and microfilm, and special collections.

A possible interpretation of “digital library” that is quite different from the concept developed in this book is a system designed to automate traditional library functions by helping librarians to manage physical books. Online public access catalogs (OPACs) are standard in libraries today. And many other functions are automated: acquisitions, loans, recalls, interlibrary services, and library planning. However, this kind of library automation system is not closely related to the digital libraries we describe in the examples that open this chapter.

The changing nature of books

The technophile visionaries whose dreams we shared above were not talking about a virtual library in the sense of an automated catalog. They wanted the full text of all documents in the library to be automated, not just a metadata surrogate. They took it for granted that books are adequately represented by the information they contain: the physical object is of no consequence.

The information in library catalogs and bibliographies can be divided into two kinds: the first having reference to the contents of books; the second treating their external character and the history of particular copies. Intellectually, only the abstract content of a book—the information contained therein—seems important. But the strong visceral element of books cannot be neglected and is often cited as a reason why book collections will never become “virtual.”

Portable writing began with the clay tablet, invented by the Sumerians in 3500 B.C. A well-known example is the Phaistos Disk from the ancient Minoan civilization (Figure 1.8a). A thousand years later, Egyptians began writing on scrolls made of papyrus plants (a practice that was adopted by various cultures over the centuries, see Figure 1.8b). In other places, animal skins were used for

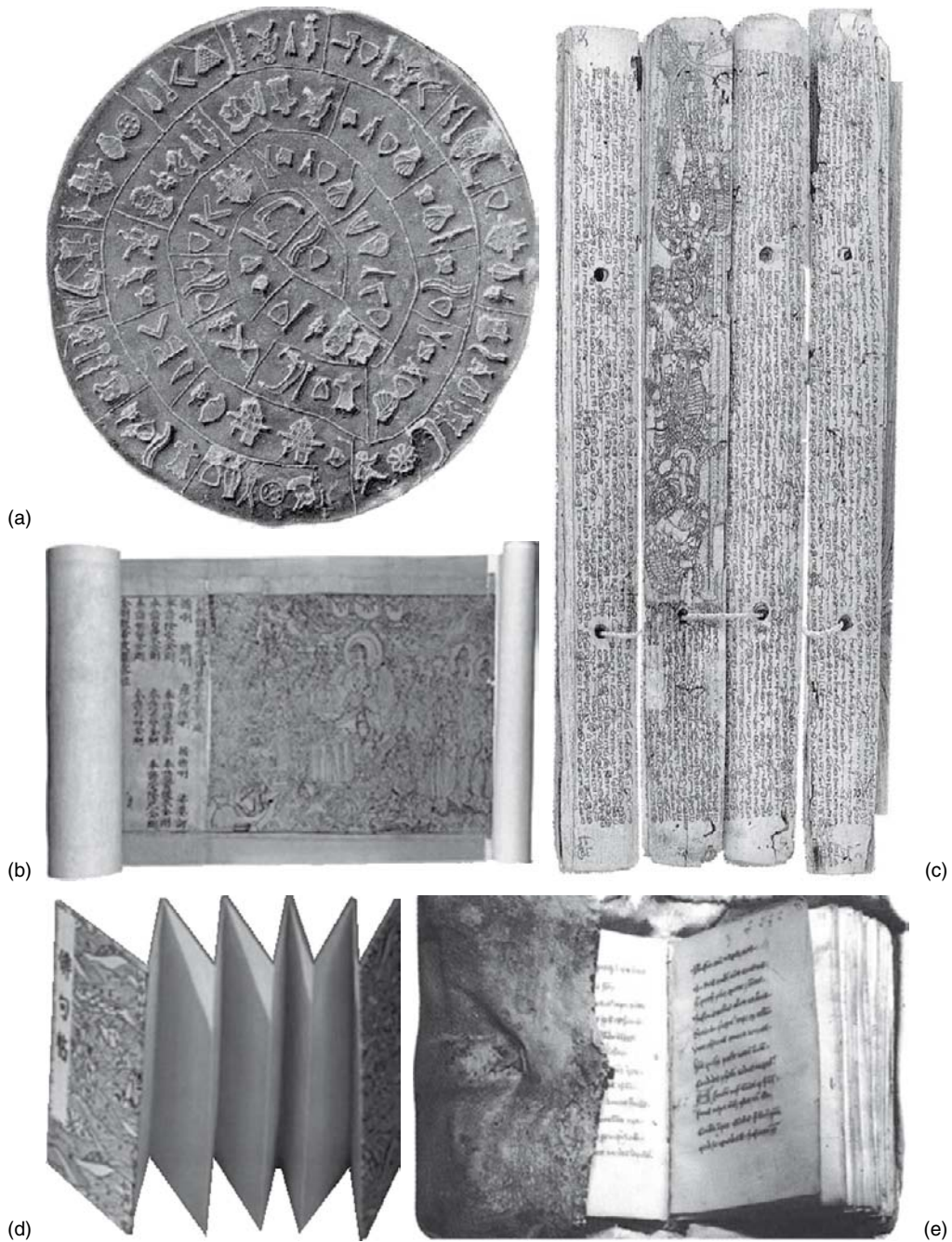


Figure 1.8: Ancient devices for portable writing: (a) clay tablet; (b) scroll; (c) palm leaves (from Thanjavur Maharaja Serfoji's Sarasvat Mahal Library, Thanjavur, Tamil Nadu, 1995); (d) concertina; (e) codex

parchment or vellum; elsewhere, bamboo, silk, and even palm leaves served as writing surfaces (Figure 1.8c). But lengthy scrolls become unwieldy—some, like the Great Harris Papyrus composed by King Ramses IV, were as long as 135 feet. A simple solution was to fold the scroll into a concertina (Figure 1.8d). Folded scrolls are more compact, making them easier to handle while reading and to store afterwards. Information can be readily accessed: when seeking particular passages, readers need not unroll and reroll the document. An unfolded concertina book is essentially a scroll, providing backward compatibility.

The book form or “codex” superseded scrolls and concertinas (Figure 1.8e). Around 200 B.C., Greeks and Romans began to write on wax tablets backed with wood, sometimes connected with cords (resembling a ring binder). Later the Romans substituted the more durable vellum for the wood panels.

Because of its numerous virtues, the codex quickly became the preferred format for all literary works. Writers can use both sides; readers can access pages randomly. Codices are easy to read, store, carry, and search. The content is well protected, and thicker pigments can be used for decoration and illustration. The only real disadvantage is that readers can’t view more than two pages at once. Books quickly became the standard document format, although concertinas are still in use today for brochures and maps.

Surprisingly, the evolution of computer output paralleled the book format’s development. Early printers used paper rolls (scrolls); later, line printers used fanfold paper (concertinas). The latter innovation was spurred by accelerated print mechanisms, but old-timers still recall the advantages of fanfolds over unruly rolls of paper. Indeed, paper rolls were eventually perforated so that they could be folded, boxed, and perused more easily. Today we print on pages and bind them into books. The parallel isn’t confined to print technology. Early computer display monitors scrolled, as do Web pages, which originated in the early 1990s. Adobe Reader and Microsoft Word Print Preview provide a paginated concertina-like view: readers scroll through the document page by page, using a scrollbar or Page Up and Page Down keys.

Bibliophiles love books as much for the statements they make as objects as for the statements they contain as text. Indeed, early books were works of art. The steles in Xi’an are a monumental example, studied as much for their calligraphic beauty as for the philosophy, poetry, and history they record, a priceless, permanent record of earlier civilizations. The *Book of Kells* in Ireland, produced by Irish monks at the scriptorium of Iona about 1,200 years ago, is one of the masterpieces of Western art. Figure 1.9 shows part of a page and illustrates the extraordinary array of pictures, interlaced shapes, and ornamental details. Giraldus Cambrensis, a 13th-century scholar, fancifully wrote that “you might believe it was the work of an angel rather than a human being.”

Beautiful books have always been highly prized for their splendid illustrations, for colored impressions, for heavily decorated illuminated letters, for being printed on uncommon paper or uncommon materials, for their unusual bindings, and for their rarity and historic significance. In the castle library of Königsburg there are 20 books bound in silver, richly adorned with large and beautifully engraved gold plates. Whimsical bindings abound: a London bookseller had Fox’s *History of King James II* bound in fox skin.

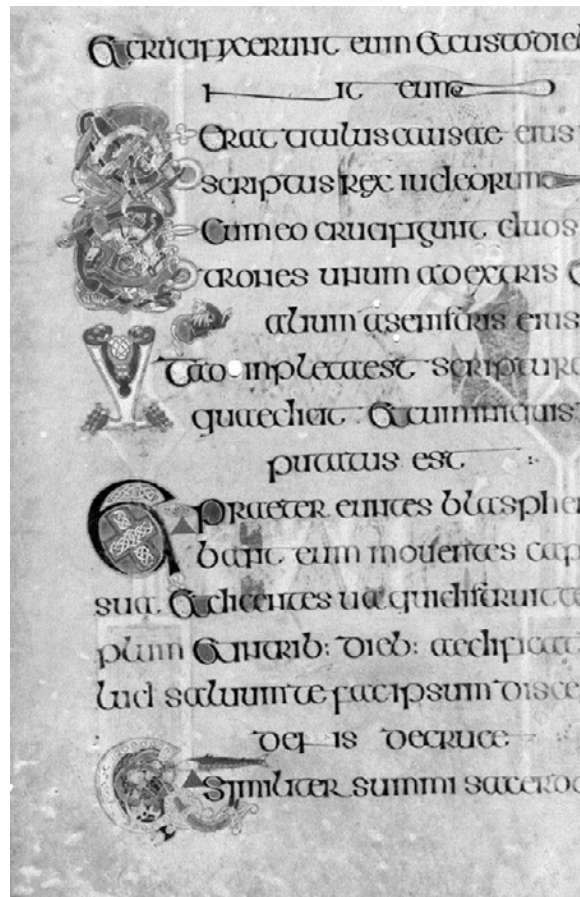


Figure 1.9: Part of a page from the *Book of Kells*

Catalogs and bibliographies are metadata: virtual information about books. In the kind of virtual library sketched in the early technodreams above, the very concept of the book as an individual physical entity seems to be at risk. However, technology has advanced to the point where it need not be: surrogates can substitute for physical works. A picture of the cover may be displayed as a “tangible”—or at least memorable—emblem of the physical book itself. Users can browse the collection using graphical techniques of virtual reality. Maybe they will even be able to caress the virtual cover, smell the virtual pages. But although it is unlikely that readers will love simulated books the way that bibliophiles love real ones, what really matters in libraries is information, even knowledge. Ask the Kataayi community.

1.3 Searching for Sophocles

Pergamos, a digital library at the University of Athens, is representative of many interesting resources that have appeared on the Web over the last decade. It contains four collections spanning

historical archives, folklore, the theater, and sheet music; and more collections are being developed. The name is an alternate spelling for Pergamum, which, as already mentioned, was the site of a significant library in its day. Here we use Pergamos to exemplify features of a real-world digital library.

The system provides an interface in Greek, English, and French. [Figure 1.10a](#) shows the home page (in English), which lists the collections available. For each collection a fitting image is shown, along with a summary of what is provided: a description of the resource, the time span it covers, who owns the material, their contact information, and access rights for the user. Some collections, such as the Historical Archive collection, include subcollections: printed documents and photography from the university archives, the Senate Secretariat, and five of the university's faculties. These can be accessed individually or collectively.

Suppose a user is interested in finding out which of the seven surviving plays by Sophocles have been performed at the university over the years. He brings up the library's search page shown in [Figure 1.10b](#) and selects the Theatrical collection from a nested list of possibilities. Next he enters *sophocles* in the search box. The result of the query, shown in [Figure 1.10c](#), is disappointing: only one matching document is found. This is an unexpectedly sparse result for a writer of such prominence, and our user mentally runs a few checks. Was there a typing mistake in the query? Could the uncapitalized initial letter have prevented matching? In both cases the answer is No: the name is correctly spelled, and in this digital library search is case-insensitive. Why then was only one match returned?

It is hard not to notice the size of a physical library. But establishing equivalent cues in the digital realm is tricky. Perhaps, wonders our user, the Theatrical collection is rather small after all? To get a handle on its scope and coverage, he clicks the Navigate button in the interface's main navigation bar, which appears on each page as a row of words beneath the masthead. A few clicks bring up [Figure 1.10d](#), which lists all items in the collection alphabetically by title. There are 633 items in all, and items 1–20 are presented in summary form: title of the show, thumbnail of the program, table displaying author, director, group, theater, and period. Not all items in the list have all these components. The page also contains navigation links: next page and direct entry to a particular page. The sort order can be changed from ascending to descending, and the sort field can also be chosen.

Clicking through a page or two, our user quickly realizes that author names, titles, and so forth are predominately entered in the Greek alphabet. When the user repeats the search in Greek, *Σοφοκλής*, twelve matches are returned ([Figure 1.10e](#)), which accords better with the user's expectations. He scrolls through the returned matches ([Figure 1.10f](#)) and selects the play *Αντιγόνη* (Antigone). [Figure 1.10g](#) shows the digitized version of the actual program produced for this show. The thumbnail shown on the left is quick to load, and the main area of the page is taken up with a larger version. A zoom control allows the user to alter the display resolution in order to see different levels of detail. There is a link to the second page (and, where relevant, subsequent pages) of the program. Digital libraries like this can be accessed by anyone, from any corner of the world. This makes it impossible to predict the uses that people will want to make of the material. For instance, research students might study patterns in scribbled annotations left on the paper documents, which would normally be viewed as ephemera, in order to help investigate a hypothesis they have developed. Most visitors are

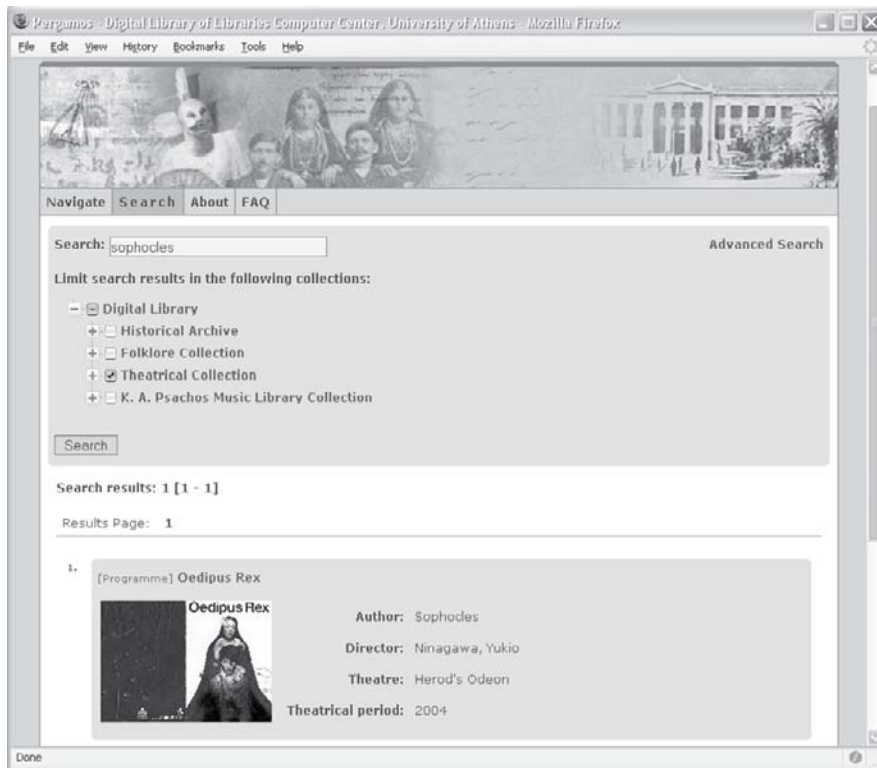


(a)



(b)

Figure 1.10: The Pergamos Digital Library: (a) home page (in English); (b) searching the Theatrical Collection for *sophocles*;



(c)

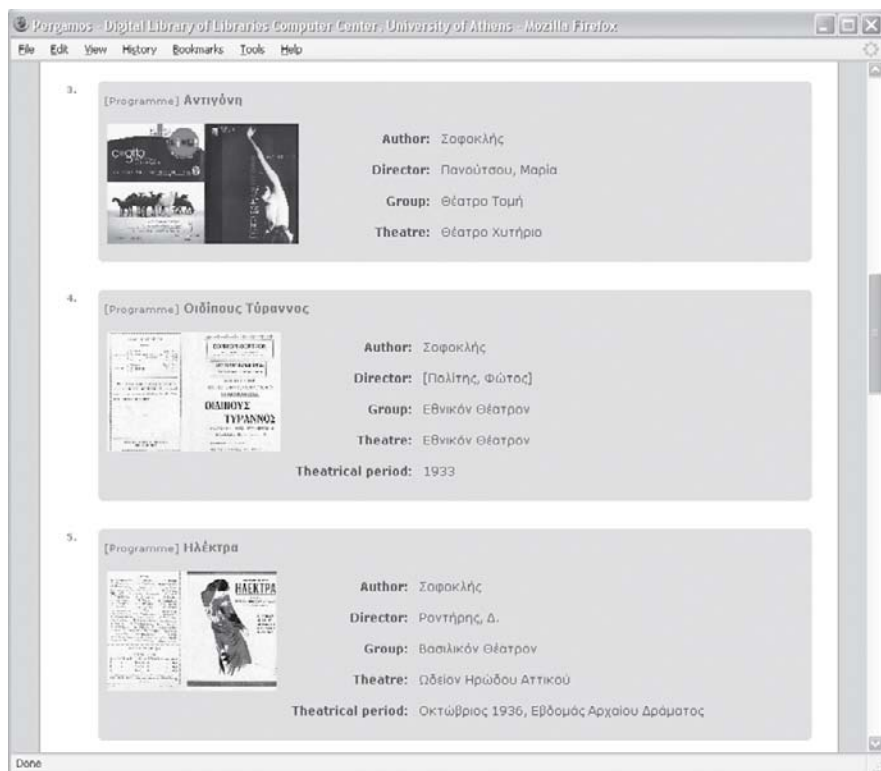


(d)

Figure 1.10, cont'd: (c) search result; (d) browsing titles alphabetically;



(e)



(f)

Figure 1.10, cont'd: (e) searching for *Σοφοκλής*; (f) search result;

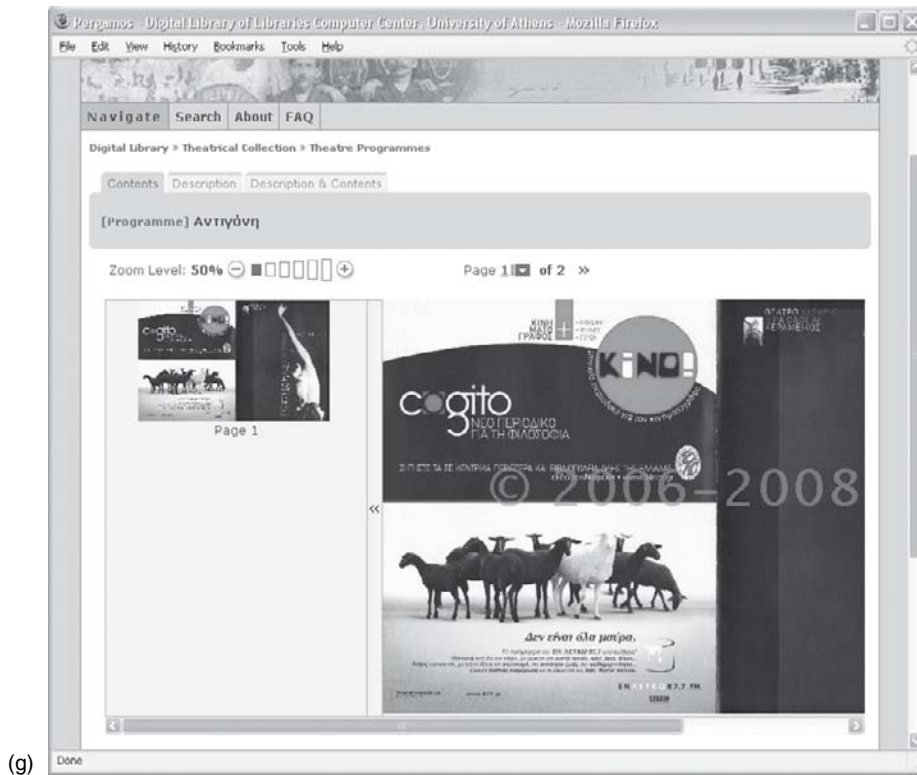


Figure 1.10, cont'd: (g) viewing the item *Αντιγόνη* (*Antigone*)

more interested in theatrical aspects. They might feel that the highest levels of magnification are overkill, but to researchers seeking faded pencil annotations they are a godsend.

Countless digital libraries on the Web adopt the basic form presented in this walk-through. They offer searching and browsing operations to locate items of interest, and ultimately show digital representations of each document, along with ancillary information (metadata). In this example the documents are digitized images (theater programs), to which access is provided through fields of descriptive text, such as title, author, and date. In the case of text-based documents like word-processed reports or Web pages, the full text can be searched as well.

1.4 Digital Libraries in Developing Countries

It sometimes happens that technological advances in developing countries leapfrog those in developed ones. This occurs because established infrastructure, a strong and necessarily conservative force, is absent. For example, in developing countries alternative energy sources, such as solar energy, are widely used in place of traditional power generation and distribution, and these countries have experienced far higher levels of mobile phone growth than developed countries have. Digital

libraries provide another example, compensating for the failure of traditional distribution mechanisms to address local requirements and to deliver information where and when it is needed.

Many current technology trends are not benefiting developing countries—indeed, some bring serious negative consequences. Just as industrialization and globalization have increased the gulf between haves and have-nots, so information and communications technology is creating a chasm between “knows” and “know-nots.” By and large, developing countries are not participating in the information revolution, although knowledge is critical for development. The knowledge gap between rich and poor is widening.

In the developing world, digital libraries provide perhaps the first really compelling *raison d’être* for computing technology. Priorities in these countries include health, food, hygiene, sanitation, and safe drinking water. Though computers are not a priority, simple, reliable access to targeted information meeting these basic needs certainly is. Digital libraries give system developers a golden opportunity to help reverse the negative impact of information technology on developing countries.

Disseminating humanitarian information

Traditional publishing and distribution mechanisms have tragically failed the developing world. Take medicine, a field of great importance in this context. Whereas a U.S. medical library subscribes to about 5,000 journals, the Nairobi University Medical School Library, long regarded as a flagship center in East Africa, received just 20 journals in 1998 (compared with 300 a decade before). In Brazzaville, Congo, the university has only 40 medical books and a dozen journals, all published before 1993, and the library in a large district hospital consists of a single bookshelf filled mostly with novels.

By decoupling production and distribution costs from intellectual property charges, digital libraries offer a lifeline. A wealth of essential humanitarian material is produced by international organizations such as the United Nations, as well as national units like the U.S. Peace Corps. Being produced by internationally oriented, nonprofit organizations, funded by all people on the planet, this information is—at least in principle—in the public domain: it could be made freely available in the form of networked digital libraries. While those 5,000 medical journals cannot be distributed for free because they are produced by commercial publishers, this problem does not arise in many areas of physics, as we have seen. The world is changing, and the rate of change will accelerate.

Disaster relief

Natural disasters like earthquakes and hurricanes and human-made ones like terrorist attacks and nuclear accidents demand an immediate and informed response. Disaster relief situations are complex and are addressed by a broad range of players in a variety of organizations acting in parallel. They present an overwhelming need for information: information that is tailored for the problem at hand, organized so that it can be accessed effectively, and distributed even in the absence of an effective network infrastructure. The response to a crisis is characterized by the generation of large amounts of

unstructured multimedia data that must be acquired, processed, organized, and disseminated sufficiently rapidly to be of use to crisis responders.

Digital library technology allows the rapid creation of organized collections of information, graced with comprehensive searching and browsing capabilities. Intelligence specific to the nature of a disaster, the geographical region, and the logistic resources available for the relief effort can all be gathered into a built-to-order digital library collection that combines targeted knowledge with general information about medicine and sanitation.

Preserving indigenous culture

Libraries and their close relatives, museums, have always been involved in preserving culture. These institutions collect literature and artifacts and use them to disseminate knowledge and understanding of different times and cultures. Digital libraries, however, open up the possibility of flexible and coherent multimedia collections that are both fully searchable and browsable in multiple dimensions—and permit more active participation by indigenous people in preserving and disseminating their own culture, as is illustrated by the example of the Zia Pueblo. The principal participants here are by definition the indigenous people themselves: the technological world assumes the role of catalyst, midwife, and consumer—once indigenous culture has been recorded, it will find a fascinated, sympathetic, and perhaps influential audience elsewhere.

Information about indigenous culture takes many guises: oral history, in the form of narration and interviews; artifacts, in the form of images and descriptions; songs, in the form of audio recordings, music transcriptions, and lyrics; and dances and ceremonies in the form of video, audio, written synopses, and interpretations. Multimedia digital libraries allow such information to be integrated, recorded, browsed, and searched, all within a uniform user interface.

Because language is the vehicle of thought, communication, and cultural identity, a crucial feature of digital libraries for culture preservation is the ability to work in local languages. This strengthens individual cultures, promotes diversity, and reduces the dominance of English in the global information infrastructure.

Locally produced information

In digital library applications for culture preservation, the relevant information is, of necessity, readily available locally. But there are countless other scenarios that involve creating and distributing locally produced information collections. At first glance one might think that the Internet includes such a wealth of content that surely there must be something of benefit to everyone. However, this ignores not only the problem of language—most information is available only in major languages like English—but also the importance of the kind of content that only local communities can generate.

Teachers prepare educational material that addresses specific community problems, and they adapt published material to employ local examples. Indigenous people have medicinal knowledge based on

local plants or long-acquired knowledge of the cultivation and protection of local species. Such knowledge is vital: more than half of the world's most frequently prescribed drugs are derived from plants or synthetic copies of plant chemicals, and this trend is growing.

Local groups assemble information collections that describe and reflect neighborhood conditions, providing new material for sociocultural studies, fostering cultural exchange while retaining diversity, and increasing international understanding. Web sites for community and social development might include information on health problems endemic to a particular African community, or information on commodity prices for a particular good traded in Brazilian markets, or examples of curricular projects suitable for use in Indian schools.

The development of content that addresses the specific needs of a particular community stimulates the demand for information technology among that community. Getting learners to produce their own content is one of the best ways to exploit information technology in learning situations. It not only improves the learning experience, but also creates material that benefits the community. Teachers and students can work together to create their own content that has value for the community, and for the nation as well.

Effective human development blossoms from empowerment. As the Chinese proverb says, "Give a man a fish and he will eat for a day; teach him to fish and he will eat for the rest of his days." Dissemination of information that originates in the developed world is useful to developing countries, as Kataayi members will testify, but a more effective strategy for sustained long-term human development is to foster the capability for creating information collections, rather than the collections themselves. This allows developing countries to participate actively in the information society, rather than observing it from outside. It stimulates the creation of new industry, and it helps ensure that intellectual property remains where it belongs, in the hands of those who produce it.

The technological infrastructure

Computers are not so hard to come by in developing countries as one might think. Their extraordinarily rapid rate of obsolescence, coupled with the developed world's voracious appetite for the latest and greatest, makes low-end machines essentially free: instead of clogging landfill sites many (although certainly not enough) find their way to developing countries. A 1998 World Bank survey of developing countries found 3 to 30 PCs per 1,000 people, depending on the poverty level. Using an estimated growth rate of 20 percent per year, we conclude that at the turn of the millennium there were 50 million PCs in developing countries, serving a population of four billion.

A more serious obstacle is that network access varies widely around the globe. Whereas in 1998 more than a quarter of the U.S. population was surfing the Internet, the figure for Latin America and the Caribbean was 0.8 percent, for Sub-Saharan Africa 0.1 percent, and for South Asia 0.04 percent. Schools and hospitals in developing countries are poorly connected. Even in relatively well-off South Africa, many hospitals and 75 percent of schools have no telephone line. In African universities, up to 1,000 people can depend on just one workstation. The Internet is failing the developing world. While global satellite communication networks may eventually bring relief, this takes time and money.

Because of the difficulty of network access, the structure and organization of digital libraries should be separated from their distribution media. Physical distribution of information on recordable devices can provide an attractive alternative to networks. Compact disk read-only memory, CD-ROM, is a practical format for areas with little Internet access. The 650-MB capacity of a CD-ROM can hold a useful volume of information, such as the 1,200 fully illustrated and fully indexed books in the Humanity Development Library. Most of the space in a collection like this is consumed by pictures: several times as many books could be included if they were not so lavishly illustrated. CDs are giving way to digital versatile disks, DVDs, which can hold from 5 to 20 GB of data. A year's supply of the 5,000 medical journals mentioned above could fit, fully indexed, on a single DVD. And save lives.

1.5 The Pen Is Mighty: Wield It Wisely

Collecting information and making it widely available to others has far-ranging social implications, and those who build digital libraries must act responsibly by making themselves aware of the legal and ethical issues that surround their particular application. Copyright is the place to begin.

Digital libraries are far more accessible than physical ones. And this brings its own problems: access to the information in digital libraries is generally less controlled than it is in physical collections. Putting information into a digital library has the potential to make it immediately available to a nearly unlimited audience.

This is great news. For the user, information becomes available wherever you are, worldwide. For the author, the potential audience is greater than ever before. And for publishers, new markets open up that transcend geographical limitations. On the other hand, authors and publishers worry that fewer copies of a work will be sold if networked digital libraries enable worldwide access to an electronic copy of it. Their nightmare is that the number of copies sold could be as few as *one*. How many books will be published online if the entire market can be extinguished by the sale of one electronic copy to a public library?

Copyright law

In copyright law, possession of a copy of a document certainly does not constitute ownership. Although there may be many copies, each document has only one copyright owner. This applies not just to physical copies of books, but to computer files, too, whether they have been digitized from a physical work or created electronically in the first place—"born digital." When you buy a copy of a document, you can resell it, but you certainly do not buy the right to make further copies and redistribute them. That right rests with the copyright owner.

Copyright subsists in the work, rather than in any particular embodiment of it. A work is an intangible intellectual object, of which a document is a physical manifestation. Lawyers use the word *subsists*, which in English means to remain or to continue in existence, because copyright has no separate existence without the work. Copyright protects the way ideas are expressed, not the ideas themselves. Two works that express the same idea in different ways are independent in copyright law.

Who owns a particular work? The creator is the initial copyright owner, unless the work is made for hire. If the work is created by an employee within the scope of her employment, or under a specific contract that explicitly designates it as being made for hire, the employer or contracting organization owns the copyright. Any copyright owner can transfer, or “assign,” copyright to another party through a written contract. Typically, an author sells copyright to a publisher (or grants the publisher an exclusive license) and the publisher reproduces, markets, and sells the work.

The copyright owner has the exclusive right to do certain things with the work: thus copyright is sometimes referred to as a “bundle” of rights. In general, there are four rights, although details vary from country to country. The *reproduction right* allows the owner to reproduce the work freely. The *distribution right* allows the owner to distribute it, but this is a one-time right: once a copy has been distributed, the copyright owner has no right to control its subsequent distribution. For example, if you buy a book, you can do whatever you want with your copy, including reselling it. The *public lending right* compensates authors for public lending of their work—although an exception is granted for not-for-profit and educational use, which do not require the copyright holder’s consent. The remaining rights, called *other rights*, include permitting or refusing public performance of the work, and making derivative works like plays or movies.

Copyright law is complex, arcane, and varies from one country to another. The British Parliament adopted the first copyright act in 1710; the U.S. Congress followed suit in 1790. Although copyright is national law, most countries today have signed the Berne Convention of 1886, which lays down a basic international framework. According to the Convention, copyright subsists *without formality*, which means that (unlike patents) it’s not dependent on registering a work with the authorities or depositing a copy in a national library. It applies regardless of whether the document bears the international copyright symbol ©. You automatically have copyright over works you create (unless they are made for hire). Some countries—including the United States—maintain a copyright registry even though they have signed the Berne Convention, which makes it easier for a copyright holder to take legal action against infringers. Nevertheless, copyright still subsists, even if the work has not been registered.

The Berne Convention decrees that it is always acceptable to make limited quotations from protected works, with acknowledgment and provided it is done fairly. The United States has a copyright principle called “fair use” that allows material to be copied by individuals for research purposes. The U.K. equivalent, which has been adopted by many countries whose laws were inherited from Britain in colonial times, is called “fair dealing” and is slightly more restrictive than fair use.

Making copies of copyrighted works for distribution or resale is prohibited. That is the main economic point of the copyright system. The Berne Convention also recognizes certain moral rights. Unlike economic rights, these cannot be assigned or transferred to others; they remain with the author forever. They give authors the right to the acknowledgment of their authorship, and to the integrity of their work—which means that they can object to a derogatory treatment of it.

The public domain

Works not subject to copyright are said to be in the “public domain,” which comprises the cultural and intellectual heritage of humanity that anyone may use or exploit. Often, works produced by the

government are automatically placed in the public domain, or else the government sets out generous rules for their use by not-for-profit organizations. This applies only in the country of origin: works produced by the U.S. government are in the public domain in the United States and its territories, but are subject to U.S. copyright in other countries.

Copyright does not last forever; when it expires, the work passes into the public domain, free of all copyright restrictions. No permission is needed to use it in any way, incorporate any of its material into other works, or make any derivative works. You can copy it, sell it, excerpt it—or digitize it and put it on the Web. The author’s moral rights still hold, however, so you must provide due attribution.

As internationally agreed in the Berne Convention, the minimum copyright duration is life plus 50 years, that is, until 50 years after the author dies. Of course, it is often difficult to find out when the author died. One way is to consult the authors’ association in the appropriate country, which maintains links to databases maintained by authors’ associations around the world.

The duration of copyright has an interesting and controversial history. Many countries specify a longer term than the minimum, and this changes over the years. The original British 1710 act provided a term of 14 years, renewable once if the author was alive; it also decreed that all works already published by 1710 would get a single term of 21 further years. The 1790 U.S. law followed suit, with a 14-year once-renewable term. Again, if an author did not renew copyright, the work automatically passed into the public domain. In 1831, Congress extended the initial period of copyright from 14 to 28 years, and in 1909 it extended the renewal term from 14 to 28 years, giving copyright a maximum duration of 56 years.

From 1962 onward, Congress enacted a series of copyright extensions, some of one or two years, others of 19 or 20 years. In 1998, it passed the Sonny Bono Copyright Term Extension Act, which extended the term of existing and future copyrights by 20 years. (The act is named in memory of former musician Sonny Bono, who, according to his widow, believed that “copyrights should be forever.”) The impetus behind the changes was the desire of large, powerful corporations to protect a minuscule number of cultural icons; opponents call them the “Mickey Mouse” copyright extensions. Many parts of the world (notably the United Kingdom) have followed suit by extending their copyright term to life plus 70 years—and in some countries (e.g., Italy) the extension was retroactive, so that works already in the public domain were suddenly removed from it.

The upshot is that copyright protection ends at different times depending on when the work was created. It also begins at different times. In the United States, older works are protected for 95 years from the date of first publication. Through the 1998 Copyright Extension Act, newer ones are protected from the “moment of their fixation in a tangible medium of expression” until 70 years after the author’s death. Works made for hire—often ones belonging to corporations—are protected for 95 years after publication or 120 years after creation, whichever comes first.

The original copyright term was one-time renewable, but few copyright holders renewed. Focusing again on the United States, in 1973 more than 85 percent of copyright holders failed to renew their copyright, which meant that, at the time, the average term of copyright was just 32 years. Today there is no renewal requirement for works created before 1978: copyright is automatically given for a period of 95 years—tripling the average duration of protection.

No copyrights will expire in the 20-year period from 1998 to 2018. To put this into perspective, 1 million patents will pass into the public domain during the same period. The effect of this extension is dramatic. Of the 10,000 books published in 1930, only a handful (less than 2 percent) are still in print. If the recent copyright extensions had not occurred, all 10,000 of these books would by now be in the public domain, their copyright having expired in 1958 (after 28 years) if it was not renewed, or 1986 (after a further 28 years) if it was renewed. Unfortunately, that is not the case.

If you want to digitize one of these 1930 books and make it available on the Internet, you must first determine its copyright status. If copyright expired in 1958 without renewal, it is already in the public domain. The official registry of copyright renewal is not available online, but new initiatives like the Copyright Renewal Database at Stanford University and the Registry of Copyright Evidence by the Online Computer Library Center (OCLC) are trying to help streamline the task. With or without the help of these newer initiatives, for works still under copyright, you will have to contact the copyright holders. If the book is out of print (98 percent of cases), they would likely be perfectly happy to give you permission. But you'd have to track these people down.

As Lawrence Lessig wrote in his book *Free Culture* (from which much of the above information was taken),

Now that technology enables us to rebuild the library of Alexandria, the law gets in the way. And it doesn't get in the way for any useful copyright purpose, for the purpose of copyright is to enable the commercial market that spreads culture. No, we are talking about culture after it has lived its commercial life. In this context, copyright is serving no purpose at all related to the spread of knowledge. In this context, copyright is not an engine of free expression. Copyright is a brake.

Relinquishing copyright

As we have explained, the Berne Convention grants copyright without formality, without registration. Anything you write, every creative act that is “fixated in a tangible means of expression”—be it a book, an e-mail, or a grocery list—is automatically protected by copyright until 50 years after you die (according to the Berne Convention's minimum restrictions) or, today in the United States, 70 years after you die (assuming you did not write your grocery list as a work made for hire). People can quote from it under the principle of fair use, but they cannot otherwise use your work until such time as copyright expires, unless you reassign the copyright.

Authors who wish to relinquish copyright must take active steps to do so. In fact, it's quite difficult. To facilitate it, a nonprofit organization called the Creative Commons has developed licenses that people can attach to the content they create. Each license is expressed in three ways: a legal version, a human-readable everyday-language description, and a machine-readable tag. Content is marked with the CC mark, which does not mean that copyright is waived but that freedoms are given to others to use the material in ways that would not normally be permissible by default under copyright.

The freedoms allowed by licensing all go beyond traditional fair use, but their precise nature depends on the choice of license. One license permits any use so long as attribution is given. Another permits only noncommercial use. A third permits any use within developing nations. Or any educational use. Or any use except for the creation of derivative works. Or any use so long as the same freedom is

given to other users. Most important, according to the Creative Commons, is that these licenses express what people can do with your work in a way they can understand and rely upon without having to hire a lawyer. The idea is to help reconstruct a viable public domain.

The term *copyleft* is sometimes used to describe a license that imposes the same terms on any derivative work: typically the rights to use, modify, and redistribute. The intention of *copyleft* is to use the facilities of *copyright* legislation to preserve these freedoms. So, if you take a *copyleft* work, make changes and distribute a modified version, then other people automatically acquire the same rights (to change and distribute) that you had received.

A prominent figure that helped establish this form of licensing is Richard Stallman. In 1984 he left MIT and founded the GNU Project and set about developing—and persuading over time others to help develop with him—a comprehensive suite of general purpose software. (The acronym, GNU, is itself a self-referential joke, standing for GNU’s Not Unix.) The produced software was made available under the terms of the GNU General Public License (GPL), encapsulating the notion of *copyleft*, and at the time a radical departure from how mainstream software was distributed. Subsequently the project extended the idea to other forms of work. For instance, the GNU Free Documentation License is the counterpart to GPL, originally designed for the manuals and documentation that accompany GNU software. Others are free to make use of the GNU licenses for their own work, and it is a popular choice for many open source projects.

Digital rights management

A treaty adopted by the World Intellectual Property Organization (WIPO) in 1996 addresses some of the copyright issues raised by digital technology and networks in the modern information era. It decrees that computer programs should be protected as literary works and that the arrangement and selection of material in databases is protected. It provides authors of works with more control over their rental and distribution rights than the Berne Convention does.

In order to understand the motivation behind this treaty, and its implications, reflect on the enormous repercussions of current innovations in electronic publishing. Publishers are accelerating production of digitized versions of their titles (e-Books) and selling them online. They continue to rely on conventional retailers for the sale of paper copies, but they are experimenting with combining e-Books with pre- and post-release paper copies. E-Book technology is being forced to standardize.

New sales models are beginning to emerge. Highly effective search engine advertising is leveling the market, providing more opportunities for small publishers and self-publishing. Physical bookstores are finding themselves bypassed. E-Books also provide preview options (flip through the pages at no cost), and can be rented on a time-metered or absolute duration basis (like video rentals), sometimes with an option to purchase. Publishers are also experimenting with print-on-demand technology.

Content owners (publishers) are adopting technical means to implement policies governing access to the information they sell. The term “digital rights management” (DRM) refers to the control and protection of digital content, including text documents, images, video, and audio. DRM technology limits what users can do with content—even when they possess it. Access can be restricted to a

particular computer: this means no lending to friends, no sharing between your home computer and your office, and no backing up on another machine. It also precludes resale, eliminating the second-hand market. Furthermore, expiry dates can be imposed, precluding permanent collections and archives. These measures go far beyond the traditional legal bounds of copyright.

In the entertainment industry, DRM schemes are used to counter perceived threats of piracy. Some complain that these schemes are concerned solely with content owners' rights, not with users' rights. DRM does not grant "permissions," which is what copyright authorizes, but instead enforces absolute, mechanical "controls." For example, the motion picture industry establishes control by compelling manufacturers to incorporate encryption into their products because it holds key patents on DVD players.

The WIPO Treaty introduces an important but controversial requirement that countries must provide effective legal measures against the circumvention of DRM schemes. One of the first national laws implementing the WIPO Treaty was the U.S. Digital Millennium Copyright Act (DMCA), which, among many other things, makes it unlawful to publish information that exposes the weaknesses of technical protection measures. In Europe, the European Council has approved the treaty on behalf of the European Community and has made directives that largely cover its subject matter.

Such legislation jeopardizes basic rights legally enshrined in the concept of copyright. DRM allows reading rights to be controlled, monitored, and withdrawn, and DMCA legislation makes it illegal for users to seek redress by taking matters into their own hands. In scholarly publishing, DRM is already well advanced. Academic libraries license access to content in electronic form, often in tandem with purchase of print versions. Because they form the entire market, they have been able to negotiate reasonable conditions with publishers. However, libraries have far less power in the consumer book market. For example, one can envisage a scenario where publishers establish a system of commercial pay-per-view libraries for e-Books and refuse public libraries access to books in a form that can be circulated.

Copyright and digitization

Many digital library projects involve digitizing documents. If the work to be digitized is in the public domain, or it attempts to faithfully reproduce a work in the public domain, you may digitize the work without securing anyone's permission. Of course, the result of your digitizing efforts is not protected by copyright either, unless you produce something more than a faithful reproduction of the original.

If material has been donated to your institution for digitizing, and the donor is the copyright owner, you can certainly go ahead, provided the donor gave your institution the right to digitize—perhaps in a written form, such as "the right to use the work for any institutional purpose, in any medium." Even without a written agreement, it may be reasonably assumed that the donor implicitly granted the right to take advantage of new media, provided the work continues to be used for the purpose for which it was donated. You do need to ensure, of course, that the donor is the original copyright owner and has not transferred copyright. For example, you cannot assume permission to digitize letters sent to the donor but written by others.

If you want to digitize documents and the above considerations do not apply, you should consider whether you can go ahead under the concept of fair use. This is a difficult judgment to make. You need to reflect on the copyright owner's concerns and address them. Institutional policies about who can access the material, backed up by practices that restrict access appropriately, can help. Finally, if you conclude that fair use does not apply, then you must obtain permission to digitize the work or acquire access to it by licensing it.

Thus, building a digital library requires serious attention to copyright. Digital library projects must be undertaken with a full understanding of ownership rights and with full recognition that permission to convert materials that are not in the public domain is essential. Because of the potential for legal liability, any prudent library builder should consider seeking professional advice. A full account of the legal situation is far beyond the scope of this book, but the "Notes and sources" section at the end of the chapter ([Section 1.8](#)) contains sources for practical information about copyright. The sources include information on how fair use can be interpreted and on the issues involved in negotiating copyright permission or licensing.

Looking at the situation from an ethical, rather than legal, perspective sheds light on the fundamental issues. It is unethical to steal: deriving profit by distributing a book for which someone else has rightful claim to copyright is wrong. It is unethical to deprive someone of the fruit of their labor: giving away electronic copies of a book for which someone else has rightful claim to copyright is wrong. It is unethical to pass someone else's work off as your own: making a digital library collection without proper acknowledgment is wrong. It is unethical to willfully misrepresent someone else's point of view: modifying documents before including them in the collection is wrong, even if the original authorship is acknowledged.

Collecting from the Web

The legal status of documents published on the World Wide Web is murky. Because activities like Web searching and archiving are in a state of rapid transition, it is impractical, and also inappropriate, for legal regulation to try to keep up with the change. If any legislation is needed, it should be designed to minimize harm to interests affected by technological change while at the same time enabling and encouraging effective lines of development. Legislators are adopting a "wait and see" policy, while leading innovators strive to ensure that what they do is reasonable and accords with the spirit—if not necessarily the letter—of copyright law.

Issues abound. Some lawyers have questioned whether it is legal even to view a document on the Web, since one's browser inevitably makes a local copy without explicit authorization. Of course, it is widely accepted that you can view Web documents—after all, that's what they're there for. Next comes the question whether you can save Web documents for personal use. Or link to them. Or distribute them to others. Note that, behind the scenes, Web documents are routinely copied and saved. For example, to economize on network traffic and to accelerate delivery, Web cache mechanisms save copies of documents locally and deliver them to other users.

The way that computers in general, and the Web in particular, operate raises the question whether the notion of a "copy" is perhaps no longer the appropriate foundation for copyright law in the digital

age. Legitimate copies of digital information are made so routinely that restrictions on the act of copying no longer serve to regulate and control use on behalf of copyright owners. Because computers make many internal copies when they are used to access information, the fact that a copy has been made says little about the legitimacy of the behavior. In the digital world, copying is so bound up with the way computers work that controlling it provides unexpectedly broad powers, far beyond those intended by copyright law.

All these points have an immediate and practical impact on digital libraries. Digital libraries are organized collections of information. The Web is full of unorganized information. Downloading parts of Web content in order to organize information into focused collections and to make the material more useful to others is a prime application area for digital libraries.

Search engines, one of the most widely used services on the Internet, are a good example. They use software “robots” to continually download huge portions of the Web and create indexes to the content. Although a service provider may retain documents on their own computers, searchers are presented with a summary and are directed to the original source documents rather than to local copies. Search engines are commercial operations, but their services are not sold directly to users; instead, their revenue is derived from advertising—in effect, a tax on the user’s attention. Although search engines are widely accepted and used, their legal status is unclear.

Web sites can safeguard against indiscriminate downloading. A generally accepted *robot exclusion protocol* allows individual Web sites to prevent their content from being downloaded and indexed. Although this protocol is entirely voluntary, widely used search engines certainly comply with it. But the onus of responsibility has been shifted. Previously, to use someone else’s information legitimately, one had to request explicit permission from the information provider. Now, search engines automatically assume permission unless the provider has set up an exclusion mechanism. This is a key development with wide ramifications. And some Web sites threaten dire consequences for computers that violate the robot exclusion protocol—for example, denial-of-service attacks on the violating computer. This is law enforcement on the wild Web frontier.

Different copyright issues are raised by projects that are archiving the entire World Wide Web. The rationale for creating such an archive is to offer services like supplying documents that are no longer available or providing a “copy of record” for publicly available documents—in effect, supplying the raw material for historical studies. However, creating an archive raises many interesting issues involving privacy and copyright, issues that are not easily resolved.

What if a college student created a Web page that had pictures of her then-current boyfriend? What if she later wanted to “tear them up,” so to speak, yet they lived on in the archive? Should she have the right to remove them? In contrast, should a public figure—a U.S. senator, for instance—be able to erase data posted from his or her college years? Does collecting information made available to the public violate the “fair use” provisions of copyright law?

Most digital libraries aim to provide more comprehensive searching and browsing services than search engines do. Like archives, they probably want to store documents locally, to ensure their continued availability. Documents are more likely to be seen as part of the library, rather than as products of their original Web site. Digital libraries are more likely to modify documents as an aid to the user, least invasively by highlighting search terms or adding metadata, more invasively by re-

presenting them in a standard format, most invasively by producing computer-generated summaries or extracting keywords and key phrases automatically.

Those responsible for such libraries need to consider carefully the ethical and legal issues involved. It is important to respect robot exclusion protocols. It is important to provide mechanisms whereby authors can withdraw their works from the library. It is helpful if explicit permission can be sought to include material. If information is automatically derived from, or added to, the source documents, it is necessary to be sensitive to possible misrepresentation.

The world is changing. Digital libraries are pushing the boundaries of what is possible by organizing anthologies of material. And they are pushing the boundaries of society's norms for distribution of intellectual property. Those who run large-scale Internet information services tell interesting "war stories" of people's differing expectations of what it is reasonable for their services to do. For example, search engine operators frequently receive calls from computer users who have noticed that some of their documents are indexed when they think they shouldn't be. Sometimes users feel their documents couldn't possibly have been captured legitimately because there are no links to them. Most search engines have a facility for locating any documents that link to a specified one and can easily locate the omitted link. On other occasions, people have put confidential documents into a directory that is open to the Web, perhaps just momentarily while they change the directory permissions, only to have them grabbed by a search engine and made available for all the world to see.

Search technology makes information readily available that may previously have been public in principle, but impossible to find in practice. When a major search engine took over the archives of a corpus of Internet discussion groups on a wide range of topics, it received many pleas from contributors to retract indiscreet postings because, now that postings were easily available for anyone to find, they were causing their authors embarrassment.

Illegal and harmful material

Some material is illegal and harmful and clearly inappropriate for public presentation (because such material is distasteful, we will not give examples). For example, a 1999 UNESCO Expert Meeting on Paedophilia on the Internet noted,

Violence and pornography have invaded the Internet. Photos and videos of children and young teenagers engaged in sexual acts and various forms of paedophilia are readily available. Reports of children being kidnapped, beaten, raped and murdered abound.... The Internet has in many cases replaced the media of paedophilic magazines, films and videos. It is a practical, cheap, convenient and untraceable means for conducting business as well as for trafficking in paedophilia and child pornography. The Internet has also become the principal medium for dialogue about paedophilia and its perpetuation.

UNESCO has taken the lead on breaking the silence on this topic and is engaged in a number of initiatives to provide safety nets for children online.

Whether information is considered harmful often depends on the cultural, religious, and social context in which it is circulated. Standards vary both within and among nations. However, the international nature of the Internet means that it is no longer possible to police the transfer of information, and sustaining local legal and cultural standards is a huge challenge facing society today. The

challenge includes the dilemma of balancing freedom of expression against citizens' rights to be protected from illegal or harmful material.

In 2000, a well-publicized example of different views on Internet access concerned online sales of Nazi memorabilia on U.S. Web sites accessed using the Yahoo Internet portal. A judge in Paris ruled that the sites are barred under French law and ordered them to be blocked. However, U.S. Web sites are governed by U.S. laws, and an American judge ruled that the First Amendment protects content generated in the United States by American companies from being regulated by authorities in countries that have laws more restrictive of freedom of expression. Suit and countersuit followed, and the matter was not settled for six years, when a U.S. court decided that Yahoo was liable for a fine levied in France.

Another challenging example is online gambling. The relevant laws are restrictive (or at best muddy) in countries like the United States, China, and Italy. Some international gambling sites claim to comply with local laws by checking the geographical origin of the user (a difficult and unreliable procedure that is easily circumvented) and by refusing to offer their services in countries where gambling is illegal.

Cultural sensitivity

Most digital libraries are produced by people from Western backgrounds, yet the majority of the world's population live in countries that have very different cultures. Some digital libraries are specifically aimed at people in different parts of the world: collections for developing countries, for example, or collections aimed at preserving and promoting indigenous cultures. It is clearly necessary for digital library developers to consider how their creations will affect other people.

First, as we already mentioned, language is the vehicle of thought, communication, and cultural identity, and so digital library users should be able to work in whatever language suits them. But the need for cultural sensitivity goes even deeper. Particular labels can have strong, unexpected connotations: certain models of cars have failed to sell in countries where the model's name had a serious negative association. Icons also have cultural implications: dogs, for example, are offensive in Arabic cultures and will have negative effects if they are offered as user interface icons. Furthermore, different cultures have different color preferences, and particular colors have different associations.

In Polynesian cultures the concept of *tapu*, usually translated as "sacred," has rich and complex connotations that are difficult for Westerners to appreciate. Many objects have different degrees of *tapu*, and Polynesians find it rude and offensive if these objects are used inappropriately, in the same way that many Westerners find blasphemy rude and offensive. An example that can affect digital library design is that representations of people—including pictures—are *tapu*, and in Polynesia it is generally inappropriate for them to be on public display.

1.6 Planning a Digital Library

Are you going to build a digital library? We hope so. And as with any complex system, it is wise to undergo a planning stage before immersing yourself in the detail. [Table 1.1](#) shows some of the ques-

Table 1.1: Pertinent questions

Users	1. Who are the intended users? 2. Where are they? 3. What computer experience do they have? 4. What languages do they know? 5. Will they need help in accessing the library? 6. Why do they want to access this material? 7. What technology (e.g., Web browser) will they use? 8. To what extent should the library record usage? 9. Can users contribute to the digital library? 10. How will you evaluate the project's success?
Material	1. What will be <i>in</i> the digital library? 2. What format is the material in now? 3. What format does it need to be in for presentation? 4. Will users need it to be in multiple formats? 5. Do the formats require conversion of the material? 6. How will you resource the conversion? 7. Are there copyright or other restrictions? 8. Will the library be public or restricted to specific clients? 9. Will you add value (i.e., metadata) to the material? 10. If so, how will you resource this activity?
Technology	1. What computers will host the digital library? 2. Who maintains them? 3. What software will be used? 4. Do you have resources to purchase/license/maintain it? 5. How will the material be converted to the delivery format? 6. How will you control access? 7. How will you interoperate with other libraries? 8. Can the material be exported from the digital library software? 9. What would this cost? 10. If you add value, can these additions be exported?

tions you should be asking. You may not be able to answer them all immediately, but you will have to answer them all eventually. When you have read this book you will be able to approach such questions with a solid understanding of the spectrum of possible answers and the implications of each one. Perhaps even more importantly, you will be able to reassess your answers when technology changes. This is what it means to understand the field of digital libraries.

The questions are divided into three categories: users, materials, and technology. Despite this neat structure, however, the issues are complex and resist any simple categorization. For example, choosing a suitable format for presenting material to users requires an understanding of:

- the background of the users and the technology they use for access;
- the original format of the material and whether it needs to be converted before presentation;

- technological options for conversion;
- the possibilities for information loss during conversion;
- resourcing requirements for conversion.

The issues are strongly interconnected, there are many trade-offs, and each collection has its own idiosyncrasies. Moreover, computer-based systems change extremely rapidly—in so-called Internet-time—and particular ways of resolving issues in [Table 1.1](#) will likely soon be out of date.

This book proceeds in roughly the same way as [Table 1.1](#). Chapter 2 focuses on the people in digital libraries. The most important people are the users (customers, if you like) and the questions about users in [Table 1.1](#) are much harder to answer for digital libraries than for traditional ones. Chapter 3 concerns the user *experience*: What might the final digital library system look like? We examine different kinds of documents and how they can be presented, multimedia as well as text, and document surrogates as well as full documents. We also consider how users access the library, and how they experience the fundamental operations of searching and browsing.

The following three chapters concern the second block of [Table 1.1](#): the material in digital libraries. Chapters 4 and 5 discuss the form of the source documents, the raw ingredients for the library. A major question in practice is whether material will be obtained from physical documents, such as books and journals, and Chapter 4 includes an extensive section on scanning and optical character recognition (the technology for producing electronic text from scanned pages). This can be a costly process, particularly if high accuracy is needed, which makes questions about resourcing the conversion particularly pertinent and concerns about the cost trade-offs particularly acute. In many digital collections, electronic conversion is by far the major cost. Another potential expense is adding value to the contents by providing metadata, including enhanced descriptions of each item. Metadata effectively binds the raw ingredients into a coherent form: Chapter 6 discusses its shape and form, and explores the role it plays bringing order and organization to a library.

The third block of [Table 1.1](#) is technology. The nitty-gritty details of implementation are given in Part II of this book. However, Chapter 7 introduces a wider set of technological issues, including the protocols and services that aid interoperability. The last three questions in [Table 1.1](#) concern risk management in an unpredictable technological environment. What if circumstances change? Through no fault of your own, software and implementation strategies you choose today may turn out badly tomorrow. If circumstances change and you wish to migrate the library to another system, you will want to preserve the investments you have made—another aspect of interoperability. Chapter 8 looks at the global challenges of internationalization, which are both human and technological—recall the example of the alphabet used when searching for Sophocles. Chapter 9, which concludes Part I, presents visions of digital libraries past, present, and future, and highlights the difficulty of long-term preservation.

Resource considerations plague almost every endeavor in life, and libraries are no exception. Several of the questions in [Table 1.1](#) ask about the resources required for different parts of the process. Our own experience is that the decision-makers who control resources are often tremendously impressed if they see a prototype of the digital library. Somehow, when it's their own material or material that

they have a stake in, even a small, rough-hewn demo can prove inspirational—and attract funding. It’s usually very worthwhile to obtain a small sample of the material and build it into a small prototype that you can show to people in a position to argue for resources to support a major effort. Think big—but, in the first instance, act small. The important thing is to act.

1.7 Implementing a Digital Library: The Greenstone Software

Part I of this book is general, drawing examples from the panoply of digital library innovation and implementation around the globe. But this is a practical book about *how to build* a digital library. Rather than attempting a high-level survey of the many different software products that are available, Part II focuses on a particular system for building digital libraries: the Greenstone Digital Library software from the New Zealand Digital Library Project at the University of Waikato. Almost all of what is presented in Part I can be implemented using the Greenstone toolkit.

Table 1.2 summarizes key features of the Greenstone software. A *library* can include multiple information collections, each individually designed to meet the needs of users and to take advantage of whatever organizational information is available (or can be provided). A *collection* comprises several (typically several thousand, or several million) documents. *Documents* are the fundamental unit from which collections are built; they may have internal structure, alternative representations, and associated files. Documents may contain text, images, sound, or video; collections may include a mixture of document types. *Metadata* is bibliographic information associated with individual documents.

Each collection provides a uniform interface through which its documents can be accessed, an interface that is designed by the librarian responsible for the collection. Prior to presentation, collections undergo a *building* process that, once established by the librarian, is completely automatic. This creates all the structures that are used to access the collection. Searching is based on indexes of full text and metadata. Browsing is based on structures built from metadata. When new material appears, it can be fully incorporated into the collection by rebuilding.

To address the exceptionally broad requirements of digital libraries, the Greenstone system is public and extensible. It is issued as free open-source software under the GNU General Public License. It is widely used throughout the developed and developing worlds. Further details, and many examples, can be obtained from www.nzdl.org. Part II of the book describes how to use the software to build digital library collections, customize them to particular requirements, and use Greenstone’s many interoperability features.

1.8 Notes and Sources

To avoid breaking up the flow of the main text, all references and explanatory notes are collected in a section at the end of each chapter. This first “Notes and sources” section describes information sources, papers, books, and other resources relevant to the material covered in Chapter 1.

We learned about the Kataayi Cooperative from Emmanuel Kateregga-Ndawulu, the chairman. If you would like to learn more about this fascinating initiative, a web search for *Kataayi* will turn up some

Table 1.2: The Greenstone digital library software

Reader's interface	Readers access collections using a Web browser
Multiplatform server	Digital library server runs on any version of Windows, Macintosh (OS X), Linux, or other Unix versions
Librarian's interface	Librarians design and build collections, and add documents and metadata, using an interactive interface
Documents	Ingests all standard document types, e.g., HTML, Word, RTF, PDF, PostScript, Powerpoint, Excel, e-mail
Multimedia	Ingests multimedia objects: images (e.g., GIF, PNG, JPEG, TIFF), audio (e.g., MP3, Ogg Vorbis, WAV), video (e.g., MPEG, Quicktime, RealMedia)
Metadata	Ingests all standard metadata types, e.g., XML, MARC, OAI, METS, CSV, CDS/ISIS, ProCite, BibTex
Metadata sets	Defaults to Dublin Core but can use any metadata set; incorporates metadata set editor to create ones
Extensibility	"Plug-ins" can be written to accommodate new document and metadata types
Searching	Full text is searchable by default; librarians can add indexes of (any combination of) metadata elements
Browsing	Librarians can add linearly or hierarchically browsable lists of any metadata elements
Access structures	All access structures are created automatically; no links are inserted by hand
Indexers	Incorporates a choice of full-text indexing software: MG, MGPP and Lucene
Incremental operation	Can operate incrementally when documents are added to or deleted from collections
Scheduled rebuilding	Includes end-user facilities for scheduling regular rebuilding of collections
Interoperability	Serves and harvests OAI-PMH; exports and ingests to/from METS and DSpace; harvests over Z39.50 and SRU; ingests MediaWikis
Multilingual content	Unicode is used internally and converted on the fly to a coding standard supported by the user's Web browser.
Multilingual interface	Reader's interface is available in 50 languages; librarian interface and documentation are available in several languages
User authentication	Built-in access control mechanism for collections and individual documents to authorized users
Logging	(Optionally) records all user actions
Dynamic operation	Collections can be updated and new ones brought online at any time, without disturbing users
Publishing collections	Collections can be published on removable media such as CD-ROM/DVD/USB flash drive
Distributed librarians	Authenticated users can modify collections on a central server using a remote version of the Librarian interface
Large scale	Existing collections have 10 M documents, 20 GB of raw text, 2 B words, 50 M unique terms, 50 GB metadata
Space efficient	Uses compression to reduce the size of indexes and text
License	Free open-source software issued under the GNU General Public License
Support	Mailing lists in several languages; support centers in India and Africa; several commercial support organizations exist

interesting information. Jon Miller kindly provided the photographs in [Figure 1.1](#) (and [Figure 9.3](#) in Chapter 9). The Humanity Development Library is produced by Michel Loots of the HumanInfo Project in Antwerp, Belgium, using the Greenstone software, and widely distributed in the developing world. The development of the physics archives is described by Paul Ginsparg, its originator, in a paper called “Winners and losers in the global research village” (Ginsparg, 1997); he is responsible for the memorable “sliced dead trees” metaphor. (Note incidentally that *adsorbed* is a physics term, not a misprint; it means the assimilation of dissolved matter by the surface of a solid.) Lloyd Smith at New Mexico Highlands University conceived of the Zia Pueblo project and came up with the vision described here; the Zia Pueblo kindly supplied the photograph in [Figure 1.2](#). The digital music library is ongoing work in the Department of Computer Science at Waikato.

Our definition of *digital library* in [Section 1.1](#) is abstracted from 10 definitions of the term *digital library* culled from the literature by Ed Fox (on the Web at <http://ei.cs.vt.edu/~dlib/def.htm>). It was the computer pioneer Maurice Wilkes who said that books would be hailed as a great advance if they were invented today. The “data ... information ... knowledge ... wisdom” sequence and characterization is due to Harold Thimbleby at Swansea University.

A good source for the development of libraries is Thompson (1997), who formulates some principles of librarianship, including the one quoted, that it is a librarian’s duty to increase the stock of his library. Thompson is the source of some of the material with which [Section 1.2](#) begins, including the metaphor about snapping the links of the chained book—in fact, he formulates open access as another principle: *libraries are for all*. The imaginative architectural developments that have occurred in physical libraries at the close of the 20th century are documented, and beautifully illustrated, by Wu (1999). Gore (1976b) recounts the fascinating history of the Alexandrian Library; he edited the book entitled *Farewell to Alexandria* (Gore, 1976a). The information on Trinity College, Dublin, was kindly supplied by David Abrahamson; that on the Library of Congress was retrieved from the Internet. Much of the other historical information is from an excellent and thought-provoking paper by Gaines that is well worth reading (Gaines, 1993). Thomas Mann (1993), a reference librarian at the Library of Congress, has produced a wonderful source of information on libraries and librarianship, full of practical assistance on how to use conventional library resources to find things.

H. G. Wells’s “world brain” idea was published in 1938 (Wells, 1938). Vannevar Bush’s vision was described in the year the United Nations was founded (Bush, 1945)—although certainly no connection between virtual libraries and the plight of developing countries was made in those days. Licklider’s vision dates from 1960 (Licklider, 1960), while the U.S. Presidential Office weighed in early in 1993 (Clinton and Gore, 1993).

The 19th-century librarian who “makes time for his fellow mortals” is Bowker (1883), quoted by Crawford and Gorman (1995). The modern term “metadata” is an impressive-sounding moniker, but the catchphrase “data about data” is glib but not very enlightening. In some sense, *all* data is about data: Where does one stop and the other begin? We return to this discussion at the beginning of Chapter 6. Meanwhile we continue to use the term freely, always in the context of digital library collections in which it is clear that the metadata is information about a particular resource.

More information on the promise of digital libraries in developing countries can be found in Witten et al. (2001). The figures on the Nairobi and Brazzaville universities are from the United Nations *Human Development Report* (United Nations, 1999), as is some of the information on Internet penetration in developing countries. Arunachalam (1998) tells how the Internet is “failing the developing world.” Statistics on the number of computers available in developing countries can be found in the World Bank’s (2000) *World Development Indicators*. Information on mobile phone penetration can be found in an International Telecommunication Union report (1999). There is much information on the “digital divide,” the widening knowledge gap between rich and poor: read the United Nations’ 1997 statement on *Universal Access to Basic Communication and Information Services*, the International Telecommunication Union’s 1998 *World Telecommunication Development Report*, or the World Bank’s 1998/99 *World Development Report*.

The Greenstone software is produced by the New Zealand Digital Library Project. The *toki* (adze) shown in [Figure 1.11](#) was a gift from the Māori people in recognition of the digital library’s contributions to indigenous language preservation; it resides in the project laboratory at the University of Waikato. In Māori culture there are several kinds of *toki*, with different purposes. This one is a ceremonial adze, *toki pou tangata*, a symbol of chieftainship. The *rau* (blade) is sharp, hard, and made of *pounamu* or greenstone—hence the Greenstone software, at the cutting edge of digital library technology. There are three figures carved into the *toki*. The forward-looking one looks out to where the *rau* is pointing to ensure that the *toki* is appropriately targeted. The backward-looking one at the top is a sentinel that guards where the *rau* can’t see. There is a third head at the bottom of the handle that makes sure that the chief’s decisions—to which the *toki* lends authority—are properly grounded in reality. The name of this taonga, or art treasure, is *Toki Pou Hinengaro*, which translates roughly as “the adze that shapes the excellence of thought.” *Haramai te toki, haumi e, hui e, tāiki e*.

Turning now to copyright, Samuelson and Davis (2000) provide an excellent and thought-provoking overview of copyright and related issues in the information age, which is a synopsis of a larger report published by the National Academy of Sciences Press (Committee on Intellectual Property Rights, 2000). [Section 1.5](#) draws heavily on this material. An earlier paper by Samuelson (1998) discusses specific digital library issues raised by copyright and intellectual property law, from a U.S. perspective. The Association for Computing Machinery has published a collection of papers on the effect of emerging technologies on intellectual property issues (White, 1999). We learned a lot from Lessig’s fine book *Free Culture* (Lessig, 2004), which has strongly influenced our perspective; the quotation is from p. 227. Lessig is also the originator of the Creative Commons, whose licenses can be found on the web—as can the GNU GFDL license.

There’s plenty of information on copyright on the Web. For example, <http://scholar.lib.vt.edu/> copyright is a useful Web site developed by staff at Virginia Tech to share what they learned about policies and common practices that relate to copyright. It includes interpretations of U.S. copyright law, links to the text of the law, sample letters to request permission to use someone else’s work, links to publishers’ e-mail addresses, advice for authors about negotiating to retain some rights, as well as current library policies. Georgia Harper at the University of Texas at Austin has created an



Figure 1.11: Māori *tōki* or ceremonial adze, emblem of the Greenstone project

excellent *Crash Course in Copyright* (www.utsystem.edu/ogc/intellectualproperty/cprtindx.htm) that is delightfully presented and well worth reading. The information about the duration of copyright protection is from Lolly Gasaway, director of the Law Library and professor of law at the University of North Carolina, through his Web site at www.unc.edu/~uncclng/public-d.htm. The core reference for copyleft is at the GNU Project: <http://www.gnu.org/copyleft>.

The source of the quotation about the hypothetical college student is Kahle (1997), a *Scientific American* article describing an Internet archiving project that we return to in Chapters 5 and 9. Information on the UNESCO initiative to attack paedophilia on the Internet can be found in Chapter 8 of its *World Communication and Information Report 1999–2000*. Elke Duncker is a marvelous source of information on cultural sensitivity and user interfaces: some of her experiences with user interface issues in different cultures are described in Duncker (2002).

Standard library automation systems are described by Cooper (1996). The first book on digital libraries is *Practical Digital Libraries* by Lesk (1997), a pioneering work that gives a fascinating early

account of this emerging field. In contrast, Crawford and Gorman (1995) fear that virtual libraries are virtual nonsense that threatens to devastate the cultural mission of libraries. Chen (1998) describes the past, present, and future of digital libraries from his perspective as Program Director of the NSF/DARPA/NASA Digital Libraries Initiative in the United States from 1993 to 1995. Sanders (1999) offers an edited collection of papers that give a librarian's perspective on many aspects of digital libraries and their use. Although Borgman's (2000) title focuses on the global information infrastructure, most of her material is relevant to the kind of digital libraries discussed here. Arms (2000) gives an authoritative, comprehensive, and balanced account of digital libraries from many different perspectives. He includes a historical perspective, a survey of the state of the art, and an account of current research.

People in digital libraries

Think about your local library: besides the books, magazines, newspapers, DVDs, and computers, the library's key ingredient is people. Many are patrons (readers), but there are also several librarians, without whom there would be no library for the patrons to patronize. There are other staff as well, working behind the scenes: librarians in the cataloging and acquisitions department, IT personnel in the technical services department, and education specialists in the outreach department.

The emphasis on people is a fundamental principle of contemporary librarianship, and stands in contrast to medieval librarianship, whose job it was to protect, revere, and even chain up the books. Ranganathan's classic work *The Five Laws of Library Science* opens with this quotation from Manu, an ancient Hindu philosopher and lawmaker:

To carry knowledge to the doors of those that lack it ... even to give away the whole earth cannot equal that form of service.

What an inspirational sentiment for budding digital librarians with a social conscience! Ranganathan himself was an influential librarian and educator who is considered the father of library science in India. Librarians worldwide apply his five laws as the foundations of their philosophy:

- Books are for use.
- Every reader has [or her] book.
- Every book finds its reader.
- Save the reader's time.
- The library is a growing organism.

Today we live in a far more complex world—you can glimpse how much more complex by looking at the American Library Association's Bill of Rights in [Figure 2.1](#). Nonetheless Ranganathan's principles remain at the center of librarians' professional values. As you can see, they are as much about people as they are about books.

The first step in building a successful digital library, therefore, is to understand the people involved. Who are the likely readers? Why would they want to access the content? Do they need help or advice? Who will install, maintain, and update the digital library software? Who will look after the hardware that holds the actual documents? Who will cope with computer and network upgrades—and who will monitor their effects? How are the traditional roles of librarians affected by new technology? Can the users of digital libraries actively contribute to their development?

Library Bill of Rights

The American Library Association affirms that all libraries are forums for information and ideas, and that the following basic policies should guide their services.

- I. Books and other library resources should be provided for the interest, information, and enlightenment of all people of the community the library serves. Materials should not be excluded because of the origin, background, or views of those contributing to their creation.
- II. Libraries should provide materials and information presenting all points of view on current and historical issues. Materials should not be proscribed or removed because of partisan or doctrinal disapproval.
- III. Libraries should challenge censorship in the fulfillment of their responsibility to provide information and enlightenment.
- IV. Libraries should cooperate with all persons and groups concerned with resisting abridgment of free expression and free access to ideas.
- V. A person's right to use a library should not be denied or abridged because of origin, age, background, or views.
- VI. Libraries which make exhibit spaces and meeting rooms available to the public they serve should make such facilities available on an equitable basis, regardless of the beliefs or affiliations of individuals or groups requesting their use.

Adopted June 18, 1948.

Amended February 2, 1961; June 28, 1967; and January 23, 1980,
inclusion of "age" reaffirmed January 23, 1996,
by the ALA Council.

Figure 2.1: American Library Association's Bill of Rights

Other chapters discuss the technical details of file formats, metadata standards, digitization of material, and document representation. Here, as we begin with the most important element, the people, we need to define the various roles that people play in digital libraries. The transition from physical to digital removes many geographical constraints, which has an enormous effect on the people involved and introduces a host of new issues that traditional librarians could hardly imagine. Next we consider the central concept of "identity": What does the new librarian know, or need to know, about who the users are and what they are doing?

The fact that users and librarians no longer interact face to face is an important change in library operation that is discussed in [Section 2.3](#). In [Section 2.4](#) we look at the issues that are raised when working with digital rather than physical material, both for readers and for librarians. Finally, in

Section 2.5, we discuss what could be the most exciting new development in information access in the future: user contributions. Previously, users were severely discouraged from making contributions to library material: readers who contributed to (wrote in) library books were considered to be “defacing” the books, which had wholly negative connotations. But because the digital world offers selective access to altered material, defacing has been rebranded as “enhancing.” The digital world embraces the ability of users to contribute their expertise by making corrections and adding annotations, tags, ratings, and even entirely new material.

2.1 Roles

Libraries are social organizations that connect readers and authors through the content of their collections. Although reader and author are the most prominent roles, numerous people work behind the scenes to enable the simple act of reading a library book. Examining the roles that people play in a physical library will help us understand how to deploy and maintain a digital one.

A fundamental property of digital libraries is that they are, well, digital. Physical libraries are actual bricks-and-mortar buildings, occupying land, with windows, walls, and a roof. Indeed, as Chapter 1 describes, the new national libraries of the 20th century’s closing decade are monumental in scale. In contrast, digital libraries occupy the far less tangible world of computers, of magnetic and optical storage mechanisms, of network connections and Web sites. These are striking differences, and they impose very different roles and skill requirements on the people who manage the collections and serve the users.

Several basic characteristics are common to most workplaces, including entrances and exits, lighting, temperature control, restrooms, security, parking space (or, more commonly, the lack thereof), and so on. Libraries have two particular extra characteristics: the patrons and the collections. Physical libraries are shared environments where many of the participants are present for relatively short periods of time. The documents and other objects have special requirements with respect to temperature, humidity, and sunlight exposure. Maintaining a suitable environment is a necessary condition for an effective physical library.

In digital libraries the computers still require space and associated services, but they need not be located at any particular physical place. Computers and collections can be distributed across different centers. Copies of the content can be replicated around the world; indeed, this is a good thing—to increase robustness, to reduce the effect of outages, and to preserve the material in the event of catastrophe. The people who manage the digital content may be physically separate from their colleagues, perhaps not even part of the same organization. Hosting of digital collections can be outsourced to the other side of the world in a manner that is obviously not an option for providing access to physical resources.

Wherever they are, maintainers of digital content have quite different concerns from custodians of physical material. They seek environmental conditions that help their computers and disk drives work smoothly, without interruption. Instead of opening hours, car parking, and patrons walking around the library, maintenance staff for a digital library worry about their network connections, firewall integrity, and the load on the Web server. Therefore, the skill sets for working in physical and digital

libraries are very different: imagine the chaos that would result from swapping network support technicians for the help-desk staff in your own library!

In practice, most of today's libraries are hybrid organizations with both physical and digital elements in their collections. This means that staff must have the skills to cover both. Although the expectation that library material be Web accessible sprang up very rapidly with the advent of the World Wide Web, the transition to a digital environment has been reasonably gradual because computers have managed the metadata of physical collections ever since online public access catalogs (OPACs) began to replace library card catalogs in the 1980s (see Chapter 6).

Global users

Access to full digital content allows users to work far outside the library walls. Library holdings can now be accessed over the global computer network provided by the Internet. Today it is perfectly normal for a user to access library material even though neither user nor content is anywhere near a library building. The fact that there is no need to walk into the library removes all geographical restrictions on access. The result is that everyone in the entire connected world becomes a potential user of digital library resources.

This change has far-reaching consequences. In service-oriented organizations you need to know whom you are supposed to serve. Moving from the physical to the networked world makes it far more difficult to identify the user population (as well as making it difficult to identify individual users—see the next section).

Figure 2.1 shows the Bill of Rights of the American Library Association, whose aim is to promote library services and education internationally. We have already pointed out it differs from Ranganaathan's simple, direct laws of library science. Notice in particular the first article: books and other library resources should be provided for the interest, information, and enlightenment of all people of the community the library serves. What is this community? It is well known and fully defined for almost every physical library, but for many digital libraries the community of users is effectively the whole connected world. The fifth article of the Bill of Rights also presents a challenge to digital libraries: a person should not be denied use of a library because of origin, age, background, or views.

The principles of the association's Bill of Rights are just as admirable today as they were when they were adopted 60 years ago. But there is a clear mismatch between the global distribution of potential users and the pragmatic local aspects of management, funding, accountability, and the legal environment. Patrons are more dispersed, but the funding base remains local. Worse still, budgets are shrinking, because librarians have to swim against a current of popular opinion that questions why, in today's culture of information overload, we need libraries at all.

Here is how a practicing librarian sees the dilemma.

It's a growing (ballooning, exploding even) challenge to balance my time between our primary patron base and the random people who find us on the Web. As a public institution, our upper administration has made it clear that, while we give priority to our primary patron base, we are to answer all questions, whenever possible.

Table 2.1: Roles of the digital and physical librarian

Role	Physical content	Digital content
Librarian	Enabling access to content Managing physical objects produced by others	Restricting access to content Publishing content produced by your organization
User support	Face-to-face consultation	Computer-based interaction
Acquisitions	Purchasing physical objects	Digitizing physical content
Serials	Selecting and processing physical journals	Negotiating electronic access to online journals
IT support	Managing an integrated library system from a large systems vendor	Running content management systems—possibly from open source providers with little backup Running a Web site Continual updating and migrating of digital content

Instead of trying to be all things to all patrons, one common approach is to restrict some services to users who can be identified through such means as usernames and passwords.

Roles of librarians

The digital environment has already significantly altered the roles of librarians and is likely to continue to change the nature of information work. [Table 2.1](#) shows some of the differences between the roles of digital and physical librarians.

Just as librarians used to deal with physical objects, now they deal with digital content. Moreover, there has been an associated move from owning information to renting it. This trend strikes at the heart of the librarian's customary way of doing things. Today, publishers rarely sell digital copies of scholarly research—they license them instead. Digital rights management technology, introduced in [Section 1.5](#), can enforce conditions that go far beyond those traditionally imposed by libraries. The effect on libraries—and society—is immense. A sea change is under way in the librarian's profession, from information-enabler to restriction-enforcer.

Acquisitions and collection development have always been major responsibilities of librarians. But now the job has changed from acquiring physical material to negotiating licenses for digital material. This is particularly striking in the realm of serials and electronic journals, but it will increasingly apply to acquisitions in general. Furthermore, libraries form consortia that negotiate group licenses with publishers at favorable rates. Because selecting electronic resources tends to be done by groups rather than individuals, this has the effect of diminishing the autonomy of individual libraries and subtly changing the role of individual librarians.

Another issue related to acquisition is digitization of physical content. As [Chapter 4](#) explains, digitization is the process of taking traditional library materials, typically in the form of books and papers, and converting them to an electronic form that can be stored and manipulated by a computer. Many libraries own special collections and unique materials like rare books and manuscripts. There is a

strong incentive to use digital library technology to make these materials available electronically, widening access to a far larger user group and increasing the library's profile and reputation.

Although physical libraries need IT support, it almost always involves installing and maintaining integrated library systems supplied by major vendors. In contrast, digital libraries frequently use open source software and often run several different software systems in parallel. This greatly extends the support requirements. Digital libraries need to establish and maintain a Web site—quite a different proposition from running a traditional library system. The software will have to be updated regularly, and occasionally the library's entire content will have to be migrated to new hardware and systems.

An overarching challenge for all librarians is to integrate access to traditional library material (owned), online journals (licensed), and information on the Web (publicly available) into a one-stop shop that recognizes the different legal status of the material. Many difficult issues arise, such as how to reconcile site licensing with walk-in library patrons. (A related question is what will happen to today's students when they graduate and find their university library's doors metaphorically closed by licensing restrictions?) Some libraries even aspire to service their patrons' e-books and personal digital assistants, which raises legal questions about delivery models and technical ones about expansion-card compatibility. Countless new roles are being forced upon the librarian.

Change

This brings us to the question of change. We stand at the epicenter of a revolution in how our society creates, organizes, locates, presents, and preserves information. Librarians are undoubtedly among those most radically affected by the tremendous explosion in networked information sparked by the Internet. Advances in information technology generate an onslaught of opportunities and problems that pose a fierce and sustained challenge to librarians' self-image. As just one example: consider the challenge to librarians inherent in Google's declaration that its mission is "to organize the world's information and make it universally accessible and useful"—a mission that librarians thought society had entrusted to them alone, and which they had been doing well for centuries.

Of course, change has been ever-present throughout the history of library development, as the synopsis in Section 1.2 illustrates. Furthermore, change is especially true in the digital realm, since technological change continually produces new forms of content and new methods of access. For library staff to work successfully under these conditions, they must continually adapt their skills.

The ultra-fluid environment places stress on managers as well, since they must organize an increasingly diverse workforce. A noted author and columnist about libraries looks for these traits when recruiting new digital librarians:

- capacity to learn constantly and quickly
- flexibility
- innate skepticism
- propensity to take risks

- public-service perspective
- aptitude for teamwork
- facility for enabling and fostering change
- ability to work independently.

This checklist recognizes the need for digital librarians to adapt to technological change. New standards, protocols, delivery mechanisms, and opportunities are continually appearing and have the potential to reshape the digital workplace beyond recognition.

In fact, whole new lines of business can spring up from nowhere. New software systems can create an entirely new area of responsibility for librarians in the space of just a few years. The growth of institutional repositories in the university and research institute sector is a case in point. An institutional repository is a system that collects, preserves, and disseminates the intellectual output of an institution. It is open to worldwide access, often without any restrictions.

With the advent of institutional repositories, librarians became responsible for soliciting new content, negotiating over copyright, managing rapidly changing software systems, and migrating content between the new systems. The new job title *institutional repository manager* hardly does justice to the complexities of this role. In an amusing article that critically analyses the current state of affairs, Dorothea Salo, Digital Repository Manager of the Wisconsin Digital Collections Center, likens herself to the innkeeper at the Roach Motel: data goes in, but it doesn't come out. She remarks that

repository management is a new subspecialty, so new that most academic librarians of my acquaintance have no idea even how to introduce repository managers to other librarians and (more importantly) to faculty.

The quotations in Table 2.2 give a flavor of the chaos and misunderstanding that surround institutional repositories, and the opposition and—occasionally—praise they evoke. However, this is tangential to our focus here, which is the diversity of skills that staff need. In the same article, a librarian brought in specifically to run the repository is described as a “maverick manager”:

Her job description usually includes policy and procedure development, outreach, training, metadata, maintenance chores such as batch imports, and permissions management; it may include programming, systems administration, or Web design as well.

Table 2.2: Quotations about institutional repositories (from Salo, 2008)

“Institutional repository? Forgive me, but—that sounds vaguely obscene.” (<i>Graduate student in psychology</i>)
“What? No! I’d never want those [preprints] on the web! They’re not authoritative! I’d never use them, either!” (<i>Senior professor of engineering</i>)
“[Engineering faculty] don’t even know the library exists. They never go there; they download all they need. The library doesn’t even register with them.” (<i>Engineering IT manager</i>)
“This is Dorothea Salo. She’s our—she does all kinds of nifty digital stuff.” (<i>Librarian</i>)
“We don’t need to be running all that fancy digital stuff. We need to hire some <i>real librarians</i> .” (<i>Librarian</i>)
“I can put all that in? That’s great! Why haven’t I heard of you before?” (<i>Faculty member, public policy</i>)

Such managers have no well-defined place in the library's organizational structure. They may report to units as varied as special collections, digital collections, or online systems.

The fact that many academic institutions have given their library staff this new role—that of publisher—attests to the uncertainty that surrounds the place of libraries in organizations today. Unfortunately, new responsibility is not necessarily accompanied by new funding—as the “roach motel” metaphor implies. All these factors play havoc with the positions and job requirements of individual librarians.

2.2 Identity

The Internet, in particular the World Wide Web, lends itself well to anonymous access. Rather, it lends itself to the *appearance* of anonymous access. In practice, as many people have discovered to their cost, users are often more identifiable than they realize.

Anonymous use

Librarians have always been concerned about protecting both freedom of expression and the privacy of patrons. With regard to the former, Article IV of the Library Bill of Rights (Figure 2.1) requires librarians to cooperate with all persons and groups concerned with supporting (i.e., “resisting abridgement of”) free expression and free access to ideas. With regard to the latter, Article III of the American Library Association's Code of Ethics, reproduced in Figure 2.2, puts it very clearly: librarians must protect each user's right to privacy and confidentiality with respect to information sought and resources used. Librarians around the world share the concerns of their American colleagues about ethical issues related to both censorship and privacy.

Most people believe that access to information benefits society as a whole. Public library services are provided without profit for society collectively—in other words, they are a “public good.” The economist Paul Samuelson was the first to develop an economic theory of public goods, which he defined as ones that

all [people] enjoy in common in the sense that each individual's consumption of such a good leads to no subtractions from any other individual's consumption of that good.

A so-called “pure” public good has the further property that no individual can be excluded from consuming it, which is particularly pertinent to Web-based digital libraries. It is hard to quantify the value to society of free access to information—or indeed the value of knowledge or education in general.

Libraries typically allow anonymous public access to physical resources, and also to some digital resources. Of course, in order to borrow materials it is necessary to provide some form of identification as surety. However, librarians guard the confidentiality of users to the greatest extent possible, on the basis that if confidentiality is compromised, freedom of inquiry is also compromised. If records of user activities are stored, it is possible that someone may later be able to retrace a user's actions, including the search terms they used and the materials they accessed.

Code of Ethics of the American Library Association

As members of the American Library Association, we recognize the importance of codifying and making known to the profession and to the general public the ethical principles that guide the work of librarians, other professionals providing information services, library trustees and library staffs.

Ethical dilemmas occur when values are in conflict. The American Library Association Code of Ethics states the values to which we are committed, and embodies the ethical responsibilities of the profession in this changing information environment.

We significantly influence or control the selection, organization, preservation, and dissemination of information. In a political system grounded in an informed citizenry, we are members of a profession explicitly committed to intellectual freedom and the freedom of access to information. We have a special obligation to ensure the free flow of information and ideas to present and future generations.

The principles of this Code are expressed in broad statements to guide ethical decision making. These statements provide a framework; they cannot and do not dictate conduct to cover particular situations.

- I. **We provide the highest level of service to all library users through appropriate and usefully organized resources; equitable service policies; equitable access; and accurate, unbiased, and courteous responses to all requests.**
- II. **We uphold the principles of intellectual freedom and resist all efforts to censor library resources.**
- III. **We protect each library user's right to privacy and confidentiality with respect to information sought or received and resources consulted, borrowed, acquired or transmitted.**
- IV. **We respect intellectual property rights and advocate balance between the interests of information users and rights holders.**
- V. **We treat co-workers and other colleagues with respect, fairness, and good faith, and advocate conditions of employment that safeguard the rights and welfare of all employees of our institutions.**
- VI. **We do not advance private interests at the expense of library users, colleagues, or our employing institutions.**
- VII. **We distinguish between our personal convictions and professional duties and do not allow our personal beliefs to interfere with fair representation of the aims of our institutions or the provision of access to their information resources.**
- VIII. **We strive for excellence in the profession by maintaining and enhancing our own knowledge and skills, by encouraging the professional development of co-workers, and by fostering the aspirations of potential members of the profession.**

Adopted June 28, 1997, by the ALA Council; Amended January 22, 2008.

Figure 2.2: American Library Association Code of Ethics

American librarians are particularly concerned about demands being made by the government under the controversial USA Patriot Act, signed into law in October 2001, which increases the ability of law-enforcement agencies to search telephone, e-mail, medical, financial, and other records, including library records. Although some librarians suggest defiance, most agree that federal requests for data should be dutifully complied with, but only when a proper court order is served, and not just because a government agent asks for information. Of course, if fewer records are kept, less information can be provided. The Patriot Act does not require additional record keeping; only that anything that exists must be made available to federal authorities. Libraries usually keep minimal records and have a policy of erasing information immediately after use.

Anonymous access is one way of ensuring that users' privacy is maintained. Patrons in physical libraries usually leave no trace of their actions (although some libraries have installed surveillance

cameras, which itself has raised concerns about invasion of privacy). The same cannot be said for digital access, because electronic fingerprints are left in the user's workstation and the library's information system. However, remote access gives an impression of anonymity, and most users are unaware of the electronic trails they leave. Those who are aware do not worry unduly, because they expect that libraries will not pass on or otherwise misuse personal data.

Their confidence may be misplaced. Privacy issues in traditional physical libraries are clearly defined and well understood, if not always agreed upon. In contrast, the issue of privacy in a digital environment is murky. It is clear that users' privacy is far less shielded from the librarian, or from those who have access to the library files, and users are therefore exposed to greater risk of disclosure from at least two sources: accidental mistakes and government agencies that can compel disclosure. Moreover, with digital technology the usage data that libraries acquire has potential interest not only for law-enforcement and security agencies, but also for commercial organizations and for the library itself, in order to assist with marketing.

Authenticated use

A practical issue in allowing public access to a digital library is whether one user's activity interferes with others. We have all experienced Web sites that are slow to respond to requests. Response time is influenced by a variety of factors (e.g., network speed), but the presence of other users obviously makes systems less responsive. This effect can be mitigated in three ways:

- technical measures, ensuring that the software is configured appropriately
- economic measures, such as purchasing more computer power and faster connections
- social means, such as restricting access to identified users.

The first two measures may have little impact if the service is popular and the user base is uncontrolled.

When establishing a digital library, you must think carefully about whether you are aiming for public access—which inevitably means global access—or a specific target group of users. One compromise that has been adopted by many digital libraries is to permit public access to metadata but to restrict access to the full digital content to registered users.

Three simple methods of restricting services are:

- logical—restricting access to an organization's network domain
- physical—restricting access to particular locations in the real world
- financial—restricting access to users who are prepared to pay (a "paywall").

Users may be asked to identify and authenticate themselves via usernames, passwords, or PINs, or by connecting to the system from pre-distributed software. In addition, users may have to supply bank account or credit card information.

While paid electronic services may seem to inevitably leave audit trails that can be used to trace the user, this is not necessarily so. Strange as it may seem, new information-security methods can arrange anonymous electronic cash transactions and guarantee the user's privacy using mathematical

techniques. These methods provide assurances that have a sound theoretical foundation (in contrast to security that depends on human devices like keeping passwords secret). Even a coordinated attack by a corrupt government with infinite resources at its disposal that has infiltrated every computer on the network, tortured every programmer, and looked inside every single transistor cannot force machines to reveal what is locked up mathematically. In the weird world of modern encryption, cracking security codes is the equivalent of solving puzzles that have stumped the world's best minds for centuries. Whether electronic money transactions leave audit trails is not dictated by technology but remains a choice for society. Anonymity is an option that society has so far declined.

Digital libraries often provide management and administrative functions through Web sites. In these cases, authenticated or restricted use is essential; otherwise, users have the same power as managers and administrators. [Section 2.5](#) explores how this idea can be used to dramatic effect to allow users to contribute to the library. [Section 7.7](#) discusses authentication in more detail.

Recording usage data

The traditional way of recording usage in libraries is to record nothing at all. Nothing, that is, unless books are borrowed, in which case an anonymous date stamp was placed inside the front cover, as [Figure 2.3](#) illustrates, and the librarian made a physical note. However, library lending is now administered by computer-based systems, moving usage records to the digital domain. In fact, digital records apply to many areas of society: daily activities such as credit card purchases, telephone calls, and airline ticket purchases are routinely stored for later analysis through techniques like data mining and cross-database linking. The patterns derived often reveal interesting information. In a digital library they might tell you what users are *actually* doing, as opposed to what you *think* they are doing. However, digital records can lead to unforeseen consequences.

Here's an example. In August 2006, the Internet services company America Online (AOL) released to the research community the records of 20 million user searches. Their intention was laudable: they wanted to advance research on searching methods, which before then had been seriously handicapped by a lack of information about actual user behavior. Of course, all personal information had been removed from the records—or so AOL thought. But pretty soon journalists from the *New York Times* were able to identify that user number 4417749 was in fact Thelma Arnold of Lilburn, Georgia, USA (they sought her permission before exposing her). They did so by analyzing the search terms she used, which apparently ranged from *numb fingers* to *60 single men* to *dog that urinates on everything*. Search by search, they reported, her identity became easier to discern. There were queries for *landscapers in Lilburn, Ga*, several people with the last name *Arnold*, and *homes sold in shadow lake subdivision gwinnett county georgia*, which reporters correlated with public databases, such as phonebooks.

This episode is a graphic illustration of the power of recorded usage data to reveal the identity of real people—even when the data has been anonymized. AOL quickly acknowledged its mistake and recalled the data. The repercussions were severe: their chief technical officer resigned, and two employees were reportedly fired. But you cannot erase information from the Web, and anyone can still download the AOL files from mirror sites. If you wish, you can easily find out more about Thelma Arnold's interests.

THE UNIVERSITY OF TEXAS AT AUSTIN THE GENERAL LIBRARIES PERRY-CASTAÑEDA LIBRARY	
DATE DUE	DATE RETURNED
JUL 10 1990 PCL	RET'D JUL 09 1990 PCL
JAN 14 1991 PCL	RET'D MAY 13 1991 PCL
APR 26 1993 PCL	RET'D MAY 03 1993 PCL
OCT 29 1993 PCL	RET'D OCT 30 1993 PCL
OCT 24 1994 PCL	DEC 03 1994 PCL
FEB 02 1995 PCL	RET'D JAN 12 1995 PCL
APR 11 1998 PCL	RET'D PCL JUN 02 1998

Figure 2.3: Recording usage with date stamps

Web servers and digital library software all routinely include the capability to record the actions of users. A typical Web server log entry looks like this:

```
130.123.128.86 - - [17/Oct/2008:15:56:08 +1300]
"GET /gsdl/cgi-bin/library.cgi?a=q ... &q=snail+farming ... HTTP/1.1"
200 7544
```

Log files can contain millions of such records—one for each time the Web server is accessed.

Here's what this rather mysterious entry means. The four-part number at the beginning is the Internet Protocol (IP) address of the request, which could be the user's computer or, more likely, a "proxy" server somewhere between the user and the Web server. The "--" on the first line sometimes gives the user id of the person requesting the document. This is determined by authentication using the Hypertext Transfer Protocol (HTTP); for almost all Web accesses, it is not defined—as in this case. Following a timestamp that specifies exactly when the request was made, the next part is the request type—in this case GET, which retrieves information from the Web server. (Another possibility is POST, which allows users to submit large quantities of data, such as uploading files.)

The next entry is the resource requested—in this case, to execute a program called `/gsdl/cgi-bin/library.cgi` with certain specified arguments given after the question mark (discussed below), and to return the output. (More commonly, requests give a simple URL without any arguments, in which case the corresponding static Web page is returned.) Following that is the protocol used, in this case HTTP version 1.1. Then comes a status code: the 200 indicates a successful request (whereas 404 is sent when the server can't find the requested information). Finally, the size of the response is indicated—a 7544-byte Web page was returned to the client.

This particular example happens to request information from a digital library—the Greenstone digital library software (the `/gsdl/cgi-bin/library.cgi` part reveals this). In fact, it results in an entry in a second log file, the digital library's own log:

```
/gsdl/cgi-bin/library.cgi its-proxy1.massey.ac.nz
[Fri Oct 17 15:57:28 NZDT 2008]
(a=q, c=demo, l=en, m=50, o=20, q=snail farming, w=utf-8,
z=130.123.128.86-950647871)
"Mozilla/5.0 [en-US] Firefox/3.0.3"
```

Despite superficial differences, this log entry provides much the same information as the Web log entry discussed above. On 17 Oct 2008 a user at `its-proxy1.massey.ac.nz` (this computer has IP address [130.123.128.86]) sent a request to Greenstone. The nature of the request is encoded in the arguments (some were omitted from the earlier example purely to make it more digestible). Interpreting the arguments reveals that the user issued the query *snail farming* (`q=snail farming`). Other arguments request the result page in the English language (`l=en`) to the given query (action `a=q`), when searching the *demo* collection (collection `c=demo`). The user's browser is Firefox version 3.03. Other arguments give the number of search results to be returned (`m=50`), the number displayed per page (`o=20`), and the encoding scheme used (`w=utf-8`). The last argument, `z`, is a "cookie" generated by the Web server: it is comprised of the user's computer's IP number followed by the time that it first accessed the digital library.

Can this information be used to identify the user? We know that the request came from `its-proxy1.massey.ac.nz`, which is a computer at Massey University in New Zealand. Its name indicates that it is a proxy server through which user requests are relayed, rather than a workstation on a particular user's desktop. The proxy server keeps its own log, which we may be able to access—if we have a search warrant. Such information was, of course, removed from the AOL data. The most interesting part is the "cookie." Web cookies are short messages that are sent by a server to a Web browser and then sent back unchanged by the browser each time it accesses that server. This identifies the user—

not by name, or as a particular person, but in a way that allows the system to tell whenever they make a subsequent request. (AOL anonymized the cookies in its data by replacing them with a unique user id—4417749 in the case of Thelma Arnold. However, as she discovered, users often can be identified by their queries.) In this case we might search out snail farmers in Palmerston North, New Zealand (where Massey University is). How many can there be?

Specialist software is used to turn logs into concise summaries of Web server or digital library usage. For example, Figure 2.4a shows the number of searches for 2008 U.S. presidential candidates on Google throughout the election year. The major spike for *Palin* occurred when she was chosen as McCain's vice-presidential running mate; that for *Obama* was when he was elected President. Figure

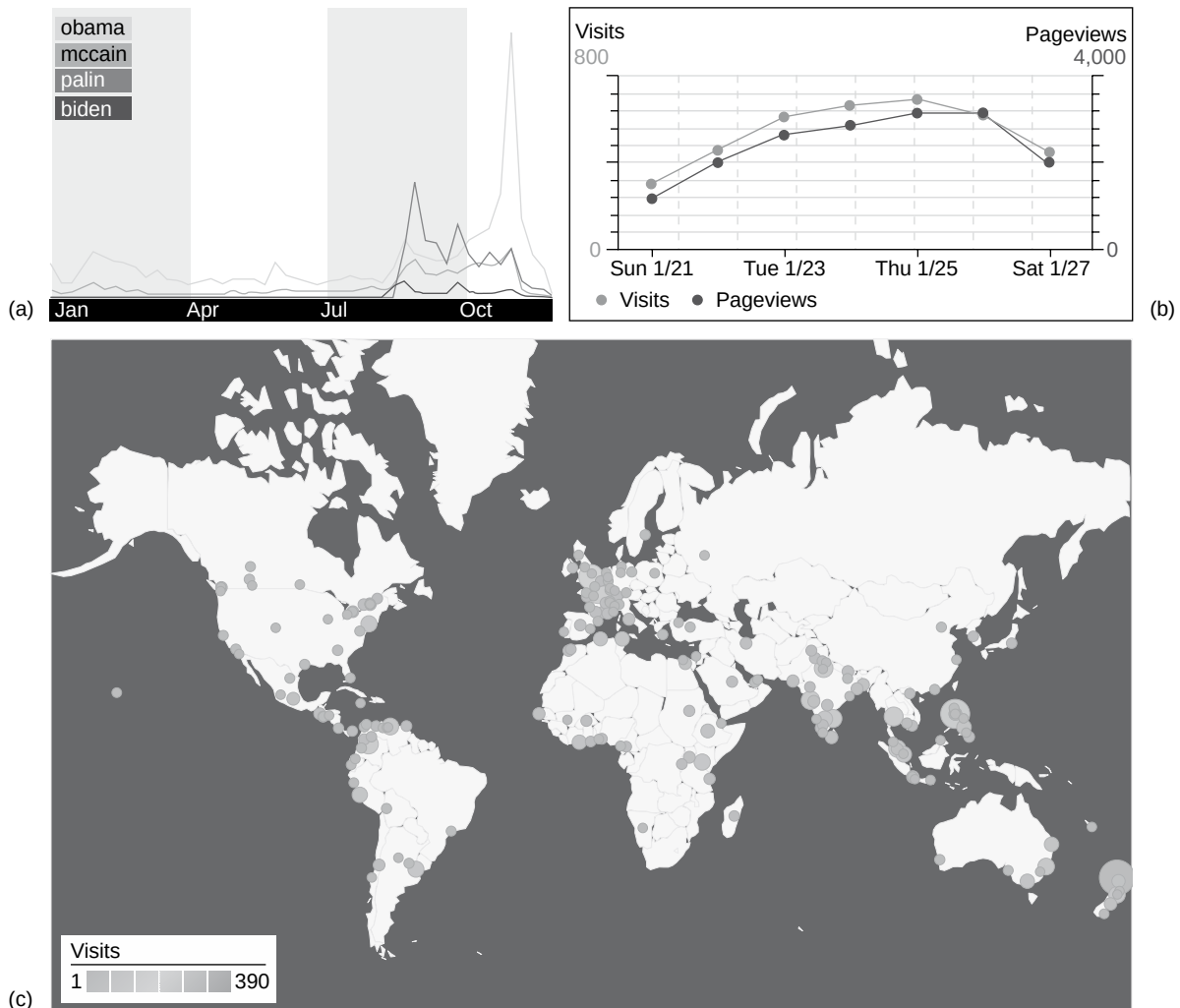


Figure 2.4: User log displays: (a) Google searches for U.S. presidential candidates during 2008 (from <http://www.google.com/intl/en/press/zeitgeist2008/politics.html>; slightly edited); (b) visits to a digital library; (c) geographical distribution of visitors

2.4b graphs the usage of a particular digital library site during a single week in 2007. It plots both *visits* (a total of 3,800), defined as a sequence of requests from a uniquely identified client that expired after 30 minutes of inactivity, and *page views* (a total of 16,800), which are requests made to the Web server for a page. Figure 2.4c shows the geographical distribution of visitors.

Such data is useful for understanding what users are doing. It also reveals what technology they use to access your resources. If you wish to enhance your document presentation using particular features of Web browsers, it is good to know which browsers are actually being used.

2.3 Help and User Support Services

As with most things in life, accessing resources in libraries does not always go smoothly. Sometimes books are not in their expected place. Sometimes the computer system doesn't work properly. And sometimes the readers don't understand the organizational structures that the librarians have constructed. For these and other reasons, libraries establish services specifically to help connect users with resources that match their information needs. These services come in a variety of forms:

- information or help desk
- telephone help lines
- bibliographic instruction
- manuals and help guides.

Users must be physically present in the library in order to benefit from some of these services—like the help desk shown in Figure 2.5. However, over time, technological changes have allowed librarians to provide enhanced service in many different ways. The invention of the electric light extended library hours into the night. The telephone allowed reference librarians to serve patrons remotely. Today, libraries send SMS text messages to mobile phones to alert users that their book loans are about to expire.

With its users distributed around the world, hailing from many cultures, and living in many time zones, how can a digital library provide support services? In a physical library with a real help desk, the number of people who can make enquiries is limited by geography and opening hours—and sometimes by the length of the line at the help desk. In contrast, as libraries go digital, they have “absent users” who work remotely, accessing services via the Web. Therefore, a library on the Web has help services available to anyone who can find them. As the librarian quoted in Section 2.1 observed, this may force library staff to prioritize services or to restrict them to certain users.

Common forms of technology used in virtual reference services include:

- telephone
- fax
- e-mail
- Web forms



Figure 2.5: Library help desk: <http://www.library.dcu.ie/images/shelpdesk.jpg>

- text messaging (i.e., SMS)
- online chat (i.e., instant messaging)
- live voice chat
- live video chat (coming soon).

For example, Figure 2.6 shows a screenshot of an online reference interview in which a patron seeks books by Kurt Vonnegut and a librarian sends her a page from the online catalog. The figure is taken from a round-the-clock reference service that offers real-time one-on-one reference assistance from professional librarians, using Web-based chat, co-browsing, and cooperative reference tools. Offering remote help is a considerable technical and organizational challenge. Although help can be regarded as separate from the actual *content* of a digital library, it is an important element in providing *access* to the content. If its content is not easily accessible, can your digital library truly be regarded as successful?

About a decade ago, forward-thinking librarians began to realize that the survival of reference services in the era of Web search hinged on remote access. Indeed, some search engine companies experimented with such services on a paid basis. For example, from 2001 to 2006 Google offered a “knowledge market” that connected users with part-time researchers who answered their questions for a small fee. The practice came under heavy fire from some quarters because it sold a service that traditionally had been provided by public libraries.

Web-based online reference services are offered by libraries all over the world: for example, *Ask a Librarian*, *Ask a Question*, *Ask Now*, and *Answers Now*. Libraries band together to provide regional coverage, both for local readers and for outsiders who seek information about the region. Users are

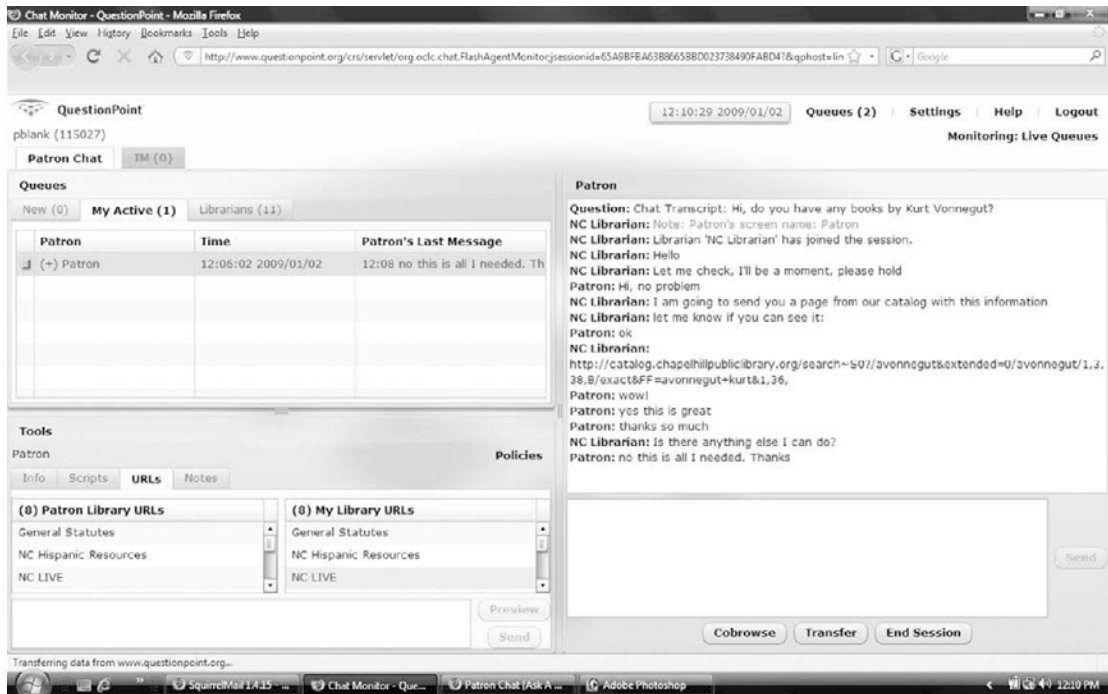


Figure 2.6: Virtual reference desk at NCKnows (www.ncknows.org)

encouraged to contact services close to where they live, and they must begin their inquiry by providing residence details before their question is addressed. However, most virtual reference services have no way of determining whether the user is telling the truth. Some U.S. libraries ask users to enter their ZIP Code to check that they reside in the appropriate area, but there is nothing to stop people from using ZIP Codes they have found in a directory.

Thinking about help and reference services for users highlights the importance of having a clear picture of the role of the library. For example, the Library of Congress's online service states that its "primary mission is to serve Members of the Congress and thereafter, the needs of the government, other libraries, and members of the public." In many digital libraries the situation is not so clear. But you must certainly consider whether you are providing services for members of your organization or for the whole world. And you must determine what form of authorization or identity system you intend to use to distinguish proper users from the random people who find you on the Web. Of course, many organizations already have a mechanism for detecting internal users, in which case it needs to be integrated, if possible, with the software of the digital library system.

2.4 Working with Digital Collections

What is the point of your digital collection? Or, more precisely, why would anyone want to access the content? The beauty of libraries is that, no matter how you answer such questions, it is impossi-

ble to predict who will find your collection useful, or when, or how. The variety of possible interactions leads to the question, What exactly will users want to *do* with your digital collection?

The take-home message is to understand what services you intend your digital library to provide to its users. To do this, you need to understand how users will interact with the content, and balance your desire to support them against the technical and organizational demands of providing ever-richer services.

Using information from digital libraries

A simple answer is that users want to read the documents—or, in the case of multimedia items, view, listen to, and interact with them. A more nuanced answer might take into account what users are trying to achieve. Do they want to save the digital object to their own personal workspace? Do they want to share it with others? Do they want to cut out a portion and use it in a new document of their own? Do they want to provide a link from their Web site to an item in the digital library?

Although it is possible to extract portions of paper documents, doing so often risks damaging the original—and incurring the wrath of librarians. You can quote text from books and papers, but quoting from multimedia objects like video or audio is much more difficult—especially if they are in analogue form. However, it is relatively easy to extract portions of digital items and re-use them. Furthermore, digital copying leaves the original content completely unaffected.

The copy and paste metaphor is familiar to anyone who has used a word processor or image editor. The same principle applies to audio and video, although the programs usually offer more controls. The ease of copying encourages users to extract things from a digital library and to repurpose them in whatever ways they see fit. Once digital content is placed online, it can immediately be copied, legally or otherwise, and used again. And copied again, and so on. Furthermore, it can never be taken back—as AOL discovered when they released their search logs.

Extracting content from a digital library can be made more difficult by using display technologies that attempt to restrict undesirable, unofficial, or illegal usage. (However, the downside is that material becomes less accessible to the intended users.) Some display formats have features that restrict their use. For example, some documents can be secured so that they can be viewed on screen but not printed (e.g., the PDF format described in Section 4.5). As a simple first step, a library should at least make the copyright status of its documents clear to end-users—although, as noted in Section 1.5, the legal considerations may differ from country to country.

Users of a digital library should be clear about any restrictions on how they can use the content. Examples include:

- Can an end-user display video content in a public venue?
- Can audio content be sampled or remixed into new musical works?
- Can search engines index the textual content?

Some digital libraries offer users an opportunity to combine (or “remix”) photographs, graphics, film clips, music, and text to create new multimedia displays. For example, a library might make available

multimedia objects pertaining to some event of local or national significance and invite patrons to use a Web-based video tool to create their own expression of what the event means to them. The newly created artifact could then be entered into the digital library.

Referring to objects in a digital library

Another aspect to the “use” of a digital object is the extent to which it can be included in the Web’s link structure. Having found something of interest, a library user might bookmark it or link to its URL. But will the URL change if other items are added to the library—that is, will it be *persistent*? Valid hyperlinks are the glue that holds the Web together, and references are much less effective if they break easily. The same question can also be asked about searches (Will the list of results for a particular search term in a digital library be persistent?) and about parts of the browsing structure provided by the library (How persistent is the list of documents whose titles begin with the letter A?).

When evaluating digital library software, you should consider what happens to the URLs of objects in the library when:

- new items are added to a collection
- existing items are deleted
- the digital library’s Web server is reorganized
- the collection is moved to another computer
- the collection migrates to a different software system.

As their name indicates, URLs (Uniform Resource Locators) are *locators* that specify how to find the file containing the information. Fortunately, it is possible to put a “redirect” on the Web server that automatically redirects users to a different location, that is, a different URL, when the original URL changes. Because they are locators, URLs do not usually survive radical events such as server reorganizations.

Many schemes have been devised to make identifiers of digital objects persist over time despite organizational changes in the underlying software systems, Web servers, and their location on the Web. We discuss these schemes in Chapter 7.

Berry-picking

When people interact with information-retrieval systems, they typically encounter not just one, but several, items of interest that they wish to pursue further. Many systems provide mechanisms for users to maintain a cache of interesting items:

- bookmarks or favorites in Web browsers
- shopping carts in e-commerce stores
- marked or tagged lists of records in library catalogs.

Here we call such lists of interesting items *berrybaskets*—a term that evokes the idea of picking the ripest and juiciest fruit from a bush—and are essential for the effective use of large collections. Figure 2.7 shows an example of this, taken from the Library of Congress’s vast online catalog. In the figure, a user is browsing the listed works of Noam Chomsky (there are over 190). Check boxes are provided down the left-hand side to select items of interest and our user, interested in the topic of propaganda, has selected relevant works. Once satisfied with the selection (perhaps visiting subsequent pages) she may save it, e-mail it to herself or a colleague, or print it as a paper record of the library visit. Commercial systems typically provide the option to purchase the items on the list.

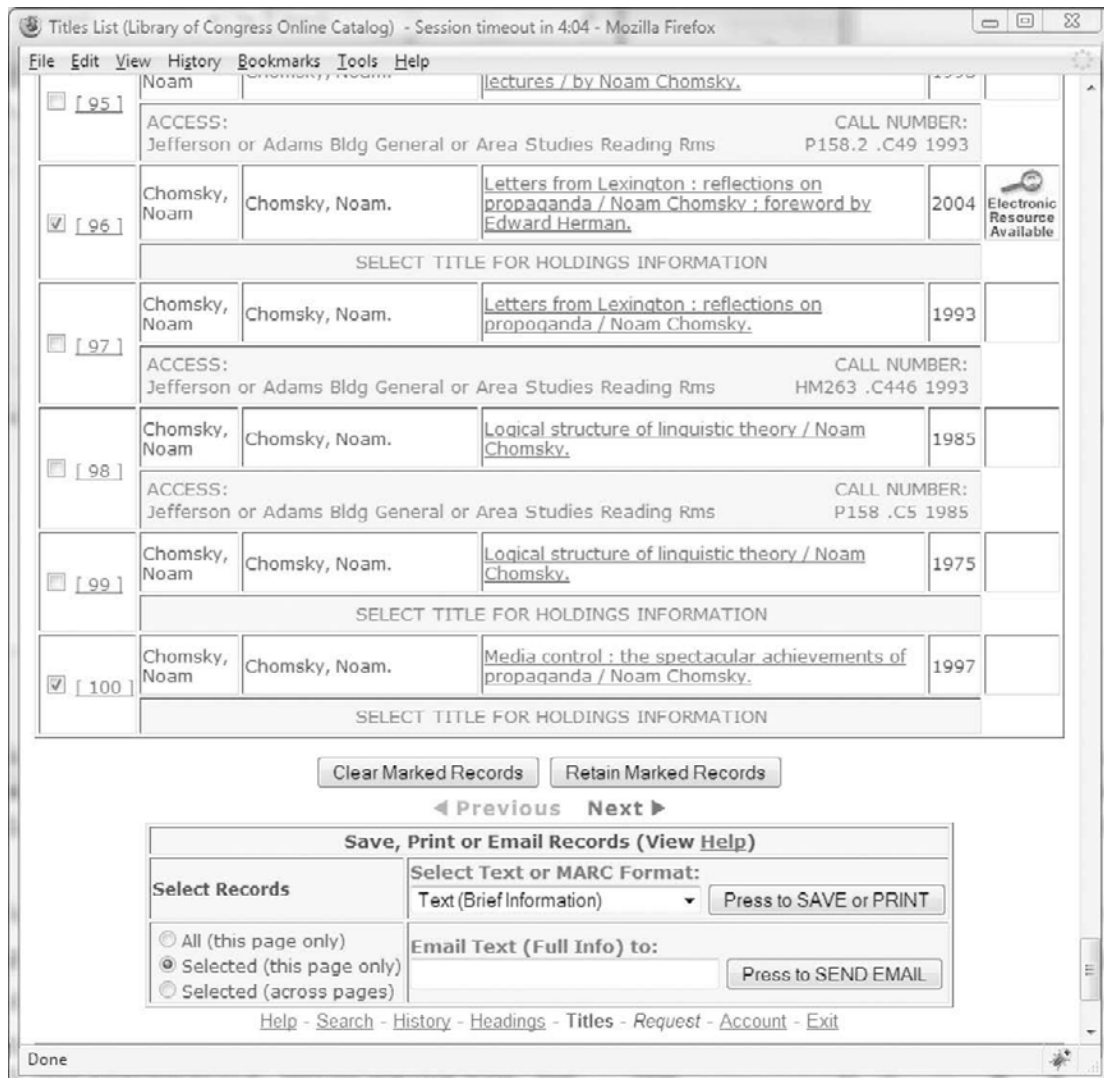


Figure 2.7: Berrybasket support provided by the Library of Congress’s online catalog

Berrybaskets raise the same questions about identity that were addressed in [Section 2.2](#). Are users—and therefore their baskets—anonymous, or do they need to provide authentication? As knowledge work becomes increasingly collaborative, people find it useful to share their lists, either publicly or with specific groups of friends or colleagues.

2.5 User Contributions

Ask not what your library can do for you, ask what you can do for your library.

Paraphrased from President Kennedy's inaugural address, 1961

Although the ALA Bill of Rights in [Figure 2.1](#) asserts that libraries are forums for information and ideas, we have seen that libraries are really providers of services and users are consumers of resources. Most libraries offer little scope for users to *contribute* content or metadata. However, many Web sites have shown that offering users a more active role in collection development can yield startling benefits.

Even before the advent of the Web, some were suggesting that user-supplied data might begin to address the information overload we all experience. Of course, librarians traditionally discourage patrons from adding value to paper materials (or defacing them, as librarians tend to see it). Neither did librarians permit users to add their own cards to the card catalog. However, with digital content, readers can choose whether to access the original version or a user-enhanced one, because user contributions are stored separately from the original content and can be combined or viewed separately as the occasion demands. An example application of this is illustrated in [Section 12.2](#) (Design pattern 5) where a user can augment a sheet music digital library with “post-it” style annotations.

Most digital library systems are still conceived of as read-only repositories, where users are consumers rather than contributors. But the much-heralded Web 2.0 revolution's aim is to harness social effects and to create Web applications that improve as more people use them. In library terms, this requires a change of mindset from creating information for patrons' use to creating an initial structure that patrons can supplement. Thus, libraries can evolve from exclusive suppliers of information that users consume into a partnership where both the library and its users supply material.

What kind of information could users supply? Here are some ideas.

Annotations

Many readers add notes in the margins of their books: to highlight important points, to make links with another concept (in another book), or to disagree with the printed text. The annotations may be personal, but they do not have to be: they could be shared with other readers. Some people might prefer annotated books over clean ones—particularly if they respect the annotator's opinion on the topic.

Keywords

A special form of annotation is the addition of keywords or tags. Keywords are terms that provide a useful summary of important topics associated with a document and can be used in a digital library

to enhance searching and browsing. Keywords and tags build bridges between the documents and the user's vocabulary and also create new associations between documents—just as traditional author-assigned keywords do.

Ratings

Ratings are another special kind of annotation: they are a quantitative assessment on a particular scale, usually numeric. We are familiar with ratings for all kinds of everyday goods and services, from films to laptops to airline food. Ratings are easy to add to a document, and numeric ones lend themselves to computational processing, such as averaging and sorting. Amazon and other e-commerce sites use ratings to enhance their display of search results, and, together with usage data, to personalize results for individual users.

Corrections

Users can also supply error reports, signaling to the librarians that something is wrong with the collection. This can be as simple as a single click marking a piece of metadata as inaccurate—perhaps a typo in the author's name. For error correction, the user interface must allow users to flag errors easily, and the interface must be backed up by an infrastructure that summarizes the feedback and communicates it to librarians. Most digital libraries do not allow users to contribute in this manner, missing out on a potentially valuable source of quality improvement.

New documents

Finally, users can, in principle, add new documents, although collection development is usually regarded as a role that needs to be performed, or at least moderated, by a trained librarian. Many Web technologies have allowed unstructured and unmoderated groups of documents to proliferate on the Internet. However, there is a middle ground between a read-only library and a chaotic user-supplied document dump.

A common example of user-initiated document submission integrated with librarian oversight is the institutional repository discussed in [Section 2.1](#), and demonstrated later in two worked examples (Sections 7.6 and 11.6). As with annotations and ratings, user-initiated document submission relies on digital library software to support both users and librarians in expanding the collection.

Partial and fluid documents

As mentioned in Section 1.2, in the first half of the 20th century, both H. G. Wells and Vannevar Bush prophesied the emergence of new and all-encompassing forms of encyclopedias and libraries. Both foresaw user contributions as the key to creating and maintaining such structures. As Vannevar Bush put it,

Wholly new forms of encyclopedias will appear, ready made with a mesh of associative trails running through them.... There is a new profession of trail blazers, those who find delight in the task of establishing useful trails through the enormous mass of the common record.

The most striking example of user contributions on the Web (or anywhere else) is the Wikipedia project. Launched in 2001 with the goal of building free encyclopedias in all languages, Wikipedia is today easily the largest and most widely used encyclopedia in existence. Wikipedia contains 10 million articles in 250 different languages. The English-language version contains 2.5 million articles totaling around 1 billion words (in 2008). Debates flare up about the quality of the articles and the reliability of the content, but Wikipedia has become a valued reference source for many Web users.

From its inception, the project offered a unique, entirely open, collaborative editing process, scaffolded by then-new wiki software for group Web-site building, and it is fascinating to see how things have flourished under this regime. Wikipedia has effectively enabled the entire world to become a panel of experts, authors, and reviewers—contributing under their own name, or, if they wish, anonymously.

Wikipedia's growth is astonishing: the acquisition of 1 billion words in seven years requires the addition of an average 400,000 words, or about five full-length novels, per day. One key reason for this growth is the absence of technical barriers: if you can view an article in a Web browser, then you can change it (all you need to do is to click the *edit* button at the top of the screen). Another advantage is that users can make incremental changes. An edit can add (or delete) large amounts of text or a single character. Text that is deleted is not removed from the system: it can be easily reinstated.

Figure 2.8 shows the beginning of the Wikipedia article *Library*. Its revision history shows that it was created on 9 November 2001 in the form of a short note (which, in fact, bears little relationship to the current version) and has been edited about 1500 times since then. Recent edits have added new links and new entries to lists, have indicated possible vandalism and its reversal, have corrected spelling mistakes, etc.

Wikipedia is very much a community effort. Each article has a discussion page that provides a forum for debate about how it might be criticized, improved, or extended in the future. The discussion page for *Library* is almost as long as the article itself and contains the following observations, among many others:

Libraries can also be found in churches, prisons, hotels etc. Should there be any mention of this?

Daniel C. Boyer 20:38, 10 Nov 2003

Libraries can be found in many places, and articles should be written and linked. A wiki article on libraries can never be more of a summary, and will always be expandable.

DGG 04:18, 11 September 2006

Discussion pages are a unique feature of Wikipedia that has no analog in traditional encyclopedias.

Wikipedia will never be finished. While this means that anyone can corrupt articles by adding untrue or irrelevant statements, and nefarious users can even vandalize them, it also means that the information can be augmented whenever the world changes. Contrast this with books, which are potentially out of date before they are printed.

Therefore, the fluidity of Wikipedia articles contrasts with the fixity of traditional library materials. User involvement in knowledge creation and distribution is arguably a vitally important innovation.

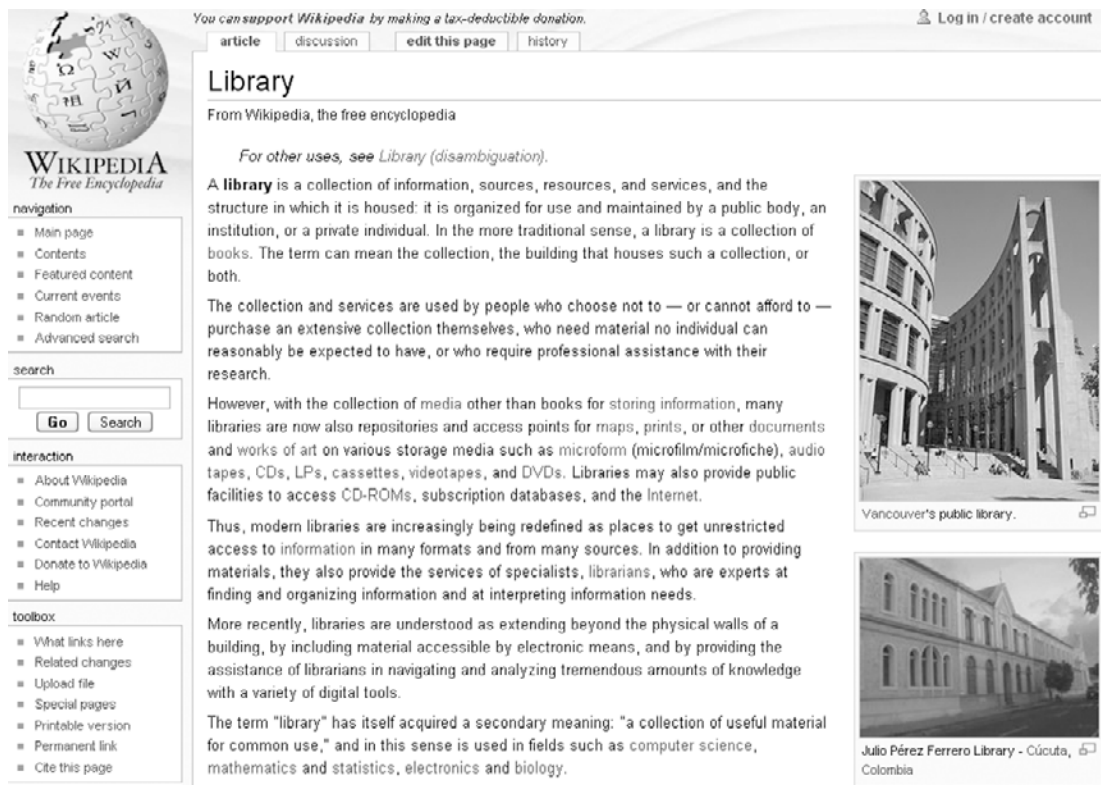


Figure 2.8: Wikipedia article *Library*

And user involvement requires the role of the digital librarian to expand again, to provide an infrastructure for knowledge production, rather than merely preserving existing content.

2.6 Notes and Sources

S. R. Ranganathan's classic book is called *The Five Laws of Library Science* (1931). The Bill of Rights in Figure 2.1 and Code of Ethics in Figure 2.2 can both be found at the American Library Association's Web site (www.ala.org). Rubin (2001) discusses ethical aspects of reference services. Anderson (2006) gives a detailed discussion of the interaction between ethics and digital libraries.

The quotation in Section 2.1 from a practicing librarian comes from Szymanski and Fields (2005). For a frank account of running an institutional repository read Salo's (2008) priceless "Innkeeper at the Roach Motel." The desired staff traits are from Roy Tennant (2004, pp. 150-151). Thornton (2000) explores the impact of electronic resources on collection development and foresees that consortia will become even more important because electronic resources, unlike traditional ones, can easily be shared.

The definition of public good at the beginning of [Section 2.2](#) is from Paul A. Samuelson's classic 1954 paper, "The Pure Theory of Public Expenditure." McCallum and Quinn (2004) and Holt et al. (1996) discuss attempts to quantify the benefits of libraries. The name of the USA Patriot Act is an acronym for Uniting and Strengthening America by Providing Appropriate Tools Required to Intercept and Obstruct Terrorism (USA PATRIOT). The Act took effect in 2001 and was so controversial that it had to be renewed by the U.S. Senate every year; most of its provisions were finally made permanent in 2006. The Electronic Privacy Information Center (EPIC) has more information (www.epic.org).

Barbaro and Zeller (2006) were the *New York Times* reporters who outed AOL user number 4417749. Many instances of unintentional information leakage to search engines relate to so-called "vanity searches," which are searches for information about oneself (Soghoian, 2007). Sturges et al. (2003) discuss the parameters of privacy in the digital library environment.

The historical observation at the beginning of [Section 2.3](#) that librarians have been able to expand services with each new technology is from Barnello (1996), who succinctly surveys more than a century of innovation in reference libraries. The "absent user" and virtual reference desks are discussed by Martell (2008), Kibbee (2006), Lankes et al. (2006), Courtney (2003), and Katz (2003). Beranek and Burke (2006) describe the development of a library telephone enquiry service. Sugimoto (2008) discusses the quality of reference transactions in academic music libraries. [Figure 2.6](#) shows the QuestionPoint software system from OCLC. There's a useful list of "Ask a Librarian" services all over the world at <http://askaquestion.ab.ca/referral.html>. Pomerantz (2005) gives a comprehensive discussion of chat-based reference services.

Here is an example of both user contributions and remixing multimedia content. Around the 90th anniversary of the signing of the armistice that marked the end of World War I, the Auckland Museum in New Zealand invited patrons to help commemorate the anniversary. The museum made available relevant photographs, graphics, film clips, and music. Through a Web-based interface, patrons could craft their own expression of what the coming home of soldiers meant to them and upload it to the site to share with others. The result can be found at <http://remix.digitalnz.org/>.

Nelson and Allen (2002) describe an interesting study of object persistence and availability in actual digital libraries. Bates (1989) is the classic reference for the idea of berrybaskets.

Before the spread of the World Wide Web, Michael Koenig (1990) suggested that user-supplied metadata is a key mechanism for addressing information overload. Annotations are thoroughly explored by Marshall (1998) and Marshall and Brush (2004). The quotation about new forms of encyclopedia is from "As We May Think," by Vannevar Bush (1945). To learn about Wikipedia, we recommend, well, the Wikipedia article on it: <http://en.wikipedia.org/wiki/Wikipedia> (available in 202 languages at the time of writing).

Presentation: User interfaces

From People to Presentation

Having reviewed the people in digital libraries and the roles they play, now we turn to presentation and what global users experience when interacting with digital libraries, which they invariably do through a Web browser.

Recall the definition of *digital library* from Chapter 1. The definition begins by stating that a digital library is:

a focused collection of digital objects, including text, video, and audio ...

and a good place to start is with the objects themselves (the documents) and how they appear on the user's screen. The next section illustrates different document displays in digital library systems. There are many possibilities, and we include a representative cross-section: structured text documents, unstructured text documents, page images, page images along with the accompanying text, audio, photographs, and videos. A digital library might include objects that manifest themselves in different forms. For example, music has many alternative representations—synthesized audio, music notation, page images, recorded audio performances—and digital libraries must cater to these optional views.

In addition to documents, digital libraries include metadata like that used conventionally for bibliographic organization. The role of metadata is considerably expanded in a digital library, and we examine some examples to convey how useful it is in helping to organize digital library collections. (Chapter 6 provides a more detailed account of metadata.)

The definition goes on to say:

... along with methods for access and retrieval ...

and this chapter illustrates different methods for access and retrieval. Conventionally these are divided into *searching* and *browsing*, although in truth the distinction is not sharp. We first examine interfaces that allow you to locate words and phrases in the full text of a document collection. Searching is also useful for metadata—such as finding words in title and author fields—and we look at interfaces that allow these searches, and combinations of them, to be expressed. It is often useful to be able to recall and modify past searches: *search history* interfaces allow you to review what you have done and reuse parts of it. Next we examine browsing interfaces for metadata—titles, authors, dates, and subject hierarchies—and show how to capitalize on the structure that is implicit within the metadata itself.

But searching and browsing aren't really different activities: they are opposite ends of a spectrum. In practice, people want to be able to interact with information collections in different ways, some involving searching for particular words, some involving convenient browsing—perhaps browsing the results of searching, or exploring various facets of the information being presented.

The final part of the definition is:

... and for selection, organization, and maintenance of the collection.

This chapter does not address this activity, except insofar as browsing structures reflect the organization of the collection. In truth, this entire book is about organizing and maintaining digital libraries, including organizing them in such a way that they are easy to maintain.

Most of the illustrations in this chapter show screen shots from a Web browser. However, the browser does not show a static Web page: the digital library software constructs pages dynamically at the time they are called up for display. The navigation bar at the top of [Figure 3.1a](#), the cover picture at the top left, the table of contents on the right, and the book title near the bottom, are all composed together with the text at display time, every time the page is accessed. The information is typically stored in some form of managed datastore, or else reconstructed on the fly from compressed index files. If you look on the computer that is the Web server for this digital library, you will not find this page stored there.

This is why in Chapter 1 we distinguish a digital library from a Web site, even one that offers a focused collection of well-organized material. The fact that documents are treated as structured objects internally enhances the prospects for providing comprehensive searching and browsing facilities. Furthermore, because all pages are generated dynamically from an internal representation, it is easy to change the entire look and feel of all pages associated with a collection without regenerating or even touching the content of the collection.

3.1 Presenting Textual Documents

If you want to build a digital library, the first questions that need to be answered are: What form are the documents in? What structure do they have? How do you want them to look?

Documents, chapters, sections

The book shown in [Figure 3.1](#), *Village-Level Brickmaking*, is from the Humanity Development Library mentioned in Chapter 1. A picture of the front cover is displayed on the left, and the table of contents appears to its right. Below is the start of the main text, beginning with title, author, and publisher. The books in this collection are generously illustrated. On the screen these images appear in-line, just as they did in the paper books from which the collection was derived. [Figures 3.1c and d](#) show some of the images, obtained by scrolling down from [Figure 3.1b](#).

Books in this collection have front-cover images, which appear at the top of any page where the book, or part of it, is displayed. This picture gives a feeling of physical presence, a reminder of the context in which you are reading. The user interface may be a poor substitute for the look and feel of

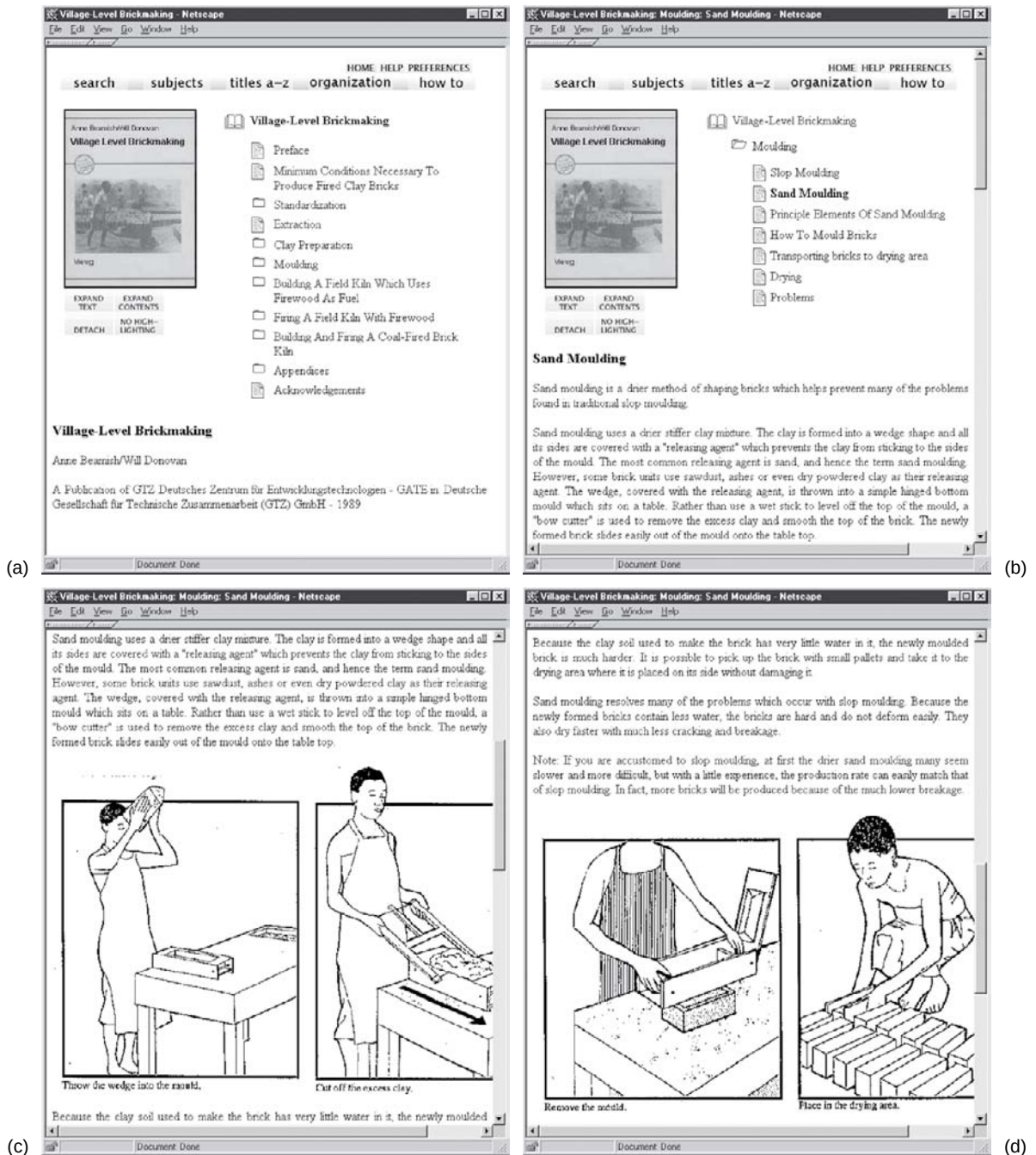


Figure 3.1: *Village-Level Brickmaking*: (a) the book; (b) the chapter on *Moulding*; (c, d) some of the pages. Beamish, A., and Donovan, W. *Village-Level Brickmaking*. Copyright © 1989 GTZ Deutsches Zentrum für Entwicklungstechnologien-GATE in Deutsche Gesellschaft für Technische Zusammenarbeit (GTZ) GmbH

a physical book—the heft, the texture of the cover, the crinkling sound of pages turning, the smell of the binding, highlighting and marginal notes on the pages, dog-eared leaves, coffee stains, the pressed wildflower that your lover always used as a bookmark—but it’s a lot better than nothing.

The books in the Humanity Development Library are structured into chapters and sections. The small folder icons in [Figure 3.1a](#) indicate chapters—there are chapters on *Standardization*, *Clay Preparation*, *Moulding*, and so on. The small text-page icons beside the *Preface*, *Extraction*, and *Acknowledgements* headings indicate leaves of the hierarchy: sections that contain text but no further subsection structure.

Clicking on *Moulding* in [Figure 3.1a](#) yields the page in [Figure 3.1b](#), which shows the chapter’s structure in the form of a table of contents. Here the user has opened the book to *Sand moulding* by clicking its text-page icon; the section heading is shown in bold and its text appears below. Other headings lead the reader to such topics as *Slop moulding*, *How to mould bricks*, and *Drying*. You can read the beginning of the *Sand moulding* section in [Figure 3.1b](#): the scroll bar to the right of the screen indicates that more text follows. [Figures 3.1c and d](#) show the effect of scrolling further down the page.

The Expand Contents button in [Figure 3.1](#) expands the table of contents into a full hierarchical structure. Similarly, the Expand Text button expands the text of the section being displayed. In [Figure 3.1a](#) it would yield the text of the entire book, including all chapters and subsections; in [Figure 3.1b](#) it would yield the complete text of the *Moulding* chapter, including all subsections. This is convenient for printing the whole book or sections of it. Finally, the Detach button duplicates this window on the screen, so that you can retain its text while continuing to browse the library in the other window. This is useful for comparing multiple documents.

As noted in Chapter 1, the Humanity Development Library is a large compendium of practical material. It covers diverse areas of human development, from agricultural practice to foreign policy, from water and sanitation to society and culture, from education to manufacturing, from disaster mitigation to microenterprises. This material was carefully selected and put together by a collection editor who acquired the books, arranged for permission to include each one, organized a massive optical character recognition (OCR) operation to convert them into electronic form, set and monitored quality-control standards for the conversion, decided what form the digital library should take and what searching and browsing options should be provided, entered the metadata necessary to build these structures, and checked the integrity of the information and the look and feel of the final product. The care and attention put into the collection is reflected by its high quality. Nevertheless, it is not perfect: there are small OCR errors, and some of the 30,000 in-text figures (of which examples can be seen in [Figures 3.1c and d](#)) are inappropriately sized. The amount of effort required to edit a high-quality collection of a thousand or more books is staggering—just ask a publisher what goes into the production of a single conventional book like the one you are reading.

Unstructured text documents

[Figure 3.2](#) shows screen shots from a far plainer collection. The documents are not presented in a hierarchical way. There are no front-cover images. In place of [Figure 3.1](#)’s picture and table of con-

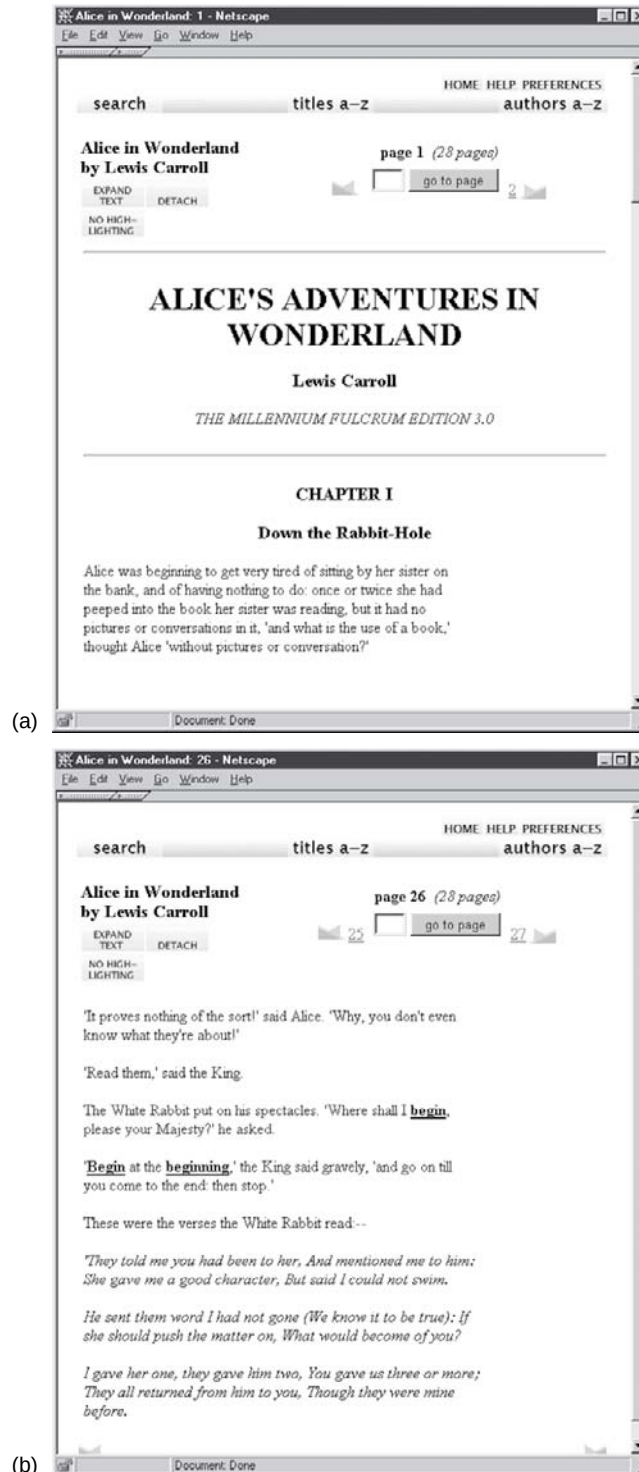


Figure 3.2: *Alice's Adventures in Wonderland*: (a) the beginning; (b) finding a quotation

tents, what [Figure 3.2](#) shows is more prosaic: the title of the book and a page selector that lets you turn from one page to another. Browsing is less convenient because there is less structure to work with. Even the “pages” do not correspond to physical pages, but are arbitrary breaks made by the computer every few hundred lines—that’s why *Alice’s Adventures in Wonderland* has only 28 pages. The only reason for having pagination at all is to prevent the Web browser from downloading the entire book every time you look at it.

In fact, this book *does* have chapters—in [Figure 3.2](#) you can see the beginning of Chapter 1, *Down the rabbit-hole*. However, this structure is not known to the digital library system: the book is treated as a long scroll of plain text. With some extra effort in setting up the collection, it would have been possible to identify the beginning of each chapter, and its title, and incorporate this information to permit browsing chapter by chapter, as has been done in the Humanity Development Library. The cost depends on how similar the books in the collection are to one another and how regular the structure is. For any given book, or any given structure, it is easy to do; but in real life large collections usually exhibit considerable variation in format. As we mentioned before, the task of proofreading thousands of books is not to be undertaken lightly.

The books in this collection are stored as raw text, with the end of each line hard-coded, rather than, say, in HTML, which is used for [Figure 3.1](#). (Chapter 4 gives details of these formats.) That is why the lines of text in [Figure 3.2](#) are quite short: they always remain exactly the same length and do not expand to fill the browser window. Compared with the Humanity Development Library, this is a low-quality, unattractive collection. Removing end-of-line codes would be trivial, but a simple removal algorithm would destroy the format of tables of contents and displayed bullet points. It is surprisingly difficult to do such things reliably on large quantities of real text—reliably enough to avoid the chore of manual proofreading.

[Figure 3.2b](#) shows page 26 of the book, where the words *begin* and *beginning* are highlighted in boldface. This is because the page was reached by a text search of the entire library contents (described in [Section 3.4](#)) to find a particular quotation. The system highlights search terms: there is a button at the top that turns highlighting off if it becomes annoying. In contrast, standard Web search engines do not highlight search terms in documents—of course, they do not serve up the target documents themselves but instead direct the user to the original source location.

Alice’s Adventures in Wonderland belongs to a collection called Project Gutenberg, whose goal is to encourage the creation and distribution of electronic text. Although the project was conceived in 1971, work on it did not begin in earnest until 1991, with the aim of producing 10,000 electronic texts within ten years. The first achievement was an electronic version of the U.S. Declaration of Independence, followed by the Bill of Rights and the Constitution. Then came the Bible and Shakespeare—unfortunately, however, the latter could not be released until much later because of copyright restrictions on the comments and notes in the particular edition that was entered. The collection was planned to double each year, with one book per month added in 1991, two in 1992, four in 1993, and so on, reaching the target of 10,000 by 2001. This schedule slipped slightly, but the target was passed in 2003.

A huge boost came from a development known as *distributed proofreading*, which provides a perfect example of the role of user contributions mentioned in [Section 2.5](#). Optical character recognition

(OCR) software is used to digitize volumes *en masse*. Then a global community of volunteers proof-reads the result and corrects errors using a specially designed Web site. Upon logging in, registered volunteers are presented with a scanned page and the corresponding text in editable form, for correction. Once corrections have been made, a second volunteer verifies the work. Developed in 2000, this approach became an official part of Project Gutenberg two years later. The Project Gutenberg library now boasts over 26,000 digitized texts and continues to grow fast.

Project Gutenberg is a grassroots phenomenon. Text is input by volunteers, each of whom can enter a book a year or even just one book in a lifetime. The project does not direct the volunteers' choice of material; instead, people are encouraged to choose books they like and to enter them in the manner in which they feel most comfortable. Central to the project's philosophy is to represent books as plain text, with no formatting and no metadata other than title and author. Professional librarians look askance at amateur efforts like this, and indeed quality control is a problem. However, dating back two decades before the advent of the World Wide Web, the Gutenberg vision is remarkably far-sighted and gives an interesting perspective on the potential role of volunteer labor in placing society's literary treasures in the public domain.

The collection illustrated in [Figure 3.2](#) represents the opposite end of the spectrum to the Humanity Development Library. It took just a few hours to download the Project Gutenberg files and to create the collection, and a few hours of computer time to build it. Despite this tiny investment of effort, it is fully searchable—which makes it indispensable for finding obscure quotations—and includes author and title lists. If you want to know the first sentence of *Moby Dick*, or whether Hermann Melville wrote other popular works, or whether “Ishmael” appears as a central character in any other books, or the relative frequencies of the words *he* and *her*, *his* and *hers* in a large collection of popular English literature, this is where to come.

Page images

[Figure 3.3](#) shows a historical collection of literature written for schoolchildren, the New Zealand School Journal, which is delivered to schools throughout New Zealand by the Ministry of Education. Dating from 1907—the first cover is the top left image in [Figure 3.3](#)—it is believed to be the longest-running serial publication for children in the world. [Figure 3.3a](#) shows the collection's home page: you click on an image to get to that issue of the journal. [Figure 3.3b](#) shows a page of the children's story “Never Shout at a Draft Horse,” represented not as text but as a facsimile of the original printed version—a decision made by the collection's designer. From a technical point of view, this decision makes a big difference: the textual content occupies only about 5 percent of the storage space required for a page image, greatly reducing the resources required to store the collection and the time needed to download each page. Of course, the picture of the horse would have to be represented as an image, just as the pictures in [Figures 3.1c and d](#) are, sacrificing some of the space gained.

A good reason for showing page images rather than extracted text is that the OCR process that identifies and recognizes the text content makes errors. When children are the target audience, it is important not to expose them to erroneous text. Of course, errors can be detected and corrected manually,

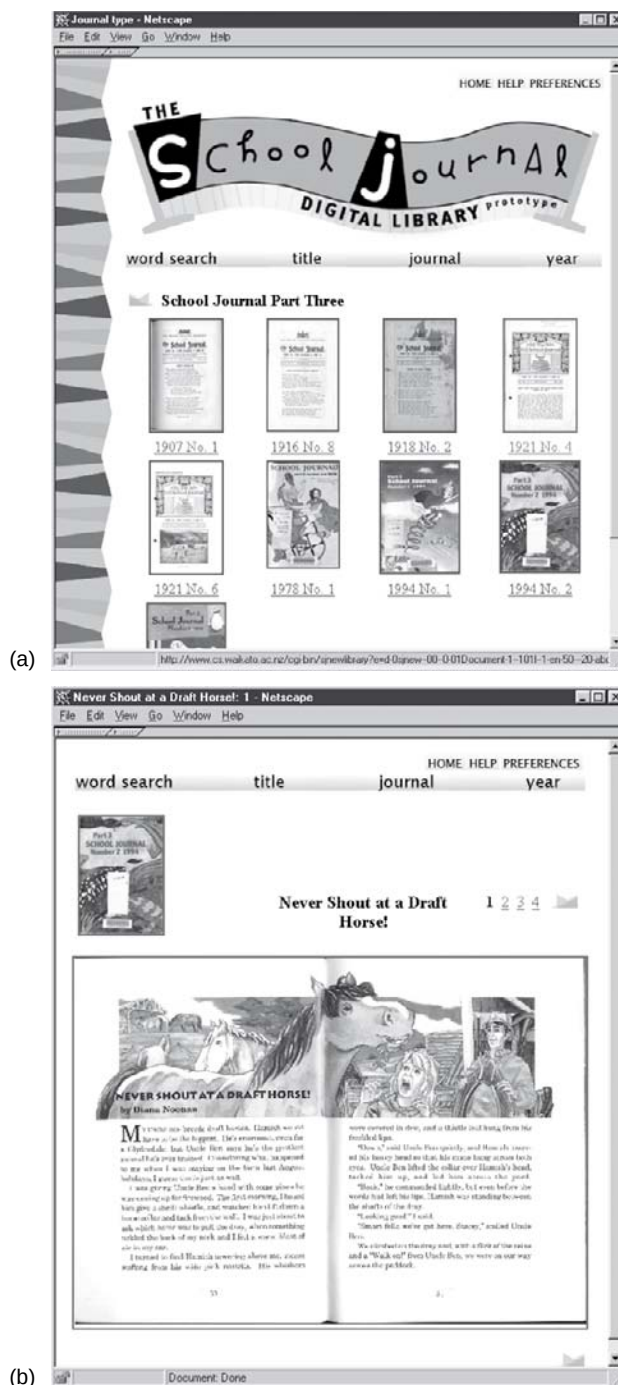


Figure 3.3: School Journal Digital Library (Learning Media Limited, Wellington, New Zealand): (a) home page; (b) the story “Never Shout at a Draft Horse!” by Diana Noonan. New Zealand School Journal, 1994, Part 3, No. 2

as in the Humanity Development Library and the Gutenberg collection, but at a substantial cost well beyond the resources that could be mustered for this particular project.

Text is indeed extracted from the New Zealand School Journal pages using OCR, and that text is used for searching, but readers never see it. The consequence of OCR errors is that some searches may not return all the pages they should. If a word on a particular page is misrecognized, a search for it will not return that page. Or, if a particular word is misinterpreted as a different one, a search for *that* word will return an extra page. However, neither of these errors was seen as a big problem—certainly not as serious as showing children corrupted text.

Figure 3.3b shows the journal cover at the top left and a page selector at the right that is more convenient to use for browsing around in the story than the numeric selector in Figure 3.2. The stories are short: “Never Shout at a Draft Horse” has only four pages.

Images with text

Figure 3.4 demonstrates a far larger collection of page images that also allows searching on extracted, OCR’d text, but in this case users are able to see the text if they so desire. This is the National Library of New Zealand’s Papers Past collection, which has 1.1 million pages of digitized national and regional newspaper and periodicals spanning the years 1839–1920.

A user interested in the suffragette movement has entered the query *Kate Sheppard* (Kate Sheppard was a prominent 19th-century campaigner who successfully campaigned to have New Zealand become the first country where women could vote). Figure 3.4a shows the beginning of one of the articles returned by the search, “To the Freewomen of New Zealand,” from p. 46 of the *Otago Witness* (Issue 2066, 28 September 1893). Notice that (unlike in the School Journal collection) the search terms are highlighted in this image. During the OCR process, the precise location of each word in the source image is stored, along with coordinate information for each article. This allows search terms to be highlighted and individual articles to be clipped out of the newspaper pages.

A link at the top takes the reader to the text shown in Figure 3.4b. Some errors are apparent, most notably the word *names*—the last word shown in Figure 3.4a—has been rendered as *niiniis*. The decision to allow readers to see this text was a good one, for then they can see what they are searching. It is also a courageous decision because it exposes errors, and most digital libraries of newspapers conceal this information. In this case, the word *names* in Figure 3.4a is badly broken up: no wonder it was misrecognized.

Below the main navigation bar is hierarchical information that pinpoints where the user is: *Papers Past > Otago Witness > 28 September 1893 > Page 46 > TO THE FREEWOMEN OF NEW ZEALAND*. Clicking *Page 46* switches from article to page mode, shown in Figure 3.4c, and the current article is highlighted. Moving the cursor around changes the highlighting from one article to another; clicking switches the highlighted item to article view.

From Figure 3.4c readers can view both a high-resolution image and a printable version. They can navigate page by page through the newspaper, and from one edition to the next. Figure 3.4d shows the contents of this edition of the *Otago Witness*, providing a useful overview. At the top, the user

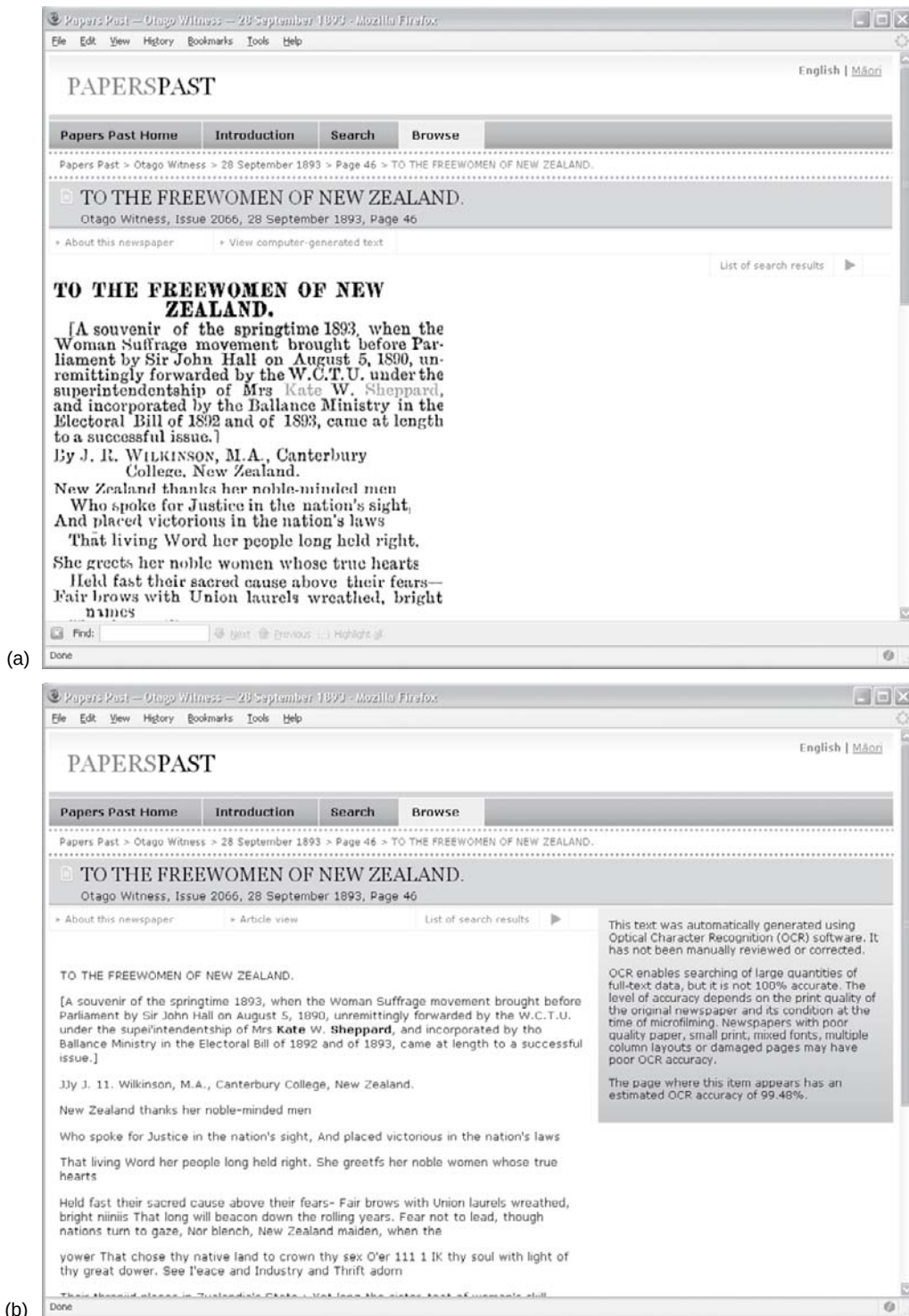


Figure 3.4: The National Library of New Zealand's Papers Past: (a) viewing a newspaper; (b) text view;

(c) 

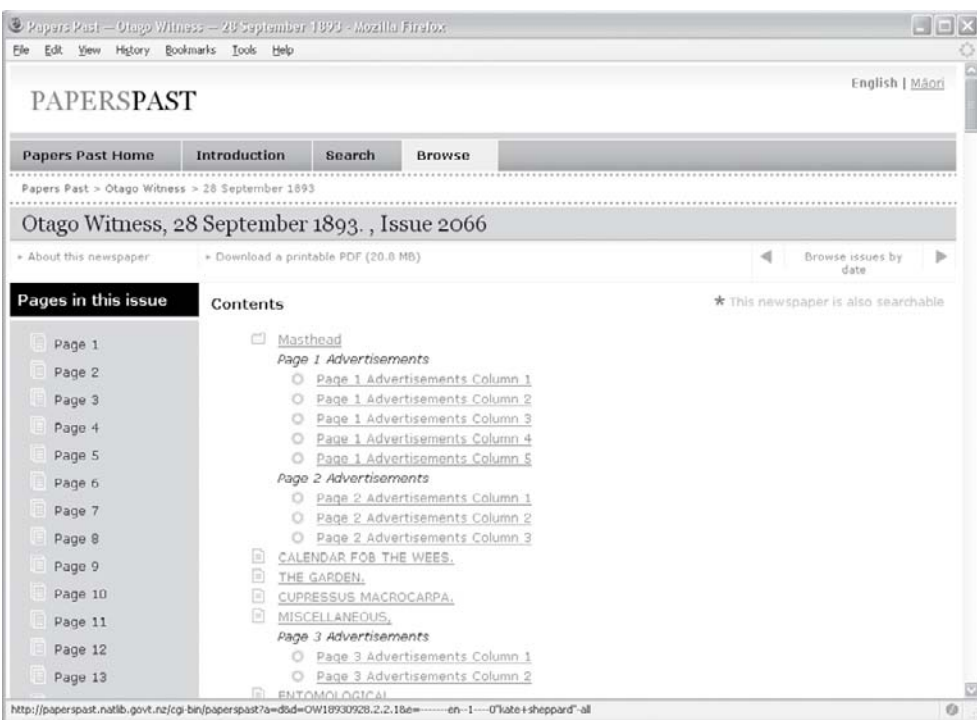
(d) 

Figure 3.4, cont'd: (c) the article in context; (d) contents of a newspaper issue

can obtain a printable version of the full edition (in the PDF, Portable Document Format, that is described in Section 4.5). The first column provides navigable links to each page; the second provides links to each article.

This is just a glimpse of what the Papers Past digital library has to offer. Such magnificent functionality does not come cheap. For example, a huge amount of metadata is involved. Normally, storage requirements for the source documents vastly outstrip those needed for the accompanying metadata. Here, however, this is reversed: for every gigabyte of text extracted by the OCR process, three times as much space is consumed by the coordinate information—metadata. In fact, the 1 MB of metadata required for a page exceeds the 0.75 MB required even for the scanned page image, let alone for the text extracted from it.

Realistic books

Section 1.2 describes how it took centuries for the written word to progress from papyrus scrolls to the book form that we use today. The book was a revolutionary development that changed the way people read written information; it is arguably one of the most important inventions in the history of thought. Consequently, it is perplexing that the scroll bar, reminiscent of long-obsolete technology, has dominated our computer interfaces for the past three decades.

The book is coming back. [Figure 3.5](#) shows an interface that much more closely resembles physical documents than the representations in earlier figures. Because it works in a standard Web browser, it can be widely deployed. Readers use a mouse to grasp the paper and sweep out the path of that point to turn the page. There is complete freedom to move the page within the constraints imposed by not moving it to a point that would tear physical paper, and the visual details follow instantly. Although the model does not look completely realistic in static pictures, it is effective in practice because it is dynamically reactive.

The book in [Figure 3.5](#), *Farming Snails I*, is from the same collection as the book in [Figure 3.1](#). Readers grasp the page anywhere along the top, right, or bottom edge—usually, but not necessarily, at a corner—by pointing their mouse there and depressing and holding down the mouse button. As they move the mouse, the page follows. If they release the button, the page either floats back to its original position or floats down to the turned position. Readers who use a touch panel instead of a mouse gain an even better sense of control.

Realistic books typically have a cover ([Figure 3.5a](#)), title page, table of contents ([Figure 3.5b](#)), and the main text ([Figures 3.5c and d](#)). Sections begin on a new page and are split into pages. Contents entries are hyperlinked, so that clicking them opens the book to that chapter. Colored tabs protruding from the page edges mark chapters, sections, or pages containing illustrations (these things are switchable). The reader turns pages by using the mouse or simply by clicking to turn them automatically. When the book is open, the cover's inside border is visible, and the reader can click this to close the book in either direction.

The presentation in [Figure 3.5a](#) was generated by the same digital library system used for [Figure 3.1](#). Any book in the Humanity Development Library can be displayed in this way; the software converts

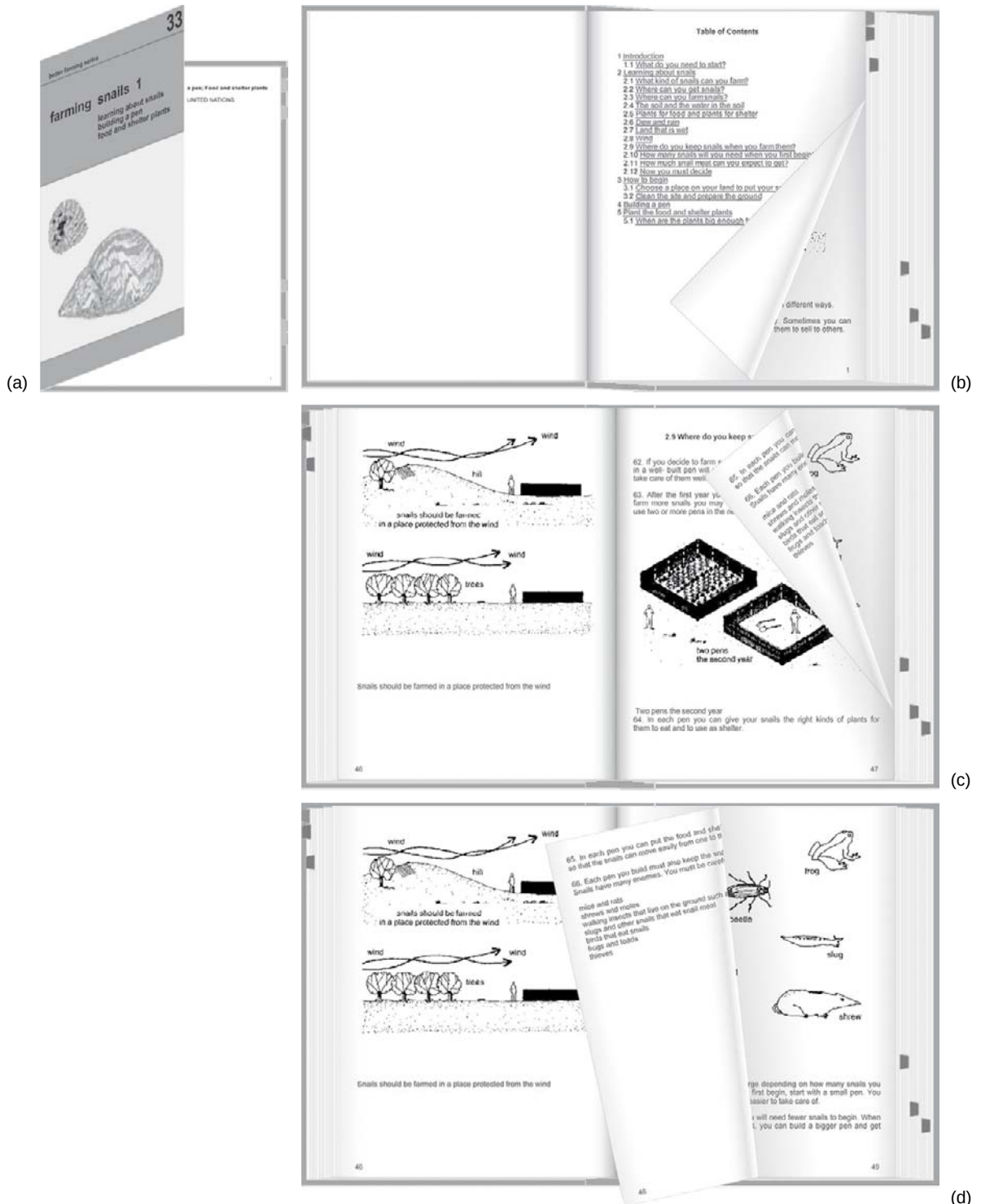


Figure 3.5: Browsing through a realistic book: *Farming Snails I*: (a) opening the front cover; (b) contents; (c, d) turning some pages. Copyright © 1986 Food and Agriculture Organization of the United Nations (FAO), Rome

it on the fly from the HTML representation. This serves to underscore the fact that digital libraries can change the entire look and feel of all documents in a collection at the touch of a button—or, as in this case, a single menu selection on a Preferences page.

3.2 Presenting Multimedia Documents

Next we consider multimedia documents: audio recordings and photographic images; video, which includes both image and audio components; and musical objects that can be presented in several different forms.

Sound and pictures

Some years ago, the public library in Hamilton, New Zealand, the small town where we live, began a project to collect local history. Concerned that knowledge of what it was like to grow up in Hamilton in the 1930s, 1940s, and 1950s would soon be permanently lost, the library decided to arrange to interview older people about their early lives. Armed with tape recorders, local volunteers conducted semistructured interviews with residents of the region and accumulated many cassette tapes of recorded reminiscences, accompanied by miscellaneous photographs from the interviewees' family albums. From these tapes, the interviewers developed a brief typewritten summary of each interview, dividing it into sections representing themes or events covered in the interview. But then the collection sat in a cardboard box behind the library's circulation desk, largely unused.

In a subsequent round of development, all the tapes and photos, along with the summaries, were digitized and made into a digital library collection. [Figure 3.6](#) shows it in use. The user is listening to a particular recording using a standard software audio-player that has regular tape-recorder functions (pause, fast forward, and so on) on the small control panel in the center. Users don't have to wait until the whole file is downloaded: the beginning starts playing while the rest is being transmitted. Behind the control panel is the interview summary. In the background on the right can be seen a photograph—in this case, of the town's celebrations on VE Day (Victory in Europe) near the end of the Second World War—and on the left is the query page that was used to locate this information.

The interview page is divided into sections, with a summary for each. Clicking on one of the speaker icons plays back the selected portion of the audio; interviews also can be played in full using buttons at the top of the page (not visible in [Figure 3.6](#)). When the tapes were digitized, timings were generated for the beginning and end of each section. Flipping through a recording in this way, scanning a brief textual synopsis and clicking on interesting parts to hear them, is far more engaging and productive than trying to scan an audiotape with a finger on the fast-forward button.

The contents of the interview pages are used for text searching. Although they do not contain full transcripts, many keywords that you might want to search on are included. In addition, brief descriptions of each photograph were entered, and they are also included in the text search. These value-adding activities were done by amateurs, not professionals. Standard techniques, such as deciding in advance on the vocabulary with which objects are described (i.e., using a *controlled vocabulary*) were not used. Nevertheless, users can easily find material of interest.

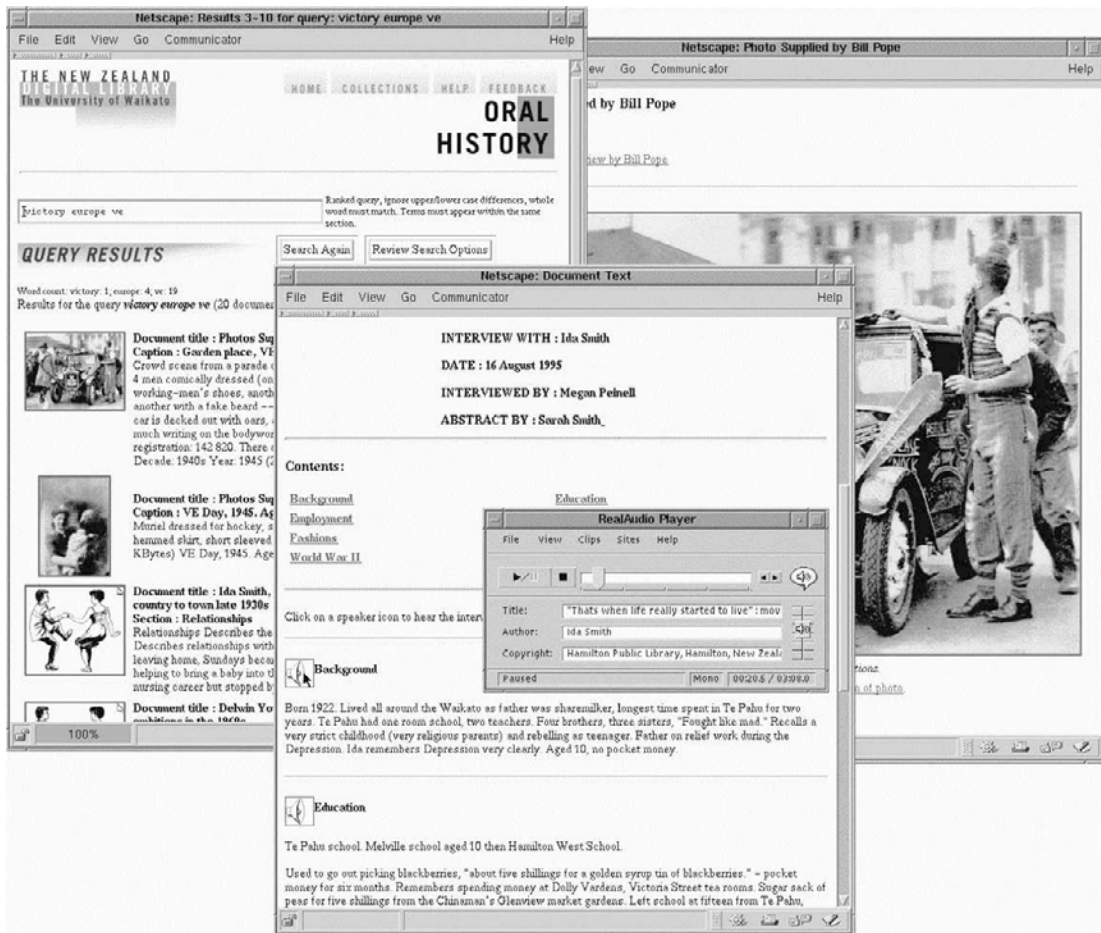


Figure 3.6: Listening to a tape from the Oral History collection. Hamilton Public Library, Hamilton, New Zealand

Consider the difference between accessing a box of cassette tapes at the library's circulation desk and searching the fully indexed, Web-accessible digital library collection depicted in Figure 3.6. Text searching makes it easy to find out what it was like at the end of the war, to study the development of a particular neighborhood, or to see if certain people are mentioned (and you can actually hear senior residents reminisce about these things). Casual inquiries and browsing are simple and pleasurable—in striking contrast to searching through a paper file, then a box of tapes, and finally trying to find the right place on the tape using a cassette player. In fact, although this collection can be accessed from anywhere on the Web, the audio files are available only on terminals in the local public library, because the interviewees were not asked for consent to broadcast their voices worldwide. The message here for those engaged in local history projects is: think big.

Video

Videos combine time-based information with a spatial image component. As with audio, time-based documents can be made more conveniently browsable by segmenting them, and videos can be automatically converted into sequences of thumbnails that correspond to scene changes. Web browsers can play video in a variety of formats, provided a suitable plug-in is installed; the digital library server can even offer users a choice of formats. The Flash video format (reviewed along with several others in Chapter 5) adopted by YouTube has enormous penetration: some surveys estimate that it is installed on 99 percent of Internet-capable computers (compared with 80 to 85 percent for its closest rivals).

In planning a digital library, such statistics are useful in deciding on the best way to deliver documents to the intended audience. Decisions need to be coupled with knowledge about what the delivery format can do (for instance, Can the video be started at particular time offset? How easily and faithfully can the source format be converted?).

The feasibility of downloading video over the Internet depends on technical factors like the bandwidth of the connection. Of course, a great deal of storage space will be required for a large collection. Perhaps the digital library designer should consider providing an audio-only representation initially. This requires far less bandwidth and, particularly in the case of fixed-camera interviews, still communicates the essential information. For movies with an important visual component, a storyboard of images could convey important details at a fraction of the cost of video.

Searching video, like searching the oral history audio, requires appropriate descriptive text. And, as is discussed in [Section 3.4](#), images can also be searched directly, using similarity based on analysis of the images themselves. This technique can be applied to key-frames chosen from the video either manually or automatically.

Music

As Chapter 1 describes, digital collections of music have the potential to capture popular imagination in ways that scholarly libraries never will. Two elements to creating such a resource that is interesting and entertaining to search and browse are (1) having different representations of the same music available, and (2) linking to external resources to locate additional, relevant information.

[Figure 3.7](#) shows a prototype digital music library motivated by these observations. Starting with scanned images of sheet music, optical music recognition (OMR) software—which is similar to OCR software but works in the domain of printed music—was used to generate a symbolic version of each song. This was paired up with the original score and accompanying metadata for title, composer and lyricist. Searching and browsing capabilities were then developed based on this digital content.

In the figure we join a user partway through seeking for the tune *Auld Lang Syne* in the digital library, having sung a few remembered notes (we return to this “query by humming” capability in [Section 3.4](#)). For now we are interested in what can be done given the result of the query, which is a ranked list of songs that match precisely or are at least similar to the query. The further down the list



Figure 3.7: Finding *Auld Lang Syne* in a digital music library

a song is, the less likely it is to be the one the user is looking for. Clicking on the speaker icon to the top matching item (which in this case happens to be *Auld Lang Syne*) results in an audio rendition of the piece, based on the symbolic representation stored in the digital library. The player can be seen in the front window of Figure 3.7. The adjacent icon displays the musical notation for the tune (also featured in the figure), which was generated from the same internal representation, this time by a music-typesetting program.

For various reasons, this computer generated playback and typesetting can be of limited quality. Not shown in Figure 3.7, but just a click away for the user, is an image of the actual book page that contains the song. Also available are the lyrics. Furthermore, the song and artist metadata for a song are automatically hyperlinked to initiate text queries on the Web using a conventional general purpose Internet search engine: a convenient (but not necessarily precise) way to locate additional information

about the song (or composer). It also enables other versions of the song to be found in, for instance, MIDI (Musical Instrument Digital Interface) format (see Section 5.6), or an actual recording (see Section 5.2) and one of those played instead.

3.3 Document Surrogates

Traditional libraries manage their holdings using catalogs that contain information about every object they own. Metadata, characterized in Chapter 1 as “data about data,” is a recently coined term for this information. Metadata is information in a structured format and its purpose is to provide a description of other data objects in order to facilitate access to them. The data objects themselves (e.g., the books) generally contain unstructured information. Sometimes, as in the Humanity Development Library in Figure 3.1, they do have some internal structure. Sometimes, as in the Project Gutenberg collection in Figure 3.2, their information has structure but that structure is not apparent to the system. However, the essential feature of metadata is that its elements are structured. Moreover, metadata elements are standardized so that the same type of information can be used in different systems and for different purposes. In the computer’s full-text copy of a book, the title, the author, the publisher, the source of the original edition, etc., may not be obvious. However, when this information is represented in metadata, in a standard way using standard elements, the computer can identify these fields and operate on them.

When users initiate a search or browse in a digital library, they are often presented with lists or displays that summarize the digital objects themselves. These summaries are known as *document surrogates*, which are concise displays that represent the actual object, typically using some of its metadata.

Metadata

Figure 3.8 is taken from a digital library collection of computer science bibliographies. It shows the result of searching for the author *Honkala*, with matching publications presented as a standard bibliographic listing: the hyperlink at the end of each item links to the source bibliography. Many of the entries have abstracts, which are viewed by clicking the page icons to the left—although here they are grayed out, indicating that abstracts are unavailable. The metadata displayed includes title, author, date, the title of the publication in which the article appears, volume number, issue number, and page numbers. As noted above, it also includes the URL of the source bibliography and the abstract (although it is debatable whether this is structured enough to constitute metadata).

Metadata has many different aspects, corresponding to various kinds of information that might be available about an item. Historical features describe provenance, form, and preservation history. Functional ones describe usage, condition, and audience. Technical ones provide information that promotes interoperability between different systems. Relational metadata covers links and citations. And, most important of all, intellectual metadata describes the content or subject. Metadata provides assistance with search and retrieval; gives information about usage in terms of authorization, copy-right, or licensing; addresses quality issues, such as authentication and rating; and promotes interoperability with other systems.

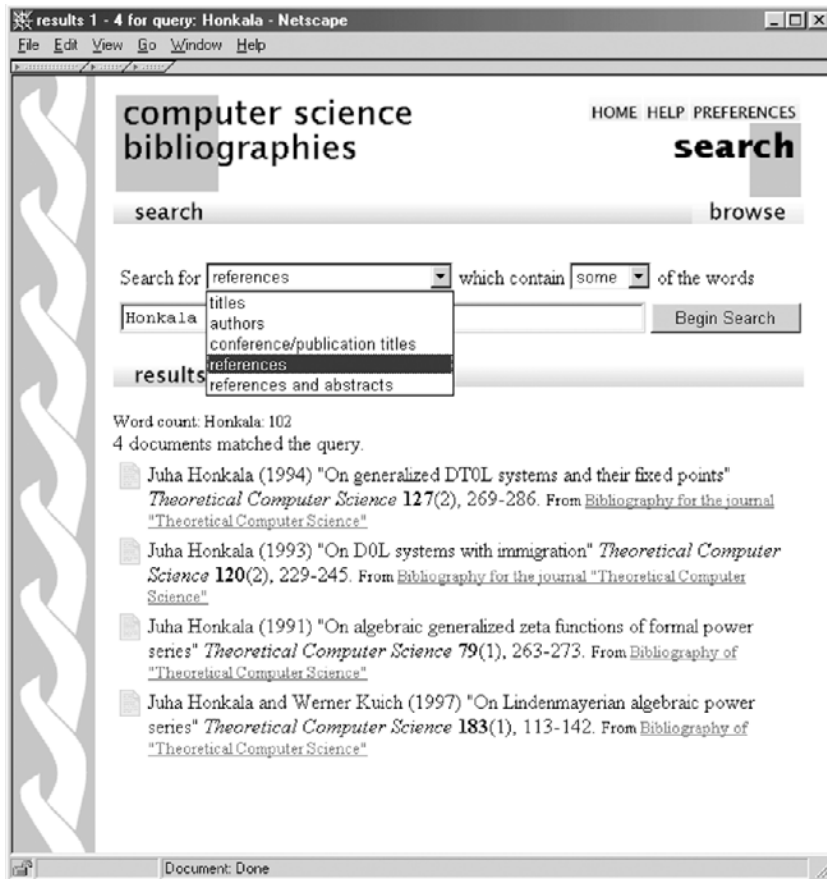


Figure 3.8: Bibliography display

Figure 3.9a shows a record retrieved over the Internet from the Library of Congress and displayed within a simple interface (although only half of the fields in the record are visible). Common fields are named, while obscure ones are labeled with identification numbers (e.g., field 35). You can see that there is some redundancy: the principal author appears in both the *Personal Name* field and in a subfield of the title; the other authors also appear further down the record in separate *Author Note-Name* fields (not shown). This metadata was retrieved using an information interchange standard (called Z39.50) that is widely used throughout the library world (see Section 7.2) and is represented in a record format called MARC (“machine-readable cataloging”) that is also used by libraries internationally (see Section 6.2).

Library metadata is standardized, but, as is often the case with standards, there are many different ones to choose from. (MARC itself comes in more than 20 variants, produced for different countries.) Non-bibliographic metadata has no widely accepted standards. Figure 3.9b shows a record from a BBC catalog of radio and television programs; the record gives such information as program title, item title, date, medium, format, several internal identifiers, a description, and a comments field.

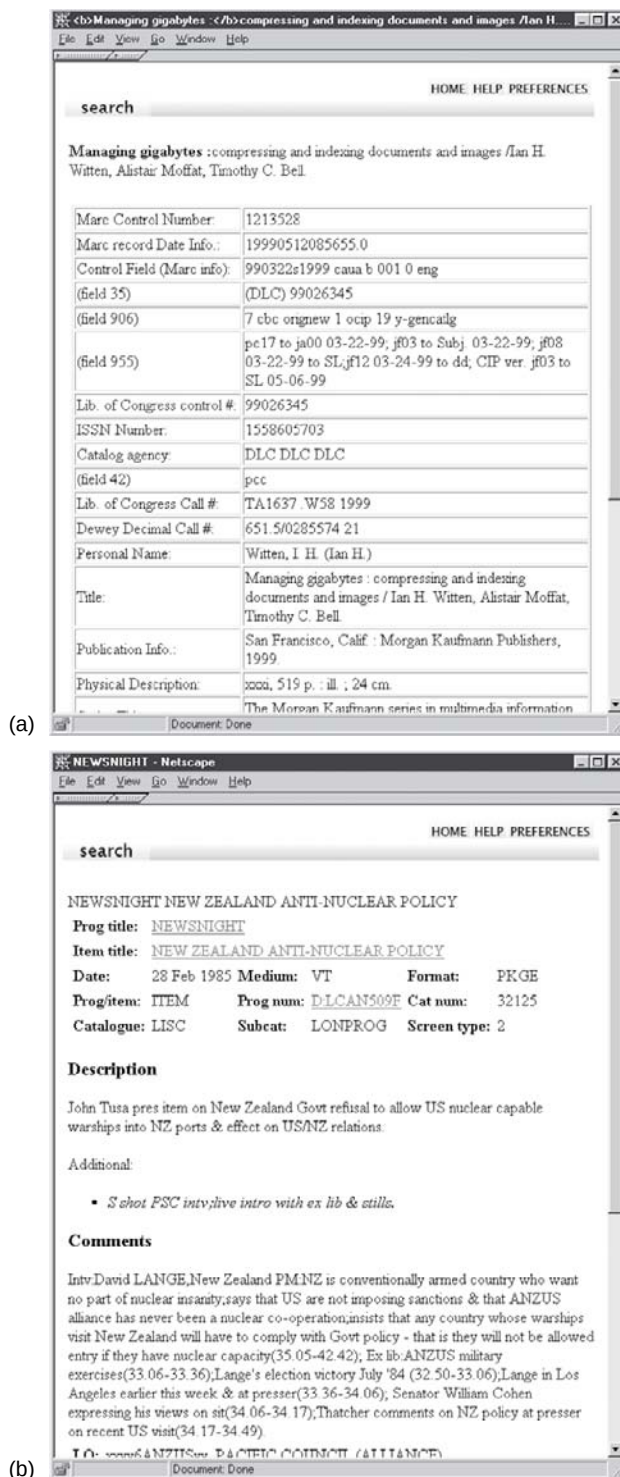


Figure 3.9: Metadata examples: (a) bibliography record retrieved from the Library of Congress; (b) description of a BBC television program

Metadata descriptions often grow willy-nilly, in which case the relatively unstructured technique of text search is a better way to locate records than using a conventional metadata database. Because of increased interest in communicating information about radio and television programs internationally, people in the field are working on developing a new metadata standard for this purpose. Developing international standards requires a lot of hard work, negotiation, and compromise; it takes years.

Surrogates can also include elements from the document's actual content. A common method of displaying full-text search results is to highlight some matching text and to use it as part of the surrogate that is displayed to the user. For example, [Figure 3.10](#) shows textual snippets from the documents in a page returned by the Google search engine. This allows users to see how their search terms interact with the document collection.

Multimedia surrogates

Textual surrogates rarely give users a good feeling for multimedia content, but some elements of multimedia can produce effective representations that help users make informed choices about which documents to investigate further. Even a predominantly textual document like a book can use its cover image as a surrogate.

It is natural to represent full-size images by miniature versions, and scaled or cropped thumbnails are effective surrogates. Temporal multimedia like audio and video are not so easily accommodated. Should an hour-long video be represented by a mini version of it? Which parts? All the way to the end? Users normally expect to be able to make selection decisions in just a few seconds, but if the surrogate appears among search results, they may face ten or twenty miniature videos.

For this reason, video surrogates are usually reduced to image key-frames or short clips (of a few seconds) that are under the user's control. Similarly, well-chosen musical excerpts can stand in for an entire symphony. Alternatively, images of CD covers or artists can be used as visual surrogates of audio. Here are some general approaches used when there are no obvious surrogates, as with complex digital objects like animations, computer programs, and data sets:

- textual metadata
- miniature version of the content (e.g., cropped or scaled images)
- extract in a different media format (e.g., image key-frames from video)
- short extract from temporal media (e.g., video, audio, animations)
- related multimedia in another format (e.g., CD cover image)
- generic icon like the symbol for a particular document format or an image representing music.

3.4 Searching

Electronic document delivery is the primary *raison d'être* for most digital libraries. But searching comes a close second—in particular, searching the full text of documents. Conventional library catalog searches are restricted to metadata, but digital libraries have access to the full content of the objects they contain. This is a great advantage.



Figure 3.10: Document snippets used as surrogates

Figure 3.11 shows a request that seeks paragraphs in English containing both the word *begin* and the word *beginning* (in order to find the *Alice in Wonderland* quotation “begin at the beginning”), and the computer’s response: a list of documents that match. These pages provide a plain and simple search mechanism with rudimentary functionality. Digital libraries, particularly ones targeted to the general public, need simple facilities that fulfill what most people want most of the time, with options for more advanced users. To quote a famous saying, “Simple things should be simple, complex things should be possible.”

In Figure 3.11 readers can specify the unit to be searched, the language, and the type of search. The selection varies from collection to collection. Searchable units typically include *paragraphs*, *sections*, and *documents*; as well as *section titles*, *document titles*, and *authors*. The first group involves the full text; the second involves metadata.

Full documents, not just the relevant paragraphs, are returned in the list of titles in Figure 3.11b—even though the search was for *paragraphs*. In this digital library system, *documents* are defined as what the user finally sees on the screen as the result of a search. If you want paragraphs to be presented individually, you must define paragraphs to be the “documents”—and arrange for them to link to the paragraphs before and after so that users can read the text in sequence. This is not hard to do.

Types of queries

In Figure 3.11 the option *all of the words* has been chosen; the alternative in this collection is *some of the words*. Queries comprise a list of *terms*—words to be sought. In a general Boolean query, terms are combined using the connectives AND, OR, and NOT. AND is the most common connective, and that is how the option *all of the words* is interpreted in Figure 3.11. A query such as

digital AND library

might be used to retrieve books on the same subject as this one. Both terms (or equivalent lexical variants) must occur somewhere in every answer. They need not be adjacent, nor in any particular order. Documents containing phrases like *library management in the digital age* will be returned, as will documents containing the text *a software library for digital signal processing*—perhaps not quite what is sought, but nonetheless a correct answer to the Boolean query. A document in which the word *library* appeared in the first paragraph and *digital* in the last also would be considered a correct result.

Retrieval systems inevitably return many irrelevant answers, which users must filter out manually. Users have to make a difficult choice between casting a broad query, to be sure of retrieving all relevant material (albeit diluted with many irrelevant answers), and a making a narrow query where most retrieved documents are of interest but others slip through the net because the query is too restrictive. A broad search that identifies nearly all the relevant documents is said to have *high recall*, while one in which nearly all retrieved documents are relevant is said to have *high precision*. Searchers must choose between high precision and high recall and formulate a query appropriately. In Web searches, for example, precision is generally preferred over recall. There is so much out there that you rarely want to find *every* relevant document (you probably couldn’t review them all anyway) and you

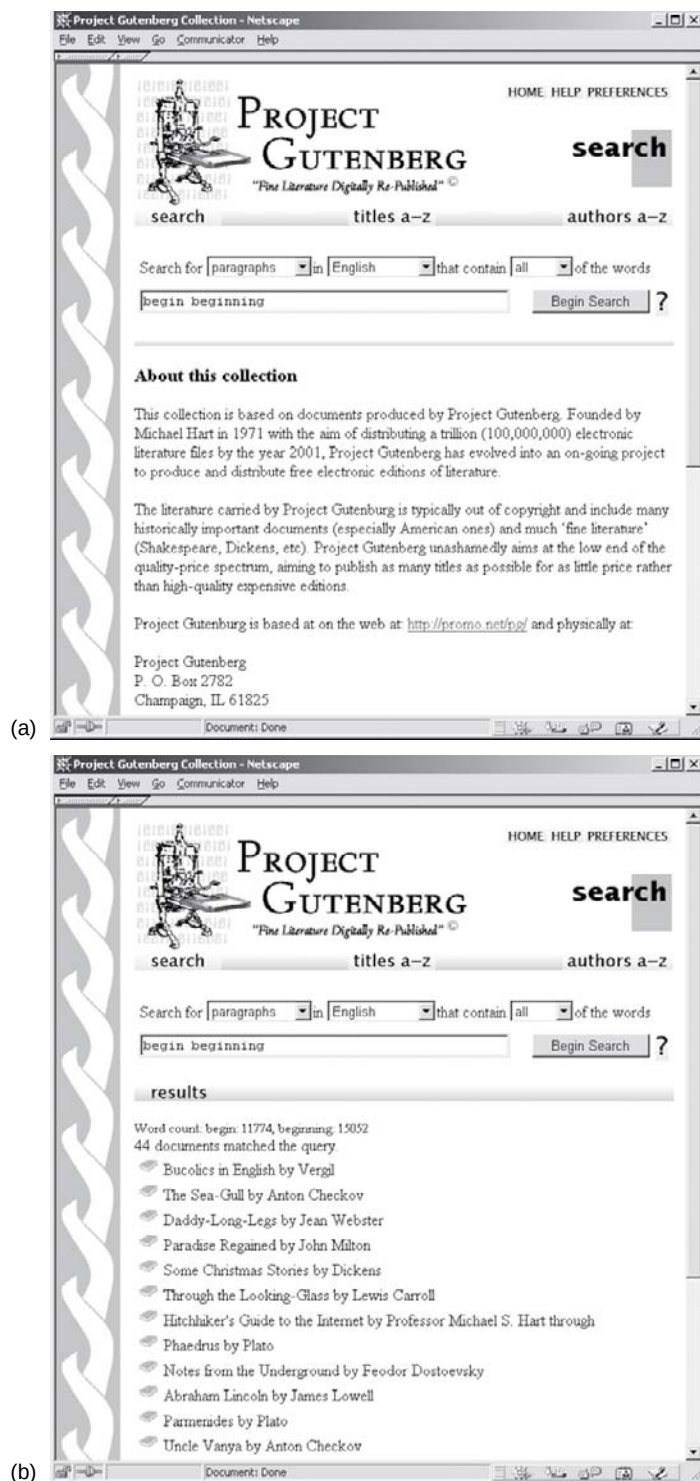


Figure 3.11: Searching for a quotation: (a) query page; (b) query response

certainly don't want to have to check through a host of irrelevant ones. However, if you are counsel for the defense looking for precedents for a legal case, you probably care a great deal about recall—you want to be sure that you have examined *every* relevant precedent, because the last thing you want is for the prosecutor to spring a nasty surprise in court.

An enduring theme in information retrieval is the tension between recall and precision.

Another problem is that small variations can yield quite different results. You might think the query *electronic AND document AND collection* is similar to *digital AND library*, but they are likely to produce very different answers. To catch all desired documents, professional librarians become adept at adding extra terms, learning to pose queries such as

(*digital OR virtual OR electronic*) AND (*library OR (document AND collection)*)

where the parentheses indicate operation order.

Until around 1990, Boolean retrieval systems were the primary means of accessing information in commercial and scientific applications. However, Internet search engines have shown that they are not the only way a database can be queried.

Rather than seeking exact Boolean answers, it is often better simply to list words that are of interest and have the retrieval mechanism supply whatever documents seem most relevant. For example, to locate books on digital libraries, the list of terms

digital, virtual, electronic, library, document, collection, large-scale, information, retrieval

is, to a nonprofessional at least, probably a clearer encapsulation of the topic than the Boolean query cited earlier.

Yet, identifying relevant documents is not just a matter of converting a list of terms to a Boolean query. It would be fruitless to connect these particular terms with AND operators, since vanishingly few documents are likely to match. (We cannot say that no documents will match. This page certainly does.) It would be equally pointless to use OR connectives, since far too many documents will match and few are likely to be useful.

The solution is to use *ranking*. In ranking, an artificial measure is used to gauge the similarity of each document to the query, and a fixed number of the closest matching documents are returned as answers. If the measure is good, high precision is guaranteed if few documents are returned, and high recall is guaranteed if many are returned. In practice, there is a trade-off: low precision invariably accompanies high recall, since many irrelevant documents come to light before the last of the relevant ones appears in the ranking. Conversely, low recall accompanies high precision, because precision will be high only near the beginning of the ranked list, at which point only a few of the total set of relevant documents will have been encountered.

A simple ranking technique is to count the number of query terms that appear somewhere in the document: this is called *coordinate matching*. A document that contains five of the query terms will be ranked higher than one containing only three, and documents that match just one query term will be ranked lower still. Obviously, coordinate matching favors long documents over short ones, since by

virtue of size alone long documents are more likely to contain a broader selection of the query terms. Furthermore, common terms in the query are treated just the same as highly specific ones—for example, the query *the digital library* might rank a document containing *the digital age* alongside or even ahead of one containing *a digital library*. Words such as *the* in the query should probably not be given the same importance as *library*. For this reason, most ranking techniques assign a weight to each term based on its frequency in the document collection, giving common terms low weight. They also compensate for the length of the document, so that long ones are not automatically favored.

It is difficult to describe ranking mechanisms in a few words. The simple interface in [Figure 3.11](#) mentions only that answers should match *some of the words*, as the alternative to the *all of the words* that is shown in the figure (but does not give any indication as to how this is done). Experience indicates that readers are rarely confused by the responses they receive—they don’t even think about the ranking mechanism.

Professional information retrieval specialists like librarians want to understand exactly how their query will be interpreted and are prepared to issue complex queries provided that the semantics are clear. For most tasks they prefer precisely targeted Boolean queries, which are especially appropriate if metadata is being searched, and particularly if professional catalogers have entered it, because the terms that are used are relatively unambiguous and predictable. Casual users, on the other hand, may not be concerned about how queries are interpreted; they prefer to trust the system to do a good job and are prepared to scroll further down the ranked list if they want to expend more effort on their search. They like ranked queries, which are especially appropriate if full text is being searched, because term usage in full text is relatively unpredictable.

Case-folding and stemming

Two common operations in querying are *case-folding* and *stemming*. [Figure 3.12](#) shows a Preferences page (which allows users to choose query options), and case-folding and stemming have been selected (near the bottom). The interface supplies three- or four-word descriptions of these operations so that their meaning can be grasped immediately by casual users.

Case-folding replaces all uppercase characters in the query with their lowercase equivalents, treating uppercase versions of words as equivalent to lowercase words (in our example queries, case-folding treats *Digital* and *DIGITAL* and *Library* and *LIBRARY* as equivalent to *digital* and *library*). Users seeking an exact match need to disable case-folding. For example, a user looking for documents containing *Digital AND Equipment AND Corporation* (a now-defunct computer manufacturer) would disable case-folding in order to avoid being flooded with answers about *corporation policy on capital equipment for digital libraries*. Thus, users must be able to specify whether case-folding applies.

Stemming reduces words by stripping off suffixes, converting them to neutral stems that are devoid of tense, number, and—in some languages—case and gender information. This relaxes the match between query terms and words in the documents so that, for example, *libraries* is deemed equivalent to *library*. Like case-folding, stemming is not appropriate for all queries, particularly those involving names and other very specific words.

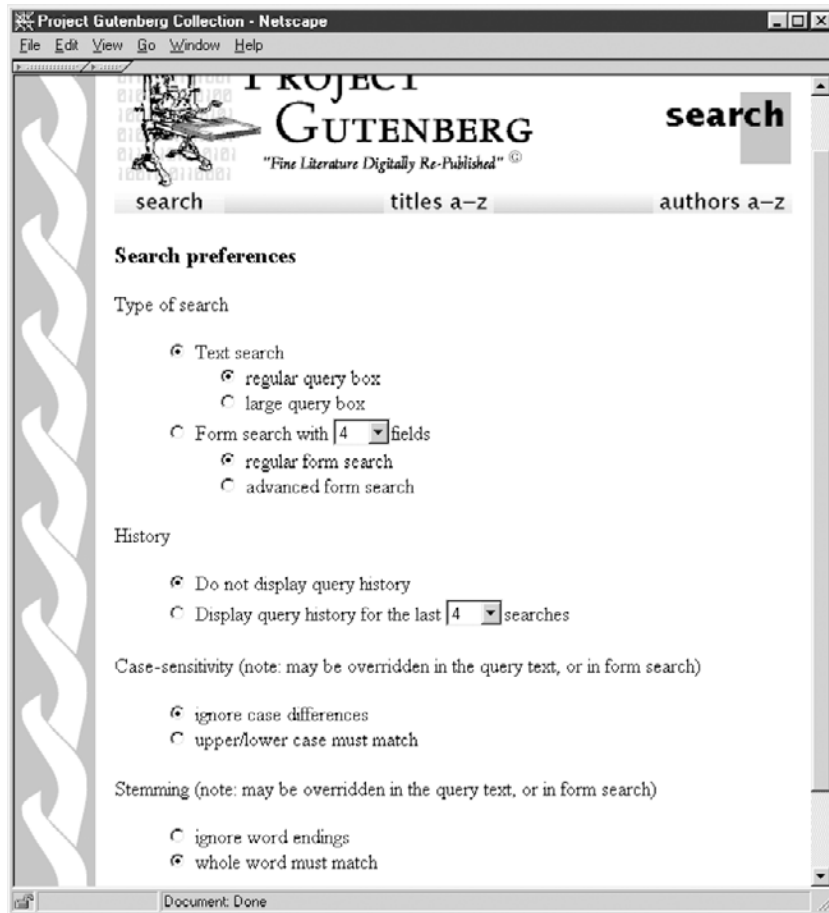


Figure 3.12: Choosing search preferences

The addition of suffixes is governed by linguistic rules. Converting the *y* in *library* to the *ies* in *libraries* is one example; another is the doubling of a final consonant, as when *stem* is augmented to *stemming*. Reducing a word to its root form, or *morphological reduction*, requires language dictionaries that include information about which rules apply to which words. However, for the purposes of information retrieval, simpler solutions suffice. All that is necessary is for different variants of a word to be reduced to an unambiguous common form—the stem. The stem does not have to be the same as the morphological root form. The desired effect will be obtained so long as all variants reduce to the same stem, and no other words do. Also, only certain kinds of suffixes need be considered. Linguists use the word *inflectional* to describe suffixes that mark a word for grammatical categories. In contrast, *derivational* suffixes modify the major category of a word—for example, when *-less* is appended to words like *hope* or *cloud* it converts a noun to an adjective. Derivational suffixes should not be stripped because they alter the meaning—radically, in this example.

Stemming is language-dependent: an algorithm for English will not work well on French and vice versa. Indeed, the concept of stemming differs widely from one language to another. Many languages use prefixes as well as suffixes to indicate derivational forms; others contain complex compound words that should be split into constituent parts before being entered into a query. Case-folding, too, is not relevant to certain languages, such as Chinese and Arabic.

Furthermore, stemming and case-folding complicate the highlighting of search terms in retrieved documents. Simply finding the stem and highlighting all words that contain it could highlight the wrong word. For example, a particular system might make *library* and *libraries* match when searching by stemming them both to *librar*, but deliberately avoid stemming *librarian* because its derivational suffix changes the meaning. However, a simple textual match with the stem will result in the word *librarian* being incorrectly highlighted in the search results. And a simplistic method may fail to highlight correct terms—a different algorithm might stem *libraries* to the root form *library* when searching, but fail to highlight the retrieved search term.

The correct procedure is to stem each word in the retrieved document using the same algorithm and then to assess if the result matches the stemmed query word. However, this procedure can be prohibitively time-consuming. A more practical alternative is to record the stemmed form of each word and to expand the query by adding all unstemmed variants prior to the highlighting operation.

Phrase searching

Users often want to search for contiguous groups of words. Indeed, most of our examples—*digital library*, *Digital Equipment Corporation*—would be better posed as phrase searches. Phrases are generally indicated in a query by using quotation marks.

Although phrase searching is simple and natural for users, it is actually quite complex to implement. Users might think the computer looks through all the documents just as they would, but faster—a lot faster. Because the computer can find the individual words in all the documents, it seems natural to conclude that it can just as easily check to determine if they come together as a phrase. However, a computer search doesn't scan through the text: that would take too long. (Computers are not all *that* fast, and for documents stored on disk, access is by no means instantaneous.)

Instead, computers first create an index that records, for each word, the documents that contain that word. Every word in the query is looked up in the index, and the computer compiles a list of retrieved document numbers for each word. Then the query is answered by manipulating the lists—in the case of a Boolean AND query, by checking which documents are in all the lists. (The process is described more in Section 4.1.)

Phrase searching changes everything. No longer can queries be answered simply by manipulating lists of document numbers. Instead, there are two options. Having treated the query as a set of individual words, one option is to look inside the documents themselves: checking through all documents that contain the query terms to see if they occur together as a phrase. This is a *postretrieval scan*. The other option is to record word numbers as well as document numbers in the index—the position of each occurrence of the word in the document as well as the documents it occurs in. We refer to

this as a *word-level index*. Then, when each word in the query is looked up in the index, the computer can tell from the list of word positions if the words occur together in a phrase, because words in a phrase will be numbered consecutively.

How phrase queries are implemented greatly affects the resources required by the system. A postretrieval scan can take a great deal of time because—depending on how common the terms in the phrase query are—many documents might have to be examined. Phrases containing unusual words can be processed quickly: few documents will include them and therefore few will need to be scanned for the occurrence of the phrase. But phrases containing common words, such as *digital library*, will require substantial computer time, and response will be slow.

With a word-level index, the exact position of each occurrence of every word is recorded, instead of just the documents in which it occurs. This makes the index significantly larger. Not only are word numbers larger than document numbers, and hence require more storage space, but any given word probably appears many times in each document, and every occurrence must be recorded.

Which mechanism is employed also affects how phrases can be used in queries. For example, people often want to specify proximity: the query terms must appear within so many words of each other, but not necessarily contiguously in a phrase. If word numbers are stored, responding to a proximity query is just a matter of checking that the positions do not differ by more than the specified amount. If phrase scanning is employed, proximity searching is far more difficult.

Users sometimes seek phrases that include punctuation and even white space. Most word-level indexes treat the documents as sequences of words, ignoring punctuation and spacing. Distinguishing between differently punctuated versions of a phrase requires a postretrieval scan even if a word-level index is available—unless the index also includes word separators.

Phrases complicate ranked searching. Here the frequency of a query term throughout the corpus is used to measure how influential that word should be in determining the ranking of each document—common words like *the* are less influential than rare ones like *aspidistra*. However, if the query contains a phrase, it should really be the frequency of the phrase, not the frequency of the individual words in it, that is used for ranking. For example, the English words *civil*, *rights*, and *movement* are used in many contexts, but the phrase *civil rights movement* has a specific meaning. The importance of this phrase relative to other words in a query should be judged according to the frequency of the phrase, not the constituent words.

Including phrases in queries complicates things technically. In practice, building digital libraries involves pragmatic decisions. Word-level indexes are recommended if phrase searching is likely to be common and if space is not a significant constraint. Simple systems use a postretrieval scan, which suffices when phrase searching is rare or if phrases contain punctuation. In either case, ranking is based on individual word frequencies, not phrase frequencies, for practical reasons.

How can we communicate these decisions to users? As we have seen, it is difficult to fully understand what is happening in phrase searching. An alternative is a more advanced search technique, like the one described in [Section 3.6](#), which provides a natural interface whose workings are easy to grasp.

Query interfaces

The search pages we have seen have minuscule query boxes, implicitly encouraging users to type just one or two terms. In reality, most queries contain only a few words. In fact, studies have shown that the most common number of terms in queries to Web search engines is—zero! People just hit the search button, or the Enter key, without typing anything, presumably by accident. The second most common number of search terms is one. Note that for single-term queries there is no difference between AND and OR, although in some systems *all of the words* returns the documents in some predetermined order—say by date—whereas *some of the words* implies ranking. For single-term queries, ranking returns documents in order of how often they contain the query term (normalized by document length). The third most common query has two terms; after that, query frequency falls off rapidly with query length.

Modern search engines easily deal with large queries: indeed, large queries can often be processed more efficiently than smaller queries because they are more likely to contain rare words that restrict the scope of the search. Figure 3.13 shows a large query box into which users can paste paragraph-sized chunks of text—and it is scrollable to facilitate even larger queries.

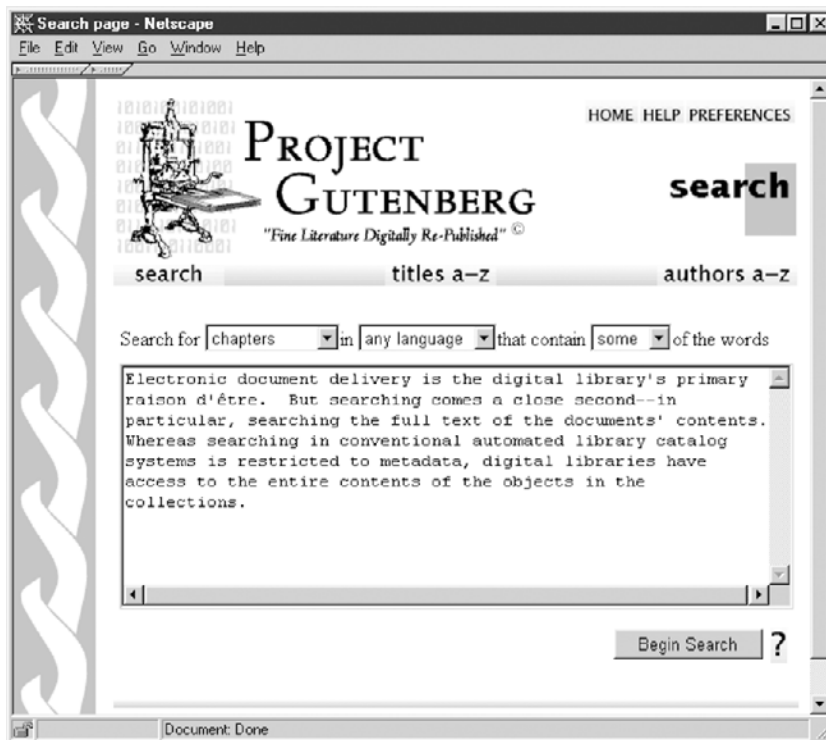


Figure 3.13: Large-query search interface

People often repeat searches, which is easy if they have access to their search history. “Those who ignore history,” to adapt George Santayana’s famous dictum, “will be forced to retype it.” New queries are often modifications of old ones—new terms are added if the query is too broad, to increase precision at the expense of recall, or terms are removed if the query is too restrictive, to increase recall. The interface in [Figure 3.14](#) presents the last four queries issued by the user. The buttons on the left move the query into the search box, where it can be modified. For example, clicking on the button to the left of the first field will place *begin beginning* in the search box. The Preferences page in [Figure 3.12](#) is used to select how much history to display.

What if users change search options, or even collections, between queries? The history display should make this explicit. Maybe users are experimenting with these options to test their effect on a query—Does the number of results change with stemming? Does one collection contain more potential answers than another? This is why details are given alongside the history item when such changes occur, as [Figure 3.14](#) shows. Normally, these details don’t appear, because users rarely alter their query options. When the details do appear, they clarify the context within which the query history should be interpreted.

Particularly for academic study, searches on different fields often need to be combined. For example, a researcher might be seeking a book by a certain author with a particular word in the title or a particular phrase in the main text. Library catalog systems have a search form that supplies several fields

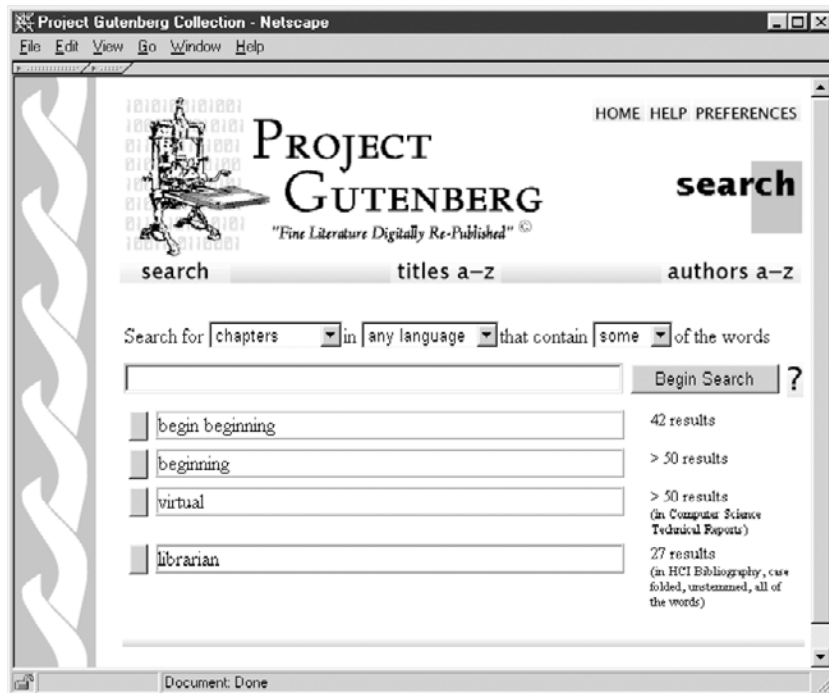


Figure 3.14: Query with history

into which specifications can be entered, like the one in [Figure 3.15a](#). Users type each part of the query into a box and use the menu to the right of the box to select the field. Finally, they decide whether the documents should satisfy *some* or *all* of these conditions. If necessary, users can go to the Preferences page to request more boxes ([Figure 3.12](#)).

More complex fielded searches can be undertaken using the form in [Figure 3.15b](#). Again, specifications are placed in the entry boxes and a field is selected for each one. Case-folding and stemming can be set for each individual field. The selection boxes that precede the fields allow the Boolean operators AND, OR, and AND NOT. This form cannot be used to specify completely general Boolean queries because users cannot control the order of operation—there is no place to insert parentheses. The specifications are executed sequentially: each one applies to all the fields beneath it.

The line near the top, just under the navigation bar, allows users to decide whether the results should be sorted into *ranked* or *natural* order (the latter is the order in which documents appear in the collection, usually by date). Users can also limit the search to certain date ranges. They can also select which date field to use, because in some collections more than one date may be associated with each document, with different dates corresponding to different kinds of metadata. For example, in a historical collection there can be a big difference between the period a document covers and the date it was written.

This advanced interface is intended for expert users. However, experts often find it frustrating to have to fill out forms: they prefer to type textual queries rather than to click between fields. Behind the form-based interface is an underlying query language, and users may prefer to use this for query entry.

Searching multimedia

The oral history collection in [Figure 3.6](#) is a searchable multimedia collection based on textual metadata painstakingly entered by hand. An intriguing alternative is to base retrieval directly on the multimedia by analyzing the content itself. For example, optical character recognition (OCR) and automatic speech recognition (ASR) turn digitized textual images and spoken audio into text. In principle, these technologies allow source content to be fed directly into full-text indexing engines and to be retrieved in just the same way as text. However, there are some caveats, because such systems are error prone.

Media like photographs and music, which have no textual representation, present even greater challenges than textual images and spoken audio. Here are two examples.

Searching music

In a digital music library ([Figure 3.7](#)) multiple representations of music can be derived and presented. Search might be based on textual metadata, making each item retrievable by title, composer, the year it was written, who performed a particular version, etc.

But text-based search does not always map well to what users can express, particularly when the underlying form is non-textual. To locate Vivaldi's *The Four Seasons*, should you search by overall title or for one of the individual parts—spring, summer, autumn and winter? In actual fact, one of the

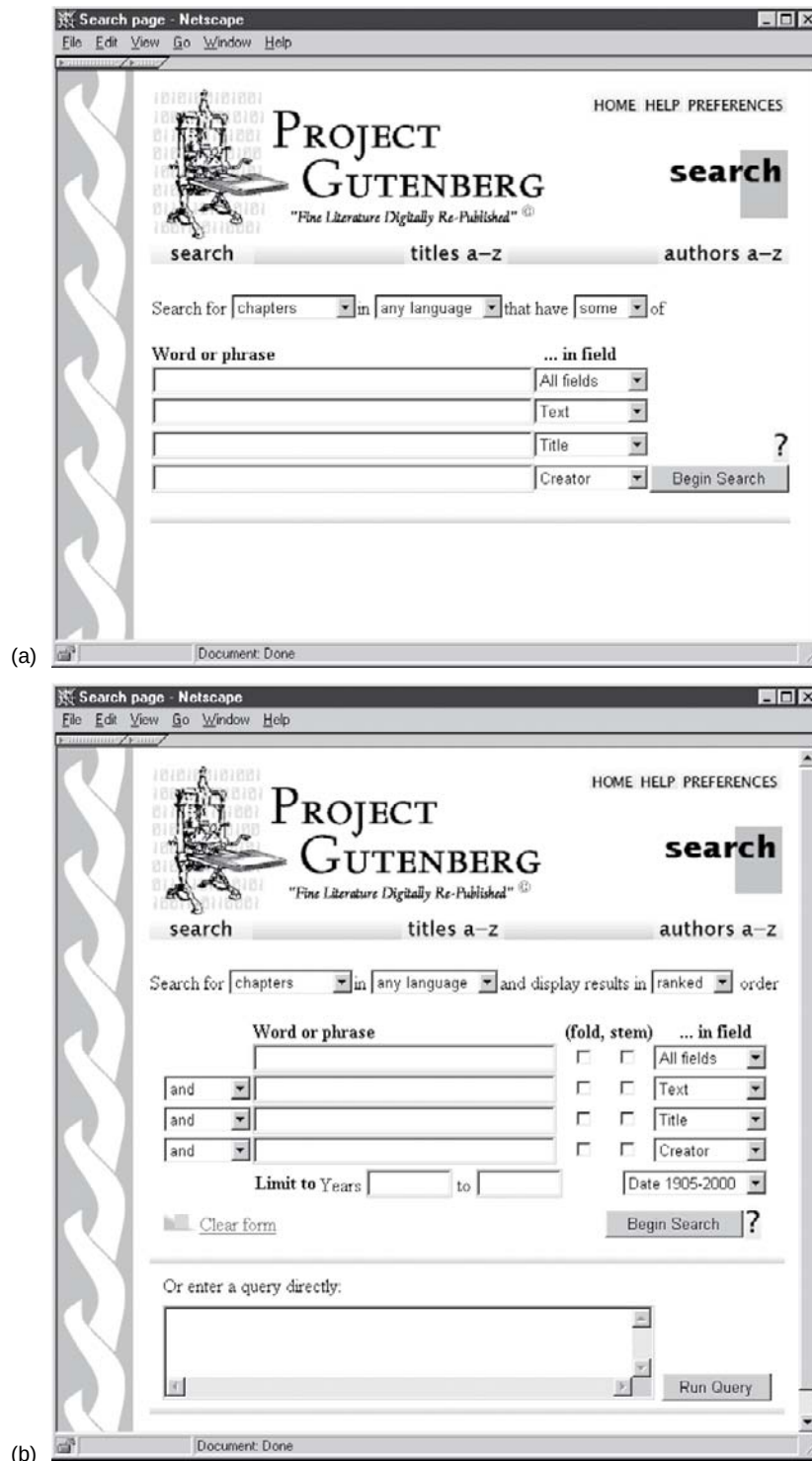


Figure 3.15: Form search: (a) simple; (b) advanced

authors tried this as a student many years ago. Entering the title query “four seasons” (with quotes) into the university library catalog returned nothing; removing the quotes produced a deluge of irrelevant matches—Vivaldi’s work certainly did not appear in the first three pages. Much time was wasted varying the composition of the query (keyword, Boolean, adding the names of seasons, etc.), to no avail. Eventually, he resorted to searching for *Vivaldi* and painstakingly working through several hundred matching items. When the sought-after work was finally located, it was revealed that it was cataloged under its Italian name (*Le quattro stagioni: La primavera, L’estate, L’autunno, L’inverno*) and could never have been found by a title-based search formulated in English. If only it had been possible to sing a few bars of one of the themes and use that for searching!

Figure 3.16 illustrates a digital library of popular MIDI tunes where searching is based upon what has been called “query by humming.” In Figure 3.16a a virtual piano keyboard is being used to tap out notes to form a symbolic query. An alternative is raw audio input, illustrated in Figure 3.16b, where the user presses a *record* button and starts to sing or hum a query—in this case, the first ten notes of *Fields of Gold* by Sting. Then the user presses *search*, which after a short delay brings up the view shown in Figure 3.16c.

During this time, the system analyzes the audio signal, segments it into individual notes and determines the pitch of each one. Then the collection is searched for matching songs. But musical themes recur in modified forms, and a user’s recollection will not necessarily match the version in the collection, so the machine is programmed to seek approximate matches. The amount by which the notes in the query have to change to match one of the songs in the collection (technically known as the *edit distance*) is used to order the results. Exact matches have an edit distance of zero and appear at the top of the list.

The result of converting the raw audio into symbolic notes is displayed in Figure 3.16c. Transcription errors may occur, depending on how clearly and accurately the user has sung, and the result is shown in music notation for the user to check. Alternatively, the query notes can be synthesized and played back. Below the query are the matching documents, with *Fields of Gold* at the top. Clicking the adjacent icon launches a MIDI player to play the song (Figure 3.16d).

Searching images

Figure 3.17 shows a system that supports text and image content queries, both individually and jointly, and illustrates the state of the art in automatic image content analysis. The user enters the query *flower*, which brings up a grid of matching thumbnails (there are 16 pages of thumbnails in all). The second thumbnail shows a carnation with a butterfly on its stem, against a defocused background of foliage. The user selects it and switches to the Content Viewer tab, which shows the larger version in Figure 3.17b, including some metadata—width and height, filename and textual keywords (in this case, *buds flora flower grass hawkweed insect insects moth nature on range*). Part of the metadata (although not visible here) is a time-stamp for each image, and a temporal view appears along the bottom of the window. The current image is in the center, flanked by its neighbors in time order, shrinking into the distance in both directions.

The user returns to the first view and right-clicks the image. This brings up a menu with options that include augmenting the search with the image itself. Selecting this and pressing the search button

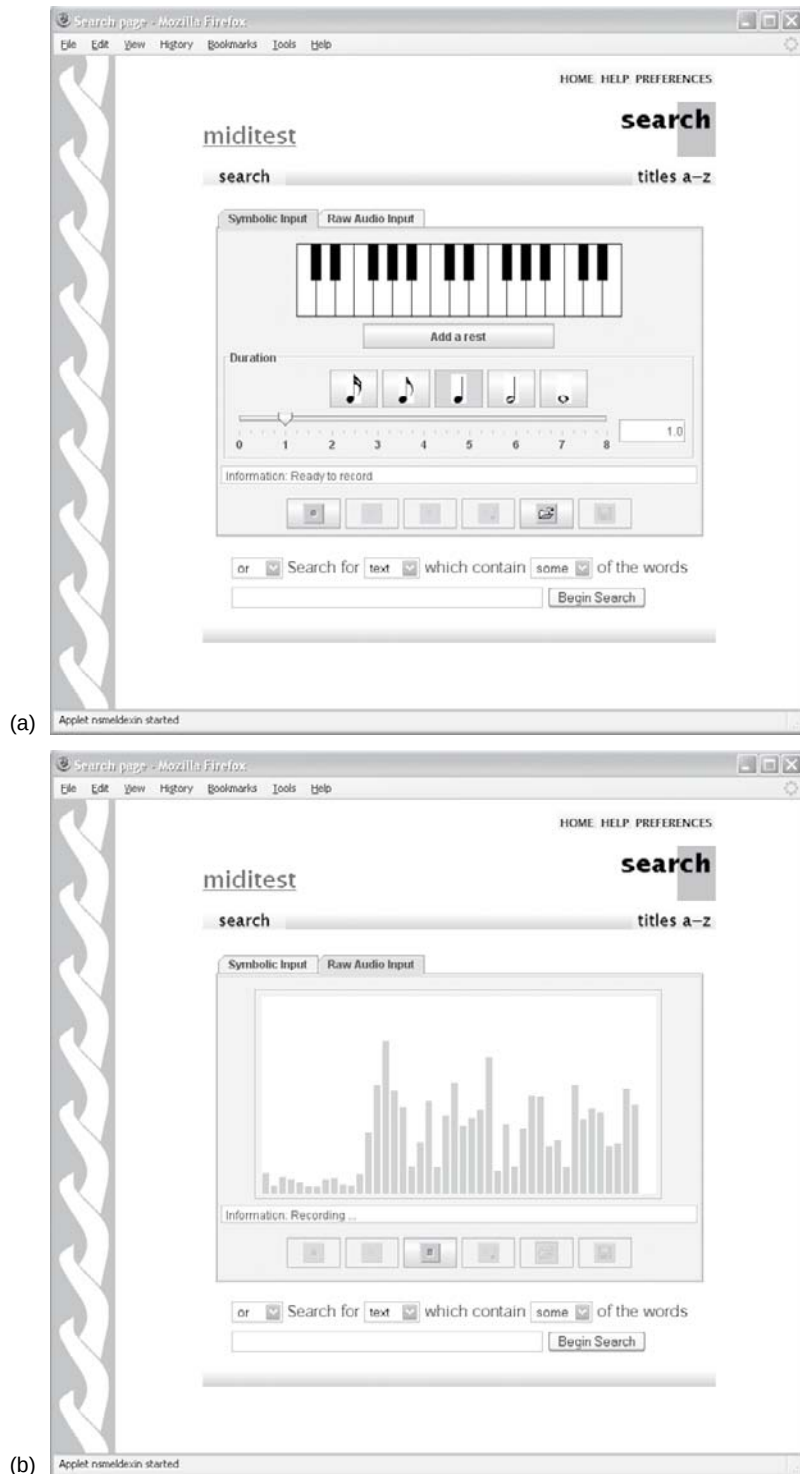
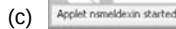


Figure 3.16: A musical digital library: (a) symbolic audio query; (b) singing a query;



The sliders to the left of the screen in [Figures 3.17a](#) and [c](#) each represent an aspect of similarity. There are nine in all, including one for text (the last five are obtained by scrolling). They are used to simultaneously control each aspect's influence on the decision. For example, setting the text slider to 0% and the others to 100% restricts the influence to features derived from the image content. (In this case, the same effect could be achieved by clearing the text query box.) The slider labels are cryptic—*Gabor 2-4*, *Convolution 2*, etc.—because the system is still an experimental research tool

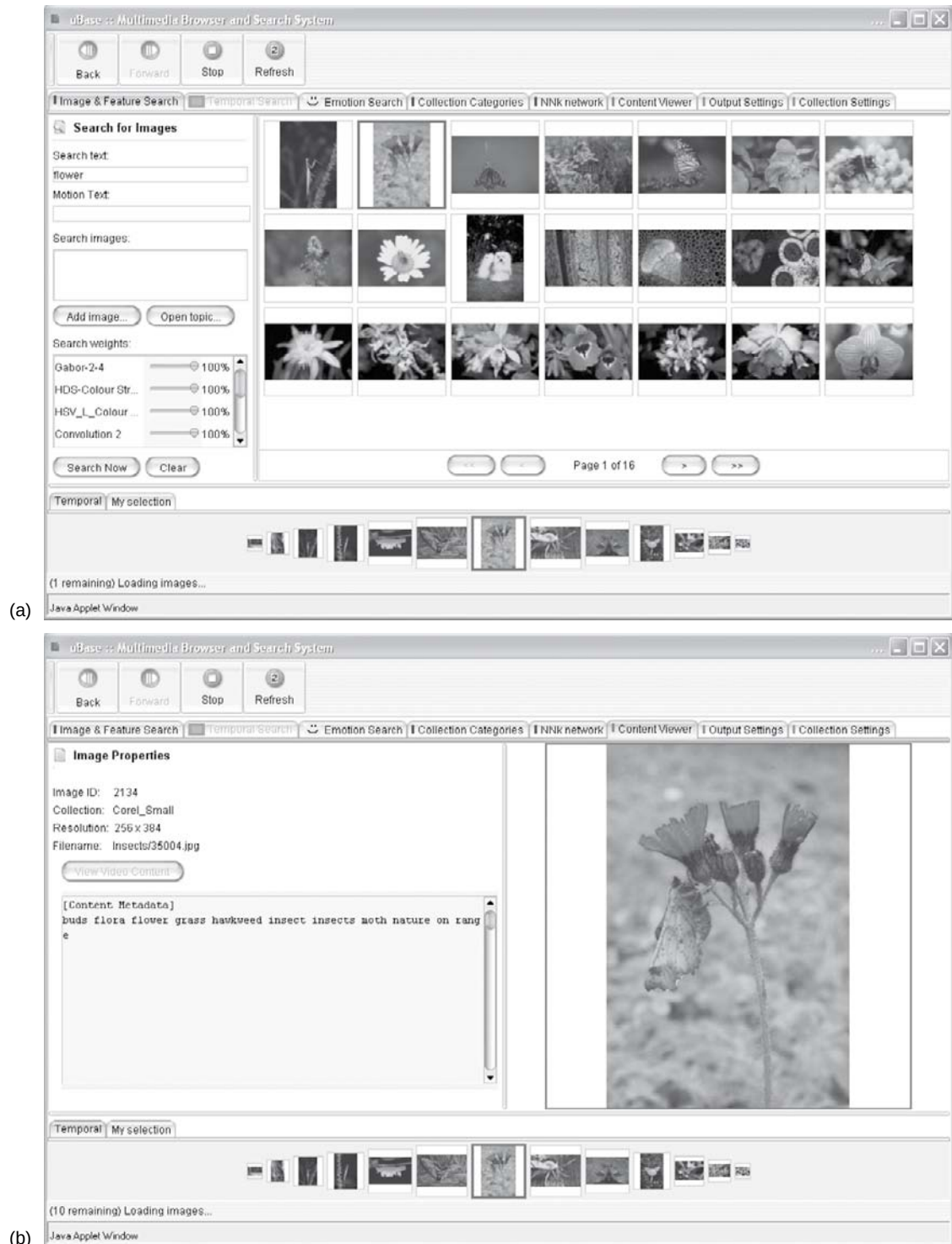


Figure 3.17: Searching image content: (a) text query based on metadata; (b) viewing an image;

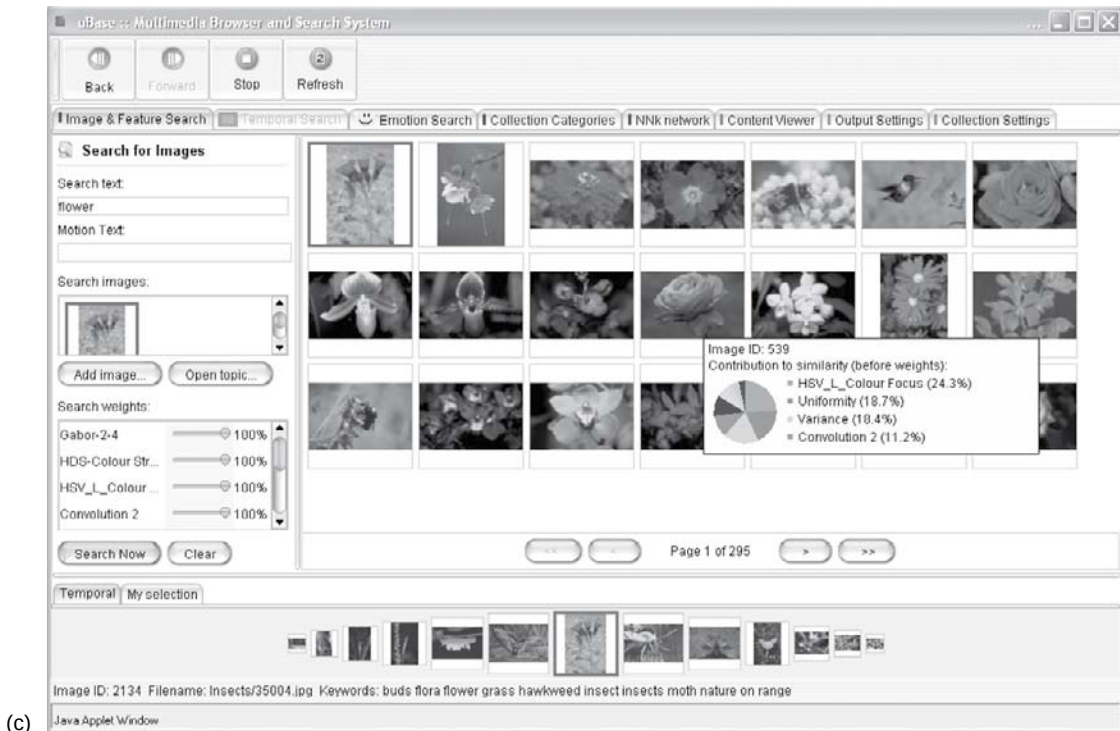


Figure 3.17, cont'd: (c) augmenting the query through content analysis

and these are technical names for the underlying algorithms. More conventional labels would have to be found for a general audience.

Returning to Figure 3.17c, combining the text query *flower* with images similar to the chosen one replaces the matching documents with ones whose hue is more reddish (although this is not apparent in the printed version of Figure 3.17c). The user's mouse is hovering over the fourth image in the central row, and details pop up explaining which features were responsible for this item's appearance so early in the result set. A feature called *HSV-L-Colour Focus* (HSV stands for hue, saturation and value) is dominant (at 24%); it is followed by features that gauge uniformity (19%), variance (18%), and convolution (11%).

3.5 Metadata Browsing

Browsing is often described as the other side of the coin from searching, but really the two are at opposite ends of a spectrum. One dictionary defines browsing as “inspecting in a leisurely and casual way,” whereas searching is “making a thorough examination in order to find something.” Other dictionaries have more verbose definitions. According to Webster's, browsing includes

- looking over casually (as a book), skimming;
- skimming through a book, reading at random passages that catch the eye;

- looking over books (as in a store or library), especially in order to decide what one wants to buy, borrow, or read;
- casually inspecting goods offered for sale, usually without prior or serious intention of buying;
- making an examination without real knowledge or purpose.

The word *browse* originally referred to animals nibbling on grass or shoots, and its use in relation to reading, which is now far more widespread, appeared much later. Early in the 20th century, the library community adopted the notion of browsing as “a patron’s random examination of library materials as arranged for use” when extolling the virtues of open book stacks over closed ones—the symbolic snapping of the links of the chained book mentioned in Section 1.2.

Searching is purposeful, whereas browsing tends to be casual. Terms such as *random*, *informal*, *unsystematic*, and *without design* are used to capture the unplanned nature of browsing and, often, the lack of a specific goal. Searching implies that you know what you’re looking for, whereas browsing implies that you’ll know it when you see it. Often, browsers are far less directed than that—perhaps just casually passing time. But the distinction between searching and browsing is not clear—pedants are quick to point out that if, when searching, you *really* know what you’re looking for, then there’s no need to find it. The truth is that we do not have a good vocabulary to describe various degrees of browsing.

The metadata provided with the documents in a collection can support different browsing activities. Information collections that are entirely devoid of metadata can be searched—that is one of the real strengths of full-text searching—but they cannot be browsed in any meaningful way unless some additional data is present. The structure that is implicit in metadata is the key to providing browsing facilities. And now it’s time for some examples.

Lists

The simplest structure is the ordered list. [Figure 3.18a](#) shows a plain alphabetical list of document titles. Notice incidentally that the alphabetizing follows the common practice of ignoring articles like *The* and *A* at the beginning of titles. For some lists the ordering may use another practice: names are conventionally alphabetized by surname, even though they may be preceded by first name and initials.

Long lists can take a long time to download and are cumbersome to scroll through. They are usually presented in alphabetic ranges, as in [Figure 3.18b](#). The user clicks a tag and a corresponding list of titles appears. In this example, the ranges have been automatically chosen so that each covers a reasonable number of documents—titles in the ranges *J–L*, *Q–R*, and *U–Y* have been merged because there are only a few under each of the letters. In fact, [Figure 3.18a](#) was generated in exactly the same way, but there are so few documents that there was just one overall range and so the alphabetic range selector was suppressed.

This scheme does not scale up well. Tabs with multiletter labels such as *Fae–Fal* are inconvenient. Although such labels are used in dictionaries and telephone directories, users generally take a stab at their desired location on the basis of the book’s bulk, and then they might employ the tabs. Going

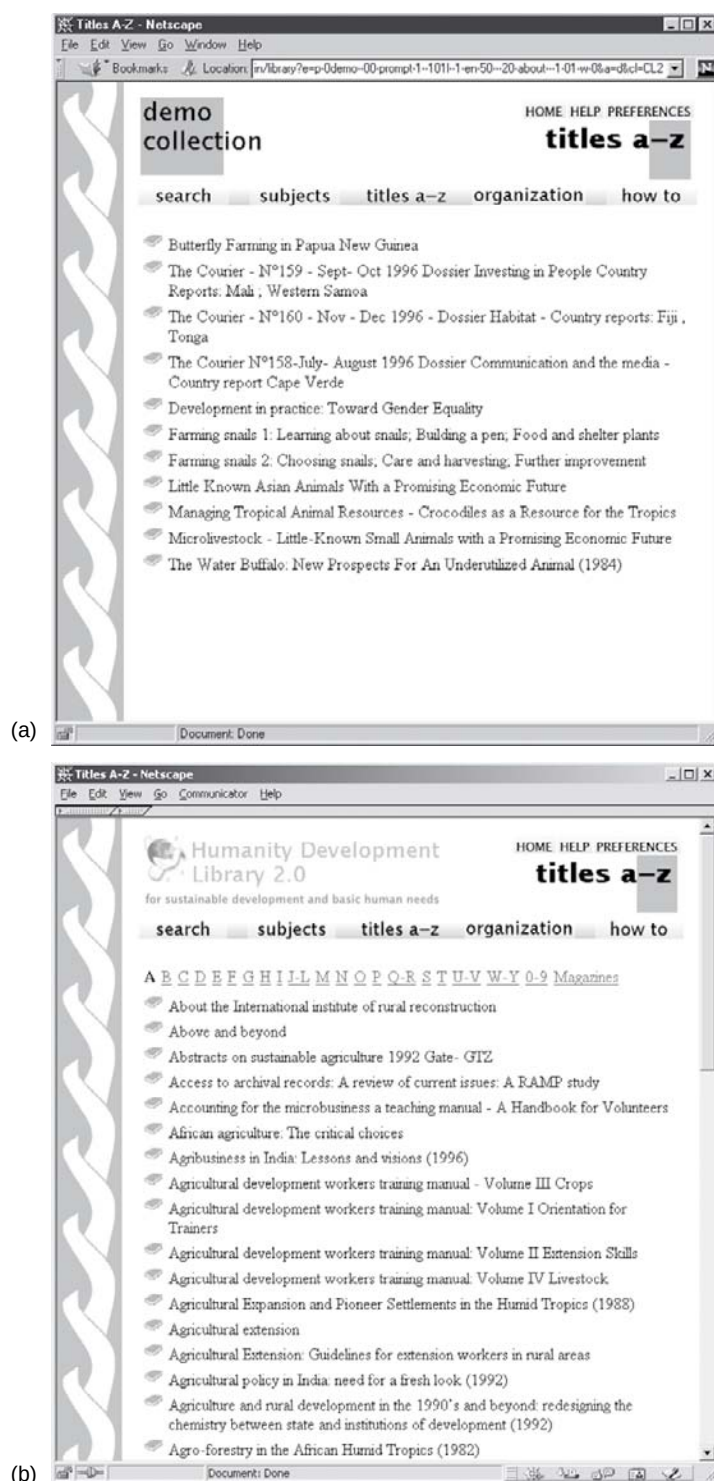


Figure 3.18: Browsing an alphabetical list of titles: (a) plain list; (b) with A-Z tags

through a sequence of decisions (*F*, *Fa*–*Far*, *Fae*–*Fal*, ...) is a tedious and unnatural way of narrowing down the search.

The final tab, 0–9, presents another snag. Users can't always know what characters titles start with—titles sometimes start with punctuation characters, arithmetic operators, or mathematical symbols. Fortunately, this is not a big problem in English because such documents rarely occur and can be dealt with by a single Miscellaneous tab.

Dates

In Figure 3.19, newspapers are browsed by date. An automatically generated selector gives a choice of years; items within each range are laid out by month. Figures 3.18 and 3.19 were created automatically based on *Title* and *Date* metadata respectively; the year ranges are chosen to put a reasonable number of items on each page.

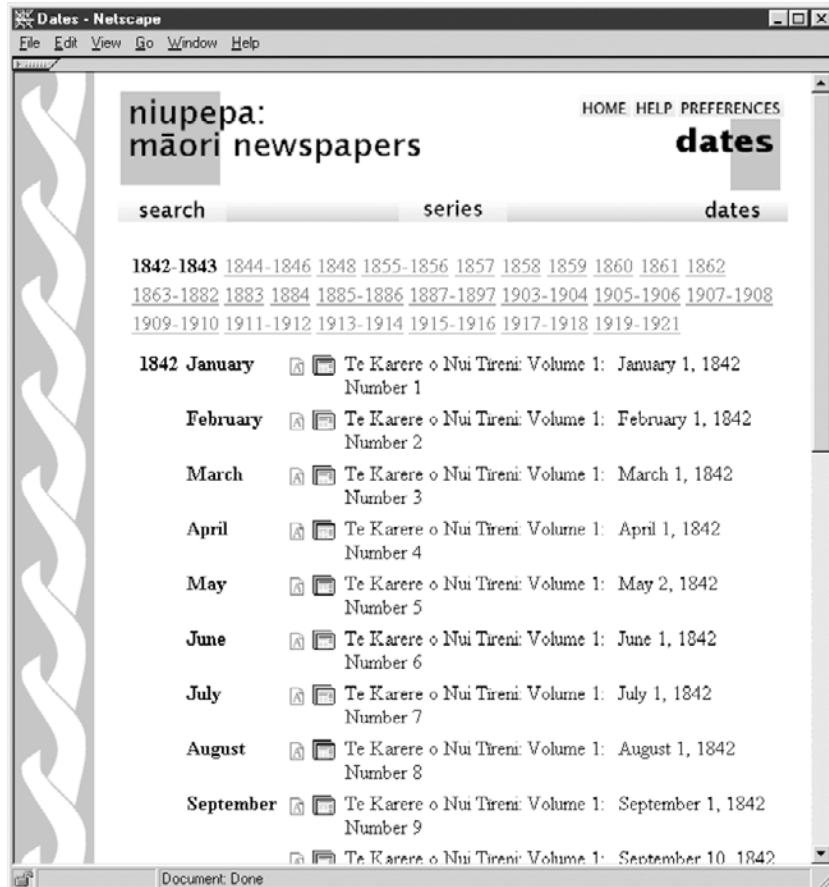


Figure 3.19: Browsing by date

Hierarchies

The browsers introduced so far are restricted to linear classifications with a limited number of documents. In contrast, hierarchical structures are used in areas that have very large numbers of items. In the library world, the Library of Congress and Dewey Decimal classifications are used to categorize printed books (enabling placement of volumes treating similar subjects on neighboring shelves). These schemes are considered *hierarchical* because the beginning parts of the code provide a rough categorization that is refined by the later characters.

Figure 3.20 shows a hierarchical display used in the Humanity Development Library. Nodes of the hierarchy are represented as bookshelves. Clicking one opens it up and displays all the nodes that lie beneath, as well as any documents at that level. For example, node 2.00 in Figure 3.20b contains one document and eight subsidiary nodes, of which one, node 2.06, is shown in Figure 3.20c. Just as bookshelf icons represent internal nodes of the hierarchy, so book icons represent documents, the leaves of the classification tree. Figure 3.20b shows a book icon for the *Earth Summit Report*, which is the only document with classification 2.00.

This hierarchical structure was generated automatically from metadata. Each document is accompanied by its associated position in the hierarchy. In fact, because documents can appear in several places, the metadata is multivalued. The hierarchical information includes names for the interior nodes, which are used to label the “bookshelves” in Figure 3.20. This particular classification scheme is nonstandard, chosen by the collection designer as being appropriate for the intended users. Some digital library systems impose uniformity; others provide flexibility for collection designers to organize things however they see fit. The latter option gives librarians freedom to exercise their professional judgment.

Facets

Given appropriate metadata, richer browsing options can be offered using the technique of *faceted classification*, which provides alternative navigation options and conveys information that helps users understand the content of the collections. Figure 3.21 illustrates facets in a search from the Australian Newspapers project. In Figure 3.21a the user has entered the query term *Waikato* (a region of New Zealand) and is viewing the result. There are over 1300 matching documents, the first ten being displayed in surrogate form, ranked, as usual, by relevance. The top item is a 1899 report in the *Northern Territory Times* that the SS *Waikato* was 42 days overdue on its trip from Vancouver to Auckland.

On the left are several categories—*facets*—that provide an alternative route for exploring and refining the results. The first, *Title*, is shown in its entirety, and the beginning of the second, *Category*, is also visible (scrolling down would reveal the rest). The labels under each facet show the number of documents tagged with that metadata value (e.g., 178 of the 1346 results came from the *Argus* newspaper), along with a visual representation as a bar graph. The user clicks on *Argus* to reveal the corresponding reduced set of matching documents in Figure 3.21b. The first matching item is a 1927 report about the Melbourne Cricket Club’s defeat of Waikato. Within this facet, the further facets displayed on the left can be explored by *Category*, *Illustrated*, *Decade*, and *Word Count*.

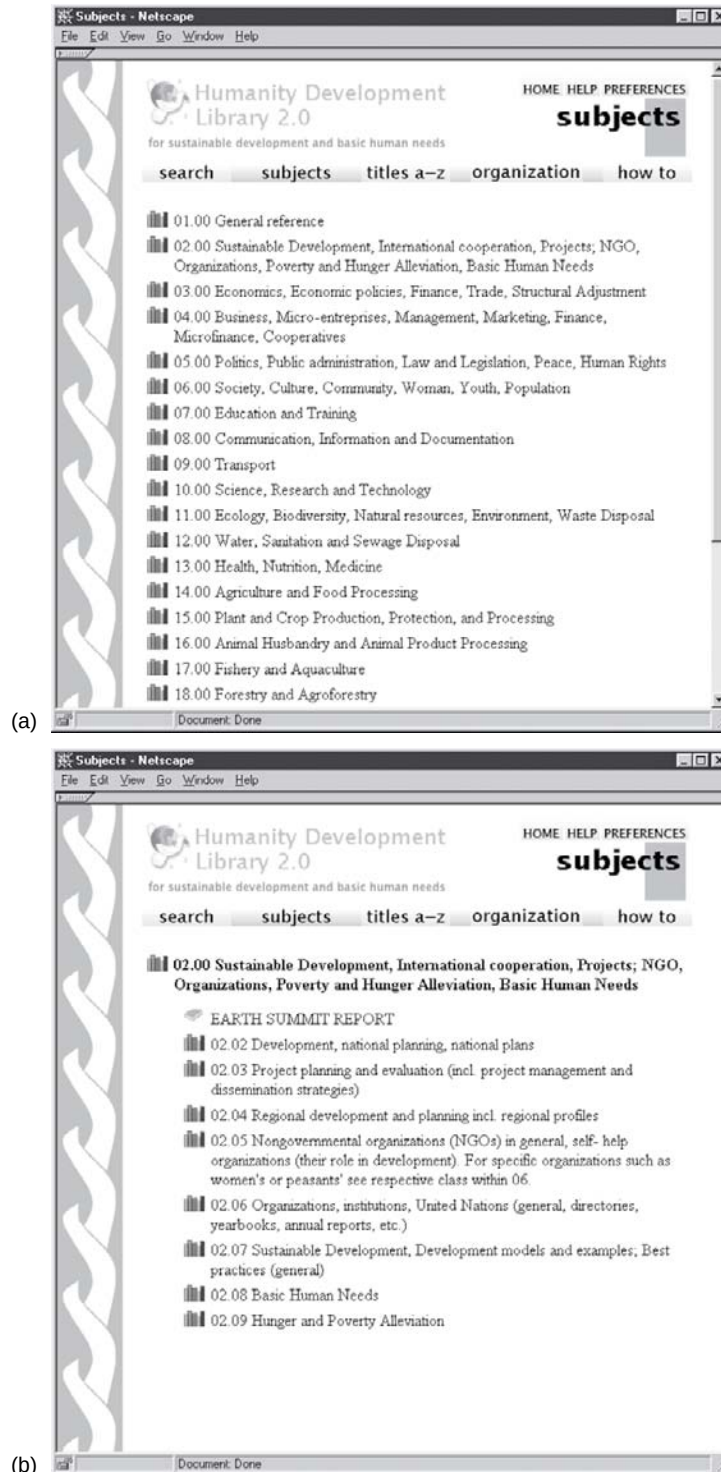


Figure 3.20: Browsing a classification hierarchy: (a) the beginning; (b) expanding *Sustainable development*;

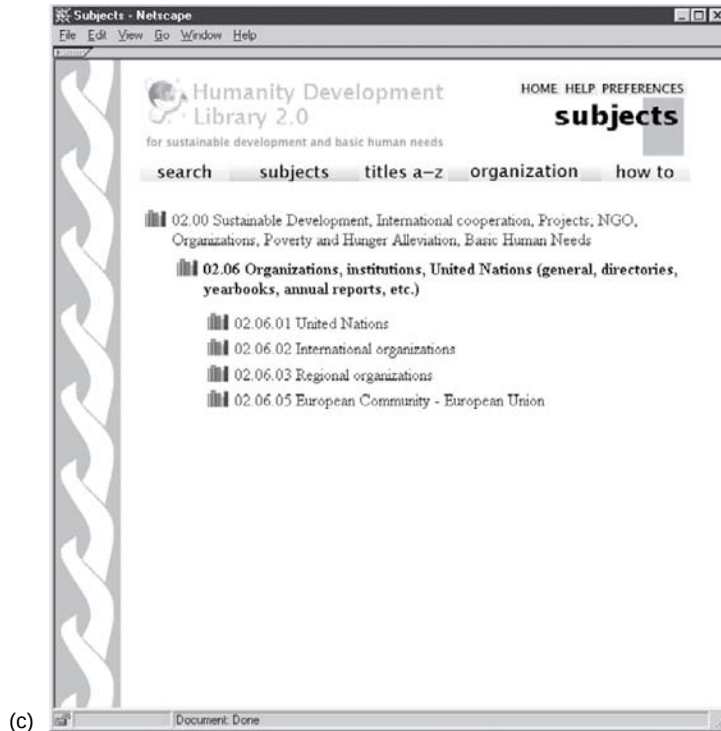


Figure 3.20, cont'd: (c) expanding *Organizations, institutions*

In general, facets use one of the forms of browsing described above. The Australian Newspapers project uses lists throughout, broken up into a series of pages that resemble search results. One could also imagine a *Subject* facet structured as a hierarchy and an *On this day* facet in calendar format. Facets are chosen by librarians based on the metadata available to guide users around collections. Not every piece of metadata makes a good facet. There are guidelines for effective facets; for example, facets should be mutually exclusive, permanent, and represent clear divisions of the items.

3.6 Putting It All Together

This section tours the Massachusetts Institute of Technology (MIT) institutional repository—a representative, publicly available digital library system—to illustrate the basic shape and form of many digital libraries today. The focus here is on the end-user's perspective. In Section 7.6 we revisit this example to explore it from a digital librarian's point of view.

An institutional repository

Figure 3.22a shows the home page of MIT's institutional repository, which is implemented in the DSpace system (see Notes and sources). It is a gateway to 30,000 digital items: technical reports,

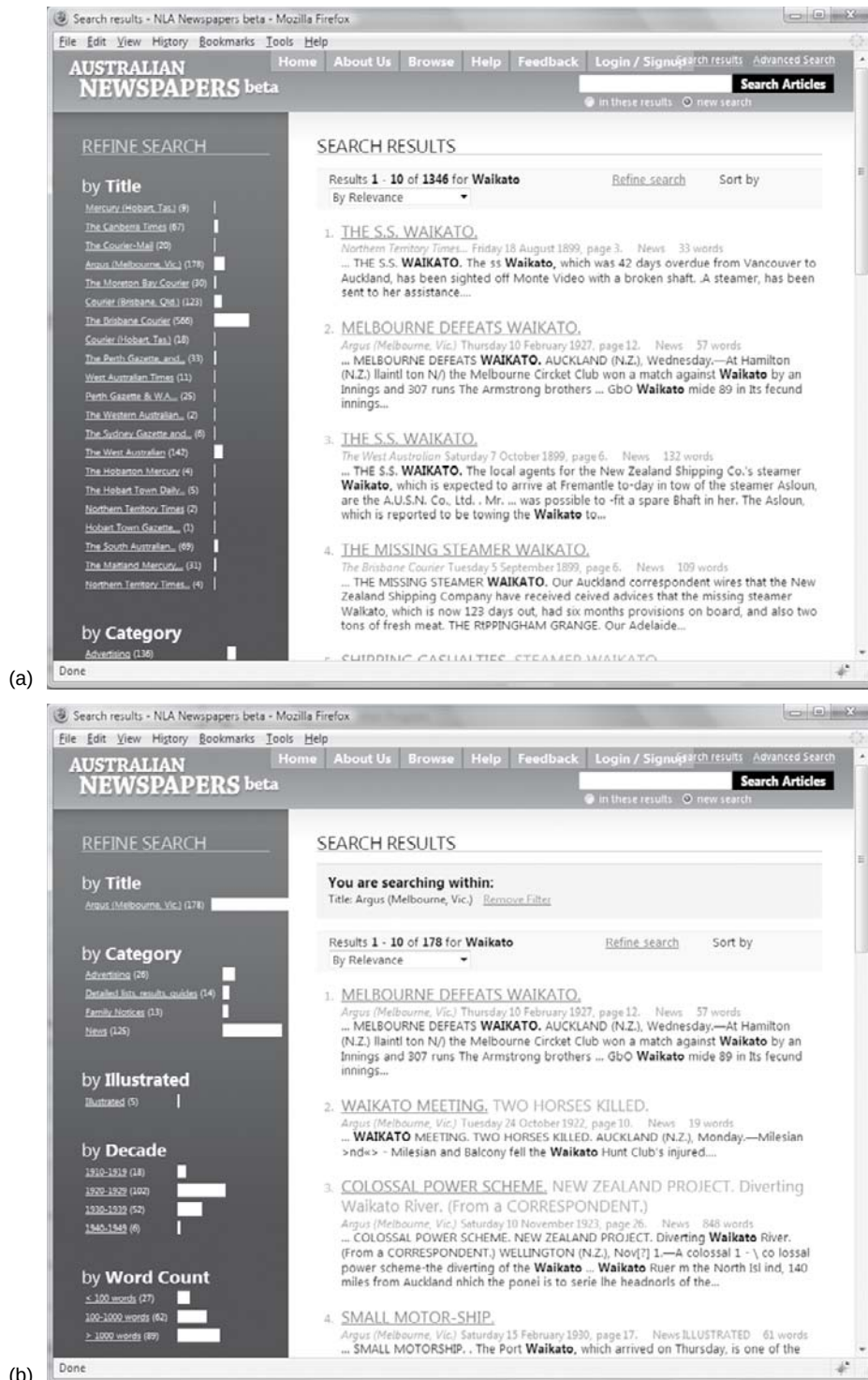


Figure 3.21: Facets in the Australian Newspapers project: (a) searching for the term **Waikato** across all newspapers; (b) exploring using the **Title** facet

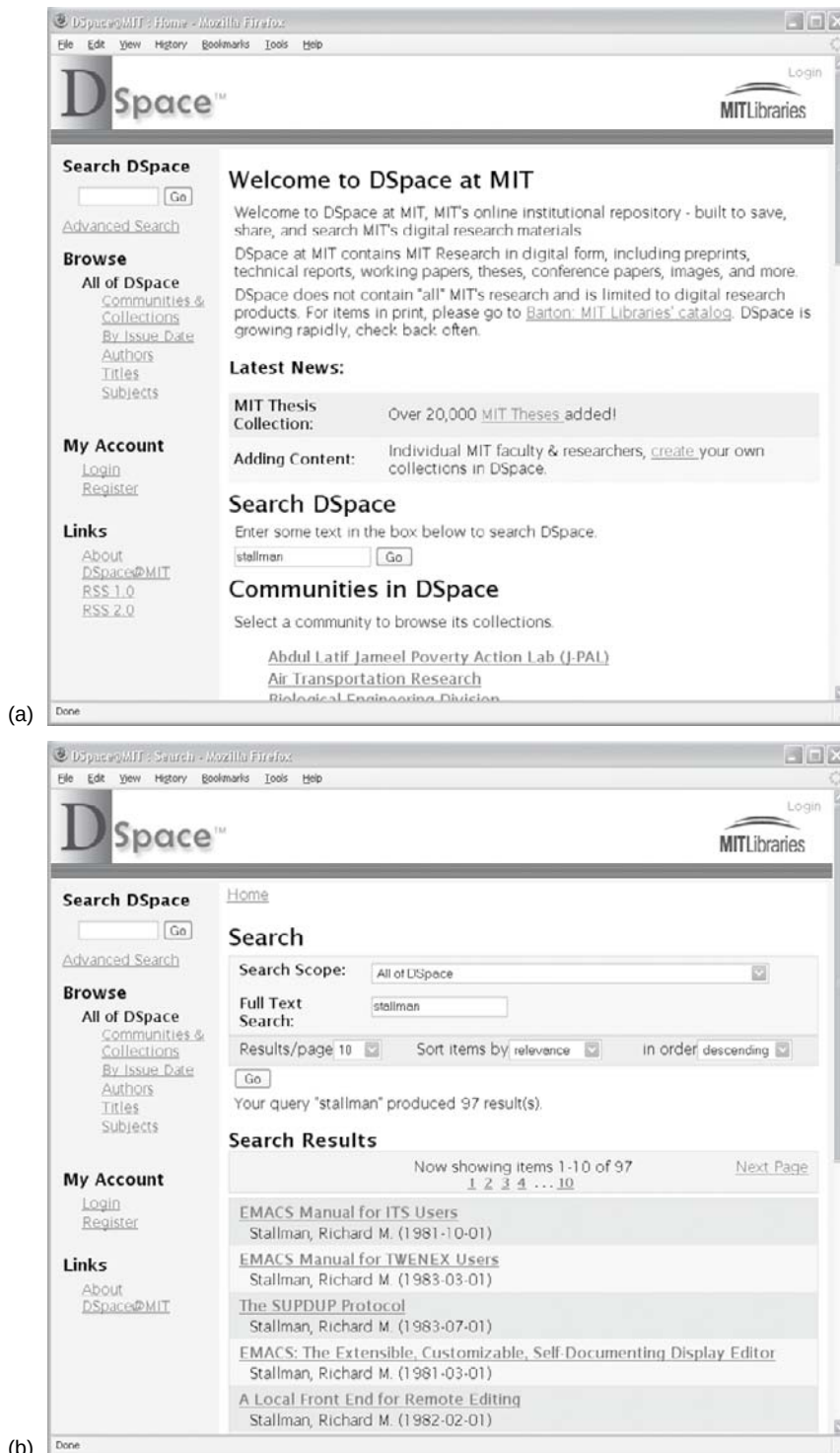


Figure 3.22: DSpace installation at MIT: (a) home page; (b) query result for Stallman;

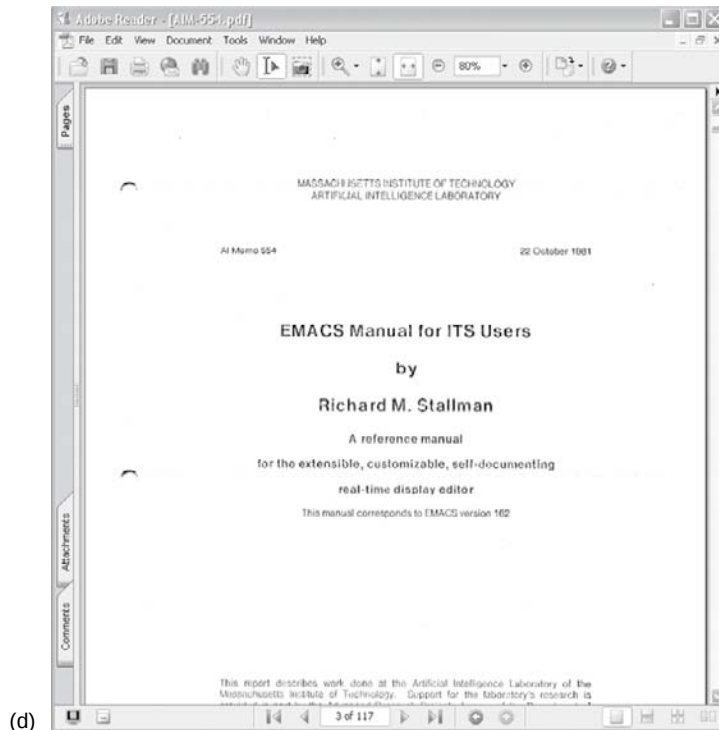
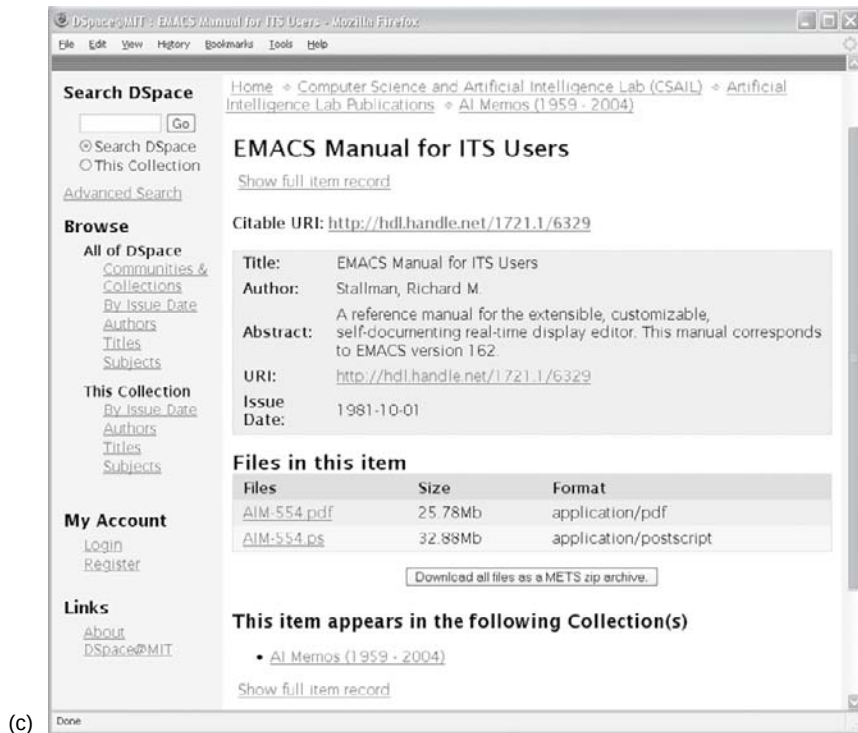
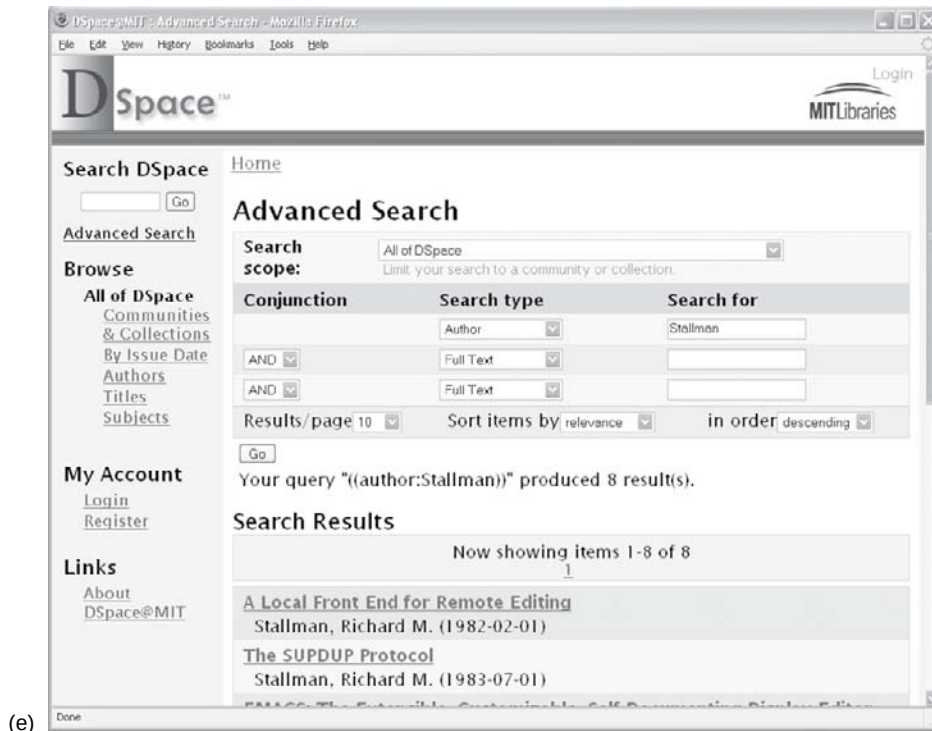


Figure 3.22, cont'd: (c) document overview; (d) the document itself;

Figure 3.22, cont'd: (e) searching by author for *Stallman*; (f) community and collections hierarchy;

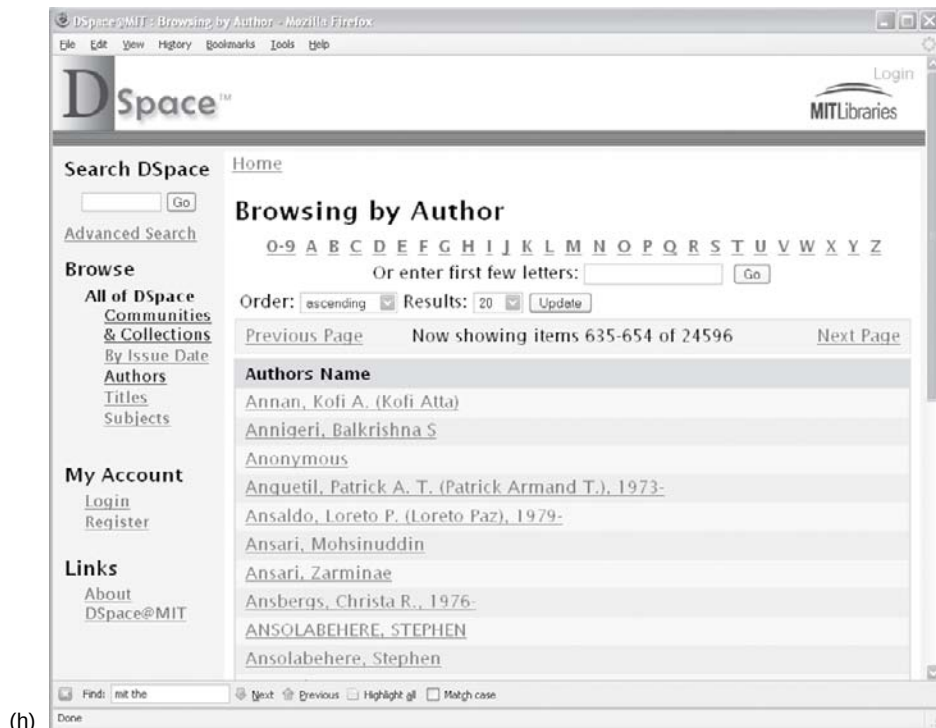
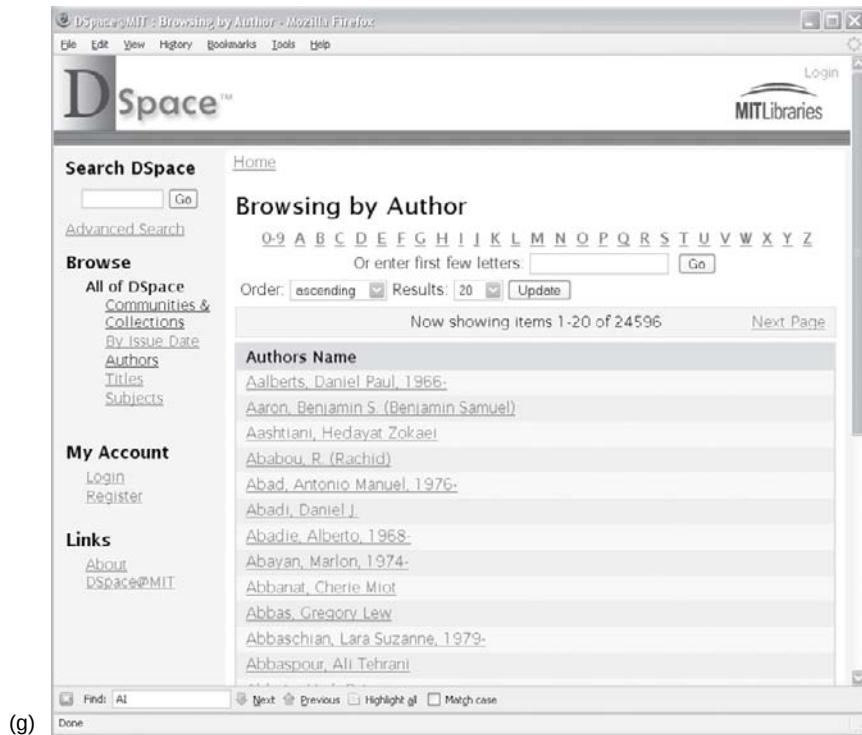


Figure 3.22, cont'd: (g) browsing by author; (h) pre-filtering authors by ann;

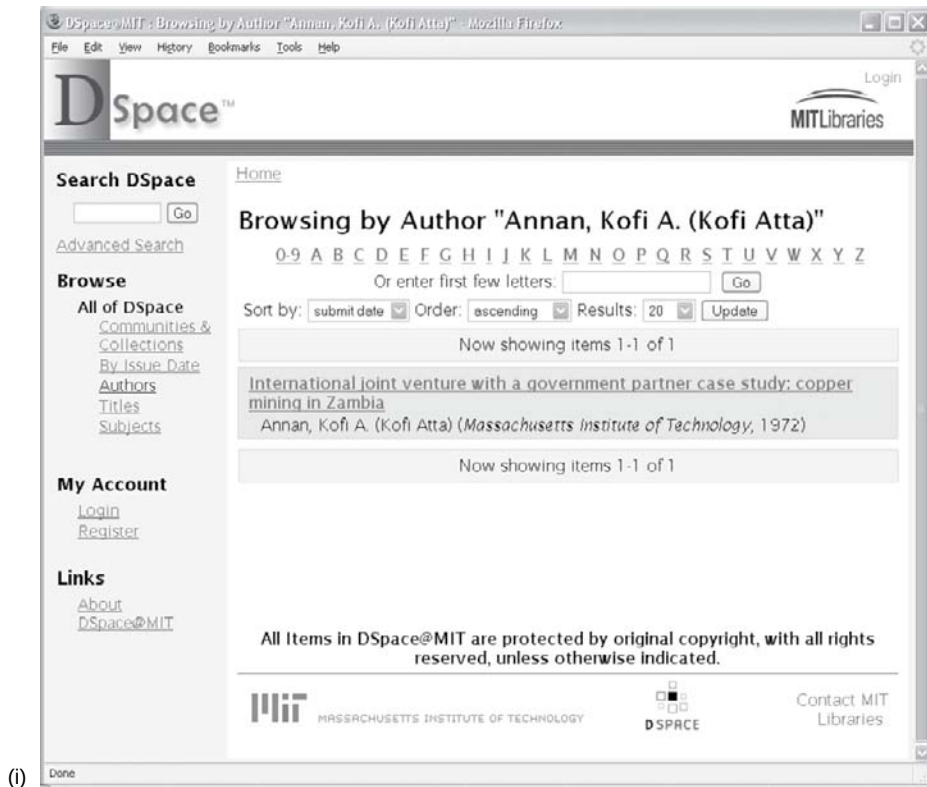


Figure 3.22, cont'd: (i) articles by *Annan, Kofi*

working papers, preprints, and theses. The repository is organized as a hierarchy: communities are the uppermost level, and they are hierarchically grouped into subcommunities and so forth, with individual collections at the bottom. Searching can be conducted at any level.

The left-hand margin of the screen provides access to all levels of the community hierarchy, plus browsing by date, author, and subject. Below this is where users register and log in. Searching and browsing are unrestricted, but an account is needed for services like requesting e-mail alerts. Registration is open to all, but MIT affiliates are distinguished from others and are granted access to restricted items as well as the ability to submit new items to the repository. The bottom two links in the margin connect users to RSS feeds, which provide information about new items that are added. Two variants are supported: RSS 1.0 and RSS 2.0 (depending on who you ask, the letters stand for *really simple syndication*, *rich site summary* or *RDF site summary*). Both are XML based (see Section 4.3 for more information on XML and Section 6.4 for a description of RDF).

A user interested in the work of Richard Stallman, founder of the GNU Project and the Free Software Foundation, enters the query *stallman* in the main search box. Figure 3.22b shows the result: 97 matches. The first hit is a 1981 manual for the open source EMACS text editor developed by Stallman, and clicking this link brings up Figure 3.22c, a document surrogate displaying basic metadata:

title, author, abstract, and issue date. The collection—*AI Memos*—is displayed at the top, along with the associated community—*Computer Science and Artificial Intelligence Lab, Artificial Intelligence Lab Publications*.

The user clicks a link to view the document. Once it has downloaded (26 MB) it appears in a new window, [Figure 3.22d](#). Despite the fact that it was born digital—as the manual itself points out, it can be viewed online as part of the Emacs help system or purchased in print for \$3.25—this item in the collection is derived from a scanned copy. This is not surprising: the document was authored a year before the inception of PostScript (described in Section 4.5).

The fact that the document is digitized accounts for its vast download size and the evident graininess of the pages. It has not undergone OCR and can only be located through metadata, not full text search. However, the result set does include later documents for which a searchable version had been generated directly from the computer file, and these are included because their full text also contains *stallman*. In fact, all matching full-text items turn out to be other authors referencing Stallman’s work. An “advanced” facility in the digital library can be used to restrict the search to the author’s name, and the more carefully constructed query in [Figure 3.22e](#) returns only eight matching documents.

Next the user clicks the *Communities and Collections* link on the left to examine the complete repository, listed alphabetically by top-level community and indented to convey the hierarchical structure. The ordering continues recursively within communities, so you can drill down and commence queries at any level. The list is very long, so our user elects to browse by author instead, clicking the appropriate link to bring up [Figure 3.22g](#). Here the user can jump immediately to any letter of the alphabet using the navigation bar; then he can step through the authors using Next page and Previous page links. Twenty authors are displayed per page; this can be changed to 5, 10, 40, 60, 80, or 100 using a pull-down menu. The sort order can also be changed from ascending to descending. Pressing *Update* reloads the page with the new setting.

As noted earlier, alphabetic browsing does not scale well. MIT’s repository has 25,000 authors and most letters are tedious to click through even 100 items at a time. However, users can enter the first few letters on the browsing page to filter the information displayed. To get to [Figure 3.22h](#) our user typed *ann*, and then selected the author *Annan, Kofi* to yield [Figure 3.22i](#), which shows a list of all the articles by the former UN Secretary-General. In this case there is just one: a 1972 Master’s thesis at the Sloan School of Management entitled *International joint venture with a government partner case study: copper mining in Zambia*. The document is viewed just as before, but only registered MIT users can print it because it belongs to the Thesis collection community. For other users, a link is provided to purchase a printable copy.

3.7 Notes and Sources

The Humanity Development Library in [Figure 3.1](#) is mentioned in Chapter 1; it is available both on CD-ROM and interactively at www.nzdl.org. More information about Project Gutenberg, from which [Figures 3.2](#) and [3.11 to 3.15](#) come, can be found at www.gutenberg.org. When we wrote the first edition of this book in 2001, it took a few hours to download the 4,500 available e-texts, and a

similar amount of time to create a digital library from it. By 2008 the collection had grown over threefold, to 17,000 e-texts, but increases in Internet connectivity and computer processing speeds meant that it took roughly the same time both to download and to build an updated version.

The New Zealand School Journal is produced by the Ministry of Education and published by Learning Media Ltd. in New Zealand. Its purpose is to provide quality writing that is relevant to the needs and interests of New Zealand children, and the ultimate aim is to foster a love of reading. Figure 3.3 is taken from a prototype collection containing a few sample journals across the entire range of years from 1907 onward; it was designed by Mankelow (1999). It gives a fascinating historical perspective on the development of attitudes toward children's education—and indeed the development of society in general—throughout the last century. For copyright reasons, the collection is not publicly available.

The National Library of New Zealand's Papers Past collection in Figure 3.4 is implemented in Greenstone, and Section 11.8 discusses its construction, size, and the resources it requires. It's on the web at <http://paperspast.natlib.govt.nz>.

The realistic book in Figure 3.5 was produced by an open source implementation that allows such books to be easily produced from HTML or PDF files (Liesaputra et al., 2009). It is available from www.nzdl.org/books. The Oral History collection in Figure 3.6 was originally produced on audio-tapes by Hamilton Public Library; Bainbridge and Cunningham (1998) describe its conversion into digital form. Early research on music collections and music searching, some of which is illustrated in Figure 3.7, is described by McNab et al. (1996) and Bainbridge et al. (1999). The 99% figure for the uptake of Flash in Web browsers comes from a June 2008 Millward Brown survey that is conducted quarterly and spans ten countries (www.adobe.com/products/player_census/flashplayer).

Figure 3.8 is from a large collection of computer science bibliographies assembled and maintained by Alf-Christian Achilles at the University of Karlsruhe, Germany (<http://liinwww.ira.uka.de/bibliography>). Figure 3.9a was obtained from the Library of Congress using Greenstone's Z39.50 client. Figure 3.9b is from a million-entry collection of metadata for the BBC radio and television archives. It was produced in the form of a coordinated set of five CD-ROMs that can be loaded onto disk and searched either individually or together in a seamlessly coordinated fashion, making any laptop into a full catalog server.

Much of the material on searching is taken from the book *Managing Gigabytes* (Witten et al., 1999), which gives a comprehensive and detailed technical account of how to compress documents and indexes, yet make them rapidly accessible through full-text queries. There are many standard reference works covering the area of information retrieval; the best-known are Salton (1989), Salton and McGill (1983), van Rijsbergen (1979), Frakes and Baeza-Yates (1992), Korfhage (1997), and Baeza-Yates and Ribeiro-Neto (1999). Lovins (1968) and Porter (1980) are two of many authors who have produced particular algorithms for stemming. Frakes surveys this area in a chapter of a book that he coedited (Frakes, 1992); Lennon et al. (1981) have also examined the problem. We return briefly to the topic of text searching in Section 4.1.

The “famous quote” about simple and complex things near the beginning of Section 3.3 is from Alan Kay, a guru in the interactive computing world; an interesting article about him appeared in *New Sci-*

entist (Davidson, 1993). George Santayana, who feared that those who ignored history would have to relive (not retype) it, was a philosopher, poet, critic, and best-selling novelist (Santayana, 1932). Oscar Wilde, on the other hand, said that “The one duty we owe to history is to rewrite it.” Perhaps it doesn’t do to take these *bon mots* too seriously.

The origins of automatic speech recognition can be traced back to 1936 at AT&T’s Bell Labs (www.dragon-medical-transcription.com/historyspeechrecognition.html). Open source speech recognition software is available from Carnegie Mellon University’s Sphinx project, and the University of Cambridge’s HTK project.

Figure 3.16 is from the Meldex collection at www.nzdl.org/meldex. Although the illustrated example focuses exclusively on musical queries, they can be combined with textual metadata queries—the text field and pull-down menu for this can be seen in the lower portion of Figure 3.16a. Studies have shown that most users can recall at least one fragment of bibliographic information when performing a music-seeking task (Bainbridge et al., 2003). The advanced image retrieval system in Figure 3.17 is the U.K. Open University’s uBase, at <http://kmi.open.ac.uk/technologies/ubase>. The system has many advanced capabilities; what we have shown is just a taster.

Chang and Rice (1993) examine the activity of browsing from a broad interdisciplinary point of view. Hyman (1972) discusses how the notion of browsing in traditional libraries was linked to the rise of the open-stack organization; the quotation near the beginning of Section 3.5 about the “patron’s random examination of library materials” is his. The brief characterization of browsing is from the *Houghton Mifflin Canadian Dictionary of the English Language*; the longer one is from *Webster’s Third New International Dictionary*.

The list of titles in Figure 3.18a shows the Demo collection, which is supplied with every Greenstone installation and contains a small subset of the Humanity Development Library; Figure 3.18b is from the full Humanity Development Library, as is Figure 3.20. The date list in Figure 3.19 was generated within the Māori-language Niupepa collection described in Chapter 8. Spiteri (1998) provides guidelines for choosing facets, and Yee et al. (2003) report on the effectiveness of facets for searching and browsing in image collections.

The example used to illustrate facets, the Australian Newspapers project, is part of Australia’s Newspapers Digitization Program at <http://ndpbeta.nla.gov.au>. In early 2009 it was still under development, and the functionality and URL may both change.

MIT’s institutional repository can be accessed at <http://dspace.mit.edu/>. It was the catalyst for the open source project DSpace, a joint venture between Hewlett Packard (www.dspace.org) and MIT. The size quoted for this repository is an estimate; we could find no definitive way to ascertain the total size.

The document viewed, Richard Stallman’s 1981 Emacs Manual, is actually a scanned version of every second page: pages 1, 3, 5; a handwritten note alerts the reader to this problem. This suggests that at some point a double-sided version was copied single-sided and the original was lost; an interesting lesson in document preservation.

Textual documents: The raw material

Documents are the digital library's building blocks. It is time to step down from our high-level discussion of digital libraries—what they are, how they are organized, and what they look like—to nitty-gritty details of how to represent the documents they contain. To do a thorough job for international documents in non-Roman alphabets, we will have to descend even further and look at the representation of the characters that make up textual documents.

A great practical problem anyone faces when writing about the innards of digital libraries is the high rate of change in the core technologies. Perhaps the most striking feature of the digital library field is the inherent tension between two extremes: the dizzying pace of technological change and the very long-term view that libraries must take. We must reconcile any aspirations to surf the leading edge of technology with the literally static ideal of archiving material “for ever and a day.”

Document formats are where the rubber meets the road. Over the last two decades a cornucopia of different representations have emerged. Some have become established as standards, either official or *de facto*, and it is on these that we focus. There are plenty of them: the nice thing about standards, they say, is that there are so many different ones to choose from!

The standard ASCII (American Standard Code for Information Interchange) code used by computers has been extended in dozens of different and often incompatible ways to deal with other character sets. Some Web pages specify the character set explicitly. Internet Explorer recognizes over 100 different character sets, mostly extensions of ASCII and another early standard called EBCDIC, some of which have half a dozen different names. Without a unified coding scheme, search programs must know about all this to work correctly under every circumstance.

Fortunately, there is an international standard for representing characters: Unicode. It emerged over the last 20 years and is now stable and widely used—although it is still being extended and developed to cover languages and character sets of scholarly interest (e.g., archaic ones). It allows the content of digital libraries, and their user interfaces, to be internationalized. The basics of Unicode are introduced in the next section: a more comprehensive account is included in Chapter 8 on internationalization. Of course, Unicode only helps with character *representation*. Language translation and cross-language searching are thorny matters of computational linguistics that lie completely outside its scope (and outside this book's scope, too).

[Section 4.1](#) discusses the representation of plain text documents, and related issues. Even the most basic character representation, ASCII, can present ambiguities in interpretation. We also sketch how an index can be created for textual documents in order to facilitate rapid full-text searching. Before the index is created, the input must be split into words. This involves a few mundane practical

decisions—and introduces deeper issues for languages that are not traditionally written with spaces between words.

Probably the single most important issue when contemplating a digital library project is whether you plan to digitize the material for the collection from ordinary books and other documents. This inevitably involves manually handling physical material, and perhaps also manual correction of computer-based text recognition. It generally represents the vast bulk of the work involved in building a digital library. [Section 4.2](#) describes the process of digitizing textual documents, by far the most common source of digital library material. Most library builders end up outsourcing the operation to a specialist; this section alerts you to the issues you will need to consider when planning this part of the project.

The next section describes HTML, the Hypertext Markup Language, which was designed specifically to allow references, or *hyperlinks*, to other files, including picture files, giving a natural way to embed illustrations in the body of an otherwise textual document

Electronic documents have two complementary aspects: structure and appearance. Structural markup makes certain aspects of the document structure explicit: section divisions, headings, subsection structure, enumerated and bulleted lists, emphasized and quoted text, footnotes, tabular material, and so on. Appearance is controlled by *presentation* or *formatting* markup that dictates how the document appears typographically: page size, page headers and footers, fonts, line spacing, how section headers look, where figures appear, and so on. Structure and appearance are related by the design of the document, that is, a catalog—often called a *style sheet*—of how each structural item should be presented. While HTML supports this separation, it does not enforce it and, worse, given the format's chaotic beginnings it quickly became augmented with a host of other facilities that blurred this distinction.

Whatever its faults, HTML, being the foundation for the Web, is a phenomenally successful way of representing documents. However, when dealing with collections of documents, different ways of expressing formatting in HTML tend to generate inconsistencies—even though the documents may look the same. Although these inconsistencies matter little to human readers, they are a bane for automatic processing of document collections. Next we describe XML, a more modern development that is intended to solve some of the problems that arise with HTML. XML is an *extensible* markup language that allows you to declare what syntactic rules govern a particular group of files. More precisely, it is referred to as a *metalanguage*—a language used to define other languages. XML provides a flexible framework for describing document structure and metadata, making it ideally suited to digital libraries. It has achieved widespread use in a short period of time—reflecting a great demand for standard ways of incorporating metadata into documents—and underpins many other standards. Among these are ways of specifying style sheets that define how particular families of XML documents should appear on the screen or printed page. There is also XHTML, a recasting of this format using the stricter syntax of XML.

Style sheets have been mentioned several times. Associated with the markup languages HTML and XML are the stylesheet languages CSS (cascading style sheets) and XSL (extensible stylesheet language), and these are introduced in [Section 4.4](#).

After thoroughly reviewing HTML and XML, we move on to more comprehensive ways of describing finished documents. Desktop publishing empowers ordinary people to create carefully designed and lavishly illustrated documents and to publish them electronically, often dispensing with print altogether. People have quickly become accustomed to reading publication-quality documents online. It is worth reflecting on the extraordinary change in our expectations of online document presentation since, say, 1990. This revolution has been fueled largely by the PostScript language and its successor, the Portable Document Format (PDF). These are both *page description* languages: they combine text and graphics by treating the glyphs that comprise text as little pictures in their own right and allowing them to be described, denoted, and placed on an electronic page alongside conventional illustrations.

Page description languages portray finished documents, ones that are not intended to be edited. In contrast, word processors represent documents in ways that are expressly designed to support interactive creation and editing. Society's notion of *document* has evolved from painstakingly engraved Chinese steles, literally "carved in stone," to hand-copied medieval manuscripts; from the interminable revisions of loose-leaf computer manuals in the 1960s and 1970s to continually evolving Web sites whose pages are dynamically composed on demand from online databases, and it seems inevitable that more and more documents will be circulated in word-processor formats.

[Section 4.6](#) describes examples of word-processor documents, starting with the ubiquitous Microsoft Word format and Rich Text Format (RTF). Word is intended to represent working documents inside a word processor. It is a proprietary format that depends on the exact version of Microsoft Word that was used and it is not always backward-compatible. RTF is more portable, being intended to transmit documents to other computers and other versions of the software. It is an open standard, defined by Microsoft, for exchanging word-processor documents between different applications.

Winds of change are sweeping through the world of document representation. XML and associated standards are beginning to be adopted for fully fledged word-processor documents. The Open Document Format (ODF) is an XML-based format that can describe word-processor documents and many more: from spreadsheets to charts, from presentations to mathematical formulae. ODF has been developed and promoted by a not-for-profit consortium called the Organization for the Advancement of Structured Information Standards, which is strongly associated with open source software, and whose mission is to drive the development and adoption of open standards for the global information society. Office Open XML (OOXML) is a similar (but, unfortunately, different) XML-based document format being developed by Microsoft to represent documents in its Office line of products. These two standards are vying to be the dominant format for describing word-processor documents, and we discuss the controversy and describe ODF in [Section 4.6](#). We also describe the format used by the LaTeX document processing system, a system which has been around for over 20 years and is widely used to represent documents in the scientific and mathematical community.

Finally, [Section 4.7](#) takes a brief look at other predominantly textual document formats: spreadsheets, presentation files, and e-mail.

This chapter gets down to details, the dirty details (where the devil is). You may find the level uncomfortably low. Why do you need to know all this? The answer is that when building digital

libraries you will be presented with documents in many different formats, yet you will yearn for standardization. You have to understand how different formats work in order to appreciate their strengths and limitations. Examples? When converted from PDF to PostScript, documents lose interactive features like hyperlinks. Converting HTML to PostScript is easy (your browser does it every time you print a Web page), but converting an arbitrary PostScript file to HTML is next to impossible if you want a completely accurate visual replica.

Even if your project starts with paper documents, you still need to know about online formats. The optical character recognition process may produce Microsoft Word documents, retaining much of the formatting in the original and leaving illustrations and pictures *in situ*. But how easy is it to extract the plain text for indexing purposes? To highlight search terms in the text? To display individual pages? Perhaps another format is preferable?

4.1 Representing Textual Documents

Way back in 1963, at the dawn of interactive computing, the American National Standards Institute (ANSI) began work on a character set that would standardize text representation across a range of computing equipment and printers. (At the time, a variety of codes were in use by different computer manufacturers, such as an extension of a *binary-coded decimal* punched card code to deal with letters—EBCDIC, or Extended Binary Coded Decimal for Information Interchange—and the European Baudot code for teleprinters that accommodated mixed-case text by switching between upper- and lowercase modes.) In 1968, ANSI finally ratified the result, called ASCII: American Standard Code for Information Interchange. Until recently, ASCII dominated text representation in computing.

ASCII

Table 4.1 shows the ASCII character set, with code values in decimal, octal, and hexadecimal. Codes 65–90 (decimal) represent the uppercase letters of the Roman alphabet, while codes 97–122 are lowercase letters. Codes 48–57 give the digits zero through nine. Codes 0–32 and 127 are *control characters* that have no printed form. Some of these govern physical aspects of the printer—for instance, BEL rings the bell (now downgraded to an electronic beep), BS backspaces the print head (now the cursor position). Others indicate parts of a communication protocol: SOH starts the header, STX starts the transmission. Interspersed between these blocks are sequences of punctuation and other nonletter symbols (codes 33–47, 58–64, 91–96, 123–126). Each code is represented in seven bits, which fits into a computer byte with one bit (the top one) free. In the original vision for ASCII, this was earmarked for a parity check.

ASCII was a great step forward. It helped computers evolve over the following decades from scientific number-crunchers and fixed-format card-image data processors to interactive information appliances that permeate all walks of life. However, it has proved a great source of frustration to speakers of other languages. Many different extensions have been made to the basic character set, using codes 128–255 to specify accented and non-Roman characters for particular languages. ISO 8859-1, from the International Organization for Standardization (the international counterpart of the American standards organization, ANSI), extends ASCII for Western European languages. For example, it

Table 4.1: The ASCII character set

Dec	Oct	Hex	Char	Dec	Oct	Hex	Char
0	000	00	NUL	44	054	2C	,
1	001	01	SOH	45	055	2D	-
2	002	02	STX	46	056	2E	.
3	003	03	ETX	47	057	2F	/
4	004	04	EOT	48	060	30	0
5	005	05	ENQ	49	061	31	1
6	006	06	ACK	50	062	32	2
7	007	07	BEL	51	063	33	3
8	010	08	BS	52	064	34	4
9	011	09	HT	53	065	35	5
10	012	0A	LF	54	066	36	6
11	013	0B	VT	55	067	37	7
12	014	0C	FF	56	070	38	8
13	015	0D	CR	57	071	39	9
14	016	0E	SO	58	072	3A	:
15	017	0F	SI	59	073	3B	;
16	020	10	DLE	60	074	3C	<
17	021	11	DC1	61	075	3D	=
18	022	12	DC2	62	076	3E	>
19	023	13	DC3	63	077	3F	?
20	024	14	DC4	64	100	40	@
21	025	15	NAK	65	101	41	A
22	026	16	SYN	66	102	42	B
23	027	17	ETB	67	103	43	C
24	030	18	CAN	68	104	44	D
25	031	19	EM	69	105	45	E
26	032	1A	SUB	70	106	46	F
27	033	1B	ESC	71	107	47	G
28	034	1C	FS	72	110	48	H
29	035	1D	GS	73	111	49	I
30	036	1E	RS	74	112	4A	J
31	037	1F	US	75	113	4B	K
32	040	20	SPAC	76	114	4C	L
33	041	21	!	77	115	4D	M
34	042	22	"	78	116	4E	N
35	043	23	#	79	117	4F	O
36	044	24	\$	80	120	50	P
37	045	25	%	81	121	51	Q
38	046	26	&	82	122	52	R
39	047	27	'	83	123	53	S
40	050	28	(	84	124	54	T
41	051	29	)	85	125	55	U
42	052	2A	*	86	126	56	V
43	053	2B	+	87	127	57	W

Continued

Table 4.1: The ASCII character set—cont'd

Dec	Oct	Hex	Char	Dec	Oct	Hex	Char
88	130	58	X	108	154	6C	l
89	131	59	Y	109	155	6D	m
90	132	5A	Z	110	156	6E	n
91	133	5B	[	111	157	6F	o
92	134	5C	\	112	160	70	p
93	135	5D	]	113	161	71	q
94	136	5E	^	114	162	72	r
95	137	5F	_	115	163	73	s
96	140	60	`	116	164	74	t
97	141	61	a	117	165	75	u
98	142	62	b	118	166	76	v
99	143	63	c	119	167	77	w
100	144	64	d	120	170	78	x
101	145	65	e	121	171	79	y
102	146	66	f	122	172	7A	z
103	147	67	g	123	173	7B	{
104	150	68	h	124	174	7C	
105	151	69	i	125	175	7D	}
106	152	6A	j	126	176	7E	~
107	153	6B	k	127	177	7F	DEL

represents *é* as the single decimal value 233 rather than the ASCII sequence “*e* followed by back-space followed by ‘.’” The latter is alien to the French way of thinking, for *é* is really a single character, generated by a single keystroke on French keyboards. For non-European languages like Hebrew and Chinese, ASCII is irrelevant. For them, other schemes have arisen: for example, GB and Big-5 are competing standards for Chinese; the former is used in the People’s Republic of China and the latter in Taiwan and Hong Kong.

As the Internet exploded into the World Wide Web and burst into all countries and all corners of our lives, the situation became untenable. The world needed a new way of representing text.

Unicode

In 1988, Apple and Xerox began work on Unicode, a successor to ASCII that aimed to represent all the characters used in all the world’s languages. As word spread, the Unicode Consortium, a group of international and multinational companies, government organizations, and other interested parties, was formed in 1991. The result was a new standard, ISO-10646, ratified by the International Organization for Standardization in 1993. In fact, the standard melded the Unicode Consortium’s specification with ISO’s own work in this area.

Unicode continues to evolve. The main goal of representing the scripts of languages in use around the world has been achieved. Current work is addressing historic languages, such as Egyptian hiero-

glyphics and Indo-European languages, and notations like music. A steady stream of additions, clarifications, and amendments eventually lead to new published versions of the standard. Of course, backward compatibility with the existing standard is taken for granted.

A standard is sterile unless it is adopted by vendors and users. Programming languages like Java have built-in Unicode support. Earlier ones—C, Perl, Python, to name a few—have standard Unicode libraries. Today’s operating systems all support Unicode, and application programs, including Web browsers, have passed on the benefits to the end user. Unicode is the default encoding for HTML and XML. People of the world, rejoice!

Unicode is universal: any document in an existing character set can be mapped into it. But it also satisfies a stronger requirement: the resulting Unicode file can be mapped back to the original character set without any loss of information. This requirement is called *round-trip compatibility* with existing coding schemes, and it is central to Unicode’s design. If a letter with an accent is represented as a single character in some existing character set, then an equivalent must also be placed in the Unicode set, even though there might be another way to achieve the same visual effect. Because the ISO 8859-1 character set mentioned above includes *é* as a single character, it must be represented as a single Unicode character—even though an identical glyph can be generated using a sequence along the lines of “*e* followed by backspace followed by ‘.’”

Round-trip compatibility is an attractive way to facilitate integration with existing software and was most likely motivated by the pressing need for a nascent standard to gain wide acceptance. You can safely convert any document to Unicode, knowing that it can always be converted back again if necessary to work with legacy software. This is indeed a useful property. However, multiple representations for the same character can cause complications. These issues are discussed further in Chapter 8 (Section 8.2), which gives a detailed account of the Unicode standard.

In the meantime, it is enough to know that the most popular method of representing Unicode is called UTF-8. UTF stands for “UCS Transformation Format,” which is a nested acronym: UCS is Unicode Character Set—so called because Unicode characters are “transformed” into this encoding format. UTF-8 is a variable-length encoding scheme where the basic entity is a byte. ASCII characters are automatically 1-byte UTF-8 codes; existing ASCII files are valid UTF-8.

Plain text

Unicode provides an all-encompassing form for representing characters, including manipulation, searching, storage, and transmission. Now we turn attention to document representation. The lowest common denominator for documents on computers has traditionally been plain, simple, raw ASCII text. Although there is no formal standard for this, certain conventions have grown up.

A text document comprises a sequence of character values interpreted in ordinary reading order: left to right, top to bottom. There is no header to denote the character set used. While 7-bit ASCII is the baseline, the 8-bit ISO ASCII extensions are often used, particularly for non-English text. This works well when text is processed by just one application program on a single computer, but when transferring between different applications—perhaps through e-mail, news, the Web, or file transfer—the

various programs involved may make different assumptions. An alphabet mismatch often means that characters in the range 128–255 are displayed incorrectly.

Plain text documents are formatted in simple ways. Explicit line breaks are usually included. Paragraphs are separated by two consecutive line breaks, or else the first line is indented. Tabs are frequently used for indentation and alignment. A fixed-width font is assumed; tab stops usually occur at every eighth character position. Common typing conventions are adopted to represent characters like dashes (two hyphens in a row). Headings are underlined manually using rows of hyphens or equal signs. Emphasis is often indicated by surrounding text with a single underscore (like this), or flanking words with asterisks (*like* *this*).

Different operating systems have adopted conflicting conventions for specifying line breaks. Historically, teletypes were modeled after typewriters. The line-feed character (ASCII 10, LF in [Table 4.1](#)) moves the paper up one line but retains the position of the print head. The carriage-return character (ASCII 13, CR in [Table 4.1](#)) returns the print head to the left margin but does not move the paper. A new line is constructed by issuing *carriage return* followed by *line feed* (logically the reverse order could be used, but the *carriage return line feed* sequence is conventional, and universally relied upon). Microsoft Windows uses this teletype-oriented interpretation. However, Unix and the Apple Macintosh adopt a different convention: the ASCII line-feed character moves to the next line *and* returns the print head to the left margin. This difference in interpretation can produce a strange-looking control character at the end of every line: ^M or “carriage return.” (Observe that CR and M are in the same row of [Table 4.1](#). Control characters in the first column are often made visible by prefixing the corresponding character in the second column with ^.) While the meaning of the message is not obscured, the effect is distracting.

People who use the standard Internet file transfer protocol (FTP) sometimes wonder why it has separate ASCII and binary modes. The difference is that in ASCII mode, new lines are correctly translated when copying files between different systems. It would be wrong to apply this transformation to binary files, however. Modern text-handling programs conceal the difference from users by automatically detecting which new-line convention is being used and behaving accordingly. Of course, this can lead to brittleness: if assumptions break down, all line breaks are messed up and users are mystified.

Plain text is a simple, straightforward, but impoverished representation of documents in a digital library. Metadata cannot be included explicitly (except, possibly, as part of the file name). However, automatic processing is sometimes used to extract title, author, date, and so on. Extraction methods rely on informal document structuring conventions. The more consistent the structure, the easier extraction becomes. Conversely, the simpler the extraction technique, the more seriously things break down when formatting quirks are encountered. Unfortunately you cannot normally expect complete consistency and accuracy in large plain text document collections.

Indexing

Rapid searching is a core function of digital libraries that distinguishes them from physical libraries. The ability to search full text adds great value when large document collections are used for study or

A full-text index to a document collection gives, for each word, the position of every occurrence of that word in the collection's text. A moment's reflection shows that the size of the index is commensurate with the size of the text, because an occurrence position is likely to occupy roughly the same number of bytes as a word in the text. (Four-byte integers, which are convenient in practice, are able to specify word positions in a 4-billion-word corpus. Conversely, an average English word has five or six characters and so also occupies a few bytes, the exact number depending on how it is stored and whether it is compressed.) We have implicitly assumed a word-level index, where occurrence positions give actual word locations in the collection. Space will be saved if locations are recorded within a unit, such as a paragraph, chapter, or document, yielding a coarser index—partly because pointers can be smaller, but chiefly because when a particular word occurs several times in the same unit, only one pointer is required for that unit.

A comprehensive index, capable of rapidly accessing all documents that satisfy a particular query, is a large data structure. Size, as well as being a drawback in its own right, also affects retrieval time, for the computer must read and interpret appropriate parts of the index to locate the desired information. Fortunately, there are interesting data structures and algorithms that can be applied to solve these problems. They are beyond the scope of this book, but references can be found in the “Notes and sources” section at the end of the chapter.

The basic function of a full-text index is to provide, for any particular query term, a list of all the units that contain it, along with (for reasons to be explained shortly) the number of times it occurs in each unit on the list. It's simplest to think of the “units” as being documents, although the granularity of the index may use paragraphs or chapters as the units instead—or even individual words, in which case what is returned is a list of the word numbers corresponding to the query term. And it's simplest to think of the query term as a word, although if stemming or case-folding is in effect, the term may correspond to several different words. For example, with stemming, the term *computer* may correspond to the words *computer*, *computers*, *computation*, *compute*, and so on; and with case-folding it may correspond to *computer*, *Computer*, *COMPUTER*, and even *CoMpUtEr* (an unusual word, but not completely unknown—for example, it appears in this book!).

When one indexes a large text, it rapidly becomes clear that just a few common words—such as *of*, *the*, and *and*—account for a large number of the entries in the index. People have argued that these words should be omitted, since they take up so much space and are not likely to be needed in queries, and for this reason they are often called *stop words*. However, some index compilers and users have observed that it is better to leave stop words in. Although a few dozen stop words may account for around 30 percent of the references that an index contains, it is possible to represent them in a way that consumes relatively little space.

A query to a full-text retrieval system usually contains several words. How they are interpreted depends on the type of query. Two common types, both explained in Section 3.4, are *Boolean queries* and *ranked queries*. In either case, the process of responding to the query involves looking up, for each term, the list of documents it appears in, and performing logical operations on these lists. In the case of Boolean queries, the result is a list of documents that satisfy the query, and this list (or the first part of it) is displayed to the user.

In the case of ranked queries, the final list of documents is sorted according to the ranking heuristic that is in place. As Section 3.4 explains, these heuristics gauge the similarity of each document to the set of terms that constitute the query. For each term, they weigh the frequency with which it appears in the document being considered (the more it is mentioned, the greater the similarity) against its frequency in the document collection as a whole (common terms are less significant). This is why the index stores the number of times each word appears in each document. A great many documents—perhaps all documents in the collection—may satisfy a particular ranked query (if the query contains the word *the*, all English documents would probably qualify). Retrieval systems take great pains to work efficiently even on such queries; for example, they use techniques that avoid the need to sort the list fully in order to find the top few elements.

In effect, the indexing process treats each document (or whatever the unit of granularity is) as a “bag of words.” What matters is the words that appear in the document and (for ranked queries) the frequency with which they appear. The query is also treated as a bag of words. This representation provides the foundation for full-text indexing. Whenever documents are presented in forms other than word-delineated plain text, they must be reduced to this form so that the corresponding bag of words can be determined.

Word segmentation

Before an index is created, the text must first be divided into words. A word is a sequence of alphanumeric characters surrounded by white space or punctuation. Usually some large limit is placed on the length of words—perhaps 16 characters, or 256 characters. Another practical rule of thumb is to limit numbers that appear in the text to a far smaller size—perhaps four numeric characters, so that only numbers less than 9,999 are indexed. Without this restriction, the size of the vocabulary might be artificially inflated—for example, a long document with numbered paragraphs could contain hundreds of thousands of different integers—which negatively affects certain technical aspects of the indexing procedure. Years, however, which have four digits, should be preserved as single words.

In some languages, plain text presents special problems. Languages like Chinese and Japanese are written without any spaces or other word delimiters (except for punctuation marks). This causes obvious problems in full-text indexing: to get a bag of words, we must be able to identify the words. One possibility is to treat each character as an individual word. However, this produces poor retrieval results. An extreme analogy is an English-language retrieval system that, instead of finding all documents containing the words *digital library*, found all documents containing the constituent letters *a b d g i l r t y*. Of course the searcher will receive all sought-after documents, but they are diluted with countless others. And ranking would be based on letter frequencies, not word frequencies. The situation in Chinese is not so bad, for individual characters are far more numerous, and more meaningful, than individual letters in English. But they are less meaningful than words. Chapter 8 discusses the problem of segmenting such languages into words.

4.2 Textual Images

Plain text documents in digital libraries are often produced by digitizing paper documents. Digitization is the process of taking traditional library materials, typically in the form of books and papers,

and converting them to electronic form, which can be stored and manipulated by a computer. Digitizing a large collection is a time-consuming and expensive process that should not be undertaken lightly.

Digitizing proceeds in two stages, illustrated in Figure 4.2. The first stage produces a digitized image of each page using a process known as *scanning*. The second stage produces a digital representation of the textual content of the pages using optical character recognition (OCR). In many digital library systems, what is presented to library readers is the result of the scanning stage: page images, electronically delivered. The OCR stage is necessary if a full-text index is to be built that will allow searchers to locate any combination of words, or if any automatic metadata extraction technique is contemplated, such as identifying document titles by seeking them in the text. Sometimes the second stage is omitted, but full-text search is then impossible, which negates a prime advantage of digital libraries.

If, as is usually the case, OCR is undertaken, the result can be used as an alternative way of displaying the page contents. The display will be more attractive if the OCR system not only is able to

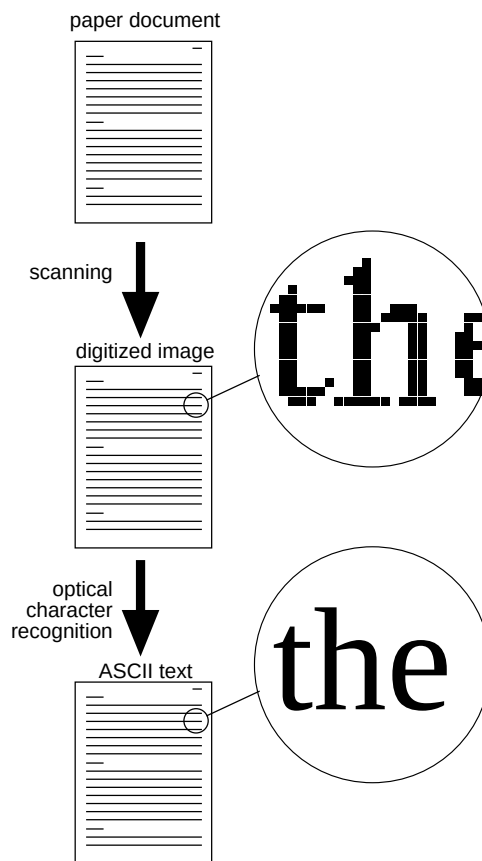


Figure 4.2: Scanning and optical character recognition

interpret the text in the page image, but also can retain the page layout as well. Whether it is a good idea to display OCR output depends on how well the page content and format are captured by the OCR process, among other things.

Scanning

The result of the first stage, scanning, is a digitized image of each page. The image resembles a digital photograph, although its picture elements or *pixels* may be either black or white—whereas photos have pixels that come in color, or at least in different shades of gray. Text is well represented in black and white, but if the image includes nontextual material, such as pictures, or exhibits artifacts like coffee stains or creases, grayscale or color images will resemble the original pages more closely. Image digitization is discussed more fully in the next chapter.

When scanning page images you need to decide whether to use black-and-white, grayscale, or color, and you also need to determine the resolution of the digitized images—that is, the number of pixels per linear unit. A familiar example of black-and-white image resolution is the ubiquitous laser printer, which generally prints 600–1200 dots per inch. Table 4.2 shows the resolution of several common imaging devices.

The number of bits used to represent each pixel also helps to determine image quality. Most printing devices are black and white: one bit is allocated to each pixel. When putting ink on paper, this representation is natural—a pixel is either inked or not. However, display technology is more flexible, and computer screens allow several bits per pixel. Color displays range up to 24 bits per pixel, encoded as 8 bits for each of the colors red, green, and blue, or even 32 bits per pixel, encoded in a way that separates the chromatic, or color, information from the achromatic, or brightness, information. Color scanners can be used to capture images having more than 1 bit per pixel.

More bits per pixel can compensate for a lack of linear resolution and vice versa. Research on human perception has shown that if a dot is small enough, its brightness and size are interchangeable—that is, a small bright dot cannot be distinguished from a larger, dimmer one. The critical size below which this phenomenon takes effect depends on the contrast between dots and their background, but corresponds roughly to a very low-resolution (640×480) display at normal viewing levels and distances.

Table 4.2: An assortment of devices and their resolutions

Device	Resolution (dpi)	Depth (bits)
Laptop computer screen (17 inch diagonal, 1680×1050 resolution)	116×116	24–32
Fax machine	200×200	1
Scanner	600×600	24
Laser printer	$600 \times 600 - 1200 \times 1200$	1
Phototypesetter	4800×4800	1

When digitizing documents for a digital library, think about what you want the user to be able to see. How closely does it need to resemble the original document pages? Are you concerned about preserving artifacts? What about pictures in the text? Will users see one page on the screen at a time? Will they want to magnify the images?

You will need to obtain scanned versions of several sample pages, chosen to cover the kinds and quality of images in the collection, and digitized to a range of different qualities (e.g., different resolutions, different gray levels, and color versus monochrome). You should conduct trials with end users of the digital library to determine what qualities are necessary for actual use.

It is always tempting to say that quality should be as high as it possibly can be. But there is a cost: the downside of accurate representation is increased storage space on the computer and—probably more importantly—increased time required for page access by users, particularly remote users. Doubling the linear resolution quadruples the number of pixels, and although this increase is ameliorated by compression techniques, users still pay a toll in access time. Your trials should take place on typical computer configurations using typical communications facilities, so that you can assess the effect of download time as well as image quality. You might also consider generating thumbnail images, or images at several different resolutions, or using a “progressive refinement” form of image transmission (see Section 5.3), so that users who need high-quality pictures can be sure they’ve got the right one before embarking on a lengthy download.

Optical character recognition

The second stage of digitizing library material is to transform the scanned image into a digitized representation of the page *content*—in other words, a character-by-character representation rather than a pixel-by-pixel one. This is known as *optical character recognition* (OCR). Although the OCR process itself can be entirely automatic, subsequent manual cleanup is invariably necessary and is usually the most expensive and time-consuming operation involved in creating a digital library from printed material. OCR might be characterized as taking “dumb” page images that are nothing more than images and producing “smart” electronic text that can be searched and processed in many different ways.

As a rule of thumb, a resolution of 300 dpi is needed to support OCR of regular fonts (10-point or greater), and 400 to 600 dpi for smaller fonts (9-point or less). Many OCR programs can tune the brightness of grayscale images appropriately for the text being recognized, so grayscale scanning tends to yield better results than black-and-white scanning. However, black-and-white images generate much smaller files than grayscale ones.

Not surprisingly, the quality of the output of an OCR program depends critically on the quality of the input. With clear, well-printed English, on clean pages, in ordinary fonts, digitized to an adequate resolution, laid out on the page in the normal way, with no tables, images, or other nontextual material, a leading OCR engine is likely to be 99.9 percent accurate or above—say 1 to 4 errors per 2,000 characters, which is a little under a page of this book. Accuracy continues to increase, albeit slowly, as technology improves. Replicating the exact format of the original image is more difficult, although for simple pages an excellent approximation will be achieved.

Unfortunately, the OCR operation is rarely presented with favorable conditions. Problems occur with proper names, with foreign names and words, and with special terminology—like Latin names for biological species. Problems are incurred with strange fonts, and particularly with alphabets that have accents or diacritical marks, or non-Roman characters. Problems are generated by all kinds of mathematics, by small type or smudgy print, and by overly dark characters that have smeared or bled together or overly light ones whose characters have broken up. OCR has problems with tightly packed or loosely set text where, to justify the margins, character and word spacing diverge widely from the norm. Hand annotation interferes with print, as does water-staining, or extraneous marks like coffee stains or squashed insects. Multiple columns, particularly when set close together, are difficult. Other problems are caused by any kind of pictures or images—particularly ones that contain some text; by tables, footnotes, and other floating material; by unusual page layouts; and by text in the image that is skewed, or lines of text that are bowed from the attempt to place book pages flat on the scanner platen, or by the book's binding if it interferes with the scanned text. These problems may sound arcane, but almost all OCR projects encounter them.

The highest and most expensive level of accuracy attainable from commercial service bureaus is typically 99.995 percent, or 1 error in 20,000 characters of text (approximately six pages of this book). Such a level is often most easily achieved by having the text retyped manually rather than by having it processed automatically by OCR. Each page is processed twice, by different operators, and the results are compared automatically. Any discrepancies are resolved manually.

As a rule of thumb, OCR becomes less efficient than manual keying when its accuracy rate drops below 95 percent. Moreover, once the initial OCR pass is complete, costs tend to double with each halving of the accuracy rate. However, in a large digitization project, errors are usually non-uniformly distributed over pages: often 80 percent of errors come from 20 percent of the page images. It may be worthwhile to have the worst of the pages manually keyed and to perform OCR on the remainder.

Human intervention is often valuable for cleaning up both the image before OCR and, afterward, the text produced by OCR. The actual recognition part can be time-consuming—maybe one or two minutes per page—and it is useful to perform interactive preprocessing for a batch of pages, have them recognized offline, and return them to the batch for interactive cleanup. Careful attention to such practical details can make a great deal of difference in a large-scale project.

Interactive OCR involves six steps: image acquisition, cleanup, page analysis, recognition, checking, and saving.

Acquisition, cleanup, and page analysis

Images are acquired either by inputting them from a document scanner or by reading a file that contains predigitized images. In the former case, the document is placed on the scanner platen and the program produces a digitized image. Most digitization software can communicate with a wide variety of image acquisition devices. An OCR program may be able to scan a batch of several pages and let you work interactively on the other steps afterward. This is particularly useful if you have an automatic document feeder.

The cleanup stage applies image-processing operations to the image. For example, a despeckle filter cleans up isolated pixels or “pepper and salt” noise. It may be necessary to rotate the image by 90 or 180 degrees, or to automatically calculate a skew angle and deskew the image by rotating it back by that angle. Images may be converted from white-on-black to the standard black-on-white representation, and double-page spreads may be converted to single-image pages. These operations may be invoked manually or automatically. If you don’t want to recognize certain parts of the image, or if it contains large artifacts—such as photocopied parts of the document’s binding—you may need to remove them manually by selecting the unwanted area and clearing it.

The page analysis stage examines the layout of the page and determines which parts to process and in what order. Again, page analysis can be either manual or automatic. The result divides the page into blocks of different types, typically text blocks, which will be interpreted as ordinary running text, table blocks, which will be further processed to analyze the layout before reading each table cell, and picture blocks, which will be ignored in the character recognition stage. During page analysis, multicolumn text layouts are detected and are sorted into correct reading order.

Figure 4.3a shows an example of a scanned document with regions that contain different types of data: text, two graphics, and a photographic image. In Figure 4.3b, bounding boxes have been drawn (manually in this case) around these regions. This particular layout is interesting because it contains a region—the large text block halfway down the left-hand column—that is clearly nonrectangular, and another region—the halftone photograph—that is tilted. Because layouts like this present significant challenges to automatic page analysis algorithms, many interactive OCR systems show users the result of automatic page analysis and offer the option of manually overriding it.

It is also useful to be able to manually set up a template that applies to a whole batch of pages. For example, you might define header and footer regions, and specify that each page contains a double column of text—perhaps even give the bounding boxes of the columns. Perhaps the page analysis process can be circumvented by specifying in advance that all pages contain single-column running text, without headers, footers, pictures, or tables. Finally, although word spacing is usually ignored, in some cases spaces may be significant—as in formatted computer programs.

Tables are particularly difficult for page analysis. For each table, the user may be able to specify interactively such things as whether the table has one line per entry or contains multiline cells, and whether the number of columns is the same throughout or some rows contain merged cells. As a last resort, it may be necessary for the user to specify every row and column manually.

Recognition

The recognition stage reads the characters on the page. This is the actual “OCR” part. One parameter that may need to be specified is the typeface (e.g., regular typeset text, fixed-width typewriter print, or dot-matrix characters). Another is the alphabet or character set, which is determined by the language used. Most OCR packages deal with only the Roman alphabet, although some accept Cyrillic, Greek, and Czech as well. Recognition of Arabic text, the various Indian scripts, or ideographic languages like Chinese and Korean calls for specialist software.

Furthermore, character-set variations occur even within the Roman alphabet. While English speakers are accustomed to the 26-letter alphabet, many languages do not employ all the letters—Māori, for



FEATURE ARTICLES GROS TITRES

Visualizing and Understanding Diagnoses

Suhayya Abu-Hakima

Knowledge Systems Laboratory
Institute for Information Technology
National Research Council, Ottawa, Canada
email: suhayya@ai.iit.nrc.ca tel: (613) 991-1231



Sommaire

Le diagnostic de systèmes physiques tels que les automobiles et les moteurs d'avion est une activité complexe. Les techniciens allient les manuels textuels (qui incluent des schémas) avec l'analyse des données mesurées pour diagnostiquer et réparer les moteurs. Les concepteurs de système à base de connaissance ont ajoutés l'heuristique, pour lier le texte avec les graphiques (hypermedia) pour simplifier la tâche des techniciens, tel qu'implanté dans le projet JETA (Halasz92). Des outils pour examiner la connaissance sont utilisés par les concepteurs de tels systèmes pour structurer et injecter la base de connaissance et sont aussi utilisés pour aider l'expert à visualiser la connaissance et les différentes relations possibles. De tels outils ont été conçus pour afficher et éditer la base de connaissance du projet JETA. Dans RATIONALE, un système de diagnostics qui raisonne en fournissant des explications, les explications sont utilisées pour comprendre le raisonnement du système [Abu-Hakima90]. Cet article argumente que même si ces approches à base de connaissances aident à la visualisation et la compréhension des systèmes physiques, ils ont besoin d'être améliorés et mieux intégrés.

Summary

Diagnosis of physical systems such as car or aircraft engines is a complex activity. Technicians combine textual manuals with schematics and some analysis of measured data to diagnose and repair engines. Knowledge-based system designers have added heuristics to link text and graphic (hypermedia) representations of manuals to simplify the tasks of the technicians as implemented in JETA [Halasz92]. Knowledge browsers are used by the developers of such systems to structure and input the knowledge base and are

also used in a limited capacity to help the domain experts visualize the knowledge and the various possible relations. Such a browser has been implemented to view and edit JETA's knowledge. In RATIONALE, a diagnostic system that reasons by explaining, explanation is used to understand system reasoning [Abu-Hakima90]. This paper argues that although these knowledge-based approaches help in the visualization and understanding of diagnoses in physical systems, they need to be improved and better integrated.

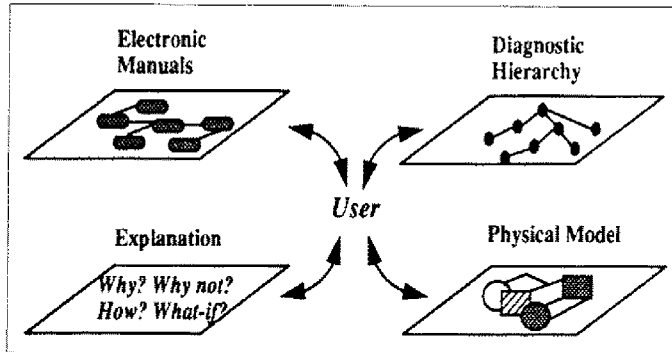
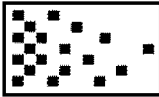


Figure 1: World of user in knowledge-based system

Introduction

Diagnosis of physical systems such as car or aircraft engines is a complex activity. Technicians combine paper manuals with schematics and some analysis of measured data to diagnose and repair engines. Knowledge-based systems provide the technicians with electronic manuals organized using hypermedia techniques as well as diagnostic hierarchies that represent the failure, test and repair actions of the diagnostic cycle. Such an approach has been followed for JETA, the Jet Engine Troubleshooting Assistant [Halasz92]. Other systems have followed modelling and simulation techniques that represent the actual physical system and attempt to diagnose it on the basis of the expected behaviour of the model [MBR91]. Some diagnostic systems



FEATURE ARTICLES GROS TITRES

Visualizing and Understanding Diagnoses

Suhayya Abu-Hakima

*Knowledge Systems Laboratory
Institute for Information Technology
National Research Council, Ottawa, Canada
email: suhayya@ai.iit.nrc.ca tel: (613) 991-1231*



Sommaire

Le diagnostic de systèmes physiques tels que les automobiles et les moteurs d'avion est une activité complexe. Les techniciens allient les manuels textuels (qui incluent des schémas) avec l'analyse des données mesurées pour diagnostiquer et réparer les moteurs. Les concepteurs de système à base de connaissance ont ajoutés l'heuristique, pour lier le texte avec les graphiques (hypermedia) pour simplifier la tâche des techniciens, tel qu'implanté dans le projet JETA (Halasz92). Des outils pour examiner la connaissance sont utilisés par les concepteurs de tels systèmes pour structurer et injecter la base de connaissance et sont aussi utilisés pour aider l'expert à visualiser la connaissance et les différentes relations possibles. De tels outils ont été conçus pour afficher et éditer la base de connaissance du projet JETA. Dans RATIONALE, un système de diagnostics qui raisonne en fournissant des explications, les explications sont utilisées pour comprendre le raisonnement du système [Abu-Hakima90]. Cet article argumente que même si ces approches à base de connaissances aident à la visualisation et la compréhension des systèmes physiques, ils ont besoin d'être améliorés et mieux intégrés.

Summary

Diagnosis of physical systems such as car or aircraft engines is a complex activity. Technicians combine textual manuals with schematics and some analysis of measured data to diagnose and repair engines. Knowledge-based system designers have added heuristics to link text and graphic (hypermedia) representations of manuals to simplify the tasks of the technicians as implemented in JETA [Halasz92]. Knowledge browsers are used by the developers of such systems to structure and input the knowledge base and are

also used in a limited capacity to help the domain experts visualize the knowledge and the various possible relations. Such a browser has been implemented to view and edit JETA's knowledge. In RATIONALE, a diagnostic system that reasons by explaining, explanation is used to understand system reasoning [Abu-Hakima90]. This paper argues that although these knowledge-based approaches help in the visualization and understanding of diagnoses in physical systems, they need to be improved and better integrated.

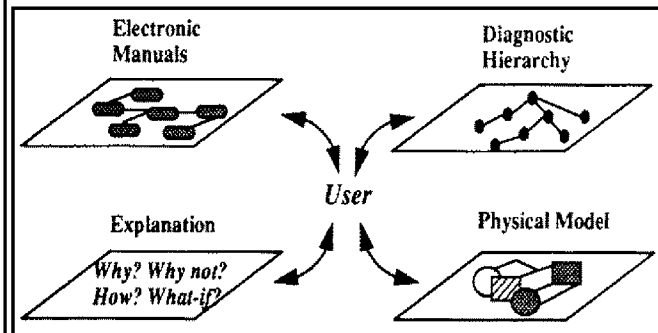


Figure 1: World of user in knowledge-based system

Introduction

Diagnosis of physical systems such as car or aircraft engines is a complex activity. Technicians combine paper manuals with schematics and some analysis of measured data to diagnose and repair engines. Knowledge-based systems provide the technicians with electronic manuals organized using hypermedia techniques as well as diagnostic hierarchies that represent the failure, test and repair actions of the diagnostic cycle. Such an approach has been followed for JETA, the Jet Engine Troubleshooting Assistant [Halasz92]. Other systems have followed modelling and simulation techniques that represent the actual physical system and attempt to diagnose it on the basis of the expected behaviour of the model [MBR91]. Some diagnostic systems

example, uses only 15. Documents in German include an additional character, ß or *scharfes s*, which is unique because, unlike all other German letters, it exists only in lowercase. (A recent change in the official definition of the German language has replaced some, but not all, occurrences of ß by ss.) European languages use accents: the German umlaut (ü); the French acute (é), grave (à), circumflex (ô), and cedilla (ç); the Spanish tilde (ñ). Documents may, of course, be multilingual.

For certain document types it may help to create a new “language” to restrict the characters that can be recognized. For example, a particular set of documents may be all in uppercase, or consist of nothing but numbers and associated punctuation.

In some OCR systems, the recognition engine can be trained to attune it to the peculiarities of the documents being read. Training may be helpful if the text includes decorative fonts or special characters like mathematical symbols. It may also be useful for recognition of large batches of text (100 pages or more) in which the print quality is low.

For example, the letters in some particular character sequences may have bled or smudged together on the page so that they cannot be separated by the OCR system’s segmentation mechanism. In typographical parlance they form a *ligature*: a combination of two or three characters set as a single glyph—such as *fi*, *fl*, and *ffl* in the font in which this book is printed. Although OCR systems recognize standard ligatures as a matter of course, printing occasionally contains unusual ligatures, as when particular sequences of two or three characters are systematically joined together. In these cases it may be helpful to train the system to recognize each combination as a single unit.

Training is accomplished by making the system process a page or two of text in a special training mode. When unrecognized characters are encountered, the user can enter them as new patterns. It may first be necessary to adjust the bounding box to include the whole pattern and exclude extraneous fragments of other characters. Recognition accuracy will improve if several examples of each pattern are supplied. When naming the pattern, its font properties (italic, bold, small capitals, subscript, superscript) may need to be specified along with the actual characters that comprise the pattern.

There is a limit to the amount of extra accuracy that training can achieve. OCR still does not perform well with more stylized typefaces, such as Gothic, that are significantly different from modern ones—and training may not help much.

Obviously, better results can be obtained if a language dictionary is used. It is far easier to distinguish letters like *o*, *0*, *O*, and *Q* in the context of the words in which they occur. Most OCR systems include predefined dictionaries and are able to use domain-specific ones containing technical terms, common names, abbreviations, product codes, and the like. Particular words may be constrained to particular styles of capitalization. Regular words may appear with or without an initial capital letter and may also be written in all capitals. Proper names must begin with a capital letter (and may be written in all capitals too). Some acronyms are always capitalized, while others may be capitalized in fixed but arbitrary ways.

Just as the language determines the basic alphabet, it may also preclude many letter combinations. Such information can greatly constrain the recognition process, and some OCR systems let users provide it.

Checking and saving

The next stage of OCR is manual checking. The output is displayed on the screen, with problems highlighted in color. One color may be reserved for unrecognized and uncertainly recognized characters, another for words that do not appear in the dictionary. Different display options can suppress some of this information. The original image is displayed too, perhaps with an auxiliary magnification window that zooms in on the region in question. An interactive dialog, similar to the spell-check mode of word processors, focuses on each error and allows the user to ignore this instance, ignore all instances, correct the word, or add it to the dictionary. Other options allow users to ignore words with digits and other nonalphabetic characters, ignore capitalization mismatches, normalize spacing around punctuation marks, and so on.

You may also want to edit the format of the recognized document, including font type, font size, character properties like italics and bold, margins, indentation, table operations, and so on. Ideally, general word-processor options will be offered within the OCR package, to save having to alternate between the OCR program and a word processor.

The final stage is to save the OCR result, usually to a file (alternatives include copying it to the clipboard or sending it by e-mail). Supported formats might include plain text, HTML, RTF, Microsoft Word, and PDF. There are many possible options. You may want to remove all formatting information before saving, or include the “uncertain character” highlighting in the saved document, or include pictures in the document. Other options control page size, font inclusion, and picture resolution. In addition, it may be necessary to save the original page image as well as the OCR text. In PDF format (described in [Section 4.5](#)), you can save the text and pictures only, or save the text under (or over) the page image, where the entire image is saved as a picture and the recognized text is superimposed upon it, or hidden underneath it. This hybrid format has the advantage of faithfully replicating the look of the original document—which can have useful legal implications. It also reduces the requirement for super-accurate OCR. Alternatively, you might want to save the output in a way that is basically textual, but with the image form substituted for the text of uncertainly recognized words.

Page handling

Let us return to the process of scanning the page images and consider some practical issues. Physically handling the pages is easiest if you can “disbind” the documents by cutting off their bindings (obviously, this destroys the source material and is only possible when spare copies exist). At the other extreme, originals can be unique and fragile, and specialist handling is essential to prevent their destruction. For example, most books produced between 1850 and 1950 were printed on acid paper (paper made from wood pulp generated by an acidic process), and their life span is measured in decades—far shorter than earlier or later books. Toward the end of their lifetime they decay and begin to fall apart (see Chapter 9).

Sometimes the source material has already been collected on microfiche or microfilm, and the expense of manual paper handling can be avoided. Although microfilm cameras are capable of very high resolution, quality is compromised because an additional generation of reproduction is inter-

posed; furthermore, the original microfilming may not have been done carefully enough to permit digitized images of sufficiently high quality for OCR. Even if the source material is not already in this form, microfilming may be the most effective and least damaging means of preparing content for digitization. It capitalizes on substantial institutional and vendor expertise, and as a side benefit the microfilm masters provide a stable long-term preservation format.

Generally the two most expensive parts of the whole process are handling the source material on paper, and the manual interactive processes of OCR. A balance must be struck. Perhaps it is worth the extra generation of reproduction that microfilm involves to reduce paper handling, at the expense of more labor-intensive OCR; perhaps not.

Microfiche is more difficult to work with than microfilm, because it is harder to reposition automatically from one page to the next. Moreover, it is often produced from an initial microfilm, in which case one generation of reproduction can be eliminated by digitizing directly from the film.

Image digitization may involve other manual processes apart from paper handling. Best results may be obtained by manually adjusting settings like contrast and lighting individually for each page or group of pages. The images may have to be manually deskewed. In some cases, pictures and illustrations will need to be copied from the digitized images and pasted into other files.

Planning an image digitization project

Any significant image digitization project will normally be outsourced. The cost varies greatly with the size of the job and the desired quality. For a small simple job, with material in a form that can easily be handled (e.g., books whose bindings can be removed), text that is clear and problem-free, and with few images and tables that need to be handled manually, you could expect to pay

- 50¢/page for fully automated OCR processing
- \$1.25/page for 99.9 percent quality (3 incorrect characters per page)
- \$2/page for 99.95 percent quality (1.5 incorrect characters per page)
- \$4/page for 99.995 percent quality (1 incorrect character every 6 pages)

for scanning and OCR, with a discount for large quantities. If difficulties arise, costs increase to many dollars per page. Using a third-party service bureau eliminates the need for you to become an expert in state-of-the-art image digitization and OCR. However, you will need to set standards for the project and to ensure that they are adhered to.

Most of the factors that affect digitization can be evaluated only by practical tests. You should arrange for samples to be scanned and OCR'd by competing companies and then compare the results. For practical reasons (because it is expensive or infeasible to ship valuable source materials around), the scanning and OCR stages may be contracted out separately. Once scanned, images can be transmitted electronically to potential OCR vendors for evaluation. You should probably obtain several different scanned samples—at different resolutions, different numbers of gray levels, from different sources, such as microfilm and paper—to give OCR vendors a range of different conditions. You should select sample images that span the range of challenges that your material presents.

Quality control is clearly a central concern in any image digitization project. An obvious quality-control approach is to load the images into your system as soon as they arrive from the vendor and to check them for acceptable clarity and skew. Images that are rejected are returned to the vendor for rescanning. However, this strategy is time-consuming and may not provide sufficiently timely feedback to allow the vendor to correct systematic problems. It may be more effective to decouple yourself from the vendor by batching the work. Quality can then be controlled on a batch-by-batch basis, where you review a statistically determined sample of the images and accept or reject whole batches.

Inside an OCR shop

Because it's labor-intensive, OCR work is often outsourced to developing countries like India, the Philippines, and Romania. Ten years ago, one of the authors visited an OCR shop in a small two-room unit on the ground floor of a high-rise building in a country town in Romania. It contained about a dozen terminals, and every day from 7:00 am through 10:30 pm the terminals were occupied by operators who were clearly working with intense concentration. There were two shifts a day, with about a dozen people in each shift and two supervisors—25 employees in all.

Most of the workers were university students who were delighted to have this kind of employment—it compared well with the alternatives available in their town. Pay was by results, not by the hour—and this was quite evident as soon as you walked into the shop and saw how hard people worked. They regarded their shift at the terminal as an opportunity to earn money, and they made the most of it.

This firm uses two different commercial OCR programs. One is better for processing good copy, has a nicer user interface, and makes it easy to create and modify custom dictionaries. The other is preferred for tables and forms; it has a larger character set with many unusual alphabets (e.g., Cyrillic). The firm does not necessarily use the latest version of these programs; sometimes earlier versions have special advantages.

The principal output formats are Microsoft Word and HTML. Again, the latest release of Word is not necessarily the one that is used. A standalone program is used for converting Word documents to HTML because it greatly outperforms Word's built-in facility. The people in the firm are expert at decompiling software and patching it. For example, they were able to fix some errors in the conversion program that affected how nonstandard character sets are handled. Most HTML is edited by hand, although they use a WYSIWYG (What You See Is What You Get) HTML editor for some of the work.

A large part of the work involves writing scripts or macros to perform tasks semiautomatically. Extensive use is made of Visual Basic for Applications (VBA). Although Photoshop is used for image work, they also employ a scriptable image processor for repetitive operations. MySQL, an open-source SQL implementation, is used for forms databases. Java is used for animation and for implementing Web-based questionnaires.

These people have a wealth of detailed knowledge about the operation of different versions of the software packages they use, and they constantly review and reassess the situation as new releases

emerge. But perhaps their chief asset is their set of in-house procedures for dividing up work, monitoring its progress, and checking the quality of the result. They claim an accuracy of around 99.99 percent for characters, or 99.95 percent for words—an error rate of 1 word in 2,000. This is achieved by processing every document twice, with different operators, and comparing the result. In 1999, their throughput was around 50,000 pages/month, although the firm's capability is flexible and can be expanded rapidly on demand. Basic charges for ordinary work are around \$1 per page (give or take a factor of two), but vary greatly depending on the difficulty of the job.

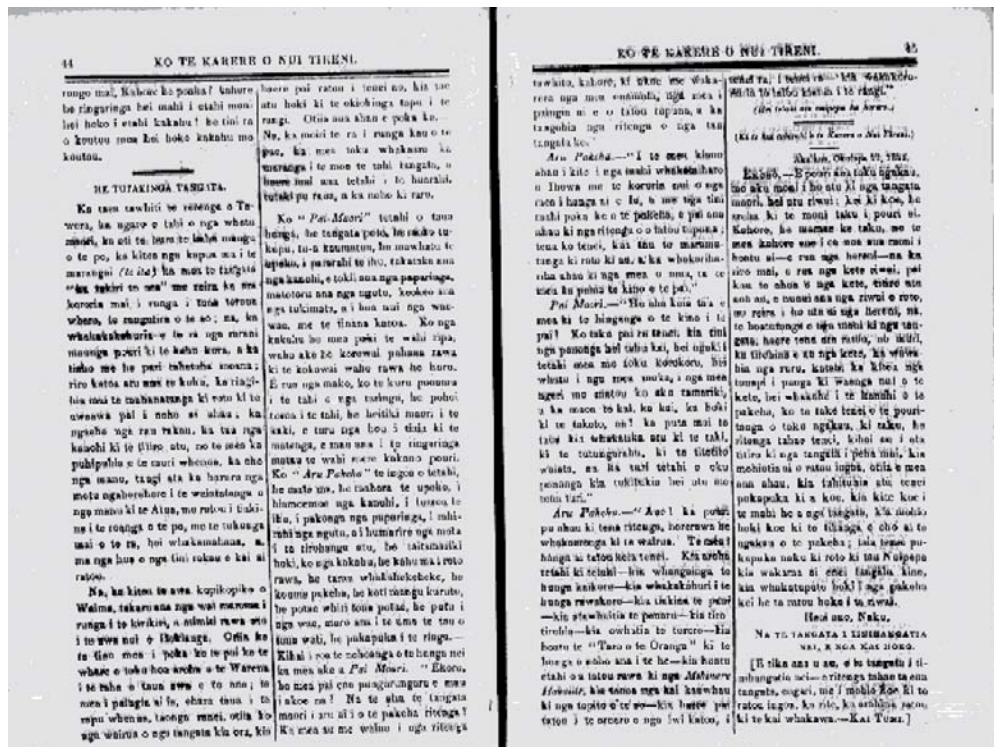
An example project

The New Zealand Digital Library undertook a project to put a collection of historical New Zealand Māori newspapers on the Web, in fully indexed and searchable form. There were about 20,000 original images, most of them double-page spreads. [Figure 4.4](#) shows a sample image, an enlarged version of the beginning, and some of the text captured using OCR. The particular image shown was difficult to work with because some areas are smudged by water-staining. Fortunately, not all the images were so poor. As you can see by attempting to decipher it yourself, high accuracy requires a good knowledge of the language in which the document is written.

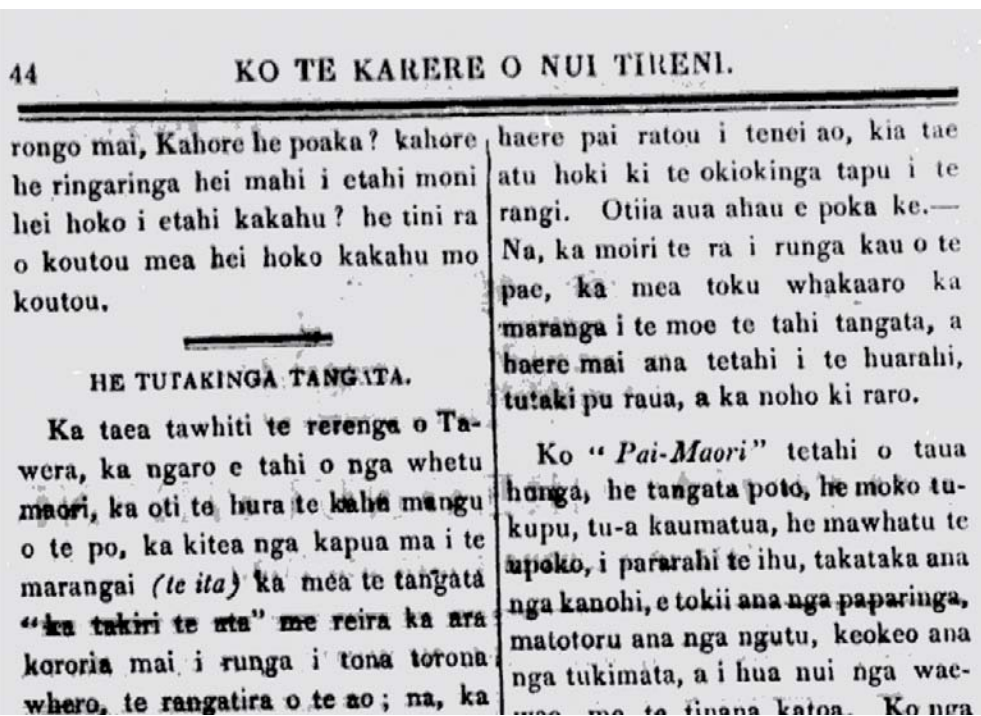
The first task was to scan the images into digital form. Gathering together paper copies of the newspapers would have been a massive undertaking, since the collection comprises 40 different newspaper titles that are held in a number of libraries and collections scattered throughout the country. Fortunately, New Zealand's national archive library had previously produced a microfiche containing all the newspapers for the purposes of historical research. The library provided us with access not just to the microfiche result, but also to the original 35-mm film master from which it had been produced. This simultaneously reduced the cost of scanning and eliminated one generation of reproduction. The photographic images were of excellent quality because they had been produced specifically to provide microfiche access to the newspapers.

Once the image source has been settled on, the quality of scanning depends on the scanning resolution and the number of gray levels or colors. These factors also determine how much storage is required for the information. After some testing, it was determined that a resolution of approximately 300 dpi on the original printed newspaper was adequate for the OCR process. Higher resolutions yielded no noticeable improvement in recognition accuracy. OCR results from a good black-and-white image were found to be as accurate as those from a grayscale one. Adapting the threshold to each image, or each batch of images, produced a black-and-white image of sufficient quality for the OCR work. However, grayscale images were often more satisfactory and pleasing for the human reader.

Following these tests, the entire collection was scanned by a commercial organization. Because the images were supplied on 35-mm film, the scanning could be automated and proceeded reasonably quickly. Both black-and-white and grayscale images were generated at the same time to save costs, although it was still not clear whether both forms would be used. The black-and-white images for the entire collection were returned on eight CD-ROMs; the grayscale images occupied 90 CD-ROMs.



(a)



(b)

Figure 4.4: (a) Double-page spread of a Māori newspaper; (b) enlarged version;

(c)	44 KO TE KARERE O NUI TIRENI. Rongo mai, Kahore he poaka? kahore he ringaringa hei mahi i etahi moni hei hoko i etahi kakahu? he tini ra o koutou mea hei hoko kakahu mo koutou. HE TUTAKINGA TANGATA Ka taea tawhiti te rerenga o Ta- wera, ka ngaro e tahi o nga whetu maori, ka oti te hura te kaha mangu o te po, ka kitea nga kapua ma te marangai (<i>te ita</i>) ka mea te tangata “ka takiri te ata” me reira ka ara kororia mai i runga i tona torona whero, te rangatira o te ao; na, ka	... haere pai ratou i tenei ao, kia tae atu hoki ki te okiokinga tapu i te rangi. Otiia aua ahau e poka ke.— Na, ka moni te ra i runga kau o te pae, ka mea toku whakaaro ka maranga i te moe te tahi tangata, a haere mai ana tetahi i te huarahi, tutaki pu taua, a ka noho ki raro. Ko “<i>Pai-Maori</i>” tetahi o taua hunga, he tangata poto, he moko tu- kupu, tu a kaumatua, he mawhatu te upoko, i pararahi te ihu, takataka ana nga kanohi, e tokii ana nga paparinga, matotoru ana nga ngutu, keokeo ana nga tukimata, a hua nui nga wae-

Figure 4.4, cont'd: (c) OCR text

Once the images had been scanned, the OCR process began. First attempts used Omnipage, a widely used proprietary OCR package. But a problem quickly arose: the Omnipage software is language-based and insists on utilizing one of its known languages to assist the recognition process. Because the source material was in the Māori language, additional errors were introduced when the text was automatically “corrected” to more closely resemble English. Although other language versions of the software were available, Māori was not among them, and it proved impossible to disable the language-dependent correction mechanism. In previous versions of Omnipage one can subvert the language-dependent correction by simply deleting the dictionary file (and the Romanian organization described above used an obsolete version for precisely this reason.) The result was that recognition accuracies of not much more than 95 percent were achieved at the character level. This meant a high incidence of word errors in a single newspaper page, and manual correction of the Māori text proved extremely time-consuming.

A number of alternative software packages and services were considered. For example, a U.S. firm offered an effective software package for around \$10,000 and demonstrated its use on some sample pages with impressive results. The same firm offers a bureau service and was prepared to undertake the basic OCR for only \$0.16 per page (plus a \$500 setup fee). Unfortunately, this did not include verification, which had been identified as the most critical and time-consuming part of the process—partly because of the Māori language material.

Eventually, we located an inexpensive software package that had high accuracy and allowed for the provision of a tailor-made language dictionary. It was decided that the OCR process would be done in house, and this proved to be an excellent decision; however, it is heavily conditioned on the unusual language in which the collection is written and the local availability of fluent Māori speakers.

A parallel task to OCR was to segment the double-page spreads into single pages for the purposes of display, in some cases correcting for skew and page-border artifacts. Software was produced for segmentation and skew detection, and a semiautomated procedure was used to display segmented and

deskewed pages for approval by a human operator. The result of these labors, turned in to a digital library, is described in Section 8.1.

4.3 Web Documents: HTML and XML

HTML, the Hypertext Markup Language, is the underlying document format of the World Wide Web, which makes it an important baseline for interactive viewing. Like all major, long-standing document formats, HTML has undergone growing pains, and its history reflects the anarchy that has characterized the Web's evolution. Since HTML's conception in 1989, its development has been largely driven by software vendors who compete for the Web browser market by inventing new features to make their product distinctive—the so called “browser wars.”

Many of the introduced features played on people's desires to exert more control over how their documents appear. Who gets to control font attributes like typeface and size—writer or reader? If you think this is a trivial issue, imagine what it means for the visually disabled. Allowing authors to dictate details of how their documents appear conflicts sharply with the original vision for HTML, which divorced document structure from presentation and left decisions about rendering documents to the browser itself. It makes the pages less predictable because viewing platforms differ in the support they provide. For example, HTML text can be marked up as “emphasized,” and while it is common practice to render such items in italics, there is no requirement to follow this convention: boldface would convey the same intention.

Out of the maelstrom, a series of HTML standards has emerged, although HTML's evolution continues—indeed, a major new revision, HTML 5, is under development although the warring parties (with vested interests) still clash: on this occasion a key controversy centers on whether or not native support for patent-free audio and video formats should be included in the standard.

The birth of HTML did not occur in a vacuum. Dipping further back in time, during the 1970s and 1980s, a generalized system for structural markup was developed called the Standard Generalized Markup Language (SGML); it was ratified as an ISO international standard in 1986. SGML is not a markup language but a metalanguage for describing markup formats. The original HTML format was expressed using SGML, and large organizations like government offices and the military also made significant use of it; however, SGML is rather intricate, and it has proven difficult to develop flexible software tools for the fully blown standard. This fact was the catalyst for the *extensible markup language*, XML.

XML is a simplified version of SGML designed specifically for interoperability over the Web. Informally speaking, it is a dialect of SGML (whereas HTML is an example of a markup language that SGML can describe). XML provides a flexible way of characterizing document structure and metadata, making it well suited to digital libraries. It has achieved widespread use in a very short stretch of time.

XML has strict syntactic rules that prevent it from describing ancient forms of HTML exactly. The differences expose parts of the early specifications that were loosely formed—ones that cause difficulty when parsing and processing documents. However, with a little trickery—for example,

judicious placement of white space—it is possible to generate an XML specification of an extremely close approximation to HTML. Put another way, you can take advantage of HTML’s sloppy specification to produce files that are valid XML. Such files have twin virtues: they can be viewed in any Web browser, and they can be parsed and processed by XML tools. The idea is formalized in HTML 5, which includes parallel specifications: one for “classic” HTML and another, XHTML, which is XML-compliant.

We start this section by describing the development of markup languages and their relation to stylesheet languages. Then we describe the basics of HTML and explain how it can be used in a digital library. Following that, we describe the XML metalanguage and again discuss its role in digital libraries. [Section 4.4](#) covers stylesheet languages for both HTML and XML.

Markup and stylesheet languages

Web culture has advanced at an extraordinary pace, creating a melee of incremental—and at times conflicting—additions and revisions to HTML, XML, and related standards. [Figure 4.5](#) summarizes the main developments.

Although it has been retrospectively fitted with XML descriptions, HTML was created before XML was conceived and drew on the more general expressive capabilities of SGML. It was also forged in the heat of the browser wars, in which Web browsers sprouted a proliferation of innovative nonstandard features that vendors thought would make their products more appealing. As a result, browsers became forgiving: they process files that flagrantly violate SGML syntax. One example is tag scope overlap—writing `<i>one <b>two </i>three </b>` to produce *one **two three***—despite SGML’s requirement that tags be strictly nested. During subsequent attempts at standardization, more tags were added that control typeface and layout, features deliberately excluded from HTML’s original design.

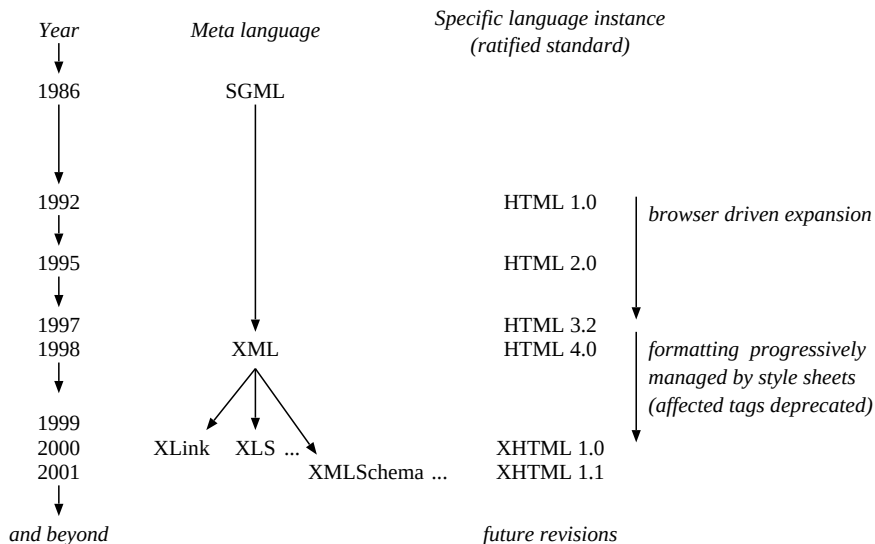


Figure 4.5: The relationship among XML, SGML, and HTML

The notion of *style sheets* was introduced to resolve the conflict between presentation and structure by moving formatting and layout specifications to a separate file. Style sheets purify the HTML markup to reflect, once again, nothing but document structure. Different documents can share a uniform appearance by adopting the same style sheet. Equally, different style sheets can be associated with the same document, for instance defining one for on-screen viewing and another for printing. Style sheets can specify a sequence—a cascade—of inherited stylistic properties and are dubbed *cascading style sheets*.

The first specification of cascading style sheets was in 1996, and this was quickly followed by an expanded backward-compatible version two years later. Style sheets can be adapted to different media by including formatting commands that are grouped together and associated with a given medium—screen, print, projector, handheld device, and so on. Guided by the user (or otherwise), applications that process the document use the relevant set of style commands. A Web browser might choose *screen* for online display but switch to *print* when rendering the document in PostScript.

Modern versions of HTML promote the use of style sheets. Moreover, they encourage them by officially deprecating formatting tags and other elements that affect presentation rather than structure. This is accomplished through three subcategories to the standard. *Strict* HTML expresses layout exclusively through style sheets: frameset commands and all deprecated tags and elements listed in the standard are excluded. In *transitional* HTML, style sheets are the principal way of specifying layout, but to provide compatibility with older browsers deprecated commands are permitted, although framesets are prohibited. *Frameset* HTML permits frameset commands and deprecated elements. Modern HTML files declare their subcategory at the start of the document. The format also adds improved support for multidirectional text (not just left to right) and enhancements for improved access by people with disabilities.

An HTML subset called XHTML has been defined that obeys the stricter syntactic rules imposed by the XML metalanguage. For instance, tags in XML are case sensitive, so XHTML tags and attributes are defined to be lowercase. Attributes within a tag must be enclosed in quotes. Each opening tag must be balanced by a corresponding closing one (there are also single tags that combine opening and closing, with their own special syntax).

The power and flexibility of XML are further increased by related standards. Three are given in [Figure 4.5](#) (there are many others). The *extensible stylesheet language* XSL described in [Section 4.4](#) represents a more sophisticated approach than cascading style sheets: it can also transform data. The *XML linking language* XLink provides a more powerful method for connecting resources than HTML hyperlinks: it has bidirectional links, can link more than two entities, and associates metadata with links. Finally, XML Schema provides a rich mechanism for combining components and controlling the overall structure, attributes, and data types used in a document.

From a technical standpoint, it is easier to work with XML and its siblings than HTML because they conform to a strictly defined syntax and are therefore easier to parse. In reality, however, digital libraries have to handle legacy material gracefully. Today's browsers cope remarkably well with the wide range of HTML files: they take backward compatibility to truly impressive levels. To help promote standardization, an open source software utility called HTML Tidy has been developed that

converts older formats. The process is largely automatic, but human intervention may be required if files deviate radically from recognized norms.

Basic HTML

Modern markup languages use words enclosed in angle brackets as tags to annotate text. For example, `<title>A really exciting story</title>` defines the title element of an HTML document. In HTML, tag names are case insensitive—`<Title>` is the same as `<title>`. For each tag, the language defines a “closing” version, which gives the tag name preceded by a slash character (/). However, closing tags can be omitted in certain situations—a practice that some decry as impure while others endorse as legitimate shorthand. For example, `<p>` is used to mark up paragraphs, and subsequent `<p>`s are assumed to automatically end the previous paragraph—no intervening `</p>` is necessary. The shortcut is possible because nesting a paragraph within a paragraph—the only other plausible interpretation on encountering the second `<p>`—is invalid in HTML.

Opening tags can include a list of qualifiers known as *attributes*. These have the form `name="value"`. For example, `` specifies an image with source file name `gsdl.gif` and dimensions 537×17 pixels.

Because the language uses characters such as `<`, `>`, and `"` as special markers, a way is needed to display these characters literally. In HTML these characters are represented as special forms called *entities* and given names like `<` for “less than” (`<`) and `>` for “greater than” (`>`). This convention makes ampersand (`&`) into a special character, which is displayed by `&` when it appears literally in documents. The semicolon needs no such treatment because its literal use and its use as a terminator are syntactically distinct. The same kind of special form can be used to specify Unicode characters beyond the ASCII range, such as ``` for `è`.

Figure 4.6 shows a sample page that illustrates several parts of HTML, along with a snapshot of how it is rendered by a Web browser. It contains “typical” code that you find on the Web, rather than exemplary HTML. Some attributes miss out double quotes (such as `align` and `valign` used in some of the table elements), and not all the elements stipulate all the attributes they should (e.g., two of the `<img>` tags are missing their `alt` attribute through which an alternative text description is given). The example would not even pass the test for transitional HTML, let alone strict HTML; however, as Figure 4.6b shows, the Web browser renders it just fine.

HTML documents are divided into a header and a body. The header gives global information: the title of the document, the character encoding scheme, any metadata. The `<meta>` tag is used in Figure 4.6a to acknowledge the New Zealand Digital Library Project as the document’s creator. *Creator* imitates the Dublin Core metadata element (see Section 6.2) that is used to represent the name of the entity responsible for generating a document, be it a person, organization, or software application; however, there is no requirement in HTML to conform to such standards. Following the header is a comment and a command that sets the background to a Polynesian motif.

This particular page is laid out as two tables. The first controls the main layout. The second, nested within it, lays out the poem and the image of a greenstone pendant. The tags `<tr>` and `<td>` are used

```

<html>
<head>
  <title>Greenstone Digital Library Software</title>
  <meta name="Creator" content="The New Zealand Digital Library project">
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<!-- Set background to Maori motif -->
<body bgcolor="#ffffff" text="#000000" background="heke.gif">
  <table align=center width="537">
    <tbody>
      <tr><td align=left>
        
        <table>
          <tbody>
            <tr valign=top>
              <td>
                <h2>Kia papapounamu te moana</h2>
                <p> kia hora te marino,<br>
                  kia tere te karohirohi,<br>
                  kia papapounamu te moana

                <p> may peace and calmness surround you,<br>
                  may you reside in the warmth of a summer's haze,<br>
                  may the ocean of your travels be as smooth as the
                  polished greenstone.

              </td>
              <td>
                
              </td>
            </tr>
          </tbody>
        </table>

        <p> Greenstone is a semi-precious stone that (like this software)
        is sourced in New Zealand. In traditional Māori society it was
        the most highly prized and sought after of all substances. It can
        absorb and hold wairua, which is a spirit or life force ...

        <ul>
          <li>Some special symbols: &lt; &amp; &quot; &auml; &#257; ;#
        </ul>

        <p><center></center>
        <p><a href="http://www.nzdl.org">New Zealand Digital Library Project</a>
      </td></tr>
    </tbody>
  </table>
</body>
</html>

```

(a)

Figure 4.6: (a) Sample HTML code involving graphics, text, and some special symbols;

to mark table rows and cells, respectively. The list item `<li>` near the end illustrates various special characters. Most take the `&...` form, but the last two (`;` and `#`) do not need to be escaped because their normal meaning is syntactically unambiguous. To generate the letter *a* with a line above (called a *macron* and used in the Māori language) the appropriate Unicode value is given in decimal (`#257`), demonstrating one way of specifying non-ASCII characters. The example illustrates several other

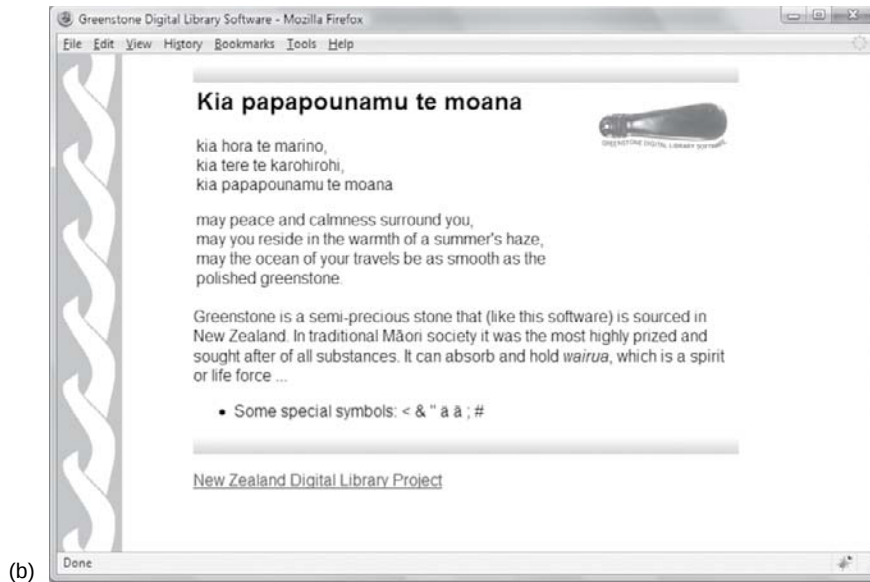


Figure 4.6, cont'd: (b) snapshot rendered by a Web browser

features, including images specified by the `<img>` tag, paragraphs beginning with `<p>`, italicized words given by `<i>`, and a bulleted list introduced by `<ul>` (for “unordered list”), along with a `<li>` tag for each list item (just one in this case).

Hyperlinks are an important feature of HTML. In the example, the tag pair `<a> ... </a>` near the end defines a link *anchor* element. The document to link to—in this case, another page on the Web—is specified as an attribute. Hyperlinks can reference PDF documents, audio and video material, and many other formats—such as the Virtual Reality Modeling Language, VRML, which specifies a navigable virtual reality experience. Browsers display the anchor text—the text appearing between the start and end hyperlink tag—differently to emphasize the presence of a link. When the hyperlink is clicked, the browser loads the new document.

HTML was originally encoded in ASCII for transmission over byte-oriented protocols. Other encoding schemes are supported by setting the *charset* attribute in the header element to the appropriate encoding name. In Figure 4.6a, line 5 sets it explicitly to UTF-8, which, as mentioned in Section 4.1, is a representation scheme for Unicode. In fact UTF-8 (which is backward-compatible with ASCII) is now the default, and the behavior would be the same if the attribute were omitted.

HTML has many more features. For example, locally defined link anchors permit navigation within a single document. Fonts, colors, and page backgrounds can be specified explicitly. Forms can be created that collect data from the user—such as text data, fielded data, and selections from lists of items.

A mechanism called *frames* allows an HTML document to be tiled into smaller, independent segments, each an HTML page in its own right. A set of frames, called a *frameset*, can be displayed

simultaneously. This is often used to add a navigation bar to every page of a Web site, along the top or down the side of the browser pane. When a link in the navigation bar is clicked, a new page is loaded into the main display frame, and the bar remains in place. Clicking on a link in the main display frame also loads the new page into the main frame.

Frames were introduced by one vendor during the browser wars and were soon supported by other browsers too. However, they have serious drawbacks. For instance, now that a browser can display more than one HTML document at a time, what happens when you create a bookmark? People often click around a site to reach an interesting document, then bookmark it in the usual way—only to find that the bookmark returns not to the intended page but to the point where the site splits into frames instead. This can be very frustrating.

Many of the effects for which frames were invented—such as persistent navigation bars—can also be accomplished by the newer and more principled mechanism of style sheets, avoiding the problems of frames; hence the three demarcated forms of HTML in more recent specifications of the standard: frameset, transitional, and strict, increasing in conformity to XML. We describe style sheets in [Section 4.4](#).

Using HTML in a digital library

As the *lingua franca* for the Web, HTML underpins virtually all digital library interfaces. Moreover, digital library source documents are often presented in HTML. This eliminates most of the difficulties associated with the plain text representation introduced earlier. For example, the HTML header disambiguates the character set, while the `<br>` and `<p>` tags disambiguate line and paragraph breaks.

To extract text from HTML documents for indexing purposes, the obvious strategy of parsing them according to a well-defined grammar quickly runs into difficulty. The permissive nature of Web browsers encourages authors to depart from the defined standard. A better way to identify and remove tags is to write them in the form of “regular expressions” (a scheme described in the next section), which generally achieves greater success for less effort, in this particular circumstance. An alternative is to use the very application that caused the complication in the first place: Web browsers. A plain text browser called *lynx* provides a fast and reliable method of extracting text from HTML documents—you give it a command-line argument (*dump*) and a URL, and it dumps out the contents of that URL in the form of plain text.

As the example in [Figure 4.6](#) illustrates, HTML allows metadata to be specified explicitly using `<meta>` tags. However, this mechanism is rather limited. For one thing, you might hesitate before tampering with source documents by inserting new metadata (perhaps determined separately, perhaps mined from the document content) in this way. When developing a digital library you need to consider whether it is wise (or even ethical, as discussed in [Section 1.5](#)) to add new information that cannot be disentangled from that present in the source document. Users might legitimately object if you serve up an altered version in place of the original.


```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Document Type Definition (DTD) for Non-Government Organizations (NGOs) -->
<!DOCTYPE NGODoc[
<!ELEMENT NGODoc (Head,Body)>
<!ELEMENT Head (Title)>
<!ELEMENT Body (#PCDATA|Agency)*>
<!ELEMENT Agency (Name,(Abbrev?|Photo*)+)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Name (#PCDATA)>
<!-- ATTLIST Name
      hq CDATA #IMPLIED>
<!ELEMENT Abbrev (#PCDATA)>
<!-- ATTLIST Photo
      src CDATA #REQUIRED
      desc CDATA "A photo">
]>
<!-- Sample content conforming to DTD -->
<NGODoc>
<Head>
  <Title>Agencies of the United Nations</Title>
</Head>
<Body>
  <Agency>
    <Name hq="Paris, France">United Nations Educational, Scientific
      and Cultural Organization</Name>
    <Abbrev>UNESCO</Abbrev>
    <Photo src="photos/unesco_hq.jpg"
      desc="Aerial photo of main UNESCO building"/>
  </Agency>
  <Agency>
    <Name hq="Rome, Italy">Food and Agricultural Organization</Name>
    <Abbrev>FAO</Abbrev>
    <Photo src="photos/fao_hq.jpg"/>
  </Agency>
  <Agency>
    <Name hq="Washington, USA">World Bank</Name>
    <Photo src="photos/worldbank_hq.jpg"/>
  </Agency>
  <!-- and so on ... -->
</Body>
</NGODoc>

```

Figure 4.7: Sample XML document

Basic XML

Figure 4.7 shows a formatted list of information about United Nations agencies encoded in XML. For each agency, the file records its full name, an optional abbreviation, and the URL of a picture of its headquarters. Included with the name is the address of the headquarters, stored as an attribute.

The file contains three broad sections, separated by comments in the form `<!-- . . . -->`. Line 1 is a header: it uses the special notation `<? . . . ?>` to denote an application-processing instruction. This syntax originates in SGML, which uses it to embed information for specific application programs that

process the document. Here it is used to declare the version of XML, the character encoding (UTF-8), and whether or not external files are used. Lines 5 to 19 dictate the syntactic structure in which the remainder of the file is expressed, in the form of a Document Type Definition (DTD). Lines 21 to 44 provide the content of the document.

The style of the content section is reminiscent of HTML. The tag specifications have the same syntactic conventions, and many tags are identical—examples are `<Head>`, `<Title>`, and `<Body>`. However, in lines 27 to 40 the markup creates structures that HTML cannot represent.

Because it is a metalanguage, XML gives document designers a great deal of freedom. In [Figure 4.7](#) the main document structure resembles HTML, but this does not have to be the case. Different element names could be chosen, and different ways could be used to express the information. For example, [Figure 4.7](#) gives the headquarters address as the *hq* attribute of the `<Name>` element. Alternatively, a new element could have been defined to contain this information. It could be constrained to appear immediately following the `<Name>` element, or left optional, or sited anywhere within the `<Agency>` element.

Structural decisions are recorded in the DTD (lines 5–19). DTD tags use the special syntax `<! . . . >` and express keywords in block capitals. For example, *ELEMENT* and *ATTLIST* are used to define elements and element attributes. Our document designer decided to capitalize the initial letter of all document elements and leave attributes in lowercase. This improves the legibility of [Figure 4.7](#) considerably.

Line 5 starts the DTD, and the square bracket syntax `[ . . . ]` indicates that the DTD will appear in-line. (It must, for line 1 declares that the file stands alone.) Alternatively, the DTD could be placed in an external file, referred to by a URL—which is the usual practice.

New elements are introduced in lines 6 to 11 by the keyword *ELEMENT*, followed by the new tag name and a description of what the element may contain. A *leaf* is an element that comprises plain text, with no markup. This is accomplished through *parsed character data* (`#PCDATA`), in which special characters may be included. For example, when the `<Title>` tag defined on line 10 is used, markup characters may appear in the title's text, encoded in the familiar HTML way—`<`, `&`; and so on. (This convention originated in SGML.)

Lines 6 to 9 describe nonleaf structures. These are defined in a form known as a *regular expression*. Here a comma signifies an ordered sequence: line 6 declares that the top-level element `<NGODoc>` contains a `<Head>` element followed by a `<Body>` element. A vertical bar (`|`) represents a choice of one element from a sequence of named elements, and an asterisk (`*`) indicates zero or more occurrences. Thus `<Body>` (line 8) is a mixture of parsed character data and `<Agency>` elements where it is permissible for nothing at all to appear. A plus sign (`+`) means one or more occurrences, and a question mark (`?`) signifies either nothing or just one occurrence. Line 9 includes all four symbols `|`, `*`, `+`, and `?`: it declares that `<Agency>` must include a name element, but that `<Abbrev>` is optional and there can be zero or more occurrences of `<Photo>` (the example is contrived: there are more concise ways of expressing the same thing). The inner pair of brackets bind these last two tags together, adding the extra stipulation that there must be at least one occurrence of the `<Abbrev>` and `<Photo>` specifications.

Attributes also give a set of possible values, but here there is no nesting. Lines 12 and 13 show an example. The attribute is signaled by the keyword *ATTLIST*, followed by the element to which it applies (*Name*), the attribute's name (*hq*), its type (*character data*), and any appearance restrictions (this one is optional). Lines 16 to 18 show another example, which introduces two attributes of the element *Photo*. Line 17 states that the *src* attribute is required, while line 18 provides a default value (namely "A photo") for the *desc* attribute.

In addition to *<* and *>*, XML incorporates definitions for *'* and *"*. These are called *entities*, and new ones can be added in the DTD using the syntax *ENTITY name "value"*. For instance, although XML does not have a definition for *à* as HTML does, one can be defined by *<!ENTITY agrave "à">*, which relies on the Unicode standard for the numeric value. Entities are not restricted to single characters, but can be used for any excerpt of text (even if it is marked up). For example, *<!ENTITY howto "How to Build a Digital Library">* is a shorthand way of encoding the title of this book.

If several elements shared exactly the same attributes, it would be tedious (and error-prone) to repeat the definitions in each element. This can be handled using a special type of entity known as a *parameter entity*. To illustrate it, Figure 4.8 shows a modified and slightly restructured version of the DTD for the document in Figure 4.7 that defines attributes *ident* and *style* under the name *sharedattrib* (lines 3–5), which is then used to bestow these attributes on the *<Title>*, *<Abbrev>*, and *<Name>* elements (lines 11–14). Parameter entities are signaled using the percent symbol (%) and provide a form of shorthand for use within a DTD.

Declaring the shared attribute *style* as *NMTOKEN* (line 4) restricts this attribute's characters to alphanumeric characters plus period (*.*), colon (*:*), hyphen (*-*), and underscore (*_*), where the first character must be a letter. Its twin *ident* is defined as *ID* (line 5), which is the same as *NMTOKEN* with the additional constraint that no two such attributes in the document can have the same value. *ID* therefore provides a mechanism for uniquely identifying elements, which in fact HTML enforces for an attribute with the particular name *id*. In XML, uniqueness can be bestowed on any attribute, whatever its name—such as *ident*.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE NGODoc[
<!ENTITY % sharedattrib
    " style NMTOKEN #IMPLIED
      ident ID #IMPLIED ">
<!ELEMENT NGODoc (Title,Body)>
<!ELEMENT Body (Name|Abbrev)+>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Abbrev (#PCDATA)>
<!ATTLIST Title %sharedattrib;>
<!ATTLIST Abbrev %sharedattrib;>
<!ATTLIST Name %sharedattrib;
            HQ CDATA #IMPLIED>
]>
```

Figure 4.8: Sample DTD using a parameterized entity

DTDs also support enumerated types, although none are present in the example. It can also include lists of tokens separated by white space (NMTOKENS) and attributes that are references to ID attributes (IDREF).

Parsing XML

A document that conforms to XML syntax but does not supply a DTD is said to be *well formed*. One that conforms to XML syntax and does supply a DTD is said to be *valid*—provided that the content does indeed abide by the syntactic constraints defined in the DTD. DTDs can be stored externally, replacing the bracketed section in [Figure 4.7](#) lines 5–19 by a URL. This allows documents with the same structure to be shared within or between organizations.

XML allows you to define new languages, and makes it easy to develop parsers for them. Generic parsers are available that are capable of parsing *any* XML file, and—if a DTD is present—also check that the file is valid. However, merely parsing a document is of limited utility. The result of a parser is just a yes/no indication of whether the document conforms to the general rules of XML (and to the more specific DTD). Far more useful would be a way of specifying what the parser should do with the data it is processing. This is arranged by having it build a parse tree and provide a programming interface—commonly called an API or *application program interface*—that lets the user traverse the tree and retrieve the data it contains.

The result of parsing any XML file is a root node whose descendants reflect both textual content and nested tags. At each tag's node are stored the values of the tag's attributes. There is a cross-platform and cross-language API called the *document object model* (DOM) that allows you to write programs that access and modify the document's content, structure, and style.

Using XML in a digital library

XML is a powerful tool. It allows file formats within an organization—or a digital library—to be rationalized and shared. Furthermore, organizations can provide an explanation of the structures used in the form of a published machine-readable document. By formulating appropriate DTDs, for instance, different organizations can develop comprehensive formats for sharing information.

A notable example is the Text Encoding Initiative (TEI), founded in 1987, which developed a set of DTDs for representing scholarly texts in the humanities and social sciences. SGML was the implementation backbone, but the work has since been reconciled with XML. These DTDs are widely used by universities, museums, and commercial organizations to represent museum and archival information, classical and medieval works, dictionaries and lexicographies, religious tracts, legal documents, and many other forms of writing.

Examples are legion. The Oxford Text Archive is a nonprofit group that has provided long-term storage and maintenance of electronic texts for scholars over the last quarter-century. Perseus is a pioneering digital library project, dating from 1985, which focuses upon the ancient Greek world. Der Junge Goethe in Seiner Zeit is a collection of early works—poems, essays, legal writings, and letters—by the great German writer Johann von Goethe (1749–1832). The Japanese Text Initiative is

a collaborative project that makes available a steadily increasing set of Japanese literature, accompanied by English, French, and German translations.

Various related standards increase XML's power and expand its applicability. Used on its own, XML provides a way of expressing a document's structural information, and/or metadata. Indeed, whether information is metadata or not is really a matter of perspective. Combined with additional standards, XML goes much further: it supports document restructuring, querying, information extraction, and formatting. The next section expands on the formatting standards, which equip XML with display capabilities comparable with HTML. More details appear in the appendix, entitled *More on markup and XML*, at the book's Web site: www.nzdl.org/howto.

4.4 Presenting Web Documents: CSS and XSL

Two kinds of style sheet can be used to control the presentation of marked-up documents. *Cascading style sheets* (CSS) produce presentable documents with minimal effort. They were developed principally in support of HTML, but also work with XML. A parallel development is the *extensible stylesheet language* (XSL) for XML (and for versions of HTML that are XML compliant). XSL performs the same services as CSS but expresses the style sheet in XML form. It also adds further power by allowing the document structure to be altered dynamically. For example, a particular element type can be constrained to appear at the top of the page, regardless of where it is actually defined.

CSS and XSL share a common heritage and are based on the same formatting model. They share the notion of a rectangular block of content, padded by specified amounts of white space and encased within four borders, which are themselves enclosed by margins. In CSS this is called a *box*, in XSL an *area*. They also share the same framework for tables.

To illustrate their similarities and differences, we give parallel examples of each. The next subsection describes CSS: an introduction to its basic structure is given through a small example in [Figure 4.10](#); for an example that illustrates tables and lists, and also the *cascading* nature of CSS, see [Figure 4.11](#); and for an example that shows context-sensitive formatting, see [Figure 4.12](#). Each of these three figures has two parts, showing the CSS code and a screenshot of the result. An additional example illustrates media-dependent formatting in CSS (see [Figure 4.13](#)). The subsection following this provides a parallel description of XSL; [Figures 4.14](#), [4.15](#), and [4.16](#) give XSL versions of the three examples (corresponding to [Figures 4.10a](#), [4.11a](#), and [4.12a](#) for CSS). A final subsection illustrates how XSL can be used for the kind of processing that CSS cannot do.

CSS

[Figure 4.9](#) shows what [Figure 4.7](#) looks like when rendered by an XML-capable Web browser. The display is rudimentary; it shows the raw XML, typically color-coded to highlight different features of the language. Depending on the browser used, it might also be possible to open and close elements interactively.

[Figure 4.10a](#) gives a style sheet for the same example, and [Figure 4.10b](#) shows the result. Improved formatting makes the three individual agency records easier to read; a different font and type size are

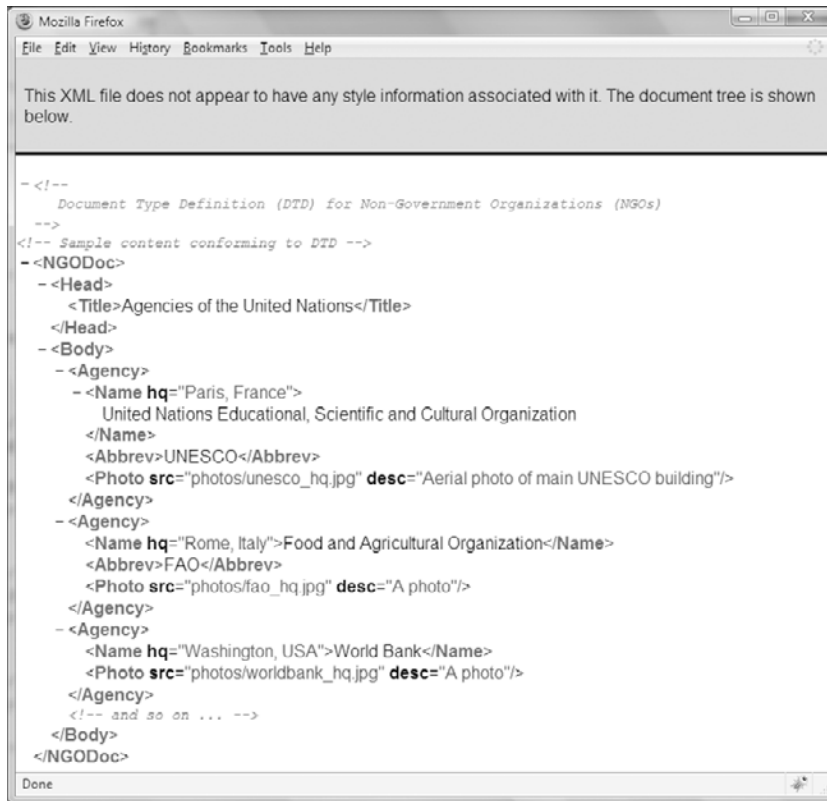


Figure 4.9: Sample XML document, viewed in a Web browser

used to distinguish the title; and the background is set to white. The style sheet is included by adding the line

```
<?xml-stylesheet href="un_basic.css" type="text/css"?>
```

just after the XML header declaration in Figure 4.7 (the text of Figure 4.10a resides in a file called *un_basic.css*).

Style sheets specify rules using *selector-declaration* pairs, such as

```
NGODoc { background: white }
```

The selector, here NGODoc, names an element type in the document. Several rules can apply to a given element—Figure 4.10a includes two for NGODoc. The declaration in braces gives formatting commands for the element—in this case setting the background to white. Declarations consist of *property: value* pairs, separated by semicolons if necessary.

There is an inheritance mechanism based upon the hierarchical document model that underpins XML. If the style sheet specifies formatting for an element, nested elements—ones beneath it in the document tree—inherit that specification. This makes style sheets concise and perspicuous. It is easy to override inherited behavior: just supply further rules at the appropriate level.

(a)

```

/* Cascading Style Sheet for UN Example */
NGODoc      { display: block }
NGODoc      { background: white; width: 7.5in }
Title       { font-family: times; /* Set font type to Times */
             font-size: 25pt;    /* Set font size to 25 point */
             font-style: bold } /* and so on ... */
Agency      { display: block;
             margin-top: 8pt; margin-bottom: 3pt;
             font-size: 16pt;
             font-family: helvetica }
Head, Body  { display: block;
             margin-top: 6pt;
             margin-left: 0.2in;
             margin-right: 5mm }
Abbrev      { display: inline; font-style: italic }
Abbrev:before { content: "(" }
Abbrev:after  { content: ")" }

```

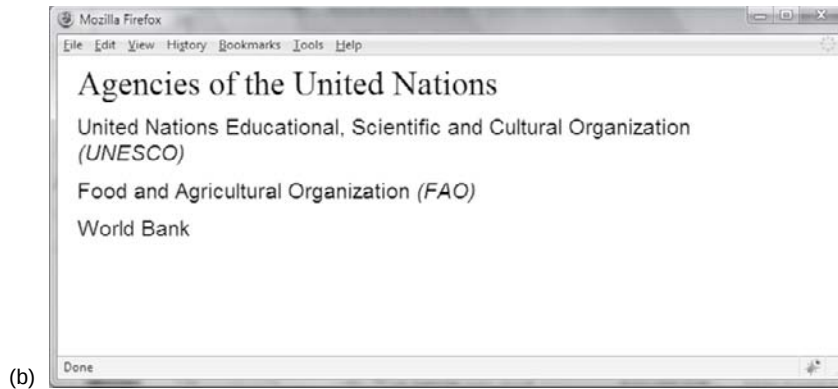


Figure 4.10: (a) Basic CSS style sheet for the United Nations Agencies example; (b) viewing the result in an XML-enabled Web browser

Although inheritance is the norm, some properties are noninheriting. What happens can be informally characterized as “intuitive inheritance” because exceptions to the rule make things behave more naturally. For example, if a background image is specified, it is tiled over the entire page. However, if nested elements inherit the background, they break up the pattern by restarting the image at every hierarchical block and subblock. Thus the *background-image* property is noninheriting. (You can override the default inheritance behavior by explicitly specifying certain properties to be inheritable or noninheritable.)

Returning to Figure 4.10a, the first rule causes the entire document to be formatted in a block. The second rule augments this element’s formatting to set the background to white and the block width to 7.5 inches. The third declares the Title font to be Times, 25 point, boldface. The fourth places the Agency record in a paragraph block with 8-point spacing above and 3-point spacing below, typeset as 16-point Helvetica. The inheritance mechanism ensures that nested elements share this typeface.

In rule five, the selector contains the two element names *Head* and *Body*, and both are assigned top, left, and right margins of 6 points, 0.2 inch, and 5 mm, respectively. Normally a style sheet would

not mix units in such a haphazard way. It is done so here to illustrate that CSS supports different units of measurement. Referring to the DTD that begins [Figure 4.7](#), the Head specification applies to the document's Title, and the Body applies to the Agency node. (Since the two specifications are the same, this effect could have been achieved more concisely by setting these properties in the NGODoc node.) Rule six adds italics to Abbrev, which already inherits 16-point Helvetica from Agency.

The last two rules use a construct known as *pseudo-elements*. The *:before* and *:after* qualifications perform the operations before and after the Abbrev element is processed—in this case placing parentheses around the abbreviation. Other pseudo-elements give access to the first character and first line of a block. The related construct *pseudo-classes* can distinguish whether a link has been visited or not, and supports interactive responses to events, such as the cursor hovering over a location.

In general, the ordering of rules in a style sheet is immaterial because every rule that matches any selector is applied. However, it is possible for rules to be contradictory—for example, the background color may be set to both red and blue. The CSS specification includes an algorithm that resolves ambiguity based upon ordering and precedence values.

Style sheets can be used in HTML documents by inserting a `<link>` tag

```
<link ref="stylesheet" type="text/css" href="example_style.css">
```

into the document's head. CSS instructions can also be embedded directly into the HTML by enclosing them within `<style type="text/css"> ... </style>` tags.

Cascading style sheets

Style sheets are *cascaded* when several are applied to the same document. [Figure 4.11](#) shows how the records in our running example can be embedded in a table and the title bulleted, along with the result viewed in a Web browser. The `@import` command incorporates the *un_basic* style in [Figure 4.10a](#) to provide a base layer of formatting rules, which will be augmented by the rules that follow. If conflicts occur, new properties take precedence. [Figure 4.11a](#) demonstrates some of CSS's table and list features.

The first rule augments the title formatting—Times font, 25 point, boldface, as defined in [Figure 4.10a](#)—with new specifications: the display type is set to *list item*, a bullet point (*disk*) is chosen to show list items, and the list is indented with a left-hand margin of 0.2 inch. These new rules create no conflict. CSS allows you to choose the symbol used for bullet points, use enumeration rather than bullets, and alter the style of enumeration (alphabetic, Roman, etc.). A counter mechanism allows such things as nested section numbering.

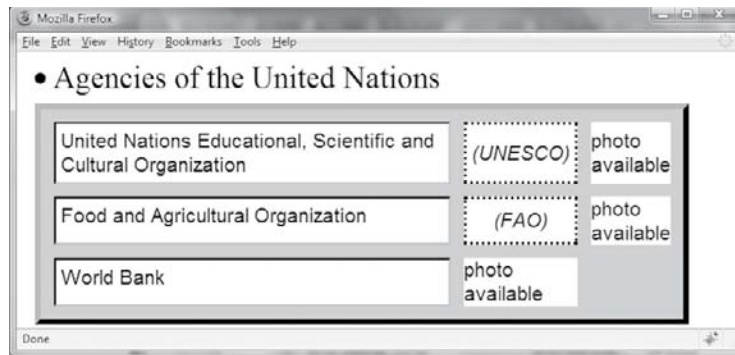
The other rules in [Figure 4.11a](#) present the document's information in tabular form. To do this, the style file maps element names to the display settings *table*, *table-row*, and *table-cell* and supplies appropriate stylistic parameters for each. First the Body element is mapped to *table*, along with values for background color and border style and size. The table layout mode is set to *auto*, causing the cell dimensions to be calculated automatically to make best use of available space. (The alternative is to specify the layout to be *fixed* and give cell width and height explicitly.) The value *separate* for *border-collapse* separates the borders of the individual cells.


```

/* Table and List item Cascading Style Sheet for UN Example */
@import url("un_basic.css");
Title { display: list-item;
        list-item: disk;
        margin-left: 0.2in; }
Body { display: table;
        table-layout: auto;
        background: silver;
        border: outset 5pt;
        border-collapse: separate;
        border-spacing: 10pt }
Agency { display: table-row; }
Name { display: table-cell;
        padding: 4pt;
        background: white;
        border: inset 2pt;}
Abbrev { display: table-cell;
        padding: 4pt;
        background: white;
        border: dotted black;
        text-align: center;
        vertical-align: middle; }
Photo { display: table-cell;
        width: 60pt;
        background: white;
        vertical-align: middle }
Photo:before { content: "photo available"; }

```

(a)



(b)

Figure 4.11: (a) Cascading style sheet illustrating tables and lists; (b) viewing the result in an XML-enabled Web browser

The next rule maps the Agency node to *table-row*, so that each agency's information is displayed in its own row. The following three rules define Name, Abbrev, and Photo to be table cells and specify some properties: a *white* background; *inset*, *dotted*, and (by omission) *plain* border styles; padded cells for Name and Abbrev (leaving space inside the border), and horizontally and vertically centered text in Abbrev.

Although the Photo elements in the XML document do not explicitly provide text information between tag pairs, they are defined as type *table-cell*, and so the table will include empty cells of

width 60 points. This illustrates a further point: the pseudo-element *before* fills the empty cell with the text “photo available.”

The end result in [Figure 4.11b](#) exhibits a small glitch: the “photo available” message on the last line appears in the second column, not the third. This reflects the structure of the XML document in [Figure 4.7](#), which lacks an abbreviation for the World Bank. CSS does not provide a general mechanism for altering document structure, although some manipulation is possible using pseudo-elements. In contrast, XSL is a more expressive language that allows the document structure to be modified.

There is far more to tables in CSS. Tables can have headers, footers, captions, or sections grouped by row or column, all of which can be structured hierarchically into the final table. CSS shares HTML’s (and XSL’s) table model, and concepts map naturally between the three.

Context- and media-dependent formatting

There’s more to cascading style sheets. Using compound selectors, rules can detect when descendant or sibling tags match a particular pattern and produce different effects. Rules can trigger when attributes match particular patterns, and this facility can be combined with compound selectors. [Figure 4.12](#) introduces some contrived formatting instructions into the running example to illustrate these points. Again it incorporates the formatting instructions of [Figure 4.10a](#) using the *@import* command.

(a)

```

/* Contextual Cascading Style Sheet instructions for UN Example */
@import url("un_basic.css");
Agency > Photo[desc]:before { content: "Available: " attr(desc) }
Body Photo[desc="A photo"]   { display: none }
Agency + Agency              { color: red;
                               background: #ffa080 }
Name[hq="Rome, Italy"]        { background: rgb(%0,%50,%0);
                               color:   rgb(255,160,80) }
Agency                       { font-size: 20pt; }

```

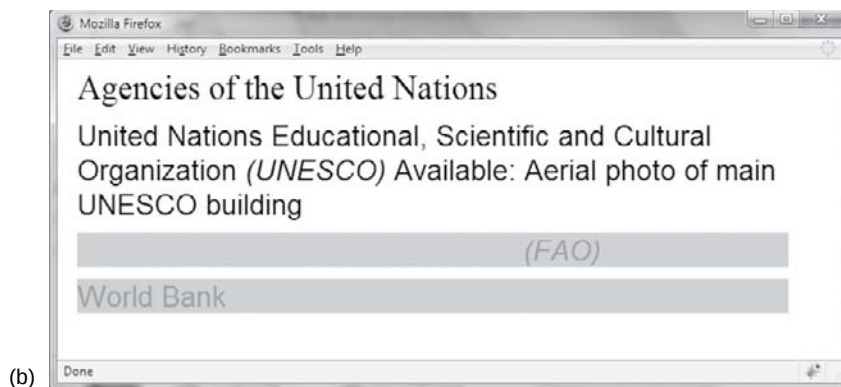


Figure 4.12: (a) Cascading style sheet illustrating context-sensitive formatting; (b) viewing the result in an XML-enabled Web browser

The first rule uses the pseudo-element *before* to tailor the content of a Photo according to the value of the *desc* attribute—but only when the Photo is a child of Agency. Omitting the > symbol would change the meaning to “descendant” rather than “child” and would trigger if the Photo node appeared anywhere beneath an Agency node.

The second rule suppresses the Photo text if its *desc* attribute matches the string “A photo.” If the first rule appeared without the second, the text “A photo” would be shown for both the FAO and the World Bank records because the document’s DTD supplies this text as the default value for *desc*.

The third rule demonstrates the + syntax that is used to specify sibling context. When one Agency node follows another at the same level in the document tree, this rule alters its background and foreground colors. In the XML document of [Figure 4.7](#), only the first Agency record in the document retains its default coloring. The rule also illustrates two different ways of specifying color: by name (red) and by specifying red, green, and blue components in hexadecimal.

The next rule prints the full name of the FAO in the same color as the background, because its *hq* attribute in the Name tag matches “Rome, Italy.” (The example would be more realistic if different colors were used here, but in this book we are restricted to black and white.) It uses a third form of color specification: *rgb()*, which gives color components in decimal—and these in fact specify the same color as in the previous hexadecimal assignment. This rule makes no sense in practice, but it explains why “(FAO)” is placed far to the right in [Figure 4.12b](#), because it is preceded by the now-invisible name.

The last rule further illustrates inheritance by setting the font size for Agency to 20 points. This overrides the 16-point value set in the initial style sheet and is inherited by descendant nodes.

A key feature of cascading style sheets is the ability to handle different media, such as screen, print, handheld devices, computer projectors, text-only display, Braille, and audio. [Figure 4.13](#) shows the idea. The *@media* command names the media type and gives rules, scoped to that media, within braces. The example first sets the Agency node globally to be a block with specified margins. Then for screen and projection media the font is set to 16-point Helvetica, while for print it is 12-point Times.

An *@import* command can be augmented to restrict the media type it applies to, for example:

```
@import url("un_audio.css") aural;
```

```
/* Different styles for different media */
Agency      { display: block;
               margin-top: 8pt;
               margin-bottom: 3pt }
@media screen, projection
  Agency      { font-size: 16pt;
               font-family: helvetica }
@media print
  Agency      { font-size: 12pt;
               font-family: times }
```

Figure 4.13: Using CSS to specify different formatting styles for different media

CSS continues to be developed. Like HTML and XHTML, the trend is to modularize the specification to make it easier for a software implementer to clarify what support is given. As with several Web technologies, the actual implementation of CSS in browsers lags behind the formal publication of the standard. Different browser versions support different subsets of the specification (often with idiosyncratic bugs). Web designers spend considerable time checking the appearance of their site in different browsers on different operating system platforms. For a digital library you should test your document display on common browsers and be aware of the browser population used to access your content. Typically, you will be able to find out about your users' browsers via your library's usage data records (see Section 2.2).

Extensible stylesheet language

XSL, the extensible stylesheet language for XML, transcends CSS by allowing the stylesheet designer to transform documents radically. Parts can be duplicated, tables of contents can be created automatically, lists can be sorted. A price is paid for this expressive power—complexity.

The XSL specification is divided into three parts: formatting objects (FO), XSL transformations (XSLT), and XPath (a way of selecting parts of a document). Formatting objects map closely to CSS instructions and use the same property names wherever possible. XSL transformations manipulate the document tree, while XPath selects parts to transform. We expand on these later in this section.

CSS can be combined with facilities such as Web-page scripting and the document object model mentioned in [Section 4.3](#) (under “Parsing XML”) to provide comparable functionality to XSL—this combination is sometimes dubbed *dynamic HTML*. Experts fiercely debate which is the better approach. We think you should know about both, since the wider context in which you work often dictates the path you must tread. There is one key difference, however, between the two approaches. Because XSL is designed to work with XML, it cannot be used with all forms of HTML—because not all forms are XML compliant. CSS operates in either setting: HTML, for which it was designed, and XML, because there is no restriction in the tag names that CSS can provide rules for.

We introduce formatting in XSL by working through the examples used to illustrate CSS. However, XSL greatly extends CSS's functionality. For example, it includes a model for pagination and layout that extends the page-by-page structure of paper documents to provide an equivalent to the “frames” used in Web pages. Its terminology is internationalized. For example, *padding-left* becomes *padding-start* to make more sense when dealing with languages that are written right to left. Similar terms are used to control the space above, below, and at the end of text, although the old names are recognized for backward compatibility.

[Figure 4.14](#) shows an XSL file for the initial version of the United Nations example. The XML syntax is far more verbose than its CSS counterpart in [Figure 4.10a](#). Beyond the initial NGODoc declaration, it includes many of the keywords we saw in the earlier version. For example, CSS's *font-size: 25pt* specification for the *Title* node now appears between nested tags whose inner and outer elements include the attributes *font-size=“25pt”* and *match=“Title”*, respectively.

```

<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format"
  version="1.0">
  <xsl:output method="xml"/>
  <xsl:template match="NGODoc">
    <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
      <fo:layout-master-set>
        <fo:simple-page-master master-name="UN-page" page-width="7.5in">
          <fo:region-body/>
        </fo:simple-page-master>
      </fo:layout-master-set>
      <fo:page-sequence master-name="UN-page">
        <fo:flow flow-name="xsl-region-body" background-color="white">
          <xsl:apply-templates/>
        </fo:flow>
      </fo:page-sequence>
    </fo:root>
  </xsl:template>
  <xsl:template match="Title">
    <fo:block font-family="Times" font-size="25pt"
      margin-left="0.2in" margin-right="5mm">
      <xsl:apply-templates/>
    </fo:block>
  </xsl:template>
  <xsl:template match="Agency">
    <fo:block space-before="8pt" space-after="3pt" font-size="16pt"
      font-family="Helvetica"
      margin-left="0.2in" margin-right="5mm">
      <xsl:apply-templates/>
    </fo:block>
  </xsl:template>
  <xsl:template match="Head|Body">
    <fo:block margin-left="0.2in" margin-right="5mm" space-before="6pt">
      <xsl:apply-templates/>
    </fo:block>
  </xsl:template>
  <xsl:template match="Abbrev">
    <fo:inline font-style="italic">
      (<xsl:value-of select="."/>)
    </fo:inline>
  </xsl:template>
</xsl:stylesheet>

```

Figure 4.14: XSL style sheet for the basic United Nations Agencies example

The style sheet is included in [Figure 4.7](#) by adding the line

```
<?xml-stylesheet href="un_basic.xsl" type="text/xsl"?>
```

just after the XML header declaration (the text of [Figure 4.14](#) resides in a file called *un_basic.xsl*), which is exactly what we did before with CSS. The result is a replica of [Figure 4.10b](#), although both standards are complex and it is not uncommon to encounter small discrepancies.

[Figure 4.14](#) begins with the obligatory XML processing application statement, followed by an `<xsl:stylesheet>` tag. As usual this is a top-level root element that encloses all the other tags. Its attri-

butes declare two namespaces: one for XSL itself, called *xsl*; the other for Formatting Objects, called *fo*. Namespaces are an XML extension that keep sets of elements designed for particular purposes separate—otherwise confusion would occur if both XSL and FO happened to include a tag with the same name (such as *block*). Once the namespaces have been set up in Figure 4.14, `<xsl:block>` specifies the XSL block tag while `<fo:block>` specifies the Formatting Objects tag.

Namespaces also incorporate semantic information. If an application encounters a namespace declaration whose value is `http://www.w3c.org/1999/XSL/Format`, it interprets subsequent tag names according to a published specification. In the following discussion we focus on a subset of Formatting Object tags typically used in document-related XML style sheets. The full specification is more comprehensive.

Returning to Figure 4.14, the next tag sets the document's output type. XSL style sheets perform *transformations*: they are used to transform an XML source document into another document. Because our style sheet is designed to format the document using Formatting Object tags, the output is set to *xml*. Other choices are *html*—in which case all the *fo*: scoped tags in the XSL file would need to be replaced with HTML tags—and *text*.

To perform the transformation, the input document is matched against the style sheet and a new document tree built from the result. First the document's root node is compared with the XSL file's `<xsl:template>` nodes until one is found whose *match* attribute corresponds to the node's name. Then the body of the XSL *template* element is used to construct the tags in the output tree. If *apply-templates* is encountered, matching continues recursively on that document node's children (or as we shall see later, on some other selected part of the document), and further child nodes in the output tree are built as a result.

In the example, the document's root node matches `<xsl:template match="NGODoc">`. This adds several *fo* tags to the output tree—tags that initialize the page layout of the final document. Eventually `<xsl:apply-templates>` is encountered, which causes the document's children `<Head>` and `<Body>` to be processed by the XSL file. When the matching operation has run its course, the document tree that it generates is rendered for viewing.

The fourth template rule specifies its *match* attribute as `Head | Body` to catch Head or Body nodes. Although it achieves the same effect as commas in CSS, this new syntax is part of a more powerful and general standard called XPath. The last template rule also introduces brackets around the abbreviation. The

```
<xsl:value-of select="." />
```

is again an XPath specification. The `"."` is a way of selecting the current position, or “here”—in this context it selects the text of the current node (Abbrev). This usage is adapted from the use of a period (`.`) in a file name to specify the current directory.

Using Formatting Objects

Formatting Objects provide similar capabilities to CSS: margins, borders, padding, foreground and background color, blocks, in-line text, tables with rows and cells, and so on. Many CSS declarations are simply mapped into *fo* tag names and attributes with the same name.

Figure 4.15 shows an XSL style sheet for the version of the United Nations Agencies example illustrated in Figure 4.11, with records embedded in a table and the title formatted with a bullet point. Like the CSS version, the file inherits from the basic XSL style sheet. This is done using the `<xsl:import>` tag, whose `href` attribute supplies the appropriate URL.

The first template rule processes the `<Title>` node, which starts by wrapping a *list-block* and *list-item* around the core information. Using a Unicode character that lies beyond the normal ASCII range, it then inserts a *list-item-label* whose content is a bullet point, before setting up the *list-item-body* with the content of the *Title* tag.

Next, instead of using `<apply-templates>` to recursively process any nested elements as was done in the first XSL example, this rule specifies `<apply-imports>`. This searches prior imported files (in the order that they were imported) for a rule that also matches the current tag (*Title*) and applies that rule as well. The result is to nest settings given in the *Title* rule of *un_basic.xsl* inside the current formatting, and then fire the `<apply-templates>` statement specified in the rule. The overall effect provides an inheritance facility similar to that of CSS.

The remaining template rules have *fo* elements for table, table row, and table cell that correspond to the same entities in CSS and are bound to the same element names in the source document. Attributes within these tags provide similar table formatting: a silver-colored table with white cells using a mixture of border styles and padding.

Some complications stem from the stricter requirements of the Formatting Objects specification. First, tables must include a body, whereas the equivalent structure in CSS is optional. In the example, the body element appears in the rule for *Body*, so this rule encodes both *table* and *table-body* elements. This is not possible in the CSS example because these two table structures are set by the *display* property, and would therefore conflict in the file. To avoid the conflict the source document would need two tag names: one mapping to *table* and the other to *table-body*.

A second complication is that *fo:blocks* cannot be placed immediately within *fo:table-body* and *fo:table-row* elements. That is why the two rules containing these elements must resort to `<xsl:apply-templates>` in their recursive processing of the document (instead of `<apply-imports>`) and duplicate the formatting attributes already present in the imported file.

Context- and media-dependent formatting

Figure 4.16 reworks the Figure 4.12 version of the United Nations example and illustrates context-based matching using contrived formatting instructions.

The key to context-based matching in XSL is the XPath mechanism. In many operating system interfaces, multiple files can be selected using wild card characters—for example, *project/*/file.html* selects all files of this name within any subdirectory of *project*. XPath generalizes this to select individual sections of a document. This is done by mapping nodes in the document tree into a string that defines their position in the hierarchy. These strings are expressed just as file names are in a directory hierarchy, with node names separated by slashes. For example, in our document *NGODoc/Body/** returns all the Agency nodes.

This idea is augmented to condition access on attributes stored at nodes. For example, *Name[@desc]* matches a *Name* node only if it has a *desc* attribute defined. Built-in predicates are supplied to

```

<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format"
  version="1.0">
<xsl:import href="un_basic.xsl"/>
<xsl:template match="Title">
  <fo:list-block>
    <fo:list-item>
      <fo:list-item-label end-indent="label-end()">
        <fo:block font-family="ZapfDingbats" font-size="9pt">
          &#9679;
        </fo:block>
      </fo:list-item-label>
      <fo:list-item-body start-indent="body-start()">
        <fo:block>
          <xsl:apply-imports/>
        </fo:block>
      </fo:list-item-body>
    </fo:list-item>
  </fo:list-block>
</xsl:template>
<xsl:template match="Body">
  <fo:table table-layout="auto" background-color="silver"
    border-style="outset 5pt" border-collapse="separate"
    Border-spacing="10pt">
    <fo:table-body>
      <xsl:apply-templates/>
    </fo:table-body>
  </fo:table>
</xsl:template>
<xsl:template match="Agency">
  <fo:table-row space-before="8pt" space-after="3pt"
    font-size="16pt" font-family="Helvetica"
    margin-left="0.2in" margin-right="5mm">
    <xsl:apply-templates/>
  </fo:table-row>
</xsl:template>
<xsl:template match="Name">
  <fo:table-cell padding="4pt" background-color="white"
    border-style="inset 2pt" display-align="center">
    <fo:block>
      <xsl:apply-imports/>
    </fo:block>
  </fo:table-cell>
</xsl:template>
<xsl:template match="Abbrev">
  <fo:table-cell padding="4pt" background-color="white"
    border-style="dotted black" text-align="center"
    display-align="center">
    <fo:block>
      <xsl:apply-imports/>
    </fo:block>
  </fo:table-cell>
</xsl:template>
<xsl:template match="Photo">
  <fo:table-cell width="60pt" background-color="white"
    display-align="center">
    <fo:block>
      Photo available <xsl:apply-imports/>
    </fo:block>
  </fo:table-cell>
</xsl:template>
</xsl:stylesheet>

```

Figure 4.15: XSL style sheet illustrating tables and lists


```

<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format"
  version="1.0">
  <xsl:import href="un_basic.xsl"/>
  <xsl:template match="Agency/Photo[@desc]">
    Available: <xsl:value-of select="@desc"/>
  </xsl:template>
  <xsl:template match="Agency/Photo[@desc='A photo']">
  </xsl:template>
  <xsl:template match="Agency">
    <fo:block space-before="8pt" space-after="3pt" font-size="20pt"
      font-family="Helvetica"
      margin-left="0.2in" margin-right="5mm">
      <xsl:apply-templates select="." mode="Extra Color"/>
    </fo:block>
  </xsl:template>
  <xsl:template match="Agency[position()>1]" mode="Extra Color">
    <xsl:attribute name="background-color">#ffa080</xsl:attribute>
    <xsl:attribute name="color">red</xsl:attribute>
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="Name[@hq='Rome, Italy']">
    <fo:inline color="rgb(255,160,80)">
      <xsl:apply-templates/>
    </fo:inline>
  </xsl:template>
</xsl:stylesheet>

```

Figure 4.16: XSL style sheet illustrating context-sensitive formatting

check the position of a node in the tree—for example, whether it is the first or last in a chain of siblings.

The first template rule in Figure 4.16 inserts the text that is stored as a Photo node’s *desc* attribute into the document, prefixed by “Available.” The second is more selective and only matches if the Photo node’s *desc* attribute contains the text “A photo”—which happens to coincide with its default given in the DTD. If it matches, no text is displayed, and recursive template matching down that part of the tree is abandoned.

The third rule, which works in conjunction with the fourth, demonstrates XSL modes. When an Agency node is first encountered, rule 3 fires and sets up the basic formatting for the block. When it comes to recursively applying the template match, it selects itself with *select="."*, switches the mode to *Extra Color*, and then rematches on Agency. This time only rule 4 can match (because of the mode), which enforces the additional requirement that Agency must be at least the second node in the file. If so, the rule uses *<xsl:attribute>* tags to augment the closest enclosing tag (the *main* block for Agency) with attributes for foreground and background colors.

Finally, the remaining rule sets the foreground color the same as the background color for any Name node whose *hq* attribute matches “Rome, Italy.”

```

<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format"
  version="1.0">
  <xsl:import href="un_basic.xsl"/>
  <xsl:template match="Body">
    <fo:block margin-left="0.2in" margin-right="5mm" space-before="6pt">
      <xsl:apply-templates select="Agency">
        <xsl:sort data-type="string" select="./Name"/>
      </xsl:apply-templates>
    </fo:block>
  </xsl:template>
</xsl:stylesheet>

```

Figure 4.17: XSL style sheet that sorts United Nations agencies alphabetically

XSL supports different output media—screen, printer, and so on—using the media attribute of `<xsl:output>`, which we have already seen used to set the output type to XML. For example, to restrict a style sheet setting to printers, add

```
<xsl:output output="xml" media="printer">.
```

Processing in XSL

Our examples have shown XSL's ability to transform the source document, but the changes have been slight (such as putting brackets around the content of an `Abbrev` tag) and could all have been achieved using CSS. [Figure 4.17](#) shows a style sheet that sorts the United Nations agencies alphabetically for display, something CSS can't do. It imports *un_basic.xsl* to provide some general formatting and then defines a rule for `Body` that performs the sorting, overriding the match that would have occurred against *Head* | *Body* in the imported file.

First a block is created that maintains the same margins and spacing provided by the basic style file. Then a recursive match is initiated on all `Agency` nodes that are descendants of the `Body` node. In earlier examples matching has been expressed by combining the opening and closing tags, as in `<xsl:apply-templates/>`. This shorthand notation is convenient for straightforward matches. Here we separate the opening and closing parts, and supply the sort criterion through the nested tag `xsl:sort`. To accomplish the desired result, the example sets the data type to *string* and specifies a sort on child nodes of `Agency` called `Name`.

This example only scratches the surface. XSL can perform a vast array of transformations—even for sorting there are many more attributes that control the ordering. Other language constructs include variables, *if* statements, and *for* statements. XSL contains many elements of programming languages, making it impressively versatile, and it is finding use in places that even its designers did not envision.

4.5 Page Description Languages: PostScript and PDF

The purpose of page description languages is to express typeset documents in a way that is independent of the particular output device used. Early word-processing programs and drawing packages incorporated code for sending documents to particular printers and could not be used with other devices. With the advent of page description languages, programs can generate documents in a device-independent format that will print on any device equipped with a driver for that language.

Most of the time, digital libraries can treat documents in these description languages by processing them using standard “black boxes”: generate this report in a particular syntax, display it here, transfer it there, and print. However, to build coherent collections from the documents, you need internal knowledge of these formats to understand what can and cannot be accomplished: whether the text can be indexed, bookmarks inserted, images extracted, and so on. For this reason we describe two page description languages in detail.

PostScript, the first commercially developed page description language, was released in 1985, whereupon it was rapidly adopted by software companies and printer manufacturers as a platform-independent way of describing printed pages that could include both text and graphics. Soon it was being coupled with software applications (notably, in the early days, PageMaker on the Apple Macintosh) that ensure that “what you see” graphically on the computer’s raster display is “what you get” on the printed page. PDF, Portable Document Format, is a page description language that arose out of PostScript and addresses many of its shortcomings.

PostScript fundamentals

PostScript files comprise a sequence of drawing instructions, including ones that draw particular letters from particular fonts. The instructions are like this: move to the point defined by these x and y coordinates and then draw a straight line to here; using the following x and y coordinates as control points, draw a smooth curve around them with such-and-such a thickness; display a character from this font at this position and in this point size; display the following image data, scaled and rotated by this amount. Instructions are included to specify such things as page size, clipping away all parts of a picture that lie outside a given region, and when to move to the next page.

But PostScript is more than just a file format. It is a high-level programming language that supports diverse data types and operations on them. Variables and predefined operators allow the usual kinds of data manipulation. New operations can be encapsulated as user-defined functions. Data can be stored in files and retrieved. A PostScript document is more accurately referred to as a PostScript *program*. It is printed or displayed by passing it to a PostScript interpreter, a full programming language interpreter.

Being a programming language, PostScript allows print-quality documents that comprise text and graphical components to be expressed in an exceptionally versatile way. Ultimately, when interpreted, the abstract PostScript description is converted into a matrix of dots or pixels through a process known as *rasterization* or *rendering*. The dot structure is imperceptible to the eye—

commonly available printers have a resolution of 600 dpi, and publishing houses use 1,200 dpi and above (see Table 4.2). This very book is an example of what can be described using the language.

Modern computers are powerful enough that a PostScript description can be quickly rasterized and displayed on the screen. This adds an important dimension to online document management: computers without the original software used to compose a document can still display the finished product exactly as it was intended to be displayed. Indeed, in the late 1980s one computer manufacturer took the idea to an extreme by developing an operating system (called NeXT) in which the display was controlled entirely from PostScript, and all applications generated their on-screen results in this form.

However, PostScript was not designed for screen displays. As is true for ASCII, limitations often arise when a standard is put to use in situations for which it was not designed. Just as ASCII is being superseded by Unicode, the Portable Document Format (PDF) has been devised as the successor to PostScript for online documents. Today the Apple Macintosh uses PDF throughout, just as the NeXT used PostScript.

The language

PostScript is page-based. Graphical marks are drawn one by one until an operator called *showpage* is encountered, whereupon the page is presented. When one page is complete, the next is begun. Placement is like painting: if a new mark covers a previously painted area, it completely obliterates the old paint. Marks can be black and white, grayscale, or color. They are “clipped” to fit within a given area (not necessarily the page boundary) before being placed on the page. This process defines the *imaging model* used by PostScript.

Table 4.3 summarizes PostScript’s main graphical components. Various geometric primitives are supplied. Circles and ellipses can be produced using the *arc* primitive; general curves are drawn using *splines*, curved lines whose shapes are controlled precisely by a number of control points. A *path* is a sequence of graphical primitives interspersed with geometric operations and stylistic attributes. Once a path has been defined, it is necessary to specify how it is to be painted: for example, *stroke* for a line or *fill* for a solid shape. The *moveto* operator moves the pen without actually drawing, so that paths do not have to prescribe contiguous runs of paint. An operator called *closepath* forms a closed shape by generating a line from the latest point back to the last location moved to. The origin of

Table 4.3: Graphical components in PostScript

Graphical primitives	Straight lines, arcs, general curves, sampled images and text
Geometrical operations	Scale, translate, and rotate
Line attributes	Width, dashed, start and end caps, joining lines/corner mitre (style)
Font attributes	Font, typeface, size
Color	Color currently in use
Paths	Sequence of graphical primitives and attributes
Rendering	How to render paths: grayscale, color, or outline
Clipping	Restricts what is shown of the path

coordinates is located at the bottom left-hand corner of a page, and the unit of distance is set to be one *printer's point*, a typographical measure whose size is 1/72 inch.

In PostScript, text characters are just another graphical primitive: they can be rotated, translated, scaled, and colored just like any other object. However, because of its importance, text receives special treatment. The PostScript interpreter stores information about the current font: font name, font type, point size, and so on, and operators like *findfont* and *scalefont* are provided to manipulate these components. There is also a special operator called *image* for sampled images.

Files containing PostScript programs are represented in 7-bit ASCII, but this does not restrict the fonts and characters that can be displayed on a page. A percentage symbol (%) indicates that the remainder of the line contains a comment; however, comments marked with a double percent (%%) extend the language by giving structured information that can be utilized by a PostScript interpreter.

Figure 4.18b shows a simple PostScript program that, when executed, produces the result in Figure 4.18a, which contains the greeting *Welcome* in five languages. The first line, which is technically a

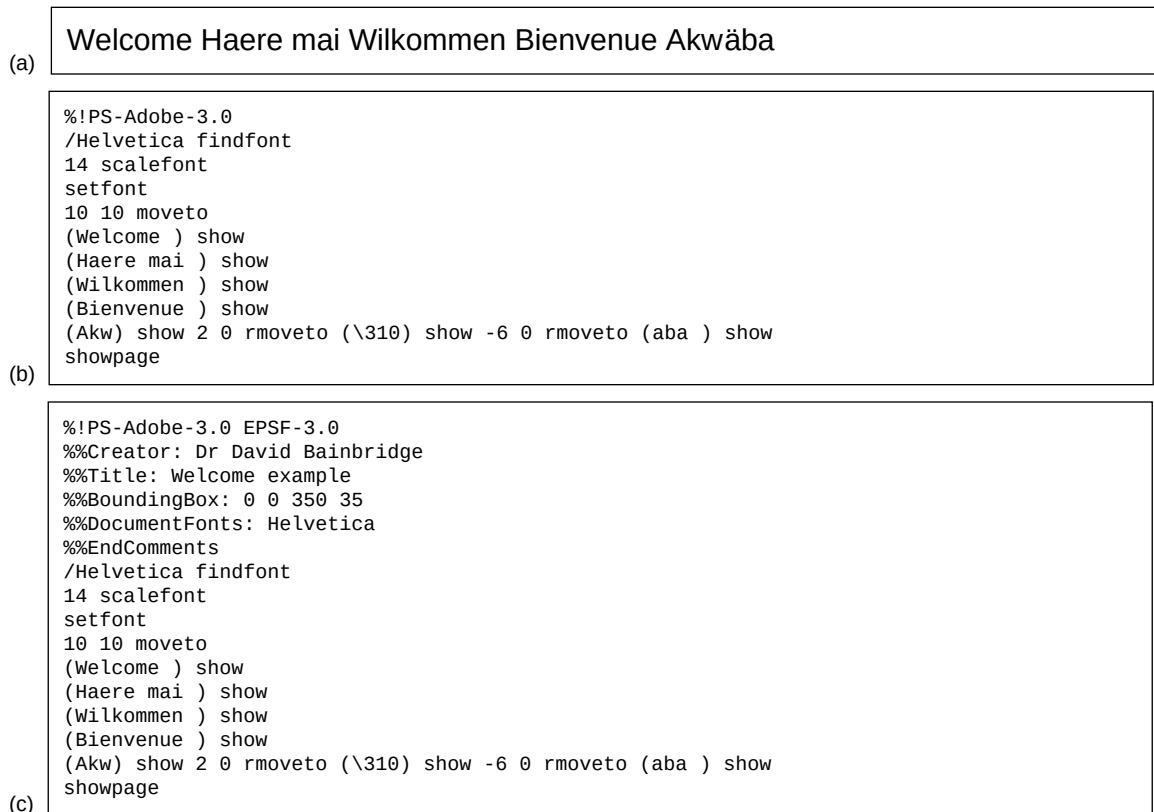


Figure 4.18: (a) Result of executing a PostScript program; (b) the PostScript program; (c) encapsulated PostScript version;

```

%PDF-1.3
1 0 obj
<< /Type /Catalog
  /Pages 2 0 R
>>
endobj

2 0 obj
<< /Type /Pages
  /Kids [3 0 R]
  /Count 1
>>
endobj

3 0 obj
<< /Type /Page
  /Parent 2 0 R
  /MediaBox [0 0 612 792]
  /Contents 4 0 R
  /Resources << /ProcSet 5 0 R
                /Font << /F1 6 0 R >>
                >>
>>
endobj

4 0 obj
<< /Length 118 >>
stream
BT
  /F1 14 Tf
  10 10 Td
  (welcome ) Tj
  (Haere mai ) Tj
  (Wilkommen ) Tj
  (Bienvenue ) Tj
  (Akw\344ba ) Tj
ET
endstream
endobj

5 0 obj
[ /PDF /Text ]
endobj

6 0 obj
<< /Type /Font
  /Subtype /Type1
  /Name /F1
  /BaseFont /Helvetica
  /Encoding /WinAnsiEncoding
>>
endobj

xref
0 7
0000000000 65535 f
0000000009 00000 n
0000000062 00000 n
0000000126 00000 n
0000000311 00000 n
0000000480 00000 n
0000000511 00000 n

trailer
<< /Size 7
  /Root 1 0 R
>>
startxref
631
%%EOF

```

(d)

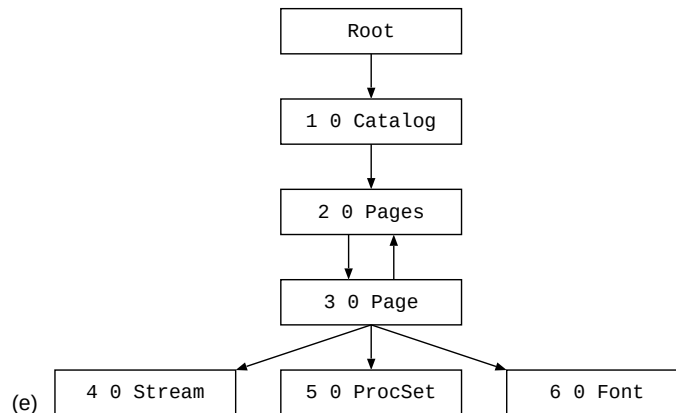


Figure 4.18, cont'd: (d) PDF version; (e) network of objects in the PDF version

comment but must be present in all PostScript programs, defines the file to be of type PostScript. The next two lines set the font to be 14-point Helvetica, and then the current path is moved to a position (10,10) points from the lower left-hand corner of the page.

The five *show* lines display the *Welcome* text (plus a space). PostScript, unlike most computer languages, uses a stack-based form of notation where commands *follow* their arguments. The *show* commands “show” the text that precedes them; parentheses are used to group characters together into text strings. In the fifth example, the text *Akw* is “shown” or painted on the page; then there is a relative move (*rmoveto*) of the current position forward two printer’s points (the coordinate specification (2, 0)); then the character `\310` is painted (octal 310, which is in fact an umlaut in the Latin-1 extension of ASCII); the current position is moved back six points; and the characters *aba* are “shown.” The effect is to generate the composite character *ä* in the middle of the word. Finally the *showpage* operator is issued, causing the graphics that have been painted on the virtual page to be printed on a physical page.

The PostScript program in [Figure 4.18b](#) handles the composite character *ä* inelegantly. It depends on the spacing embodied in the particular font chosen—on the fact that moving forward two points, printing an umlaut, and moving back six points will position the forthcoming *a* directly underneath. There are better ways to accomplish this, using, for instance, `ISOLatin1Encoding` or composite fonts, but they are beyond the scope of this simple example.

Evolution

Standards and formats evolve. There is a tension between stability, an important feature for any language, and currency, the need to extend in response to the ever-changing face of computing technology. To help resolve the tension, *levels* of PostScript are defined. The conformance level of a file is encoded in its first line, as can be seen in [Figure 4.18b](#) (*PS-Adobe-3.0* means Level 3 PostScript). Care is taken to ensure that levels are backward-compatible.

What we have described so far is basic Level 1 PostScript. Level 2 includes

- improved virtual memory management
- device-independent color
- composite fonts
- filters.

The virtual memory enhancements use whatever memory space is available more efficiently, which is advantageous because PostScript printers sometimes run out of memory when processing large documents. Composite fonts, which significantly help internationalization, are described below. Filters provide built-in support for compression, decompression, and other common ways of encoding information.

Level 2 was announced in 1991, six years after PostScript’s original introduction. The additions were quite substantial, and it was a long time before it became widely adopted. Level 3 (sometimes called PostScript 3) was introduced in 1998. Its additions are minor by comparison, and include

- more fonts, and provision for describing them more concisely;
- improved color control and smoother shading;
- advanced processing methods that accelerate rendering.

While PostScript *per se* does not impose an overall structure on a document, applications can take advantage of a prescribed set of rules known as the *document structuring conventions* (DSC). These divide documents into three sections: a prolog, document pages, and a trailer. The divisions are expressed as PostScript “comments.” For example, `%%BeginProlog` and `%%Trailer` define section boundaries. Other conventions are embedded in the document—such as `%%BoundingBox`, discussed below. There are around 40 document structuring commands in all.

Document structuring commands provide additional information about the document but do not affect how it is rendered. Since the commands are couched as comments, applications that do not use the conventions are unaffected. However, other applications can take advantage of the information.

Applications that generate PostScript, such as word processors, commonly use the prolog to define procedures that are helpful in generating document pages and use the trailer to tidy up any global operations associated with the document or to include information (such as a list of all fonts used) that is not known until the end of the file. This convention enables pages to be expressed more concisely and clearly.

Encapsulated PostScript

Encapsulated PostScript is a variant designed for expressing documents of a single page or less. It is widely used to incorporate artwork created using a software application like a drawing package into a larger document, such as a report being composed in a word processor. Encapsulated PostScript is built on top of the document-structuring conventions.

Figure 4.18c shows the *Welcome* example expressed in Encapsulated PostScript. The first line is augmented to reflect this (the encapsulation convention has levels as well; this is EPSF-3.0). The `%%BoundingBox` command that specifies the size of the drawing is mandatory in Encapsulated PostScript. Calculated in points from the origin (bottom left-hand corner), it defines the smallest rectangle that entirely encloses the marks constituting the rendered picture. The rectangle is specified by four numbers: the first pair give the coordinates of the lower left corner, and the second pair define the upper right corner. Figure 4.18c also shows document-structuring commands for the creator of the document (more commonly it gives the name and version number of the software application that generated the file), a suitable title for it, and a list of fonts used (in this case, just Helvetica).

An Encapsulated PostScript file contains raw PostScript along with a few special comments. It can be embedded verbatim, header and all, into a context that is also PostScript. For this to work properly, operators that affect the global state of the rendering process must be avoided. These restrictions are listed in the specification for Encapsulated PostScript and in practice are not unduly limiting.

Fonts

PostScript supports two broad categories of fonts: base and composite fonts. Base fonts accommodate alphabets up to 256 characters. Composite fonts extend the character set beyond this point and

also permit several glyphs to be combined into a single character—making them suitable for languages with large alphabets, such as Chinese, and with frequent character combinations, such as Korean.

In Figure 4.18b the *findfont* operator is used to set the font to Helvetica. This searches PostScript's font directory for the named font (*/Helvetica*), returning a *font dictionary* that contains all the information necessary to render characters in that font. Most PostScript products have a built-in font directory with descriptions of 13 standard fonts from the Times, Helvetica, Courier, and Symbol families. Helvetica is an example of a base font format.

The execution of a *show* command such as (*Welcome*) *show* takes place in two steps. For each character, its numeric value (0–255) is used to access an array known as the *encoding vector*. This provides a name, such as */W* (or, for nonalphabetic characters, a name like */hyphen*). This name is then used to look up a glyph description in a subsidiary dictionary. A *name* is one of the basic PostScript types: it is a label that binds itself to an object. The act of executing the glyph object renders the required mark. The font dictionary is a top-level object that binds these operations together.

In addition to the built-in font directory, PostScript lets you provide your own graphical descriptions for the glyphs, which are then embedded in the PostScript file. You can also change the encoding vector.

Font formats

The original specification for PostScript included a means of defining typographical fonts. At the time there were no standard formats for describing character forms digitally. PostScript fonts, which were built into the LaserWriter printer in 1985 and subsequently adopted in virtually all typesetting devices, sparked a revolution in printing technology. However, to protect its investment, Adobe, the company that introduced PostScript, kept the font specification secret. This spurred Apple to introduce a new font description format six years later (and this format was subsequently adopted by the Windows operating system). Adobe then published its format.

Level 3 PostScript incorporates both ways of defining fonts. The original method is called Type 1; the rival scheme is TrueType. For example, Times Roman, Helvetica, and Courier are Type 1 fonts, while Times New Roman, Arial, and Courier New are the TrueType equivalents.

Technically, the two font description schemes have much in common. Both describe glyphs in terms of the straight lines and curves that make up the outline of the character. This means that standard geometric transformations—translation, scaling, rotation—can be applied to text as well as to graphic primitives. One difference between Type 1 and TrueType is the way in which curves are specified. Both use spline curves, but the former uses a kind of cubic spline called a *Bézier curve*, whereas the latter uses a kind of quadratic spline called a *B-spline*. From a user perspective these differences are minimal—but they do create incompatibilities.

Both representations are resolution independent. Characters may be resized by scaling the outlines up or down—although a particular implementation may impose practical upper and lower limits. It is difficult to scale down to very small sizes. When a glyph comprises only a few dots, inconsistencies arise in certain letter features depending on where they are placed on the page, because even though

the glyphs are the same size and shape, they sit differently on the pixel grid. For example, the width of letter stems may vary from one instance of a letter to another; worse still, when scaled down, key features may disappear altogether.

Both Type 1 and TrueType deal with this by putting additional information called *hints* into fonts to make it possible to render small glyphs consistently. However, the way that hints are specified is different in each case. Type 1 fonts give hints for vertical and horizontal features, overshoots, snapping stems to the pixel grid, and so on, and in many cases there is a threshold pixel size at which they are activated. TrueType hints define flexible instructions that can do much more. They give the font producer fine control over what happens when characters are rendered under different conditions, but to use them to full advantage, individual glyphs must be manually coded. This is such a daunting undertaking that, in practice, many fonts omit this level of detail. Of course this does not usually affect printed text, because even tiny fonts can be displayed accurately, without hinting, on a 600-dpi device. Hinting is only really important for screen displays.

Composite fonts

Composite fonts became standard in Level 3 PostScript. They are based on two key concepts. First, instead of using a single dictionary for mapping character values, as base fonts do, composite fonts use a hierarchy of dictionaries. At its root, a composite font dictionary directs character mappings to subsidiary dictionaries, which can contain either base fonts or further composite fonts (up to a depth limit of five). Second, the *show* operator no longer decodes its argument one byte at a time. Instead, a *font number* and *character selector* pair are used. The font number locates a dictionary within the hierarchy, while the character selector uses the encoding vector stored with that dictionary to select a glyph description name to use when rendering the character. This latter step is analogous to the way base fonts are used.

The arguments of *show* can be decoded in several ways. Options include 16 bits per font number and character selector pair, separated into one byte each (note that this differs from a Unicode representation), or using an escape character to change the current font dictionary. The method used is determined by a value in the root dictionary.

Compatibility with Unicode

Character-identifier keyed, or *CID-keyed*, fonts provide a newer format designed for use with Unicode. They map multiple byte values to character codes in much the same way that the encoding vector works in base fonts—except that the mapping is not restricted to 256 entries. The CID-keyed font specification is independent of PostScript and can be used in other environments. The data is also external to the document file: font and encoding-vector resources are accessed by reading external files into dictionaries.

OpenType is a new font description that goes beyond the provisions of CID-keyed fonts. It encapsulates Type 1 and TrueType fonts into the same kind of wrapper, yielding a portable, scalable font platform that is backward-compatible. The basic approach of CID-keyed fonts is used to map numeric identifiers to character codes. OpenType includes multilingual character sets with full Unicode support, and extended character sets that support small caps, ligatures, and fractions—all

within the same font. It includes a way of automatically substituting a single glyph for a given sequence (e.g., the ligature *fi* can be substituted for the sequence *f* followed by *i*) and vice versa. Substitution can be context sensitive. For example, a *swash* letter, which is an ornamental letter—often a decorated italic capital—used to open paragraphs, can be introduced automatically at the beginning of words or lines.

Text extraction

It is useful to be able to extract plain text from PostScript files. To build a full-text index for a digital library, the raw text needs to be available. An approximation to the formatting information may be useful too—perhaps to display HTML versions of documents in a Web browser. For this, structural features like paragraph boundaries and font characteristics must be identified from PostScript.

PostScript allows complete flexibility in how documents are described—for example, the characters do not have to be in any particular order. In practice, actual PostScript documents tend to be more constrained. However, the text they contain is often fragmented and inextricably muddled up with other character strings that do not appear in the output. Figure 4.19 shows an example, along with the text extracted from it. Characters to be placed on the page appear in the PostScript file as parenthe-

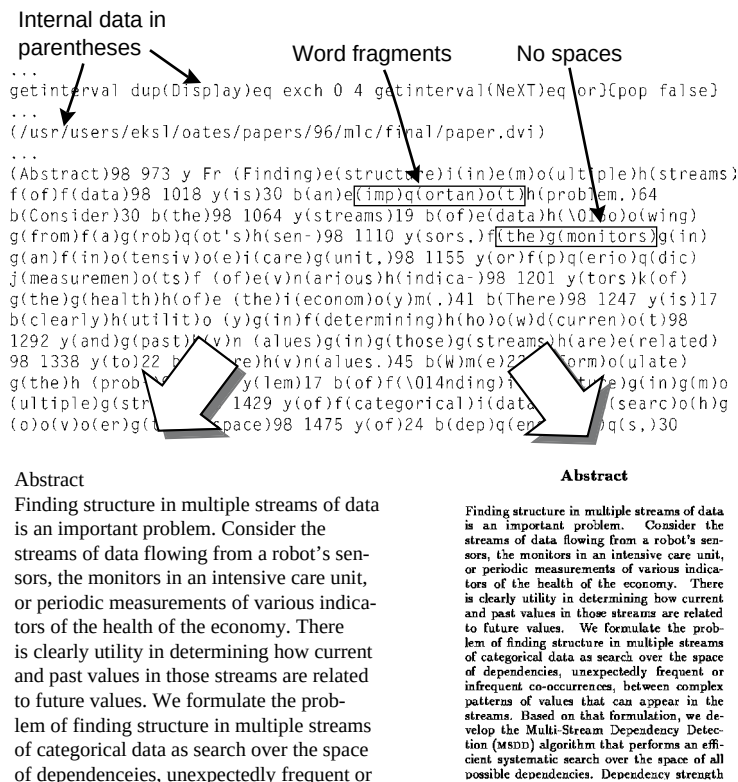


Figure 4.19: A PostScript document and the text extracted from it

sized strings. But font names, file names, and other internal information are represented in the same way—examples can be seen in the first few lines of the figure. Also the division of text into words is not immediately apparent. Spaces are implied by the character positions rather than being present explicitly. Text is written out in fragments, and each parenthetical string sometimes represents only part of a word. Deciding which fragments to concatenate is difficult. Although heuristics might be devised to cover common cases, they are unlikely to lead to a robust solution that can deal satisfactorily with the variety of files found in practice.

This is why text extraction based on scanning a PostScript document for strings of text meets with limited success. It also fails to extract any formatting information. Above all, it does not address the fundamental issue that PostScript is a programming language whose output, in principle, cannot be determined merely by scanning the file—for example, in a PostScript document the raw text could be (and often is) compressed, to be decompressed by the interpreter every time the document is displayed. As it happens, this deep-rooted issue leads to a solution that is far more robust than scanning for text, can account for formatting information, and decodes any programmed information.

If a PostScript code fragment is prepended to a document and the document is then run through a standard PostScript interpreter, the placement of text characters can be intercepted, producing text in a file, rather than pixels on a page. The central trick is to redefine PostScript's *show* operator, which is responsible for placing text on the page. Regardless of how a program is constructed, all printed text passes through this operator (or a variant, as mentioned later). The new code fragment redefines it to write its argument, a text string, to a file instead of rendering it on the screen. Then, when the document is executed, a text file is produced instead of the usual physical pages.

A simple text extraction program

The idea can be illustrated by a simple program. Prepending the incantation `/show { print } def`, shown in [Figure 4.20a](#), to the document of [Figure 4.19](#) redefines the *show* operator. The effect is to define the name *show* to be *print* instead—and therefore print the characters to a file. The result appears at the right of [Figure 4.20a](#). One problem has been solved: winnowing the text destined for a page from the remainder of the parenthesized text in the original file.

The problem of identifying whole words from fragments must still be addressed, for the text in [Figure 4.20a](#) contains no spaces. Printing a space between each fragment yields the text in [Figure 4.20b](#). Spaces do appear between each word, but they also appear within words, such as *m ultiple* and *imp ortan t*.

To put spaces in their proper places, it is necessary to consider where fragments are placed on the page. Between adjacent characters, the print position moves only a short distance from one fragment to the next; if a space intervenes, the distance is larger. An appropriate threshold will depend on the type size and should be chosen accordingly; however, we use a fixed value for illustration.

The program fragment in [Figure 4.20c](#) implements this modification. The symbol *X* records the horizontal coordinate of the right-hand side of the previous fragment. The new *show* procedure obtains the current *x* coordinate using the *currentpoint* operator (the *pop* discards the *y* coordinate) and subtracts the previous coordinate held in *X*. If the difference exceeds a preset threshold—in this case, five points—a space is printed. Then the fragment itself is printed.

(a)	<pre>/show { print } def</pre>	Finding structure in multiple streams of data is an important problem. Consider the streams of data flowing from a robot's sensors, the monitors in an intensive care unit, or periodic measurements of various indicators of the health of the economy. There is clearly utility in determining how current and past values in those streams are related to future values
(b)	<pre>/show { print () print } def</pre>	Finding structure in multiple streams of data is an important problem. Consider the streams of data flowing from a robot's sensors, the monitors in an intensive care unit, or periodic measurements of various indicators of the health of the economy. There is clearly utility in determining how current and past values in those streams are related to future values
(c)	<pre>/X 0 def /show { currentpoint pop X sub 5 gt { () print } if dup print systemdict /show get exec currentpoint pop /X exch def } def</pre>	Finding structure in multiple streams of data is an important problem. Consider the streams of data flowing from a robot's sensors, the monitors in an intensive care unit, or periodic measurements of various indicators of the health of the economy. There is clearly utility in determining how current and past values in those streams are related to future values.
(d)	<pre>/X 0 def /protoshow { currentpoint pop X sub 5 gt { () print } if dup print systemdict exch get exec currentpoint pop /X exch def } def</pre>	<pre>/show { /show protoshow } def /kshow { /kshow protoshow } def /widthshow { /widthshow protoshow } def /ashow { /ashow protoshow } def /awidthshow { /awidthshow protoshow } def</pre>

Figure 4.20: Extracting text from PostScript: (a) Printing all fragments rendered by *show*; (b) putting spaces between every pair of fragments; (c) putting spaces between fragments with a separation of at least five points; (d) catering for variants of the *show* operator

In order to record the new x coordinate, the fragment must actually be rendered. Unfortunately, [Figures 4.20a and b](#) have suppressed rendering by redefining the *show* operator. The line `systemdict /show get exec` retrieves the original definition of *show* from the system dictionary (`systemdict /show get`) and executes it (`exec`) with the original string as argument. This renders the text and updates the current point, which is recorded in X on the next line. Executing the original *show* operator provides a foolproof way of updating coordinates exactly as they are when the text is rendered. This new procedure produces the text in [Figure 4.20c](#), in which all words are segmented correctly. Line breaks are detected by analyzing vertical coordinates in the same way and comparing the difference with another fixed threshold.

PostScript (to be precise, Level 1 PostScript) has four variants of the *show* command—*ashow*, *widthshow*, *awidthshow*, and *kshow*—and they should all be treated similarly. In [Figure 4.20d](#), a procedure is defined to do the work. It is called with two arguments, the text string and the name of the appropriate *show* variant. Just before it finishes, the code for the appropriate command is located in the system dictionary and executed.

Second, line breaks in PostScript documents are designed for typeset text with proportionally spaced fonts. The corresponding lines of plain text are rarely all of the same length. Moreover, the best line wrapping often depends on context—such as the width of the window that displays the text. Paragraph breaks, on the other hand, have significance in terms of document content and should be preserved. Line and paragraph breaks can be distinguished in two ways. Usually paragraphs are separated by more vertical space than lines are. In this case, any advance that exceeds the nominal line space can be treated as a paragraph break. The nominal spacing can be taken as the most common nontrivial change in y coordinate throughout the document.

Third, more complex processing is needed to deal properly with different fonts. For instance, ligatures, bullets, and printer's quotes (“ ‘ ’ ” rather than ‘ ’ ”) are non-ASCII values that can be recognized and mapped appropriately. Mathematical formulas with complex sub-line spacing, Greek letters, and special mathematical symbols are difficult to deal with satisfactorily. A simple dodge is to flag unknown characters with a question mark, because there is no truly satisfactory plain-text representation for mathematics.

Finally, printed pages often appear in reverse order. This is for mechanical convenience: when pages are placed face up on the output tray, the first one produced is the last page of the document. PostScript's document-structuring conventions include a way of specifying page ordering, but it is often not followed in actual document files. Of several possible heuristics for detecting page order, a robust one is to extract numbers from the text adjacent to page breaks. These are usually page numbers, and you can tell that a document is reversed because they decrease rather than increase. Even if some

numbers in the text are erroneously identified as page numbers, the method is quite reliable if the final decision is based on the overall majority.

Using PostScript in a digital library

Some early digital libraries were built from PostScript source documents, with contemporary versions shifting to PDF (discussed below) or a combination of the two. PostScript's ability to display print-quality documents using a variety of fonts and graphics on virtually any computer platform is a wonderful feature. Because the files are 7-bit ASCII, they can be distributed electronically using lowest-common-denominator e-mail protocols. And although PostScript predates Unicode, characters from different character sets can be freely mixed because documents can contain many different fonts. Embedding fonts in documents makes them faithfully reproducible even when sent to printers and computer systems that lack the necessary fonts.

The fact that PostScript is a programming language, however, introduces problems that are not normally associated with documents. A document is a program. And programs crash for a variety of obscure reasons, leaving the user with at best an incomplete document and no clear recovery options. Although PostScript is supposed to be portable, in practice people often experience difficulty printing PostScript files—particularly on different computer platforms. When a document crashes, it does not necessarily mean that the file is corrupt. Just as subtle differences occur among compilers for high-level languages like C++, the behavior of PostScript interpreters can differ in unpredictable ways. Life was simpler in the early days, when there was one level of Postscript and a small set of different interpreters. Now, with the proliferation of PostScript support, any laxity in the code an application generates may not surface locally, but instead cause unpredictable problems at a different time on a computer far away.

Trouble often surfaces as a *stack overflow* or *stack underflow* error. Overflow means that the available memory has been exceeded on the particular machine that is executing the document. Underflow occurs when an insufficient number of elements are left on the stack to satisfy the operator currently being executed. For example, if the stack contains a single value when the *add* operator is issued, a stack underflow error occurs. Other complications can be triggered by conflicting definitions of what a “new-line” character means on a given operating system—something we have already encountered with plain text files. Although PostScript classes both the carriage-return and line-feed characters (*CR* and *LF* in Table 4.1) as white space (along with “tab” and “space,” *HT* and *SPAC*, respectively), not all interpreters honor this.

PostScript versions of word-processor files are invariably far larger than the native format, particularly when they include uncompressed images. Level 1 does not explicitly provide compressed data formats. However, PostScript is a programming language and so this ability can be programmed in. A document can incorporate compressed data so long as it also includes a decompression routine that is called whenever the compressed data is handled. This keeps image data compact, yet retains Level 1 compatibility. Drawbacks are that every document duplicates the decompression program, and decompression is slow because it is performed by an interpreted program rather than a precompiled

one. These are not usually serious. When the document is displayed online, only the current page's images need be decompressed, and when it is printed, decompression is quick compared with the physical printing time. Note that PostScript based digital library repositories commonly include Level 1 legacy documents.

The ideas behind PostScript make it attractive for digital libraries. However, there are caveats. First, it was not designed for online display. Second, if advantage is taken of additions and upgrades, such as those embodied in comments, encapsulated PostScript, and higher levels of PostScript, digital library users must upgrade their viewing software accordingly (or, more likely, some users will encounter mysterious errors when viewing certain documents). Third, extracting text for indexing purposes is not trivial, and the problem is compounded by international character sets and creative typography.

Portable Document Format: PDF

PDF is a page description language that arose out of PostScript and addresses its shortcomings. It has precisely the same imaging model. Page-based, it paints sequences of graphical primitives, modified by transformations and clipping. It has the same graphical shapes—lines, curves, text, and sampled images. Again, text and images receive special attention, as befits their leading role in documents. The concept of current path, stroked or filled, also recurs. PDF is device independent and expressed in ASCII.

There are two major differences between PDF and PostScript. First, PDF is not a full-scale programming language. (In reality, as we have seen, this feature limits PostScript's portability.) Gone are procedures, variables, and control structures. Features like compression and encryption are built in—there is no opportunity to program them. Second, PDF includes new features for interactive display. The overall file structure is imposed, rather than being embodied in document-structuring conventions as with PostScript. This provides random access to pages, hierarchically structured content, and navigation within a document. Also, hyperlinks are supported.

There are many less significant differences. Operators are still postfix—that is, they come after their arguments—but their names are shorter and more cryptic, often only one letter, such as *S* for stroke and *f* for fill. To avoid confusion among the different conventions of different operating systems, the nature and use of white space are carefully specified. PDF files include byte offsets to other parts of the file and are always generated by software applications (rather than being written by hand as small PostScript programs occasionally are).

PDF has been through several versions since its introduction in 1993. In 1999, JavaScript support was added for greater interactivity (Version 1.3), and in 2000 (Version 1.5) support for JPEG 2000 was added (see Section 5.3). Adobe has released successive free versions of its Reader application (and associated browser plug-ins) to enable users to access the enhanced features. Although other readers are available, most users view PDF documents in Adobe applications. In 2008, PDF (Version 1.7) became an ISO international standard.

Different subsets of PDF have been defined to target specific user groups. Of particular relevance to digital librarians is the subset aimed at long-term archiving, known as PDF/A. PDF/A documents are

intended to be self-contained and static. They require all fonts to be embedded, no use of external resources, no JavaScript actions, and device-independent color spaces. The first archival standard, PDF/A-1, is based on PDF version 1.4. The majority of applications that generate PDF provide options that allow users to save their document in whatever version they require. As with other evolving file formats, there is a trade-off between using the latest new features and ensuring that your documents can be widely read.

Inside a PDF file

Figure 4.18d is a PDF file that produces an exact replica of Figure 4.18a. The first line encodes the type and version as a comment, in the same way that PostScript does. Five lines near the end of the first column specify the text *Welcome* in several languages. The glyph *ä* is generated as the character `\344` in the Windows extended 8-bit character set (selected by the line starting `/Encoding` in the second column), and `Tj` is equivalent to PostScript's *show*. Beyond these similarities, the PDF syntax is far removed from its PostScript counterpart.

PDF files split into four sections: header, objects, cross-references, and trailer. The header is the first line of Figure 4.18d. The object section follows and accounts for most of the file. Here it comprises a sequence of six objects in the form `<num> <num> obj ... endobj`; these define a graph structure (explained below). Then follows the cross-reference section, with numbers (eight lines of them) that give the position of each object in the file as a byte offset from the beginning. The first line says how many entries there are; subsequent ones provide the lookup information (we expand on this later). Finally comes the trailer, which specifies the root of the graph structure, followed by the byte offset of the beginning of the cross-reference section.

The object section in Figure 4.18d defines the graph structure in Figure 4.18e. The root points to a *Catalog* object (number 1), which in turn points to a *Pages* object, which points to (in this case) a single *Page* object. The *Page* object (number 3) contains a pointer back to its parent. Its definition in Figure 4.18d also includes pointers to *Contents*, which in this case is a *Stream* object that produces the actual text, and two *Resources*, one of which (*Font*, object 6) selects a particular font and size (14-point Helvetica), while the other (*ProcSet*, object 5) is an array called the *procedure set* array that is used when the document is printed.

A rendered document is the result of traversing this network of objects. Only one of the six objects in Figure 4.18d generates actual marks on the page (object 4, *stream*). Every object has a unique numeric identifier within the file (the first of the `<num>` fields). Statements such as `5 0 R` (occurring in object 3) define references to other objects—object 5 in this case. The 0 that follows each object number is its *generation number*. Applications that allow documents to be updated incrementally alter this number when defining new versions of objects.

Object networks are hierarchical graph structures that reflect the nature of documents. Of course they are generally far more complex than the simple example in Figure 4.18e. Most documents are composed of pages; many pages have a header, the main text, and a footer; documents often include nested sections. The physical page structure and the logical section structure usually represent parallel hierarchical structures, and the object network is specifically designed for describing such structures—indeed, any number of parallel structures can be built. These object networks are quite

different from the linear interpretation sequence of PostScript programs. They save space by eliminating duplication (of headers and footers, for example). But most importantly they support the development of online reading aids that navigate around the structure and display appropriate parts of it, as described in the next subsection.

The network's root is specified in the trailer section. The cross-reference section provides random access to all objects. Objects are numbered from zero upward (some, such as object 0, may not be specified in the object section). The cross-reference section includes one line for each, giving the byte offset of its beginning, the generation number, and its status (*n* means it is in use, *f* means it is free). Object 0 is always free and has a generation number of 65,536. Each line in the cross-reference section is padded to exactly 20 bytes with leading zeros.

To render a PDF document, you start at the end. PDF files always end with `%%EOF`—otherwise they are malformed and an error is issued. The preceding `startxref` statement gives the byte offset of the cross-reference section, which shows where each object begins. The `trailer` statement specifies the root node.

The example in [Figure 4.18d](#) contains various data types: *number* (integer or real), *string* (array of unsigned 8-bit values), *name*, *array*, *dictionary*, and *stream*. All but the last have their origin in PostScript. A dictionary is delimited by double angle brackets, `<< . . . >>`—a notational convenience that was introduced in PostScript Level 2. The *stream* type specifies a “raw” data section delimited by `stream ... endstream`. It includes a dictionary (delimited by angle brackets in object 4 of [Figure 4.18d](#)) that contains associated elements. The preceding `/Length` gives the length of the raw data, 118 bytes. Optional elements that perform processing operations on the stream may also be included—`/Filter`, for example, specifies how to decode it.

PDF has types for *Boolean*, *date*, and specialized composite types such as *rectangle*—an array of four numbers. There is a *text* type that contains 16-bit unsigned values that can be used for Unicode text (the UTF-16 variant described in Chapter 8), although non-Unicode extensions are also supported.

Features of PDF

The PDF object network supports a variety of different browsing features. [Figure 4.21](#) shows a document—which is in fact the language reference manual—displayed using the Acrobat PDF reader. The navigation panel on the left presents a hierarchical structure of section headings known as *bookmarks*, which the user can expand and contract at will and use to bring up particular sections of the document in the main panel. This simply corresponds to displaying different parts of the object network tree illustrated in [Figure 4.18e](#), at different levels of detail. Bookmarks are implemented using the PDF object type *Outline*.

Thumbnail pictures of each page can also be included in this panel. These images can be embedded in the PDF file at the time it is created, by creating new objects and linking them into the network. Some PDF readers are capable of generating thumbnail images on the fly even if they are not explicitly included in the PDF file. Hyperlinks can be placed in the main text so that you can jump from one document to another. For each navigational feature, corresponding objects

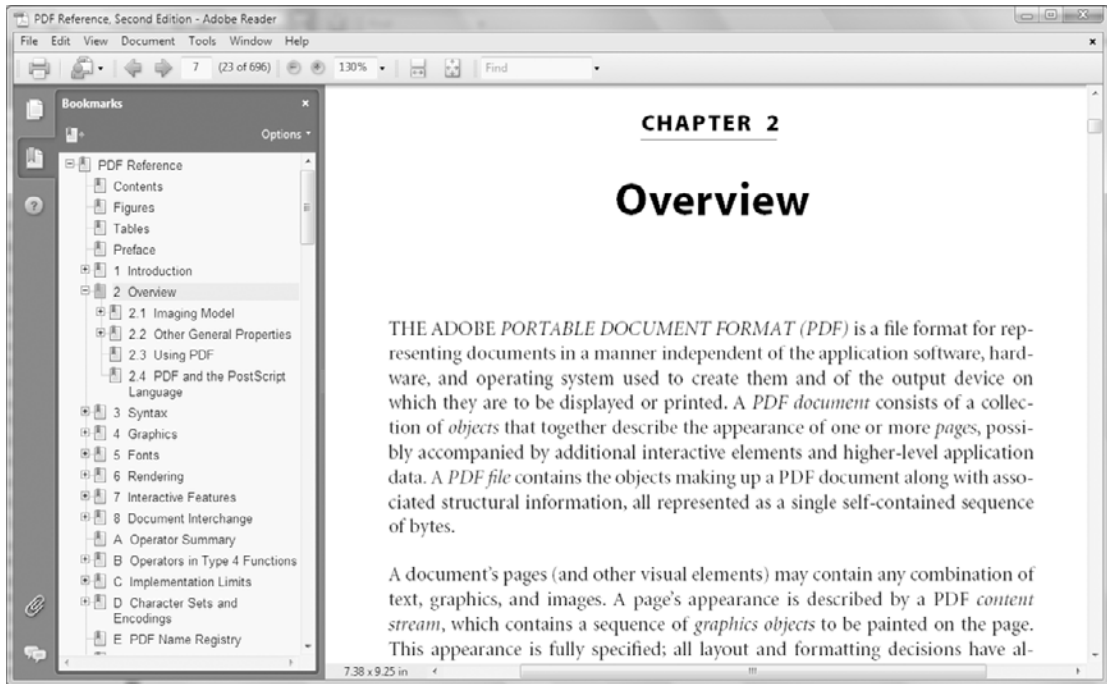


Figure 4.21: Reading a bookmark-enabled PDF document with Acrobat

must appear in the PDF network, such as the *Outline* objects mentioned earlier that represent bookmarks.

PDF has a *searchable image* option that is particularly relevant to collections derived from paper documents. Using it, invisible characters can be overlaid on top of an image. Highlighting and searching operations utilize the hidden information, but the visual appearance is that of the image. Using this option, a PDF document can comprise the original scanned page, backed up by text generated by optical character recognition. Errors in the text do not mar the document's appearance at all. The overall result combines the accuracy of image display with the flexibility of textual operations such as searching and highlighting. In terms of implementation, PDF files containing searchable images are typically generated as an output option by OCR programs (see [Section 4.2](#)). They specify each entire page as a single image, linked into the object network in such a way that it is displayed as a background to the text of the page.

There are many other interactive features. PDF provides a means of annotation that includes video and audio as well as text. Actions can be specified that launch an application. Forms can be defined for gathering fielded information. PDF has moved a long way from its origins in document printing, and today its interactive capabilities rival those of HTML.

Compression is an integral part of the language and is more convenient to use than the piecemeal development found in PostScript. It can be applied to individual stream components and helps

reduce overall storage requirements and minimize download times—important factors for a digital library.

Linearized PDF

The regular PDF file structure makes it impossible to display the opening pages of documents until the complete file has been received. Even with compression, large documents can take a long time to arrive. *Linearization* is an extension that allows parts of the document to be shown before downloading finishes. Linearized PDF documents obey rules governing object order but include more than one cross-reference section.

The integrity of the PDF format is maintained: any PDF viewer can display linearized documents. However, applications can take advantage of the additional information to produce pages faster. The display order can be tailored to the document—the first pages displayed are not necessarily the document's opening pages, and images can be deferred to later.

Security and PDF documents

The PDF document format has four features related to information security:

- encryption
- digital rights management
- phoning home
- redaction.

PDF files can be encrypted so that a password is needed to edit or view the contents. Two separate encryption systems are defined within PDF; it also includes a way in which third-party security schemes can be used for documents.

A separate facility is provided for PDF files to embed *digital rights management* (DRM) restrictions that can limit copying, editing, or printing. DRM restrictions provide only limited security, however, because they depend on the reader software to obey them. Alternatively, if you want to print a document that does not allow printing, you could use a screen-capture tool to capture the page images and print them. Of course, the resolution will suffer, but there are tools that convert PDF documents to high-resolution images, so this need not be a problem. If you need a fresh copy of the PDF document, just OCR the images.

Like HTML files, PDF files can submit information to a Web server. This facility could be used to make documents “phone home” when they are opened, and report the network (IP) address of the reader's computer. Many PDF readers will notify the user via a dialogue box that the document's supplier is auditing usage of the file and offer the option of withdrawing or continuing.

Redaction means removing information from documents. In the old days, secure redaction was achieved by physically cutting out parts of the text with scissors or knife and photocopying it against a black background. A similar effect can be achieved less securely using a black marker to strike through the text. However, as many people have discovered to their cost, covering up information in an electronic file is not the same as deleting it.

Whereas redacting paper documents is safe and easy, PDF files have created a trap for the unwary. The graphical tools available in Adobe Acrobat can be used to make it appear as though material has been redacted when in fact it has not, because the text remains in the PDF file and can still be extracted. For example, you might use a highlighter tool as a black marker, or use a rectangle tool to cover text. But these tools annotate; they do not redact. There have been several incidents where organizations have tried, and failed, to redact information in PDF files. For instance, in a 2008 legal case involving Facebook, some settlement details were kept confidential—the press were barred from the courtroom. However, when the improperly redacted PDF transcript was released, a simple copy-and-paste operation revealed the hidden text.

One sure way of redaction is to print out the document, redact the paper version, and scan it in again. Alternatively, you could visually cover the text using graphical tools, convert the PDF file to TIFF images (see Section 5.3), and convert these back to PDF. Finally, Adobe's Acrobat Professional program allows true redaction on the PDF form of the document.

PDF and PostScript

PDF is a sophisticated document description language that was developed by Adobe as a successor to PostScript. It addresses various serious deficiencies that had arisen with PostScript, principally lack of portability. While PostScript is a programming language, PDF is a format, and this bestows the advantage of increased robustness in rendering. Also, PostScript has a reputation for verbosity that PDF has avoided (PostScript now incorporates compression, but not all software uses it). Another feature of PDF is that metadata can be included in PDF files using the Extensible Metadata Platform, XMP (see Section 6.3).

PDF incorporates additional features that support online display. Its design draws on expertise that ranges from traditional printing to hypertext and structured document display. It is a complex format that presents challenging programming problems. However, a wide selection of software tools is readily available. There are utilities that convert between PostScript and PDF. Because they share the same imaging model, the documents' printed forms are equivalent. But PDF is not a full programming language, so when converting PostScript to it, loops and other constructs must be explicitly unwound. In PostScript, PDF's interactive features are lost.

Today, PDF is the format of choice for presenting finished documents online. But PostScript is pervasive. Any application that can print a document can save it as a PostScript file, whereas standard desktop environments sometimes lack software to generate PDF. From a digital library perspective, collections (for example, CiteSeer for Scientific publications) frequently contain a mixture of PostScript and PDF documents. The problems of extracting text for indexing purposes are similar and can be solved in the same way. Some software viewers can display both formats.

4.6 Word-Processor Documents

When word processors store documents, they do so in ways that are specifically designed to support editing. Microsoft Word—currently a leading product—is a good example. Three different styles of document format are associated with Word: Rich Text Format (RTF), a widely published specifica-

tion dating from 1987; a proprietary internal format that we call simply *native Word*, which has evolved over many years; and an XML-based format called Microsoft Office Open XML, which is planned for 2010. We also describe the Open Document format, which is based on XML, XSL, and associated standards, and does not differ markedly from Microsoft's product. We end this discussion by describing an example of a completely different style of document description language, LaTeX, which is widely used in the scientific and mathematical community. LaTeX is very flexible and is capable of producing documents of excellent quality; however, it has the reputation of being difficult to learn and unsuitable for casual use.

RTF is designed to allow word-processor documents to be transferred between applications. Like PostScript and PDF, it uses ASCII text to describe page-based documents that contain a mixture of formatted text and graphics. Unlike them, it is specifically designed to support the editing features we have come to expect in word processors. For example, when Word reads an RTF document generated by WordPerfect or OpenOffice (or vice versa), the file must contain enough information to allow the program to edit the text, change the typesetting, and adjust the pictures and tables. This contrasts with PostScript, where the typography of the document might as well be engraved on a Chinese stone tablet. PDF, as we have seen, supports limited forms of editing—adding annotations or page numbers, for example—but is not designed to have anything approaching the flexibility of RTF.

Many online documents are in native Word format, a binary format that is more compact than RTF, and thus yields faster download and display times. Native Word also supports a wider range of features and is tightly coupled with Internet Explorer, Microsoft's Web browser, so that a Web-based digital library using Word documents can provide a seamless interface. But native Word has disadvantages. Non-Microsoft communities may be locked out of digital libraries unless other formats are offered. Although documents can be converted to forms like HTML using scriptable utilities, Word's proprietary nature makes this challenging—and it is hard to keep up to date. Even Microsoft products sometimes can't read Word documents properly; indeed, opening a file in a version of Word other than the one with which it was created can cause incorrect display of the document. Native Word is really a family of formats rather than a single one and it has nasty legacy problems.

Rendering documents in XML ensures—at least in theory—that they can be read even when the software that created them is not available, provided that details of the format have been published. Microsoft has been working on XML versions of its internal document formats since before 2000, and a new file format for Word was announced in 2002. In 2004, the European Union recommended that Microsoft standardize this, and the following year Microsoft announced it would do so. The result is Office Open XML (OOXML), which was approved as an ISO international standard in 2008. At the time of writing, Microsoft Office 14 (the working title of the next version) has been billed as the first version to support this standard. It also appeared that a service pack would be released to enable Word 2007 to read and write ISO standard OOXML files.

The Open Document Format for Office Applications (ODF) is another standard that represents word-processor documents in XML format. It was created under the auspices of the Organization for the Advancement of Structured Information Standards and is supported by several office productivity tools, most notably the open-source OpenOffice suite. Both ODF and OOXML are billed as a good

solution for long-term document presentation, and in fact the differences between them are not large. However, the debate between their proponents is heated.

Rich Text Format: RTF

Figure 4.22a recasts the *Welcome* example of Figure 4.18 in minimal RTF form. It renders the same text in the same font and size as the PostScript and PDF versions, although it relies on defaults for such things as margins, line spacing, and foreground and background colors.

RTF uses the backslash (\) to denote the start of formatting commands. Commands contain letters only, so when a number (positive or negative) occurs, it is interpreted as a command parameter—thus `\yr2001` invokes the `\yr` command with the value 2001. The command name can also be delimited by a single space, and any symbols that follow—even subsequent spaces—are part of the parameter. For example, `{\title Welcome example}` is a `\title` command with the parameter *Welcome example*.

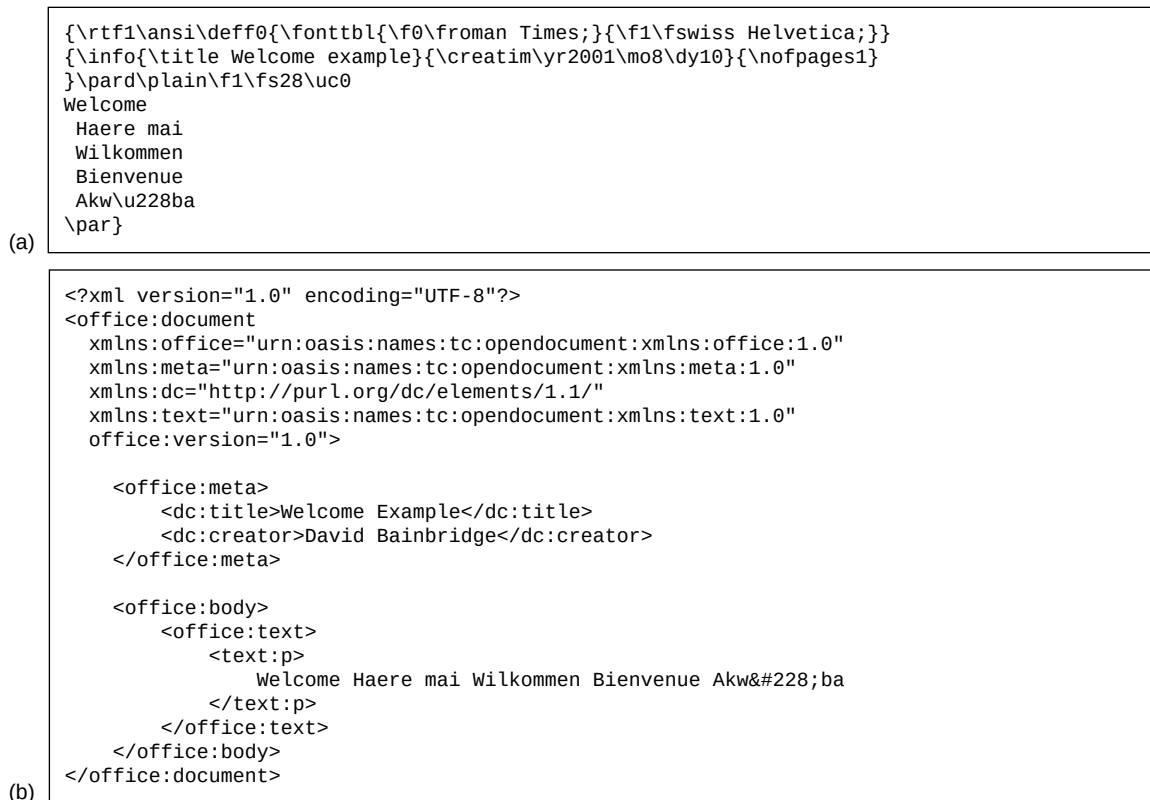


Figure 4.22: More ways of producing the document of Figure 4.18a: (a) RTF specification; (b) OpenDocument

Braces {...} group together logical units, which can themselves contain further groups. This allows hierarchical structure and permits the effects of formatting instructions to be lexically scoped. An inheritance mechanism is used. For example, if an instruction is not explicitly specified at the current level of the hierarchy, a value that is specified at a higher level will be used instead.

Line 1 of [Figure 4.22a](#) gives an RTF header and specifies the character encoding (ANSI 7-bit ASCII), default font number (0), and a series of fonts that can be used in the document's body. The remaining lines represent the document's content, including some basic metadata. On line 3, in preparation for generating text, `\pard` sets the paragraph mode to its default, while `\plain` initializes the font character properties. Next, `\f1` makes entry 1 in the font table—which was set to Helvetica in the header—the currently active font. This overrides the default, set up in our example to be font entry 0 (Times Roman). Following this, the command `\fs28`—whose units are measured in half points—sets the character size to 14 points.

Text that appears in the body of an RTF file but is not a command parameter is part of the document content and is rendered accordingly. Thus lines 4 through 8 produce the greeting in several languages. Characters outside the 7-bit ASCII range are accessed using backslash commands. Unicode is specified by `\u`: here we see it used to specify the decimal value 228, which is LATIN SMALL LETTER A WITH DIAERESIS, the fourth letter of *Akwäba*.

This is a small example. Real documents have headers with more controlling parameters, and the body is far larger. Even so, this example is enough to illustrate that RTF, unlike PostScript, is not intended to be laid out visually. Rather, it is designed to make it easy to write software tools that parse document files quickly and efficiently.

RTF has evolved through many revisions—over the years its specification has grown from 34 pages to over 240. Additions are backward-compatible to avoid disturbing existing files. In [Figure 4.22a](#)'s opening line, the numeric parameter `\rtf1` gives the version number, 1. The format has grown rapidly because, as well as keeping up with developments like Unicode in the print world, it must support an ever-expanding set of word-processor features, a trend that continues.

Basic types

Now we flesh out some of the details. While RTF's syntax has not changed since its inception, the command repertoire continues to grow. There are five basic types of command: *flag*, *toggle*, *value*, *destination*, and *symbol*.

A *flag* command has no argument. (If present, arguments are ignored.) One example is `\box`, which generates a border around the current paragraph; another is `\pard`, which—as we have seen—sets the paragraph mode to its default. A *toggle* command has two states. No argument (or any nonzero value) turns it on; zero turns it off. For example, `\b` and `\b0` switch boldface on and off, respectively. A *value* command sets a variable to the value of the argument. The `\deff0` in [Figure 4.22a](#) is a value command that sets the default font to entry zero in the font table.

A *destination* command has a text parameter. That text may be used elsewhere, at a different destination (hence the command's name)—or not at all. For example, text given to the `\footnote` command appears at the bottom of the page; the argument supplied to `\author` defines metadata that does not

actually appear in the document. Destination commands must be grouped in braces with their parameter—which might itself be a group. Both commands specified in `{\info{\title Welcome example}}` are destination commands.

A *symbol* command represents a single character. For instance, `\bullet` generates the bullet symbol (•), and `\{` and `\}` produce braces, escaping their special grouping property in RTF.

Backward compatibility

An important symbol command that was built in from the beginning is `\*`. Placed in front of any destination command, it signals that if the command is unrecognized it should be ignored. The aim is to maintain backward compatibility with old RTF applications.

For instance, there was no Unicode when RTF was born. An old application would choke on the *Welcome* example of [Figure 4.22a](#) because the `\u` command is a recent addition. In fact it would ignore it, producing *Akwba*—not a good approximation.

The `\*` command provides a better solution. As well as `\u`, two further new commands are added for Unicode support. Rather than generating *Akwäba* by writing *Akw\u228ba*—which works correctly if Unicode support is present but produces *Akwba* otherwise—one instead writes

```
{\upr{Akwaba}}{\*\ud{Akw\u228ba}}}
```

The actions performed by the two new commands `\upr` and `\ud` are very simple, but before we reveal what they are, consider the effect of this command sequence on an older RTF reader that does not know about them. Unknown commands are ignored but their text arguments are printed, so when the reader works its way through the two destination commands, the first generates the text *Akwaba* while the second is ignored because it starts with `\*`. This text is a far more satisfactory approximation than *Akwba*. Now consider the action of a reader that knows how to process these directives. The first directive, `\upr`, ignores its first argument and processes the second one. The second directive, `\ud`, just outputs its argument—it is really a null operator and is only present to satisfy the constraint that `\*` is followed by a destination command.

File structure

[Figure 4.23](#) shows the structure of an RTF file. Braces enclose the entire description, which is divided into two parts: header followed by body. We have already encountered some header components; there are many others. A commonly used construct is *table*, which reserves space and initializes data—the font table, for example. The *table* command takes a sequence of items—each a group in its own right, or separated using a delimiter, such as semicolon—and stores the information away so that other parts of the document can access it. A variety of techniques are deployed to retrieve the information. In a delimited list, an increasing sequence of numeric values is implied for storage, while other tables permit each item to designate its numeric label, and still others support textual labels.

The first command in the header must be `\rtf`, which encodes the version number followed by the character set used in the file. The default is ASCII, but other encoding schemes can be used. Next, font data is initialized. There are two parts: the default font number (optional) and the font table

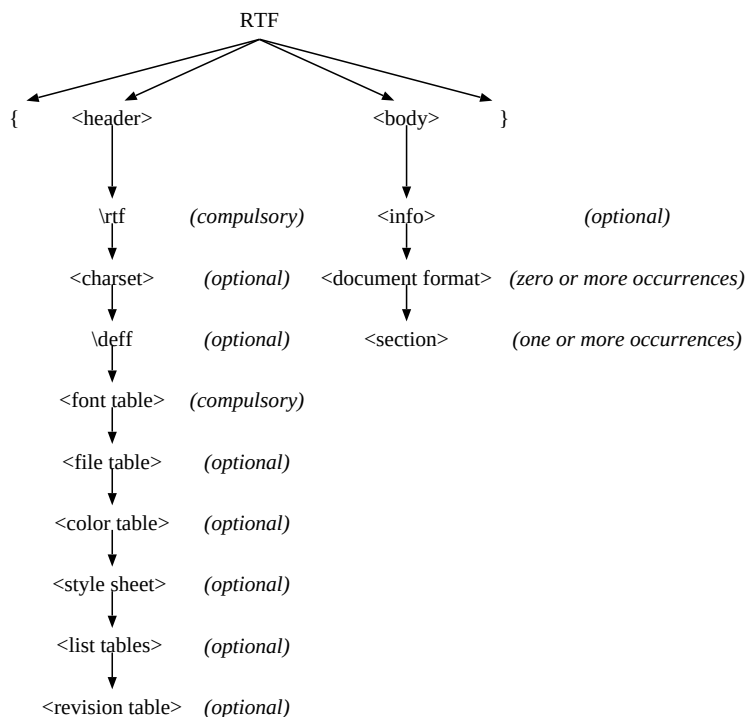


Figure 4.23: Structure of an RTF file

(mandatory). Both appear in the *Welcome* example, although the font table has many more capabilities, including the ability to embed fonts.

The remaining tables are optional. The file table is a mechanism for naming related files and is only used when the document consists of subdocuments in separate files. The color table comprises *red*, *green*, and *blue* value commands, which can then be used to select foreground and background colors through the commands `\cf1` and `\cb2`, respectively. The style sheet is also a form of table. It corresponds to the notion of *styles* in word processing. Each item specifies a collection of character, paragraph, and section formatting. Items can be labeled; they may define a new style or augment an existing one. When specified in the document body, the appropriate formatting instructions are brought to the fore. List tables provide a mechanism for bulleted and enumerated lists (which can be hierarchically nested). Revision tables provide a way of tracking revisions of a document by multiple authors.

The document body contains three parts, shown in Figure 4.23: top-level information, document formatting, and a sequence of sections (there must be at least one). It begins with an optional information group that specifies document-level metadata—in our example this was used to specify the title. There are over 20 fields, among them author, organization, keywords, subject, version number, creation time, revision time, last time printed, number of pages, and word count.

Next comes a sequence of formatting commands (also optional). Again there are dozens of possible commands: they govern such things as the direction of the text, how words are hyphenated, whether backups are made automatically, and the default tab size (measured in *twips*, an interestingly named unit that corresponds to one-twentieth of a point).

Finally, the last part of the body specifies the document text. Even here the actual text is surrounded by multiple layers of structure. First the document can be split into a series of sections, each of which consists of paragraphs (at least one). Sections correspond to section breaks inserted by an author using, for instance, Microsoft Word. Sections and paragraphs can both begin with formatting instructions. For sections, formatting instructions control such things as the number and size of columns on a page, page layout, page numbering, borders, and text flow and are followed by commands that specify headers and footers. For paragraphs, formatting instructions include tab settings, revision marks, indenting, spacing, borders, shading, text wrapping, and so forth. Eventually you get down to the actual text. Further formatting instructions can be interspersed to change such things as the active font size.

Other features

So far we have seen how RTF specifies typographic text, based around the structure of sections and paragraphs. It has many other features. Different sampled image formats are supported, including open standards like JPEG and PNG (see Section 5.3) and proprietary formats like Microsoft's Windows Metafile and Macintosh's PICT. The raw image data can be specified in hexadecimal using plain text (the default) or as raw binary—in which case care must be taken when transferring the file between operating systems (recall the discussion of FTP's new-line handling in [Section 4.1](#)).

Built into many word processors are tools that draw lines, boxes, arcs, splines, filled-in shapes, text, and other vector graphic primitives. RTF contains over 100 commands to draw, color, fill, group, and transform such shapes. The resulting shapes resemble the graphical shapes that can be described in PostScript and PDF.

Authors use annotations to add comments to a document. RTF can embed within a paragraph a destination command with two parts: a comment and an identifying label (typically used to name the person responsible for the annotation).

Field entities introduce dynamically calculated values, interactive features, and other objects requiring interpretation. They are used to embed today's date, the current page number, mathematical equations, and hyperlinks into a paragraph. They bind a field instruction command together with its most recently calculated value—which provides backup should an application fail to recognize the field. Accompanying parameters influence what information is displayed, and how. Fields allow metadata such as title and author to be associated with a document, and this information is stored in the RTF file in the form of an *\info* command. RTF uses the field mechanism to support indexes and a table of contents.

In a word-processor document, bookmarks are a means of navigation. RTF includes *begin-* and *end-bookmark* commands that mark segments of the text along with text labels, accessible through the word-processing application. Microsoft has a scheme called *object linking and embedding* (OLE) that

places information created by one application within another. For example, an Excel document can be incorporated into a Word file and still function as a spreadsheet. RTF calls such entities *objects* and provides commands that wrap the data with basic information, such as the object's width and height and whether it is linked or embedded.

Commands in the document format section control the overall formatting of footnotes (which in RTF terminology includes endnotes). The `\footnote` command is then used within paragraphs to provide a footnote mark and the accompanying text.

RTF tables are produced by commands that define cells, rows, and the table itself. Formatting commands control each component's dimensions and govern how text items are displayed—e.g., pad all cells by 20 twips, set this cell's width to 720 twips and center its text, and so on. However, there is a twist. While the other entities described earlier are embedded within a paragraph, an RTF table is a paragraph and cannot be embedded in one—this definition reflects the practice visible in Word, where inserting a table always introduces a new paragraph.

Using RTF in a digital library

When building a digital library collection from RTF documents, the format's editable nature is of minor importance. Digital libraries generally deal with completed documents—information that is ready to be shared with others. What matters is how to index the text and display the document.

To extract rudimentary text from an RTF file, simply ignore the backslash commands. The quality of the output improves if other factors, such as the character set, are taken into account. Ultimately, full-text extraction involves parsing the file. RTF was designed to be easy to parse. Three golden rules are emphasized in the specification:

- Ignore control words that you don't understand.
- Always understand `\*`.
- Remember that binary data can occur when skipping over information.

RTF files can usually be viewed on Macintosh and Windows computers. Different formats may be more appropriate for other platforms or for speedier access. For example, software is available to convert RTF documents to HTML.

Native Word formats

For much of its history the native Microsoft Word format has been proprietary and its details shrouded in mystery. Although Microsoft has published “as is” their internal technical manual for the Word 97 version, the format continued to evolve. Finally, in 2008, Microsoft made the specification publicly available. (Of course, this does not help people with legacy documents.) Native Word is primarily a binary format, and the abstract structures deployed reflect those of RTF. Documents include summary information, font information, style sheets, sections, paragraphs, headers, footers, book-marks, annotations, footnotes, embedded pictures—the list goes on. The native Word representation provides more functionality than RTF and is therefore more intricate.

A serious complication is that documents can be written to disk in Fast Save mode, which no longer preserves the order of the text. Instead, new edits are appended, and whatever program reads the file must reconstruct its current state. If this feature has been used, the header marks the file type as “complex.”

Using native Word in a digital library

To extract text from Word documents for indexing, one solution is to first convert them to RTF, whose format is better described. The Save As option in Microsoft Word does this, and the process can be automated through scripting. (Visual Basic is well suited to this task.) It may be more expeditious to deliver native Word than RTF because it is more compact. However, non-Microsoft users will typically need a more widely supported option, although this trend is shifting with the advent of OpenOffice.

Word has a Save As HTML option. While the result displays well in Microsoft’s Internet Explorer browser, it is generally less pleasing in other browsers (although it can be improved by performing certain postprocessing operations). Public domain conversion software cannot fully implement the Fast Save format because of lack of documentation and may generate all the text in the file rather than just the text in the final version. The solution is simple: switch this option off and resave all documents (using scripting).

Office Open XML: OOXML

Office Open XML is a new standard designed by Microsoft to represent, in human-readable XML form, Word and other Microsoft Office documents (spreadsheets, presentations, etc.). It was designed specifically for the features in Microsoft Office and is constrained by the need to be backward-compatible with documents created in the binary format. There has been vigorous discussion of the relative merits of OOXML and the Open Document (ODF) format (described in the next section). ODF was designed as a general office document markup language, unconstrained by the features of existing word processors. It is helpful to keep in mind the fact that these two languages were designed for different purposes and will probably be used in different ways.

Although the next section contains a brief technical account of ODF, we omit a technical description of OOXML. One reason is that these are extremely complex standards—the OOXML specification is more than 6,000 pages long, and ODF’s contains 722 pages. Instead, we give a flavor of the debate, which is often heated. Critics of OOXML complain that (for example)

- The standard is controlled by a single commercial company (Microsoft).
- It does not follow established standards for dates, graphics, formulas.
- Parts of it use non-XML formatting codes and are unreadable by XML parsers.
- Components of OOXML can be linked to Windows applications and can be understood only in a Windows context.

Proponents counter these criticisms by pointing out that naturally some OOXML elements are not supported by other software or other formats. Indeed, a project to translate documents from OOXML

to ODF (funded in part by Microsoft) found that not all aspects of Office documents could be faithfully rendered. The problem basically stems from the fact that its design requirements force OOXML to capture all features of Microsoft Office, no matter how idiosyncratic or arcane they appear to others.

On the other side, critics of ODF complain that (for example)

- It lacks standard ways of rendering macros, tables, and mathematical symbols.
- It ignores the semantics of spreadsheet formulae.
- The documentation is insufficiently detailed for vendors who seek to build their own ODF software.
- It is not well enough defined to support fully interoperable applications.

In practice, its proponents say, ODF developers will naturally look to OpenOffice for canonical implementations that clarify features like spreadsheet formulae (which seem to have become a specific bone of contention). The code is there for anyone to view, and the XML output can be trivially inspected—providing a supplement to the standard in the form of an operational implementation. Of course, this is a very different philosophy from Microsoft's.

Although there are certainly differences between the two standards, advocates on both sides usually admit that the differences are not essential—certainly not when it comes to the interoperability and preservation of textual documents. Perhaps the real issue is the potential effect of each standard in the marketplace. ODF may be specified in less detail, but it offers the flexibility to create new products, which encourages competition. OOXML is aimed at reproducing Office documents and makes it easier for other products to work with Office—but does not encourage vendors to replace it.

At any rate, OOXML will be a great improvement over the native Word formats for designers of digital libraries.

Open Document format: ODF

The Open Document format can describe word-processor documents, spreadsheets, presentations, drawings, graphics, images, charts, mathematical formulas, and documents that combine elements of any of these. They are stored in files whose extensions contain “.od” followed by a single letter indicating the document type: *.odt* for text documents, *.ods* for spreadsheets, *.odp* for presentation programs, *.odg* for graphics.

A basic ODF document is an XML file with `<document>` as its root element. (Open Document files can also take the format of a ZIP compressed archive containing a number of files and directories as described below.) Figure 4.22b shows the *Welcome* example of Figure 4.18a in minimal ODF form. It renders the same text as the PostScript, PDF, and RTF versions, although it uses the default font and size. After the obligatory XML processing application statement, the document is defined by its root element, `<office:document>`, in the *office* namespace. This begins with four namespace definitions. In practice, ODF files usually define more namespaces—including ones for Formatting Objects (called *fo*) used in Section 4.4—but these four are enough to make this rudimentary example work. The namespaces that begin with *urn* are Uniform Resource Names (see Section 7.3).

Metadata can be stored along with documents, either with a set of metadata elements that are pre-defined in Open Document or using any user-defined metadata set. We have illustrated this by including two metadata elements specified in the Dublin Core standard (see Section 6.2). The namespace beginning with *http://* is for the Dublin Core standard, and is only necessary because the example uses it to specify some metadata.

Following the metadata is the text of the document, nested inside the appropriate *office* tags. The `<text:p>` tag specifies a paragraph. As you can see, Unicode characters can be included using the ordinary `#` notation of HTML and XML.

In order to make the text appear in Helvetica font as it does in Figure 4.18a, the `<text:p>` statement near the end of Figure 4.22b needs to be augmented with a *style-name* attribute to read

```
<text:p text:style-name="Normal">
  Welcome Haere mai Willkommen Bienvenue Akw&#228;ba
</text:p>
```

To make this work, the Normal text style must be explicitly defined using a statement like

```
<style:style style:name="Normal">
  <style:text-properties style:font-name="Helvetica"/>
</style:style>
```

In reality the specifications are slightly more convoluted and verbose, which is why we have not included the details here. One cardinal advantage of XML-style document specifications over the others discussed previously is that it is very easy to create examples and study them to see what is going on.

Open Document files

Figure 4.22b shows a single file containing an entire document in the root XML tag `<office:document>`. This represents the document content, metadata, and any styles defined in the document (there are none in Figure 4.22b). It also includes any document “settings,” which are properties that are not related to the document’s content or layout, like the zoom factor or the cursor position (again, there are none here).

These four parts are usually separated by placing the components in separate XML files (*content.xml*, *meta.xml*, *settings.xml*, and *styles.xml*) instead. The single top-level `<office:document>` would be replaced by four top-level roots, one for each file:

- `<office:document-content>`
- `<office:document-meta>`
- `<office:document-styles>`
- `<office:document-settings>`

The *content* file is the most important, and carries the actual content of the document (except for binary data, like images). The *style* file contains most of the stylistic information—Open Document makes heavy use of styles for formatting and layout.

Although [Figure 4.22b](#) shows a readable XML file, ODF files are compressed into ZIP archives to reduce their size. Furthermore, an additional file (called *mimetype*) must be present in the archive, containing a single line specifying the document type—whether a textual document, spreadsheet, presentation file, or graphics. This makes the file extension (*.odt*, *.ods*, *.odp*, or *.odg*) immaterial to the format: it's only there for the user's benefit.

Formatting

Open Document has a comprehensive repertoire of formatting controls that dictate how information is displayed. Style types include:

- paragraph styles
- page styles
- character styles
- frame styles
- list styles.

There are many attributes that dictate the style of specific parts of the text, paragraphs, sections, tables, columns, lists, and fills. Characters can have their font, size, and other properties set. The vertical arrangement of paragraphs can be controlled through attributes that keep lines together and avoid widows and orphans; other attributes (such as “drop caps”) provide special formatting for parts of a paragraph.

The usual range of document structuring options is provided, including headings at different levels, numbered and unnumbered lists, numbered paragraphs, and change tracking. Section attributes can be used to control how the text is displayed. Documents can include hyperlinks, bookmarks, and references. Text fields can contain automatically generated content, and there are mechanisms for generating tables of contents, indexes, and bibliographies.

Page layout is controlled by attributes such as page size, number format, paper tray, print orientation, margins, borders, padding, shadow, background, columns, print page order, first page number, scale, table centering, maximum footnote height and separator, and many layout grid properties. Headers and footers can have defined fixed and minimum heights, margins, border line width, padding, background, shadow, and dynamic spacing.

Using ODF in a digital library

Like any well-defined XML-based standard, the Open Document format makes it easy to handle, process, and re-present documents in the way that digital libraries do. As the name implies, the standard is “open.” The natural verbosity of XML is curbed using a standard compression mechanism, ZIP, which renders the resulting files significantly smaller than other document files, such as Word's *.doc* files. Yet the information is readable and processable. To see the contents of an *.odt* file, you first decompress it, and then the data is exposed in simple text-based XML files whose content can be easily examined, modified, and processed.

The use of separate files for the content, metadata, style, and settings is designed to make it easy to process these components in different ways. Most digital libraries will simply ignore the settings. The fact that metadata resides in its own file eliminates the need to parse the entire document just to determine its metadata. The fact that style information is separate means that indexers can focus on the textual content of documents. Future digital libraries that deal only with such documents will be easier to construct and maintain, because they will not have to grapple with the complexity that is presently required to isolate such components as plain text and metadata in document formats like PostScript, PDF, and native Word. Legacy documents, unfortunately, will continue to pose problems.

Scientific documents: LaTeX

LaTeX—pronounced *la-tech* or *lay-tech*—takes a completely different approach to document representation. Word processors present users with a “what you see is what you get” interface that is specifically intended to hide the gory details of internal representation. In contrast, LaTeX documents are expressed in plain ASCII text and contain typed formatting commands: they explicitly and intentionally give the user direct access to all the internal representation details. Any text editor on any platform can be used to compose a LaTeX document. To view the formatted document, or to generate hard copy, the LaTeX program converts it to a page description language—generally PostScript, but PDF and HTML are possible too.

LaTeX is versatile, flexible, and powerful. It can generate documents of exceptionally high typographical quality. The downside, however, is an esoteric syntax that many people find unsettling and hard to learn. It is particularly good for mathematical typesetting and has been enthusiastically adopted by members of the academic, scientific, and technical communities. It is a nonproprietary system, and excellent implementations are freely available.

Figure 4.24 shows a simple example. Commands in the LaTeX source (Figure 4.24a) are prefixed by the backslash character, \. All documents have the same overall structure. They open with `\documentclass`, which specifies the document’s principal characteristics (article, report, book, etc.) and gives options, such as paper size, base font size, and whether to print single-sided or back-to-back. Then follows a preamble that gives an opportunity to set up global features before the document content begins. Here “packages” of code can be included. For example, `\usepackage{epsfig}` allows Encapsulated PostScript files, generally containing the artwork for figures, to be included.

The document content lies between `\begin{document}` and `\end{document}` commands. This `\begin ... \end` structure is used to encapsulate many structural items: tables, figures, lists, bibliography, abstract. The list is endless, because LaTeX allows users to define their own commands. Furthermore, you can wrap up useful features and publish them on Internet sites so that others can download them and access them through `\usepackage`.

As a document is written, most text is entered normally. Blank lines are used to separate paragraphs. A few characters carry special meaning and must therefore be “escaped” by a preceding backslash whenever they occur in the text; Figure 4.24 contains examples. Structural commands include `\section`, which generates an automatically numbered section heading (`\section*` omits the numbering,

while `\subsection`, `\subsubsection`, ... are used for nested headings). Formatting commands include `\emph`, which uses italics to emphasize text, and `\`, which superimposes an umlaut on the character that follows. There are hundreds more.

The last part of Figure 4.24a specifies a mathematical expression. The `\begin{displaymath}` and `\end{displaymath}` commands switch to a mode that is tuned to represent formulas, which activates additional commands specially tailored to this purpose. LaTeX contains many shortcuts—for example, math mode can alternatively be entered by using dollar signs as delimiters.

```
\documentclass[a4paper,11pt]{article}
% This is a comment
\author{I. H. Witten and D. Bainbridge}
\title{Welcome example}
\date{10 August 2001}

\begin{document}

\maketitle
\section{Introduction}

% This is another comment.

Welcome, Haere mai, Willkommen, Bienvenue, Akw\ "aba

\section{Syntax}

LaTeX syntax is a little bit like RTF. It uses the \backslash
character for special formatting commands: what you
see as the end result is certainly \emph{not} what you type. One
important difference from RTF is that it is designed to be generated
by people, not automatically generated by computer. This means that
a written file can be more liberal with its use of white space
and this does not affect the overall prose.
If you really need extra spaces you need to do it \ \ like \ \ this.

Special symbols include: \{ \} \% \_ \# \&.
Speech marks are done '' like this''.

A blank line is used to separate paragraphs. It supports all the
usual document structures:
\begin{itemize}
\item bullet point list
\item enumerated list
\item tables and figures
\item drawn graphics
\item \ldots
\end{itemize}

In particular latex has a powerful maths mode capable of expressing
complex equations. A rudimentary example is:
\begin{displaymath}
x \geq \sum_{i=0}^{\infty} \frac{1}{i^2 \pi}
\end{displaymath}

\end{document}
```

(a)

Figure 4.24: (a) LaTeX source document;

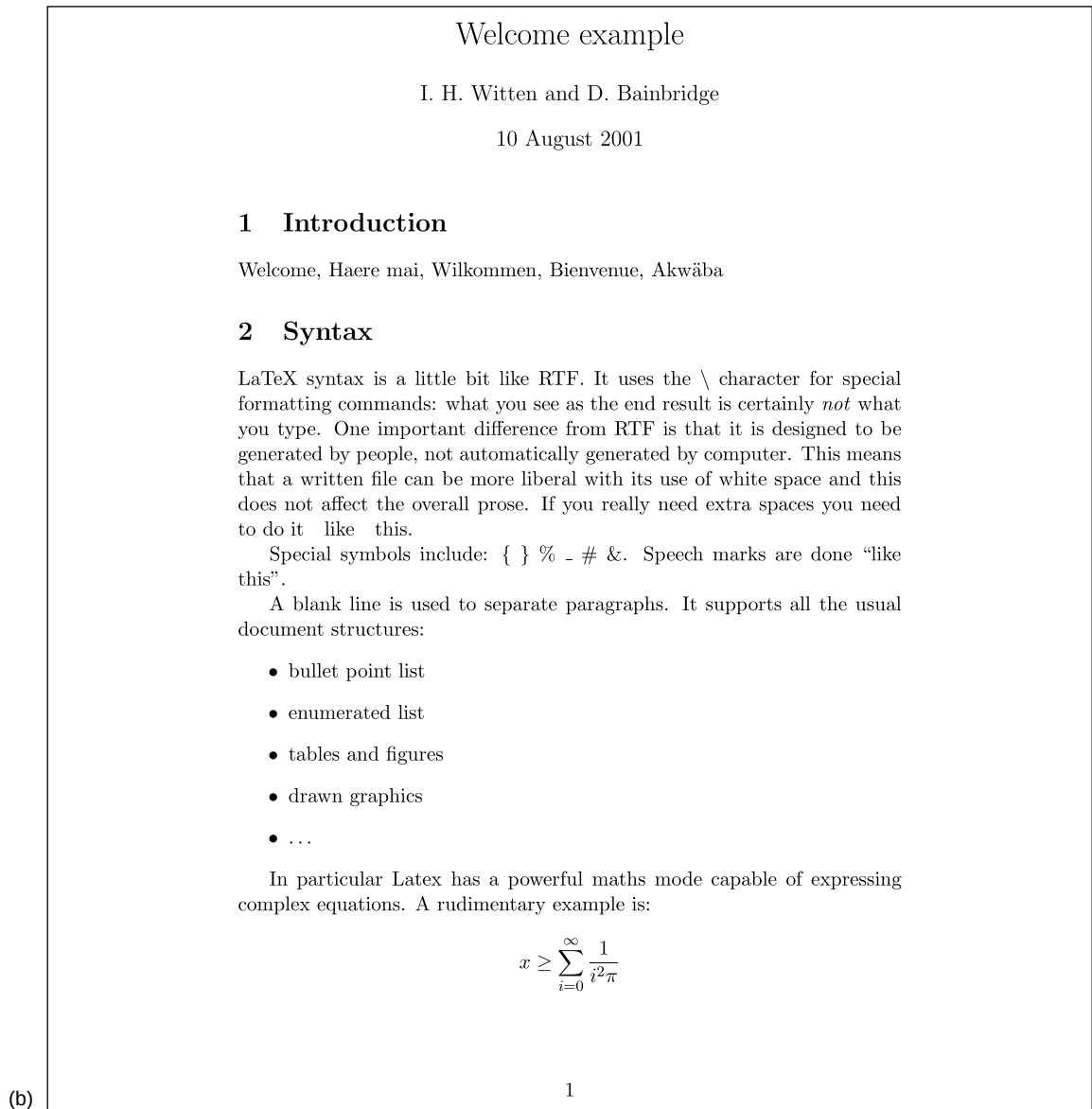


Figure 4.24, cont'd: (b) printed result

Using LaTeX in a digital library

LaTeX is a popular source format for collections of mathematical and scientific documents. Of course these documents can be converted to PostScript or PDF and handled in this form instead—which allows them to be mixed with documents produced by other means. However, this lowest-common-denominator approach loses structural and metadata information. In the case of LaTeX, such information is signaled by commands for title, abstract, nested section headings, and so on.

If, on the other hand, the source documents are obtained in LaTeX form and parsed to extract structural and metadata information, the digital library collection will be richer and provide its users with more support. It is easy to parse LaTeX to identify plain text, commands and their arguments, and the document's block structure.

However, there are two problems. The first is that documents no longer occupy a single file—they use external files such as the “packages” mentioned earlier—and even the document content can be split over several files if desired. In practice it can be surprisingly difficult to obtain the exact set of supporting files that were intended to be used with a particular document. Experience with writing LaTeX documents is necessary to understand which files need copying and, in the case of extra packages, where they might be installed.

The second problem is that LaTeX is highly customizable, and different authors adapt standard commands and invent new ones as they see fit. This makes it difficult to know in advance which commands to seek to extract standard metadata. However, new commands in LaTeX are composites of existing ones, and one solution is to expand all commands to use only built-in basic features.

4.7 Other Documents

Many other document types might be included in a digital library. Prominent among them are multimedia documents, which are discussed in the next chapter, but some other kinds of predominantly textual documents are worth a brief mention.

Spreadsheets and presentation files

We have mentioned spreadsheets and presentation files (such as PowerPoint) when discussing Office Open XML and the Open Document format. Both spreadsheets and presentation files have traditionally been encoded in proprietary binary file formats, like native Word documents. They may be presented to the user either in their native form or as PDF image files (or both). Of course, this presentation loses much information, notably the dynamic functionality of a spreadsheet involving formulas and the dynamic aspects of a presentation.

In order to index such files, the text can be extracted using the same procedure as for native-format text documents: use a Save As option to generate a more text-friendly form, such as ASCII (using the CSV or comma-separated values format) or XML for spreadsheets, and HTML or PDF for presentations. In future, open XML-based formats, such as OOXML or ODF (which, as we discuss above, apply to these files as well as to textual documents), will make life much easier.

E-mail

E-mail documents are also candidates for inclusion in digital libraries. Early in the history of international computer networks, there were multiple e-mail clients having various incompatible formats. Seeking interoperability, the U.S. Department of Defense funded efforts to create standards. The result is a set of international standards for e-mail and e-mail extensions called Multipurpose Internet Mail Extensions (MIME). This is the *de facto* standard on the Internet.

However, corporate e-mail systems often use their own internal format and communicate with servers using a vendor-specific, proprietary protocol. The servers act as gateways for sending and receiving messages over the Internet, which involves undertaking any necessary reformatting. For mail sent and received within a single company, the entire transaction may take place within the corporate system.

Mail is not usually sent directly to a digital library but is imported from wherever it happens to be stored. This can be the user's e-mail client, or their server, or both places. There are two common standard formats for mailboxes (*Maildir* and *mbox*), but several prominent clients use their own proprietary format and conversion software is needed to transfer mail between them—or to ingest it into digital libraries.

Internet e-mail messages have a header and body, separated by a blank line. The former contains metadata and is structured into fields, such as sender, receiver, date, title, and other information. It also includes the clock time and time zone, which together define the actual time the message was sent. The body contains the message content in the form of unstructured text, and sometimes ends with a signature block.

E-mails often contain attachments, which are files that are sent along with the message, encoded as part of the message to which they are attached. In the Internet MIME format, messages and their attachments are sent as a single multipart message, using an encoding scheme (called “base64”) for non-text attachments that represents binary information as printable ASCII.

There are many problems with e-mail as it is used today. For example, a standard way to quote text is to start each line with the “>” character, possibly followed by a space. Unfortunately e-mail readers usually wrap lines to fit the screen size, or to some predetermined maximum length. Quoting makes the lines overflow. Figure 4.25 shows a message that has been quoted several times. The mailer has tried to wrap the lines, but it has put only a single quote on the continuation lines, not realizing that the text being wrapped has four levels of quotes. Messages become unintelligible after a few cycles of such mutilation.

4.8 Notes and Sources

The ASCII code is central to many pieces of hardware and software, and the code table turns up everywhere in reference material, printed manuals, online help pages, Web sites, and even this book! At its inception, few could have predicted how widely it would spread. Of historical interest is the ANSI (American National Standards Institute, 1968) standard, a version of which was published (prior to final ratification) in *Communications of the ACM* (Gorn et al., 1963).

The principal aim of ASCII was to unify the coding of numeric and textual information—the digits and the Roman alphabet—between different kinds of computer equipment. However, decisions about other symbols have had a subtle yet profound influence. In the 1960s few keyboards had a backslash key, yet so ingrained is its use today to convey special meanings—such as a directory separator and to protect the meaning of certain characters in certain contexts—that computing would be almost inconceivable without it. It is here that the concept of *escaping* appeared for the first time.

```

> > Subject: australian
> > Date: Tue, 1 Apr 2008 19:59:42 EST
> >
> > >> An Australian had been drinking at a pub all night. The bartender
> > >> finally said that the bar is closing. So the Australian stood up
> > >>
> > to
> > >> leave and fell flat on his face. He tried to stand one more
> > time;
> > >> same result. He figured he'll crawl outside and get some fresh
> > air
> > and
> > >> maybe that will sober him up.
> > >>
> > >> Once outside he stood up and fell flat on his face. So he
> > decided
> > >> to crawl the 4 blocks to his home. When he arrived at the door
> > he
> > >> stood up and again fell flat on his face. He crawled through
> > the
> > door
> > >> and into his bedroom. When he reached his bed he tried one more

```

Figure 4.25: Effect of multiple quoting on an e-mail message

The official title for Unicode is tongue-twisting and mind-boggling: The International Standard ISO/IEC 10646-1, Information Technology—Universal Multiple-Octet Coded Character Set (UCS). Chapter 8 gives more information about the Unicode standard, particularly with regard to its use for non-English languages.

Technical details of searching and indexing are presented in *Managing Gigabytes* (Witten et al., 1999), which gives a comprehensive and detailed technical account of how to index documents and to make them rapidly accessible through full-text queries. This and other relevant books on information retrieval are included in the “Notes and sources” section of Chapter 3 (Section 3.7).

High-performance OCR products are invariably commercial: we know of no public-domain systems that attain a level of performance comparable to commonly used proprietary ones. One of the most accurate free software OCR engines currently available is Tesseract (<http://code.google.com/p/tesseract-ocr/>), which was originally developed at Hewlett-Packard from 1985 to 1995, and, after lying dormant for ten years, was taken over by Google and released under the Apache open source license. Another project is Ocrad, the Gnu OCR project (www.gnu.org/software/ocrad).

The interactive OCR facilities described in [Section 4.2](#) are well exemplified by the Russian OCR program FineReader (ABBYY Software, 2000), an excellent example of a commercial OCR system. Lists of OCR vendors are easily found on the Web, as are survey articles that report the results of performance comparisons for different systems. The newsgroup for OCR questions and answers is comp.ai.doc-analysis.ocr. Price-Wilkin (2000) gives a nontechnical review of the process of creating and accessing digital image collections, including a sidebar on OCR by Kenn Dahl, the founder of a leading commercial OCR company. The OCR shop we visited in Romania is *Simple Words* (now at

www.itprovision.com), a well-organized and successful private company that specializes in high-volume work for international and nongovernment organizations.

The Māori language has fifteen sounds: the five vowels *a*, *e*, *i*, *o*, and *u*, and ten consonant sounds written *h*, *k*, *m*, *n*, *p*, *r*, *t*, *w*, *ng*, and *wh*. Thus the language is written using fifteen different letters. The first eight consonant sounds are pronounced as they are in English; the last two are digraphs pronounced as the *ng* in *singer* and the *wh* in *whale*, or as *f*. Each vowel has a short and long form, the latter being indicated by a macron, as in the word *Māori*.

The ß or *scharfes s* character in German has been the source of great controversy in recent years. In 1998 a change in the official definition of German replaced some, but not all, occurrences of ß by *ss*. However, spelling reform has proven unpopular, and the traditional spelling system is still used by some newspapers and periodicals, including the widely read daily tabloid *Bild-Zeitung* and two of Germany's most respected newspapers, the *Frankfurter Allgemeine Zeitung* and *Die Welt*.

All file formats described in this chapter are well documented, except where commercial interests prevail. There is a rich vein of online resources, as you might expect from the area and the working habits of those in it, which are easy to find using Internet classification directories and resources like the Wikipedia.

The definitive guides to PostScript and PDF are the reference manuals produced by Adobe (1999, 2000), the company responsible for these formats. They practice what they preach, putting their manuals online (in PDF format) on their Web site (www.adobe.com). A supplementary tutorial and cookbook (Adobe, 1985) gives worked examples in the PostScript language. *Ghostscript* is a software PostScript interpreter that provides a useful means of experimentation. Released under the GNU Public License, it is available for all popular platforms (www.ghostscript.com). Our description of PostScript and PDF mentions several graphics techniques, such as transformations, clipping, and spline curves, that are explained in standard textbooks on computer graphics (e.g., Foley et al., 1990). The technique for extracting plain text from PostScript files is described by Nevill-Manning et al. (1998). The PDF redaction failure involving Facebook and another social networking site, ConnectU, is described in Benetton (2009). The problem is serious enough that the U.S. National Security Agency provides guidelines on how to redact properly (National Security Agency, 2006). Background information on the archival PDF/A format is at www.pdfa.org.

An initial description of Microsoft's RTF format appears in *Microsoft Systems Journal* (Andrews, 1987), and its continued expansion is documented through the Microsoft Developers' Network. Subscribers receive updates on CD-ROM, while the same information is provided for general consumption at <http://msdn.microsoft.com>. At one stage, an internal technical document describing the native Word format was published through this outlet, but it has since been discontinued. Information about OOXML can be found here. The ODF manual and other documentation are at the Organization for the Advancement of Structured Information Standards Web site, www.oasis-open.org.

LaTeX, which is based on the TeX system invented by Knuth (1986), is described in many books, such as Lammport (1994). A useful online source is Tobias Oetiker's *Not so short introduction to LaTeX2e* at <http://tobi.oetiker.ch/lshort>. Bountiful collections of packages can be found on Internet sites like the Comprehensive TeX Archive Network (www.ctan.org).

Multimedia: More raw material

The pace and scope of change in global multimedia usage are staggering. YouTube, iTunes, and Flickr are just a few examples of how people around the world create and share material in newly connected and digitally complex ways that were unanticipated even a few years ago. And these media forms are finding their way into digital libraries.

This chapter begins by describing audio, image, and video formats, and goes on to discuss so-called “rich media,” a term coined to emphasize the composition of, and interaction with, different types of media. As shown in Chapter 4, representation formats for textual documents vary widely depending on the purpose. By comparison, media have less variation in the basic representation. An audio signal is a one-dimensional sequence of numeric values representing the evolution of amplitude over time, images are a two-dimensional grid of pixel values, and video combines the two.

The raw, quasi-analog nature of multimedia means that file size bloats quickly, and so compression schemes are typically built into the storage formats. [Section 5.1](#) briefly introduces pertinent techniques, along with commonly used data transformations for information reduction. Compressing files reduces their size by an order of magnitude in some cases, but it can have side effects that need to be considered when designing digital libraries. If resources allow, people often digitize material at high precision for archival purposes, and—either at the same time or on the fly at presentation time—generate a lower-resolution representation suitable for transmission over the Internet and display in a Web browser.

Although audio is often a component of video, we treat it separately because of its prominence in our daily lives and on the Web. [Section 5.2](#) starts with the uncompressed waveform and moves to increasingly complex compression techniques, as exemplified by particular file formats. The early formats WAV, AIFF, and AU are still in widespread use because they are easy to read and process, and they use either rudimentary compression techniques or none at all. The ubiquitous MP3, which is part of the MPEG video standard, is changing how our society stores, shares, and listens to music; it deploys far more sophisticated compression techniques. Apple designed the AAC format as a successor to MP3 for use in its iPod and iTunes products, but uptake elsewhere has been sluggish because of MP3’s phenomenal grip on the market. Ogg Vorbis, another competitor of MP3, has been specifically designed to be free of all patent restrictions—which is not the case for MP3.

[Section 5.3](#) reviews image storage. We describe the prominent formats GIF, PNG, JPEG, and SVG in detail, and touch on others such as TIFF and JPEG 2000. GIF and PNG are suitable for representing artificially produced images, such as text, computer-generated artwork, and logos. JPEG is designed for continuous-tone images like photographic portraits and landscapes. In addition to raster graphics,

SVG (an acronym for scalable vector graphics) can represent graphics (such as line art) that can be rendered without loss of accuracy at arbitrarily fine resolutions. SVG is based on concepts that closely resemble those of the page description languages covered in Chapter 4.

Section 5.4 focuses principally on MPEG video and describes the various parts that make up this suite of standards. We also provide an overview of Theora, Ogg's patent-free video counterpart, along with Apple's QuickTime, Microsoft's AVI, and Adobe's Flash Video formats.

These media formats expose the underlying interplay between signal quantization, sampling rate, and internal redundancy. As with textual documents, you have to understand how different formats work in order to appreciate their strengths and limitations. Examples? Converting images from GIF to JPEG often reduces file size but degrades picture quality irreversibly. Most audio editing programs decompress MP3 files before processing them, which can lead to degradation in quality and even introduce audible artifacts—especially when a file is edited repeatedly. This chapter will equip you with the necessary knowledge to think through such implications.

Multimedia digital library projects often involve digitization—perhaps more so than with text. For example, the starting point for many textual libraries is a scattered collection of cognate documents, already in digital form, for which some kind of unified searching and browsing facilities are desired. For multimedia, however, the starting point is often a collection of interviews or similar material, developed over many years, that has languished, barely used, on analog tapes. Publishing this material as an online digital library is a far more substantial undertaking.

When digitizing multimedia artifacts, you must set parameters that determine the resolution with which they are captured: sample rate for audio, dots per inch for images, and both for video. You must also decide how many bits to use to represent each element. Different settings are appropriate for archiving and for delivery over the Internet. What are the implications in terms of disk storage? What are the indexing options? Which formats do Web browsers support? Which browsers predominate among the target audience? The information presented in this chapter will help you evaluate the merits of various approaches.

5.1 Introducing Compression and Transforms

Compression techniques reduce the size of files without losing important information. For delivery of digital library content over the Web, smaller files translate into faster delivery, which is often a paramount consideration. When selecting a compression scheme, it may be important to take into account not only the amount of compression achieved, but also the time and memory required for compression and decompression.

Compression and prediction are two sides of the same coin. If you can predict some of the information in a file, you can compress it by removing that information; conversely, decompression is tantamount to predicting what information has been removed and re-inserting it. Compression can be *lossless*, which means that the decompressed result is exactly the same file as the original, or *lossy*, in which case the decompressed file is not guaranteed to be exactly the same as the original but includes minor distortions that, in the wider scheme of things, are often not noticeable. Lossy techniques can

achieve substantial gains in the amount of compression at the expense of a slight degradation in quality.

Only lossless compression makes sense for text representation, and that is what is incorporated into the PostScript, PDF, and Open Document formats discussed in Chapter 4. For digitized images and audio, however, lossless compression helps somewhat, but the nature of multimedia data means that not much is saved. By permitting small (usually imperceptible) distortions, lossy techniques can achieve far greater compression. Multimedia signals often include noise, which is inherently incompressible—because it’s impossible to predict. The process of lossy compression not only results in smaller files but in some cases actually *improves* the fidelity of the signal by reducing noise.

Lossy techniques can obtain remarkably compact representations for digitized images and audio—10% of the original file size or less. However, exact representations are essential in certain situations, such as certified copies of medical or legal images, and archival copies of historical documents for which the requirements of future scholars cannot be anticipated. In these cases, lossless techniques must be used.

Basic compression techniques

A group of techniques known as dictionary methods often yield good lossless compression for a fairly small investment in time and memory space. The idea is to maintain a “book of codes” for commonly occurring letter sequences and replace these sequences wherever they occur by their position in the codebook. Braille, for example, is based on this principle. When text is represented, certain sequences of two or three letters (commonly occurring combinations such as *ch* and *for*) are represented by a single code. Figure 5.1 shows a few Braille codes, including examples of compositions and contractions.

Unlike Braille, however, good computer-based compression methods build up the codebook as they go along—thereby *adapting* to the character of the text that is being processed. In one scheme, previ-

Letters and numbers	a, 1 ● ○ ○ ○ ○ ○	b, 2 ● ○ ● ○ ○ ○	c, 3 ● ● ○ ○ ○ ○	d, 4 ● ● ○ ● ○ ○	...
Context and punctuation	number follows ○ ● ○ ● ● ●	capital letter follows ○ ○ ○ ○ ○ ●	comma ○ ○ ● ○ ○ ○	semicolon ○ ○ ● ○ ● ○	...
Character pairs	ch ● ○ ○ ○ ○ ●	gh ● ○ ● ○ ○ ●	sh ● ● ○ ○ ○ ●	th ● ● ○ ● ○ ●	...
Contractions	and ● ● ● ○ ● ●	for ● ● ● ● ● ●	of ● ○ ● ● ● ●	the ○ ● ● ○ ● ●	...

Figure 5.1: A selection of example characters in Braille

ously encountered strings are encoded as their position in the codebook, and then a new entry is added to the codebook comprising this string plus the next character in the text. The codebook is initialized to contain one entry for each letter of the alphabet. When the second character of the text is transmitted, a two-letter sequence—the first two letters of the text—will be added to the codebook. Many more two-letter sequences will probably be added before the first three-letter one is. Over time, the length of sequences in the codebook grows. Eventually it may threaten to overwhelm the available space, whereupon some pruning strategy must be applied.

In another scheme, the codebook is not kept explicitly. Instead, repeat occurrences of previously seen letter sequences are encoded by a pointer back to where the previous sequence was, followed by a count indicating the number of characters in the repeated sequence. The algorithm must detect repeated substrings efficiently, and an upper bound is imposed on the size of the back pointer, restricting the range of previous characters that must be considered. Again, because the codebook is constantly evolving, the algorithm is adaptive.

Two other important kinds of lossless compression are Huffman coding and arithmetic coding. Both serve the same end: to encode frequently occurring letters or words (or other tokens) in fewer bits than letters or words that occur rarely. The principle is the same as in Morse code, where common letters such as *E* and *T* are encoded economically (with a single dot and dash, respectively) compared with infrequent ones like *X* (dash, dot, dot, dash) and *Y* (dash, dot, dash, dash). However, unlike Morse code, Huffman and arithmetic coding do not use a predetermined codebook. The Huffman technique produces a codebook based on a particular file or set of files and stores or transmits it before the coding proper begins. In arithmetic coding, the codebook is adapted continually as transmission proceeds to reflect the relative frequencies with which tokens have been seen. Both methods yield compression that is close to optimal for a given frequency distribution of the tokens. With modern processors, compression and decompression is fast enough that some file systems transparently perform lossless compression on the fly when data is written to and read from disk.

Turning to lossy compression, a simple example is *quantization*, which is the process of subdividing the possible range of a measurement into steps and assigning each one a value—for example, rounding a price to the nearest dollar, or the amplitude of an audio signal to the nearest 8 bits. The steps need not be equally spaced; in general, they are drawn up in advance and recorded in a codebook.

Quantization can also be applied to vectors. Vector quantization reduces vectors in a high-dimensional space to a finite set of values (a codebook) drawn from a lower-dimensional space. Typically, vectors for audio or images are formed by transforming the data to another domain (such as the frequency domain—see the next subsection). Reducing the number of dimensions saves space (compression) but also reduces precision (hence lossy). There is no single, universal best scheme for vector quantization: a good codebook depends on the kind of signal being compressed. What vector quantization schemes seek are mappings to a lower-dimensional space that minimize the loss of precision whilst working reliably across a wide range of input data. To achieve greater compression, data is coded in terms of indexes into the codebook rather than as raw low-dimensional vectors.

Transforms

For lossy compression, higher compression is often achieved by transforming the input signal first. Audio (a time signal) and images (spatial signals) can both be converted to the frequency domain using the Fourier transform. The original signal can be recovered by an inverse transformation. The basic procedure is lossless and achieves no compression, but tricks like reducing the level of numerical precision of the transformed version can be applied to reduce the amount of information stored. This can often be engineered to achieve substantial compression with a loss of detail that is largely imperceptible to the human ear or eye. With audio, for example, weak frequency components that lie close to some dominant frequency can be removed, or represented with less precision, without adversely affecting playback. Given enough care in the elimination step, listeners will be unable to discern any difference because of how the human auditory system works. The technique, called acoustic masking, is central to MP3 compression.

Such transforms are an integral part of all but the most rudimentary multimedia formats. Several are mentioned in this chapter: the Fourier transform, Discrete Cosine Transform, and Wavelets. The Fourier transform is the *pater familias* of the transform world. The principles it embodies (which are beyond the scope of this book) are the key to the other transforms as well. They work in similar ways but exhibit different properties—such as how they deal with discontinuities in the data. These properties can be crucial in achieving good performance across the range of input signals likely to be encountered.

Each transform has an inverse version that maps it back to the original domain, where it can be heard or viewed. Mathematically, the inverse is almost identical to the original transform, with minor adjustments (such as negating certain components or normalizing values by a particular constant), and they have the same computational cost. Expressed in one-dimensional form, the transform applies directly to audio processing. Through the algebraic device of *separability*, it can be generalized to two dimensions and applied to images. [Figure 5.2](#) shows the idea. The basic one-dimensional calculation is applied to the input data horizontally, and then the same calculation is applied vertically to the result. (The order of horizontal and vertical processing is unimportant. The same result is achieved by processing first vertically, then horizontally.)

The Fourier transform

The process of transforming a time-based signal to the frequency domain is central to many representation techniques. Here is a brief technical overview, using the audio signal in [Figure 5.3a](#) as an example. The signal contains two sine waves: one at 261 Hz, corresponding to middle C, and the other at 440 Hz, corresponding to the A above. The composite signal sounds like two notes, C and A, played together as a chord. [Figure 5.3b](#) shows the result of applying the Fourier transform, which calculates frequency components, to this signal. Informally, it works by successively trying out sine waves at ever-increasing frequencies and noting where strong matches occur (correlate) with the original signal. The x- and y-axes show frequency and amplitude, and distinct peaks corresponding to the sine waves occur at 261 Hz and 440 Hz. Starting with [Figure 5.3b](#) and calculating the inverse Fourier transform yields the original time-based signal. If the 261 Hz component were set to zero first, the inverse transform would yield a signal corresponding to the A tone alone.

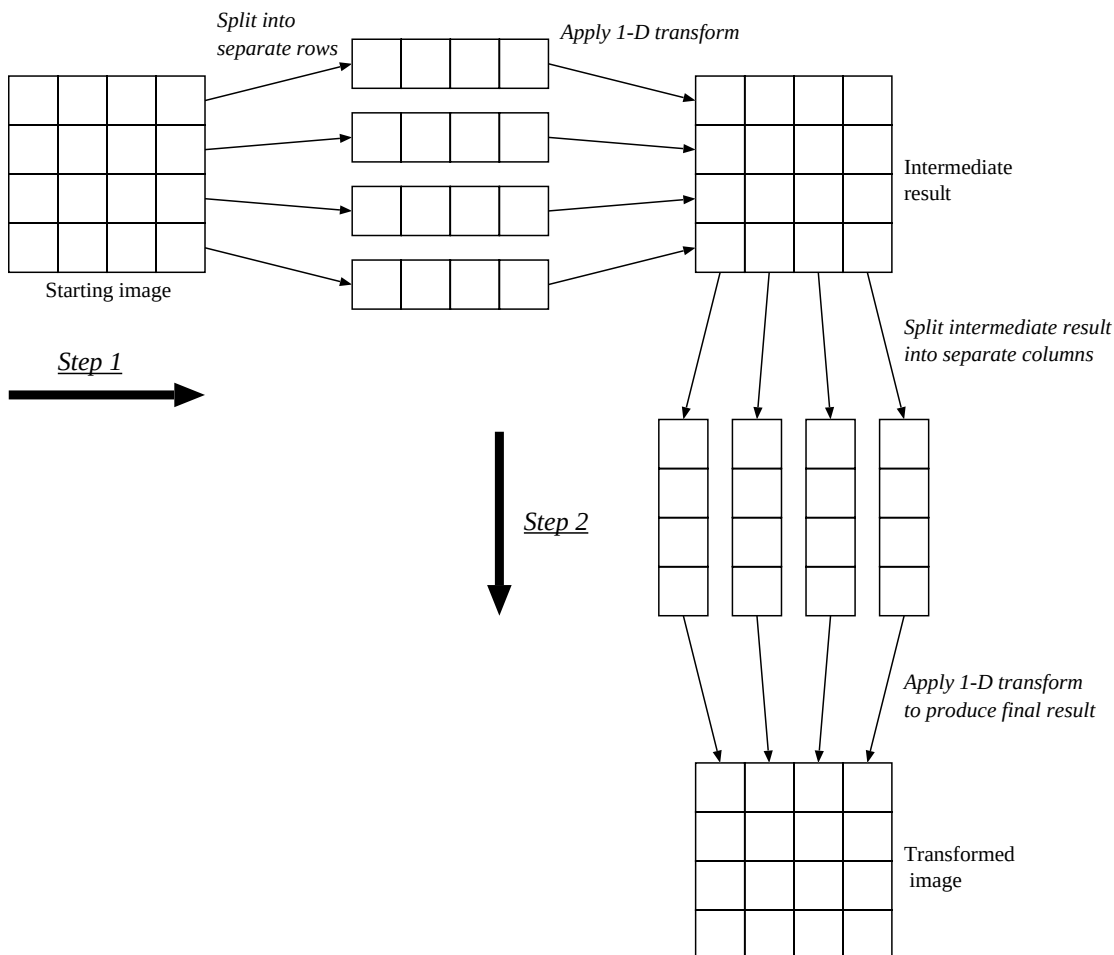


Figure 5.2: Transforming a two-dimensional image

Figure 5.4a shows a 50-ms excerpt taken from a commercial popular music CD, containing vocals and backing music. Although the signal is far more complex than our artificial example, the same principle applies: it is made up by superimposing a large number of fundamental sine waves, and the Fourier transform seeks them out. Figure 5.4b shows the result.

It is unusual to apply this sort of transform to a whole song, however. Over an entire song, most of the frequencies in the audible hearing range will occur, so this would neither be useful in terms of achieving compression, nor insightful in terms of determining what frequencies are present. Instead, a technique called *windowing* is used to subdivide the audio into shorter sections, over which the frequency spectrum does not vary much.

Windows are usually quite short. For audio, window duration is typically measured in milliseconds. A typical example is a 1024-sample window, which corresponds to around 23 ms for CD-quality audio. The value of 1024 is chosen in preference to a round number like 1000, because the algorithm

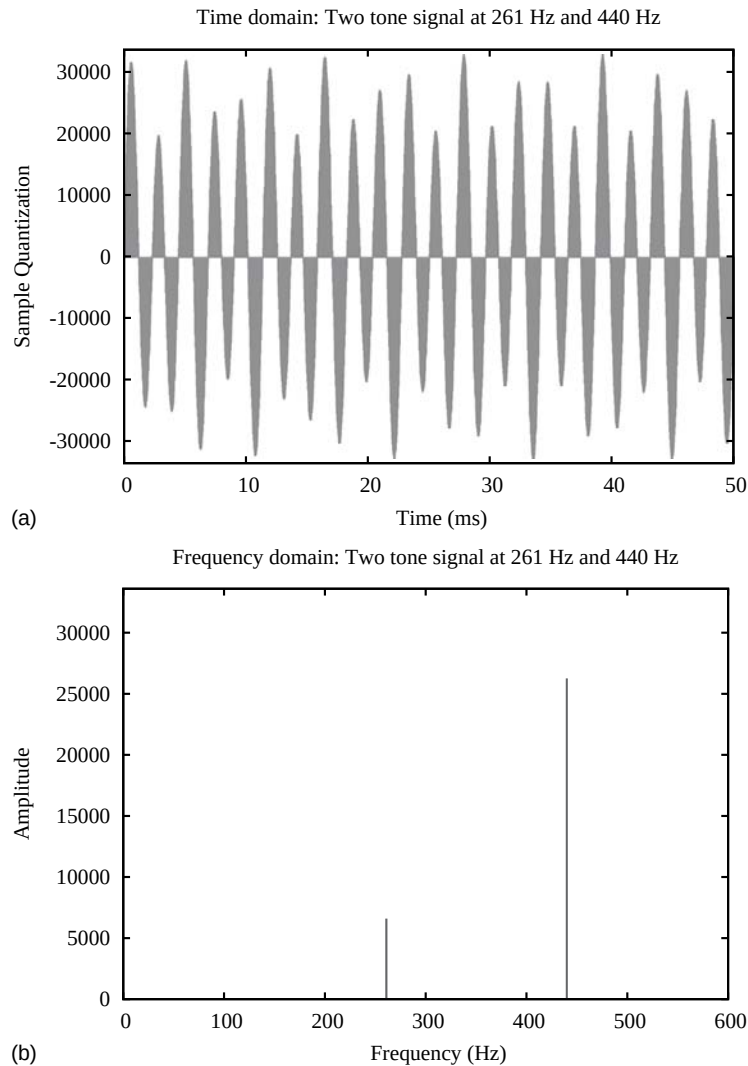


Figure 5.3: Two sine waves: (a) the composite signal; (b) result of applying the Fourier transform

used works most efficiently when the value is a power of two, such as $2^{10} = 1024$. Each section of the signal is weighted by a function that tapers off at each end and is zero outside the window. Many different window functions have been investigated. For example, a “Hamming” window is based upon a cosine calculation that peaks to one at the center of the window.

5.2 Audio

The most prevalent form of audio media we encounter in our daily lives is the compact disk, and it is so pervasive that we invariably refer to it by the acronym CD. Although a CD is recorded digitally, it

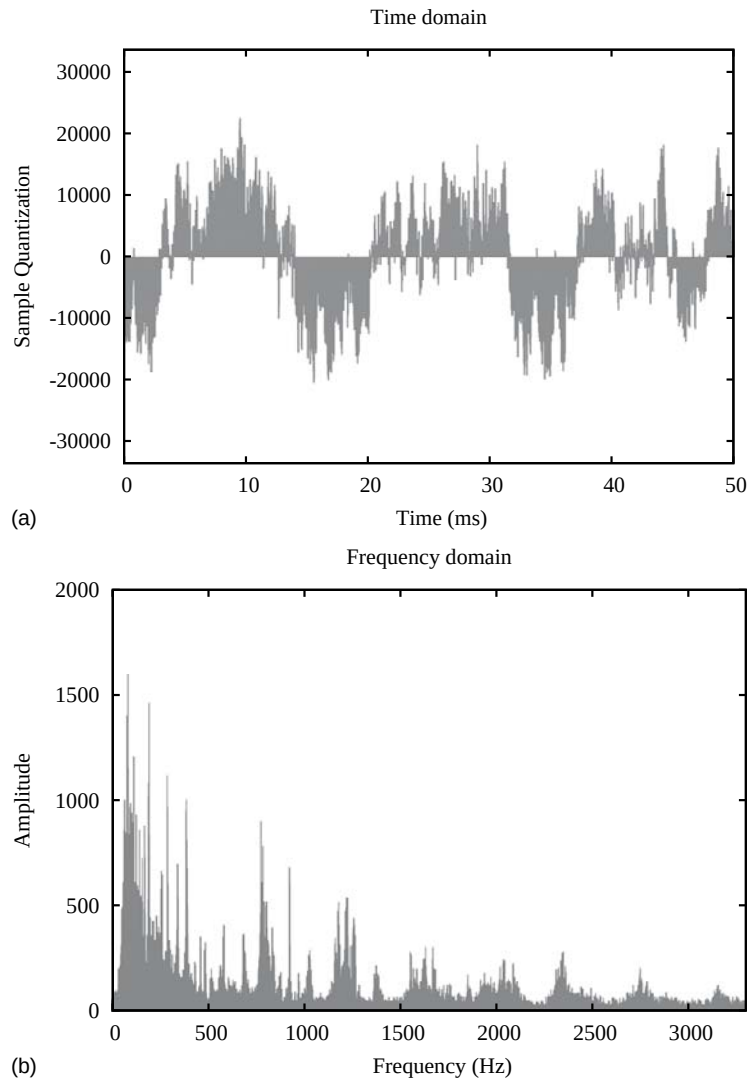


Figure 5.4: Audio from a commercial CD: (a) the signal; (b) result of applying the Fourier transform

uses no compression whatsoever—in order to make it as straightforward as possible to build physical devices that read data from the disk at the required speed and convert it to an electrical signal that can be pumped out to loudspeakers. A CD is recorded in a form called pulse code modulation.

Pulse code modulation: PCM

Figure 5.5 shows both analog and digital forms of audio. Analog recordings capture sounds directly from the world around us, using a transducer to convert changes in air pressure into another, entirely analogous, representation, such as an electrical signal. For example, some microphones employ a

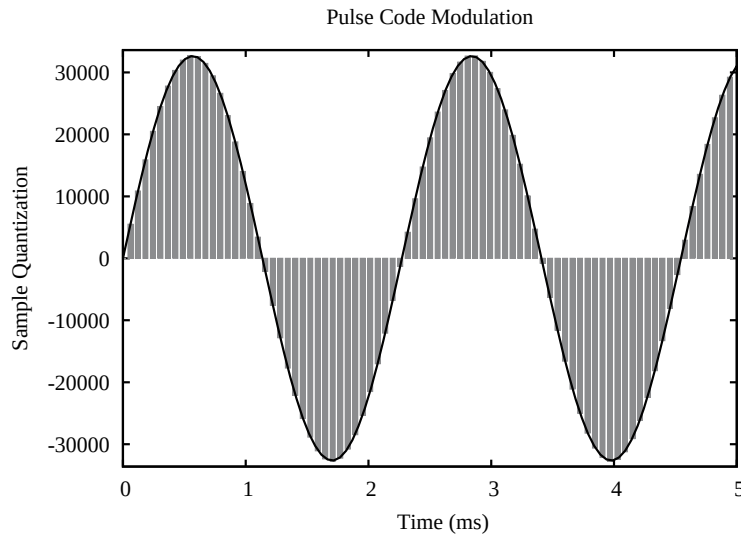


Figure 5.5: Comparison of analog and digital audio signals

vibrating cone in a magnetic field to convert sounds in the real world into electrical signals. Connect this signal to an oscilloscope whose beam scans from left to right at constant speed, and whose vertical dimension shows the analog signal at that instant of time, and put the microphone to someone's chest: the result is the classic green-line heart-beat signature seen in TV hospital dramas. Audio signals for speech and music can be displayed in the same way, but they are far less regular than a heartbeat.

When computers record audio, the analog signal is converted to digital form (and vice versa for playback). At regular intervals of time, a snapshot is taken of the value of the continuously changing signal, rounding it to the closest numeric value the digital system can represent—thus quantizing in both time and amplitude. [Figure 5.5](#) shows the distinction: the analog signal is represented by a smooth, continuously changing curve; the digital one by a series of regularly spaced vertical bars that snap to certain values on the y-axis. This is what is meant by pulse code modulation, a term that is a carryover from old-fashioned radio.

The signal in [Figure 5.5](#) is a sine wave with a frequency of 440 Hz. This corresponds to a monotonic beep with a pitch of concert A. To represent any signal digitally with high fidelity, it must be sampled with sufficient precision—in terms of both the number of samples taken per second and the numeric resolution used to express each one. The precision should be enough that a person cannot distinguish it from the original live sound when it is converted back to analog and replayed. In [Figure 5.5](#) the dynamic range (y-axis) is -32768 to $+32767$; a common choice because it is the largest range that can be represented using 16-bit numbers. The sampling rate (x-axis) in the figure is not realistic: to emphasize the steps in the signal, only every fourth sample has been plotted.

Any physical medium used to store analog signals (such as magnetic tape or vinyl) is limited in the fidelity it can capture. However, the term “analog” is also used to convey the sense of an abstract

form that captures a pure representation of the sound. This abstract form, a smooth, continuously changing line, can be modeled by mathematical formulas that can be manipulated algebraically.

Fortunately for digital librarians and others that deal in digital media, the mathematical formulas of the analog world can be translated into discrete forms suitable for computer processing. As a rough generalization, integral signs in analog formulas correspond to summation signs in digital ones, which can be computed by programs that iterate over the stream of digital values, processing them in some formulaic way that results in a particular effect—such as eliminating noise to make the signal clearer.

A fundamental theoretical result helps determine the minimum sampling rate that ensures a fair digital representation of an analog signal: knowing the highest frequency that can be present, one samples at twice that value. Humans can discern frequencies up to 20,000 Hz, so audio should be sampled at 40,000 Hz. Modern workstations can comfortably work fast enough to process such signals in real time.

Returning to the CD, playing audio can be conceptualized as reading a stream of numbers—for stereo, two streams. The sample rate is 44,100 samples/second, a value deliberately set to be just above twice the frequency limit of human hearing. In practice, playing audio from digital devices involves more controls. Users must be able to adjust the volume, skip forward and backward over tracks, and change the left-right balance and overall frequency profile—at the very least. This is achieved by appropriate manipulations of the stream of numbers. Volume, for instance, is implemented by multiplying all values in the data stream by a constant value.

Variants of PCM

The data stored on a CD is encoded as *linear PCM*. A 16-bit value is used for each channel (that is the range represented in [Figure 5.5](#)), and the sampling rate is 44.1 kHz. This means that $2 \times 16 \times 44,100 \times 60 = 84,672,000$ bits of storage are needed for every minute of audio. One byte is 8 bits, so this is roughly 10 MB. A standard CD holds 74 minutes of audio, corresponding to about 750 MB of data.

The human auditory system is essentially logarithmic in response to sound. An audio technique called *companding* exploits this to obtain higher perceptual quality for a given bit rate. Instead of representing the value of each sample on a linear scale, as illustrated in [Figure 5.5](#), a logarithmic one is used. The effect is to capture large positive and negative values more coarsely, with greater resolution for values close to zero.

[Figure 5.6](#) illustrates the principle with a 440 Hz sine wave. It shows a signal that has been compressed logarithmically, and then re-expanded. At the scale of [Figure 5.6a](#) it is hard to see much difference from [Figure 5.5](#). But upon enlarging the plot near the horizontal axis ([Figure 5.6b](#)), it is apparent that the quantized values hug the analog contour far more closely than they do near the signal's extremity ([Figure 5.6c](#)). Note that [Figure 5.6b](#) uses a much larger vertical scale than [Figure 5.6c](#).

Companding is a blend of compressing and expanding. The original signal is captured at high precision using linear PCM, reformulated on the logarithmic scale using far fewer bits for storage, and

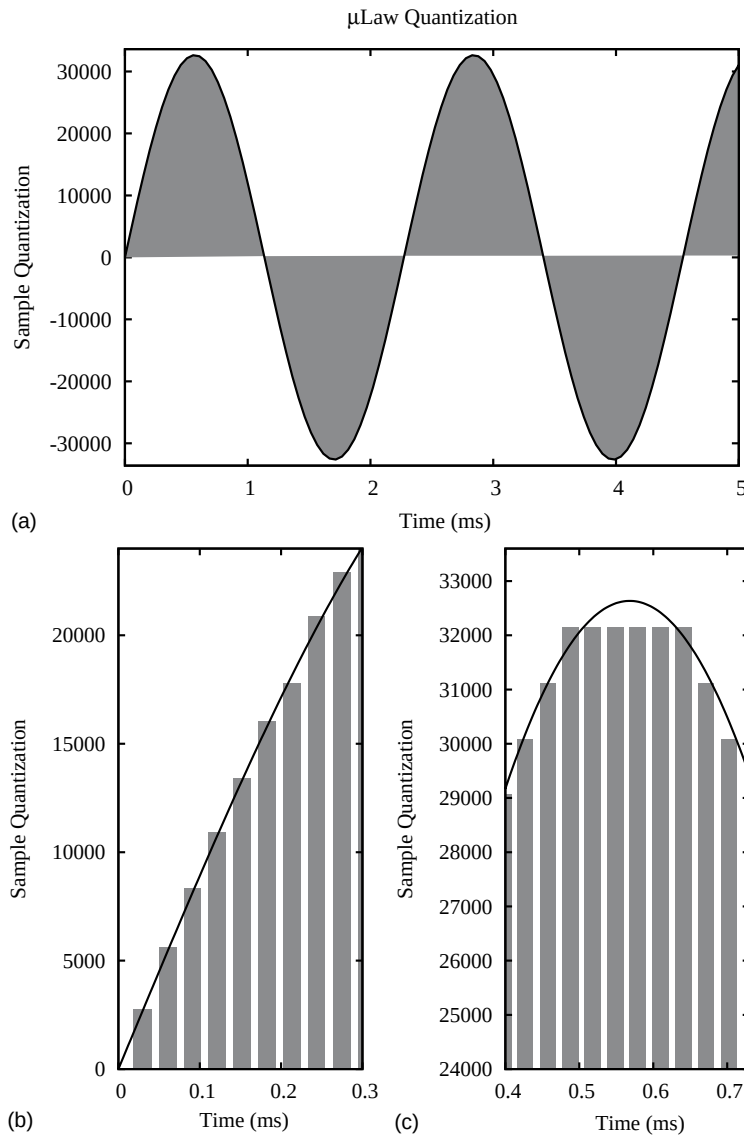


Figure 5.6: The result of compressing and re-expanding a 440 Hz tone: (a) overview; (b) high precision in low numeric values; (c) low precision in high numeric values

re-expanded for playback. The compression is lossy because the resulting signal differs slightly from the original linear PCM version. However, the loss of precision occurs in places where human listeners are unlikely to notice them. Companding is a standard technique used in telephony—the μ -law in the United States and the A-law in Europe compress 14-bit and 13-bit linear PCM signals, respectively, sampled at 8 kHz, into 8 bits per sample to yield a 64 kbit/s digital signal. The early audio formats described below support data represented in this way (among other options).

There are many more advanced ways of compressing audio signals. *Adaptive differential pulse code modulation* (ADPCM) predicts the value of the signal based on the preceding sample and encodes the difference using dynamically varying quantization steps. Speech data is unique, because it is always produced by a particular system—the human vocal tract. Although certainly not fixed, this system does not vary markedly from one person to another. The technique of *linear predictive coding* (LPC) takes advantage of this to accomplish greater compression. GSM, the world’s most popular standard for mobile phones, uses a variant known as *regular pulse excited* LPC.

Early formats: WAV, AIFF, AU

Formats that evolved during the early days of computer audio—WAV (Microsoft and IBM), AIFF (Apple), and AU (Sun Microsystems)—are still widely used today. Although each one originated on a particular computer architecture, today they are platform independent. They are simple and yield relatively large files compared to modern representations. In each case, the file begins with a header that encodes the sample rate, quantization details, and the like; this is followed by the data itself. Some use rudimentary compression techniques; others have no compression at all—reflecting the need in the early days for algorithms that worked well on primitive computers.

Along with the sample rate, number of channels, and bits per channel, the header of a WAV file indicates the compression scheme used. One option is uncompressed linear PCM; others are μ -law, A-law, ADPCM, and GSM. The number of channels can be as large as 65,535, and the sample rate can be very high—in excess of 4 GHz, which is approximately the frequency used by microwave ovens. In reality, WAV files are typically LPCM, mono or stereo, with 8 or 16 bits per sample, at rates varying from 8 to 44.1 kHz.

WAV is an example of the Resource Interchange File Format (RIFF), also from Microsoft and IBM, which is in turn based on the Interchange File Format (IFF). At the center of the design is the idea of *chunks*. A file is represented as a sequence of chunks, each corresponding to a logical unit of information or raw data in the file. Although chunks vary in size, they always begin with 8 bytes that are interpreted as follows. The first four represent a 4-character tag, such as *FORM* or *LIST*, that defines the chunk type; the remaining four represent an unsigned integer (0–65535) that specifies the size of the chunk (minus the 8 bytes just decoded). WAV defines chunk types, such as *fmt* (which stores the header information detailed above) and *data* (which stores the raw audio content). Chunks can be defined to contain sub-chunks that follow the same 8-byte header convention.

Apple’s AIFF format predates WAV by three years. Although similar, it conforms directly to the IFF standard rather than being a variant of it. Header information specifying the sample rate (among other things) is stored in a *COMM* chunk, and *SSND* is used for the raw audio. Initially, only linear PCM was included, but Apple soon added support for compression. The result was AIF-C (C for “compressed”), which has a similar file structure to AIFF but is not backward-compatible, because some of the defined chunks (such as *COMM*) have extra fields. Strictly speaking, AIF-C is a separate format.

The first four characters of a WAV file are *RIFF*, and (four bytes later) its type is *WAVE*. The first four characters of an AIFF file are *FORM*, and its type is *AIFF*. For the compressed version, the first

four characters are the same, but the type is changed to *AIFC* to make the distinction explicit. WAV files have the extension *.wav*, and the extension for AIFF files is usually either *.aif* or *.aiff*. For AIFC files, the official extension is *.afc*, but the fact that this format can represent uncompressed files as well has led software vendors to continue using the extensions *.aif* or *.aiff*. When a software application reading such a file encounters the field *AIFC* rather than *AIFF* it simply ensures that the appropriate chunk definitions are used.

The third major format is AU, from Sun Microsystems. This was originally a headerless 8-bit μ -law data file encoded at 8 kHz, but over time it was upgraded to support a wider range of compression schemes, quantization, and sample rates. AU files now have a header that begins with four bytes representing the characters *.snd* in ASCII. There follow two numeric values: the offset to where the raw audio data starts, and the total length of the file (if known; otherwise, a fixed maximum value is used). Possible encoding methods include linear PCM, μ -law, A-law, and ADPCM. The number of bits per sample is combined with the encoding representation rather than being a separate header field—for linear PCM it can be specified as 8, 16, 24, or 32 bits per sample.

The WAV and AIFF/AIFC formats provide additional facilities that are specified by different chunk types. Each has over a dozen types, some of which are shown in Table 5.1. For WAV, a *cue* is a way to mark a point of interest in the audio data, such as the beginning or end of a verse. A *playlist* specifies the order in which a series of cue points should be played. Cue points can be labeled using the *label* chunk, and further annotated with a *note* chunk. *Labeled text* chunks allow text to be associated with a section of audio rather than a specific point. All of these textual annotations are actually sub-chunks, embedded in an *associated data list* chunk.

Sampler and *instrument* chunks allow WAV files to serve as sound banks for sampled instruments that use the MIDI format described in Section 5.6. A modern synthesizer loaded with the sound bank could produce a range of musical notes containing that sound, depending on how hard and for how long the key was pressed. The higher the desired sound quality, the more samples need to be captured and stored using this representation. These chunk types use cue points to delimit which part of the file provides a particular instrument sound. They specify how it should be looped to stretch it out over time; the basic pitch; and what range of notes should be pitch-shifted to produce other notes (among other things). Today you can buy sound banks for particular instruments, and they are becoming available online in the public domain.

AIFF has similar chunk types; Table 5.1 shows a selection. *Marker* is the equivalent of *cue*, and the *comments* chunk resembles the *note* chunk except that a time-stamp is included for when the comment was made. There are text items—*name*, *author*, *copyright*, *annotation*—that relate to the

Table 5.1: Additional chunk types for WAV and AIFF

WAV	Cue	Playlist	Associated data list	Label
	Note	Labeled text	Sampler	Instrument
AIFF	Marker	Comments	Name	Author
	Copyright	Annotation	MIDI data	Instrument

overall file, rather than being associated with particular cue points. *MIDI data* and *instrument* are AIFF's tools for representing sampled instruments in a sound bank. All this textual information in WAV and AIFF files is a rich source of metadata in a digital library.

MPEG audio: MP3 and its siblings

MPEG compresses audio using a lossy technique that can achieve near-perfect CD-quality playback despite being reduced to 1/12 of the original size. It exploits a phenomenon known as *acoustic masking*: the human auditory system cannot perceive low-amplitude frequency components that are dominated by nearby high-amplitude components. These masked frequencies can be encoded using fewer bits than normal, or dropped entirely. *Temporal masking* is a related phenomenon: listeners do not hear weak sound elements that occur temporally adjacent to stronger ones. Again, these elements can be encoded at lower precision or dropped from the encoded signal altogether.

It is not easy to choose which elements to attenuate. In fact, the MPEG standard introduces an interesting idea: although the decoder is precisely defined, the encoder is intentionally left underspecified. Encoders must abide by some basic rules, but it is up to each particular implementation to decide which elements to select and how many bits to use for encoding. For instance, encoders can trade compression rate and sound quality for simplicity and encoding speed.

The MPEG standards committee publishes a reference implementation that is intended as a starting point for developing an encoder, but it is not an industrial-strength implementation. Once the developer has decided what to encode in their implementation, any decoder that conforms to the standard can reconstruct the audio since decoding is done without knowing how the values used for encoding were chosen. Future encoders may well improve on the quality ratings available today, and pre-existing decoders will still be able to play the files produced.

The MPEG standard embraces three different schemes—known as *layers*—for audio compression. The most versatile and widely used of these is MPEG Layer III, commonly known as MP3. Layers are designed so that decoders are backward-compatible: Layer II decoders can also handle Layer I data, and Layer III decoders handle all layers. As noted above, decoders are easier to write than encoders. Layer I is designed to be easy to implement and fast to process: it has a target rate of 192 kbit/s per channel. Layers II and III are 128 and 64 kbit/s per channel, respectively, corresponding to 6:1 and 12:1 compression. All three support sample rates of 32 kHz, 44.1 kHz, and 48 kHz.

It took researchers a great deal of time and effort to come up with the details of MP3. As noted above, it hinges on acoustic and temporal masking, but the real difficulty is to formulate the right mix of concepts and carry them through to a precise definition for the standard.

The encoding process first splits the signal into 32 bands, each corresponding to a predetermined frequency range—just like the sliders of a graphics equalizer on a stereo system. Each band is then transformed to the frequency domain as described at the end of [Section 5.1](#), where it is analyzed to identify masked frequencies (i.e., good candidates for compression). The transform used is called the Modified Discrete Cosine Transform, which is closely related to the Fourier transform. The idea of splitting the audio into frequency ranges and transforming each one is called a hybrid filter.

Next the coefficients are quantized and encoded. MPEG Layers I and II use bands of equal frequency, while layer III (MP3) uses unequal ones that are psycho-acoustically motivated. In addition, layers II and III exploit temporal masking, achieved by taking three consecutive overlapping windows and analyzing them. Layer III also takes stereo redundancy into account.

In addition to all this, audio can be encoded into MP3 using either a fixed or a variable bit rate. The complexity of the frequencies represented in audio can vary throughout a recording, and higher compression rates can be obtained during simpler passages without sacrificing perceived quality. The encoder can exploit this by varying the transmission rate, saving space in simpler sections and providing more detail in complex ones.

The audio quality of MPEG has been evaluated in extensive perceptual tests. Listeners are played three versions of several audio recordings: the original, the MPEG encoding, and the original (again) randomly transposed. They are asked to score the second and third versions using a standard scale of impairment that ranges from “very annoying” to “perfect.” At a rate of 128 kbit/s per channel, even experts cannot reliably distinguish MPEG from the original recording. At 64 kbit/s per channel, Layer II was judged to lie between annoying and slightly annoying. The same rate with Layer III scored significantly higher—just noticeably different. This result has led to the widely quoted claim that CD-quality audio encoded as MP3 (that is, MPEG Layer III audio) achieves minimal perceptible degradation with only 8 percent of the original file size.

Here is a summary of the key technical points of MP3:

- sample rates: 32–48 kHz
- bit rates: 32–320 kbit/s
- hybrid filter bank
- fixed or variable bit rate.

A fixed set of sample rates and bit rates is provided by the standard. A sample rate of 44.1 kHz, corresponding to the standard for CDs, is invariably used. 128 kbit/s is widely used as the bit rate, although (as noted above) people can sometimes discern distortion. With the increasing use of broadband, the higher 192 kbit/s rate, which allays this concern, is becoming more common. MPEG-2 expands the range of these two parameters—for example, it extends the sample rate down to 8 kHz.

Although MPEG is an open standard, various companies hold patents on some of the techniques it uses. Potential conflicts are resolved by requiring all technology involved to be licensable on “fair and equal terms” as part of the ISO standardization process. The intention is that these open standards not only facilitate data interchange, but do so using sophisticated techniques that rival state-of-the-art proprietary development. This means that more companies adopt the standard, boosting its likelihood of gaining widespread adoption—with benefits for all.

Post-MP3 formats: AAC, Ogg Vorbis, FLAC

Several new audio compression techniques have been developed to improve on the MP3 model for perceptually based lossy compression. Among them are Advanced Audio Coding (AAC) and Ogg

Vorbis. Developed with the benefit of hindsight, they typically achieve greater compression than MP3, or alternatively retain better quality for the same file size. In both AAC and Ogg Vorbis, whose performance is largely comparable, attention was focused on mitigating the undesirable audio artifacts that occur when encoding at low bit rates.

AAC was designed to be MP3's logical successor and was officially introduced into the MPEG-2 standard in 1997 as Part 7. Two years later, with some refinement, it became MPEG-4 Part 3. It is used by Apple in its iPod and iPhone products, the iTunes store, and QuickTime multimedia files. Technically, the format includes an expanded range of sample rates, with the option of using an alternative windowing function that helps mitigate leakage from one spectral band to another. It also applies the Modified Discrete Cosine Transform directly to the audio signal, rather than MP3's hybrid filter approach that separates the signal into bands before transforming it.

Here is a summary of the key technical points of AAC:

- sample rates: 8–96 kHz
- Kaiser-Bessel based window function
- Modified Discrete Cosine Transform
- employs larger block sizes.

The aim of the Ogg project is to define a range of multimedia formats that are unencumbered by patents. Ogg itself is a container format suitable for streaming various media forms, and the term Ogg Vorbis refers to a lossy audio representation scheme encapsulated within this container. (We will meet the Ogg container again in [Section 5.4](#) when we describe its video format.)

When audio is encoded as an Ogg Vorbis file, the user can specify a quality parameter ranging from –1 to 10, which controls the overall file size generated. Although this value does not correspond to a precise bit rate, there are some broad equivalences. Qualities of 0 and 10 are roughly equivalent to 64 kbit/s and 400 kbit/s. The default setting is 3, which gives smaller file sizes and significantly greater fidelity than MP3 compression (approximately 110 kbit/s, compared to 128 kbit/s). In MP3, when the bit rate is very low, a metallic sound appears; Vorbis includes a noise floor to achieve a more natural effect under these conditions.

Here is a summary of the key technical points of Ogg Vorbis:

- sample rate: 8–48 kHz, 16-bit and above
- fixed or variable bit rate: 16–128 kbit/s per channel
- Modified Discrete Cosine Transform
- vector quantization
- noise floor and residual.

Unlike MP3 and the other post-MP3 formats, the Free Lossless Audio Codec (FLAC) represents an audio signal in a way that the original can be reconstructed exactly, not approximately. It typically reduces file sizes by 30 to 50 percent for music, with significantly greater compression for voice

recordings. This contrasts with general-purpose lossless compression algorithms like ZIP, which typically reduces CD-quality audio by only 10 to 20 percent. While the file sizes obtained cannot rival lossy standards like MP3 and its successors, lossless compression is particularly suited to archival applications. It is an open, royalty-free standard and is well supported by many software applications.

FLAC works on linear PCM audio with samples from 4 to 32 bits, any sampling rate from 1 Hz to 1 MHz, and from 1 to 8 channels. Stereo and surround-sound channels can be grouped to increase compression by taking advantage of inter-channel correlations.

Metadata can be included in MP3 files by using tags, and both Ogg Vorbis and FLAC include a similar scheme (see Chapter 6, Section 6.3, which addresses metadata in general).

Replaying audio

One annoying problem that many people notice with their personal audio players is that tracks obtained from different sources play back at different volumes, so you have to keep adjusting the volume manually to obtain a pleasant effect.

A scheme called *replay gain*, already in use in some audio players, solves this problem. Working with single tracks or albums, it normalizes the perceived volume of digital audio formats like those surveyed above. First the entire file is scanned to measure the perceived loudness and peak levels. Then the *gain*, or difference between perceived loudness and a predefined target value, is calculated. This enables playback of the file at a similar volume to other tracks. The concept can be mapped neatly into a digital library if both the gain and the peak value are stored as metadata when the file is ingested (possibly in the audio file itself). When the track is played, these metadata values are retrieved and used to automatically adjust the default volume used.

An audio digital library

Figure 5.7 shows a selection of snapshots from the Internet Archive, focusing on its audio content and in particular the Audio Books and Poetry subcollection. The Internet Archive is a vast treasure trove of resources spanning many forms of digital media—text, audio, and moving images. Each media type is gathered together into a collection, differentiated by visual cues, such as the background color used.

As shown in Figure 5.7a, the Audio Archive contains over 200,000 digital recordings, ranging from book and poetry readings to Grateful Dead concerts, from Old Time Radio shows (such as *Dick Tracy* and *The Goons*) to original music uploaded by users. Highlights of the collection, such as a featured recording and the top ten most accessed items, appear in columns to the left and right.

The other media types in the Internet Archive are accessible from the top line of links, which include access points to general information about the archive. The next line gives quick links to the subcollections, which are also listed below in the central column at the bottom of the screen. The final line allows searching, and also lets users contribute content to the library. By default, search applies to the audio collection, but a pull-down menu allows users to search at broader (across all media types) or narrower (subcollection) levels.

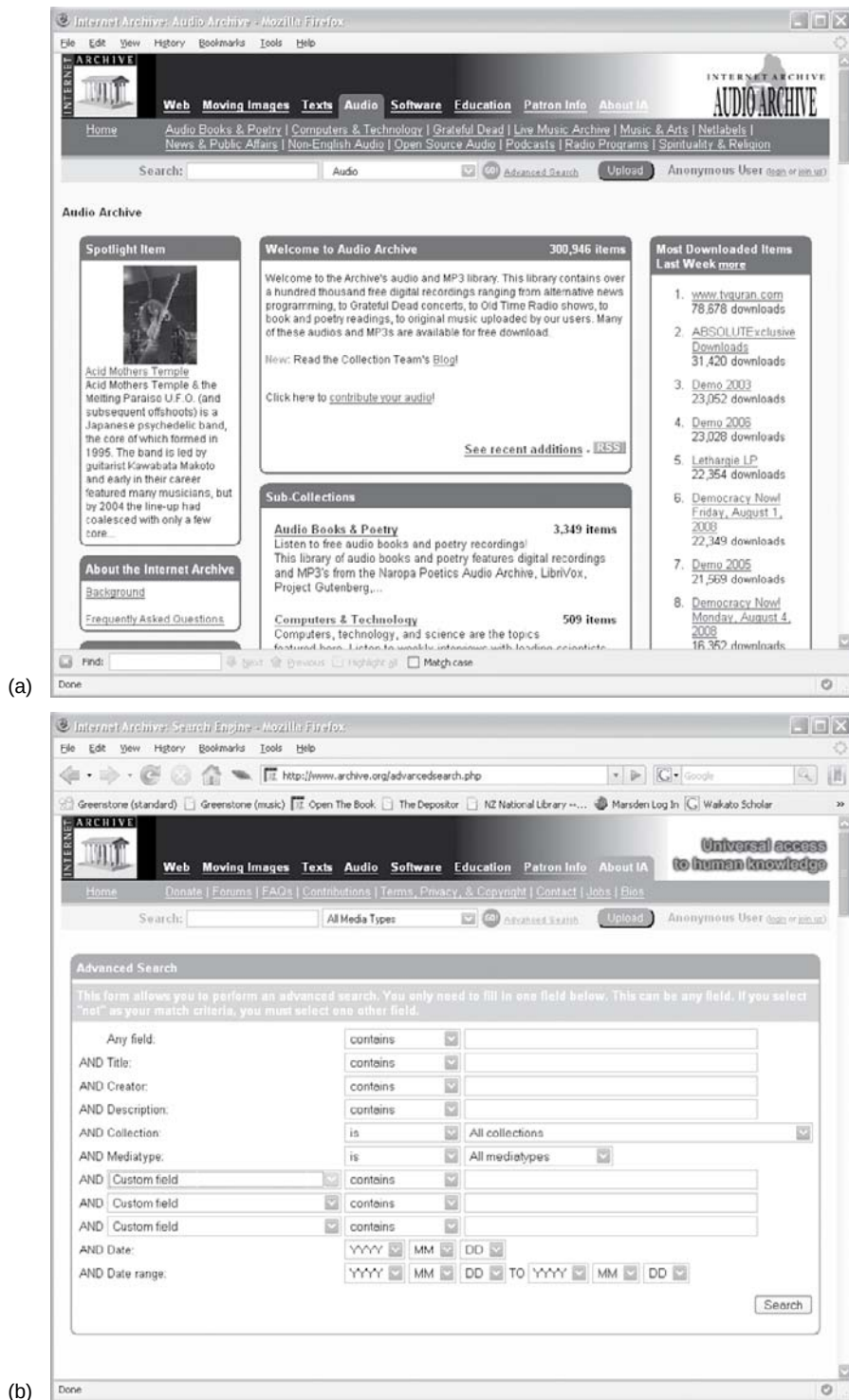
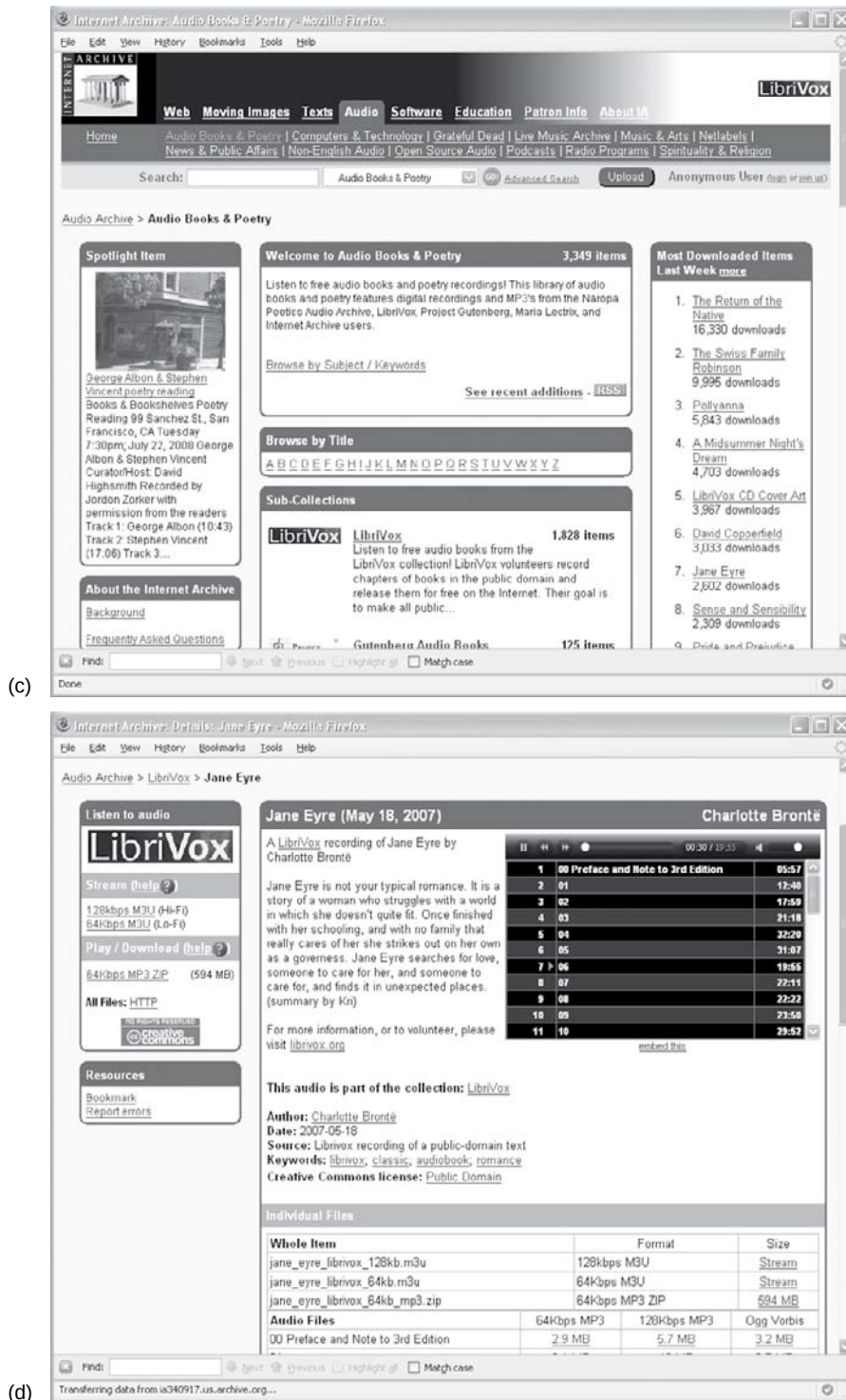


Figure 5.7: Audio content at the Internet Archive: (a) Audio Archive home page; (b) advanced search page;

Figure 5.7, cont'd: (c) Audio Books and Poetry subcollection; (d) *Jane Eyre*

Browsing and searching in this digital library is based on metadata. A horizontal navigation bar displays the letters of the alphabet. Clicking on a letter leads to a page with a maximum of 50 items, including links to the next and previous pages where appropriate.

Advanced search, like quick search, can be used to search across all collections, a single collection, or a subcollection. [Figure 5.7b](#) shows the usual fielded Web form (cf. [Figure 3.15](#)), beginning with the basic metadata fields Title, Creator, Description, Collection, and Media type. The additional “custom” fields include roughly 100 centrally managed metadata items, such as Added Date, Digital Publication Date, Language, and Stream Only, along with over 600 user-defined tags—such as Codec, Camera Operator, Camera, and even “Cake-rating”!

The access mechanisms that this digital library provides apply to collections of any media type: they work just as well for text and moving images. To support cross-collection searching in a meaningful way, the metadata terms indexed must be uniform. This is why the advanced search page displays the five basic metadata fields prominently. Uncontrolled tags like Camera Man (or is it Camera Operator?) are less useful because they make users work harder to formulate queries that satisfy their needs.

Although the collection can be searched and browsed regardless of the media type, when users reach actual documents, what happens then depends on what they’re trying to view (or hear). In [Figure 5.7c](#) the user has entered the Audio Books and Poetry subcollection and clicked on *Jane Eyre* in the list of the top ten most viewed items. [Figure 5.7d](#) is the result. It displays metadata and provides various forms of access to the audio. It presents a synopsis of the plot, along with information such as author, keywords, and the date it was added to the library. The black menu at the top right tabulates the chapters. A user clicks a chapter to have it read and can also adjust the volume, jump to another chapter, and even seek forward and backward within a chapter.

Alternative forms of audio are provided for users without broadband access. Hyperlinks at the left offer to stream the audio at 128 kbit/s and 64 kbit/s, dubbed Hi-Fi and Lo-Fi, respectively—although with lossy compression, as discussed above, fidelity does not correspond directly to data rate. The Lo-Fi version can also be downloaded. This audio book lasts for more than three hours, making the download substantial—600 MB—even with the impressive compression of MP3 and Ogg Vorbis. Further down the page in [Figure 5.7d](#), individual chapters can be downloaded as either 64 or 128 kbit/s MP3, or Ogg Vorbis.

At the end of the table are links to textual information about the book: text files, metadata, and reviews. The information is provided in XML (see [Chapter 4](#)), which is both machine and human readable. However, most of today’s Web browsers simply display the raw XML.

In summary, the Internet Archive provides a metadata-based digital library that supports browsing and searching. Document surrogates take the form of HTML files that wrap up many different ways in which users can access the documents. Both standard metadata and user-defined tags are provided and made available along with the full text and book reviews. The approach works equally well for video—or, for that matter, for scanned images that have been OCR’d.

5.3 Images

As explained in Section 4.2, a digitized image comprises a regular matrix of tightly packed black-and-white or colored dots called picture elements or *pixels*. Table 4.2 shows the resolution of an assortment of devices. Represented in a computer file in the obvious way, images occupy an inordinate amount of space compared with text documents. A standard $8\frac{1}{2} \times 11$ inch page, scanned with 1 bit/pixel at a resolution of 300 dpi, produces a file of just over 1 MB. Using 8-bit grayscale increases this to 9 MB; a 24-bit full-color image occupies 28 MB. Higher spatial resolution greatly amplifies the problem: a 600 dpi scanner quadruples the size to over 100 MB in the full-color case. Even a 1 TB disk can hold only 10,000 such images.

Images are not normally stored this way. Instead, they are compressed. As with audio, both lossless and lossy methods are both used. Scanned images (again like audio) may contain *digitization noise*, artifacts caused by the process of digitization itself. A lossy compression scheme that can eliminate the noise achieves economy while simultaneously *increasing* image quality. Such techniques can be used to obtain remarkably compact representations, particularly for grayscale and photographic images. Depending on the image content, each pixel can sometimes be compressed from its original representation in one to four bytes to well under a single bit with a barely perceptible loss of quality. However, as noted earlier, there are situations (e.g., medical, legal, and archival documents) where exact representation is essential and compression must be lossless.

A rudimentary approach to compressing images is to apply a standard utility designed for general-purpose compression of computer files. This treats the file as a one-dimensional sequence of bytes. Compression can be greatly improved by acknowledging the two-dimensional image structure and exploiting the data's spatial characteristics. For example, most images contain large areas whose color and texture are constant or slowly varying, interrupted by regions of abrupt change, such as lines or edges. Sophisticated algorithms identify these regions and exploit them for compression.

There are countless compressed image formats, and we restrict attention to the most popular ones used in Web applications and digitization projects. We begin by describing the GIF and PNG formats for lossless images, and then review JPEG, a comprehensive international standard for high-performance lossy compression of photographic images. They are all supported by modern Web browsers and related plug-in technologies, such as Flash and Java applets. Next we discuss *progressive refinement*, which allows an image to be displayed rough-hewn before it has been received in full. GIF, PNG, and JPEG all have progressive refinement modes.

Following this, we describe two additional formats: TIFF and JPEG 2000, which are often used in digital libraries during digitization and for archiving and are capable of both lossy and lossless representation. (However, they are not often used directly on the Web.) Then we take a tour of a digital library of photographs.

Finally, we describe another format, scalable vector graphics (SVG), which allows images to be defined in a structured way. In doing so, the format provides a richer set of options than purely raster-based formats for the development of searching and browsing operations in a digital library.

Lossless compression: GIF and PNG

Two lossless image compression formats are in widespread use today: the pervasive (but proprietary) Graphics Interchange Format, GIF (officially pronounced *jiff* but often heard as *gif*) and an open standard called Portable Network Graphics or PNG (pronounced *ping*). JPEG also has a lossless mode, but it is rarely used in practice.

GIF was originally specified in 1987 and for many years was the most widely used lossless image compression format. It was intended as an exchange medium for images that could be displayed on a variety of graphics hardware platforms, and it was adopted by CompuServe in order to minimize the time required to download pictures over modem links. On Web pages, GIF is most commonly used for stylistic images, logos, line drawings, and the like.

The GIF scheme applies to images in which each pixel is represented by eight bits or less. The code for a pixel can either be a grayscale value in its own right or an index into a lookup table, called a *color map*, that represents a palette of colors for the image. The color map is stored with the image and comprises up to 256 different colors (corresponding to 8-bit pixel codes), with a full 24-bit color specification for each—eight bits for the three primary colors, red, green, and blue. Thus color images are represented in terms of a small palette whose values are chosen from the full range of possible colors. In a subsequent modification to the original GIF specification, one of the 256 colors is reserved as a transparent value that shows the background on which the image appears.

The color map can be tailored for each individual image and stored along with it as a prefix to the image file. Alternatively, a group of images can share the same map, or it can be omitted entirely. If present, the color map is uncompressed and occupies up to 768 bytes (3 bytes for each of 256 possible colors).

The sequence of 8-bit pixel values that represent the image content is compressed using LZW, which is a particular incarnation of one of the dictionary methods for compression introduced in [Section 5.1](#). This scheme was designed for text compression, not images, and exploits the fact that in text many short “phrases” (of a few letters, or a few words) tend to repeat. It works by accumulating a list of phrases as they are encountered in the text file being processed. This strategy of adapting to the particular input file makes the process language independent, and also means that it can be sensibly applied to files other than text—images, for instance.

GIF files can contain either a single image or a sequence of images, the latter giving rise to the animated GIFs seen on the Web. A feature is included that makes it easy to skip through the sequence without decompressing each individual image.

In 1995, Unisys announced that royalties would be levied on programs implementing GIF because of a long-standing patent they held on the LZW compression scheme. This prospect caused widespread dismay because GIF was the primary means of storing images on the World Wide Web. In an attempt to avoid bad press, Unisys subsequently refined its position, but the uncertainty instilled by the original announcement made developers wary of using the format for years to come. The key international patent expired in 2004, removing all doubt: anyone can now use the GIF format without infringement.

Unisys's surprise announcement catalyzed the development of a new lossless image format, PNG, specifically intended for the public domain. At its core is a general-purpose compression scheme called *gzip*. This is a public-domain open source compression utility, based on another of the dictionary methods described in [Section 5.1](#). *Gzip* performs better than LZW, and PNG usually achieves greater compression than GIF.

More importantly, PNG provides better compression because it works in a way that acknowledges the two-dimensional structure of the image, rather than treating it simply as a one-dimensional sequence of bytes. It defines *filters* that can be applied to the pixel values before compression. The *horizontal difference* filter subtracts the previous pixel value from the current one, so that pixel differences are encoded, rather than pixel values. (In fact, the difference operation is applied to individual bytes rather than to pixel values. Each pixel may be represented by more than one byte, in which case the difference is taken between corresponding bytes. All PNG filters operate byte-wise.) The *vertical difference* filter subtracts the value of its neighbor in the row above. The *average difference* filter subtracts the average of the neighbors above and to the left. A further filter performs a slightly more complex operation involving a nonlinear function of the neighboring pixel values.

It is up to the encoder how it uses the filters—or indeed whether it uses them at all. The PNG standard recommends that encoders optimize the filter for each scan-line by trying all possibilities and using a heuristic criterion to select the best, possibly choosing a different filter for each scan-line. Combining this improvement with the advantage of *gzip* over LZW, PNG improves compression by around 10 to 30 percent over GIF, depending on the particular image being encoded.

PNG incorporates several other practical improvements. Pixels are not restricted to 8-bit values: they can be drawn from a 256-color palette, but they can include up to 16 bits of grayscale values or 48 bits of full-color information. There are 256 possible transparency values, so a picture can fade gradually into the background. There is a *gamma correction* feature that helps compensate for differences in how computer monitors interpret color values.

However, while GIF supports animated images, PNG does not. For animated images this is officially left to a related but different standard, Multiple-image Network Graphics or MNG (pronounced *ming*). There is an unofficial extension called Animated Portable Network Graphics (APNG), which is (for the main part) backward-compatible. It is supported by some Web browsers, notably Firefox, but not Internet Explorer which (decoding it as a standard PNG file) instead shows a static version of the first frame of the animation.

Lossy compression: JPEG

JPEG, named after the Joint Photographic Experts Group that designed it, is a comprehensive standard for representing compressed continuous-tone images. It was approved in 1994 after many years of development. It is a general-purpose standard and underlies a gamut of image communication services and image applications—desktop publishing, graphic arts, color facsimile, newspaper wire-photo transmission, and medical imaging—as well as hardware devices like digital cameras. It is complex, and a great deal of care went into its specification. It has become the standard technique for compressing still images and is universally used for representing photographic images on the Web.

JPEG's compression algorithm works well enough to produce excellent image quality at around 1 bit/pixel—the same as an uncompressed bilevel image. This represents impressive compression: grayscale or color images are digitized at anything from 8 to 32 bits/pixel. No lossless compression scheme could reduce continuous-tone pictures to anything like this level. At this rate, some loss is inevitable—you have to accept approximate rather than exact reproduction. The compression method was selected from among many candidates, based on an assessment of subjective picture quality.

JPEG is divided into a baseline system that offers a limited set of capabilities and a set of optional extended features. The baseline gives a plain, lossy, high-compression image coding and decoding capability. It operates on images in which each pixel is represented by 8 bits. The algorithm encodes color image components independently and is suitable for use with commonly used color spaces, such as red, green, and blue (RGB) and CMYK (cyan, magenta, yellow, and black). Higher-resolution options, with more bits per pixel, are among the extended system features.

Figure 5.8 depicts the encoding and decoding processes. Images are first divided into 8×8 pixel blocks. The discrete cosine transform (previously encountered in MP3 compression) is applied to each block; the transform is configured in this case to map the 64 pixel values into 64 spatial frequency coefficients. The transform is reversible: the 64 coefficients characterize the block exactly and can be used to faithfully reproduce it. They represent image components at different spatial frequencies.

Transforming one set of 64 numbers into another set of 64 numbers does not seem to achieve much. However, the spatial frequency coefficients are far more suitable for lossy quantization than the pixel values themselves. There are blocks in which sample values vary slowly from point to point—indeed

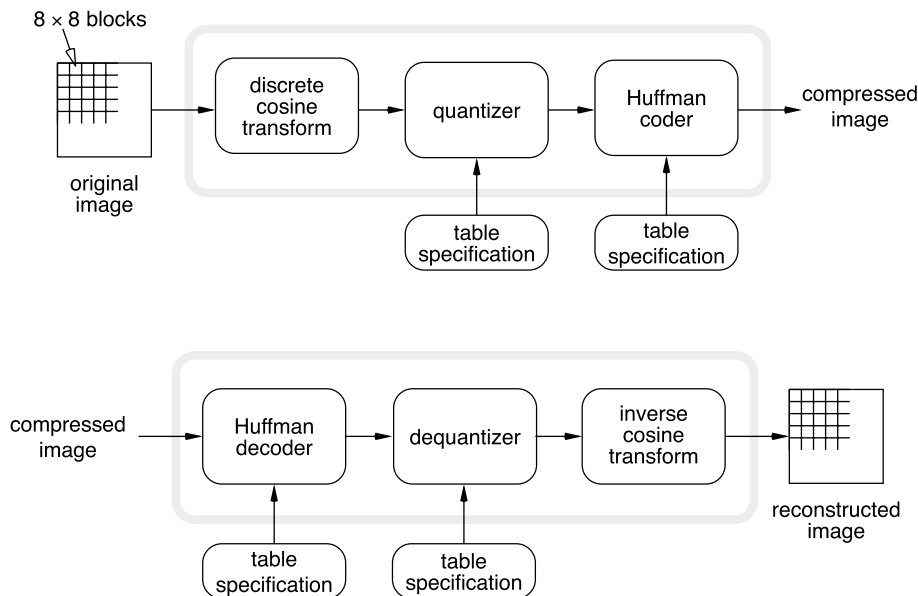


Figure 5.8: Encoding and decoding processes in baseline JPEG

this is the case for most parts of nearly all images, particularly ones of natural objects—and in these the transformation concentrates most of the signal in the lower spatial frequencies. For a typical 8×8 sample block from a typical source image, many of the higher spatial frequencies have zero or negligible amplitude and need not be encoded at all. Some pictures do have significant higher spatial frequencies—for example, ones that contain regular patterns, such as brick walls or tiled roofs—but these are the exception rather than the rule, particularly for natural images.

To show what the discrete cosine transform coefficients mean, [Figure 5.9](#) presents an image after each successive coefficient has been calculated. The original 512×512 image has been divided into an 8×8 matrix of blocks, each block having 64×64 pixels. JPEG actually uses blocks of 8×8 pixels, but we use larger ones to demonstrate the effect. The first version of the image is made from each block's zero-frequency coefficient. This represents the overall grayness of the block, so at this stage each block is a uniform shade of gray. After inclusion of the second coefficient for each block, horizontal cross-sections through the blocks still have uniform shading, although vertical cross-sections do not. By the time the 32nd coefficient is combined, the image is starting to look much like the final version. However, it would require $64 \times 64 = 4,096$ coefficients per block for full, error-free reconstruction of our example picture. The corresponding figure for JPEG is $8 \times 8 = 64$ coefficients.

Following the discrete cosine transform, the nonzero coefficients are independently rounded to discrete values by a uniform quantizer—this is the lossy part. The quantizer step size (that is, the difference between successive levels) is different for each coefficient. The encoder may specify with each picture the quantization tables that are to be used for that picture—they could be derived for each picture independently and included with it. Alternatively, it may specify that previously installed tables are to be used instead. The JPEG standard includes an example set of quantization tables that are particularly appropriate for natural scenery.

The zero-frequency coefficient is highly correlated from block to block, because it is basically the average intensity over the block. Hence it is differentially encoded—in other words, it is represented by the difference between the zero-frequency value of this block and the previous one. The other 63 coefficients are not differentially encoded. They are sorted from low to high frequency using the zigzag sequence of [Figure 5.10](#), in the order indicated by the arrowheads; then they are coded to reduce statistical redundancy.

The baseline system represents the coefficients using Huffman coding, mentioned in [Section 5.1](#). First, since many of them are zero, the number of zero coefficients before the next nonzero one is specified—a kind of run-length encoding—and only the nonzero coefficients are coded. As noted earlier, Huffman coding works by using a codebook that gives each coefficient value a particular code. An example codebook is provided in the JPEG specification, but the encoder can determine codes appropriate to the particular picture and embed them in the compressed data stream.

A key feature of JPEG is its ability to control the number of bits per pixel used to represent an image. It accomplishes this by varying the amount of detail lost, which has a corresponding effect on the overall quality of the reconstructed image. [Figure 5.11](#) shows three stages of lossy transmission from a JPEG-like encoding scheme, at 0.1, 0.2, and 1.0 bit/pixel, respectively. The differences in resolution are just discernible—despite the low-quality halftone reproduction—if you look closely,

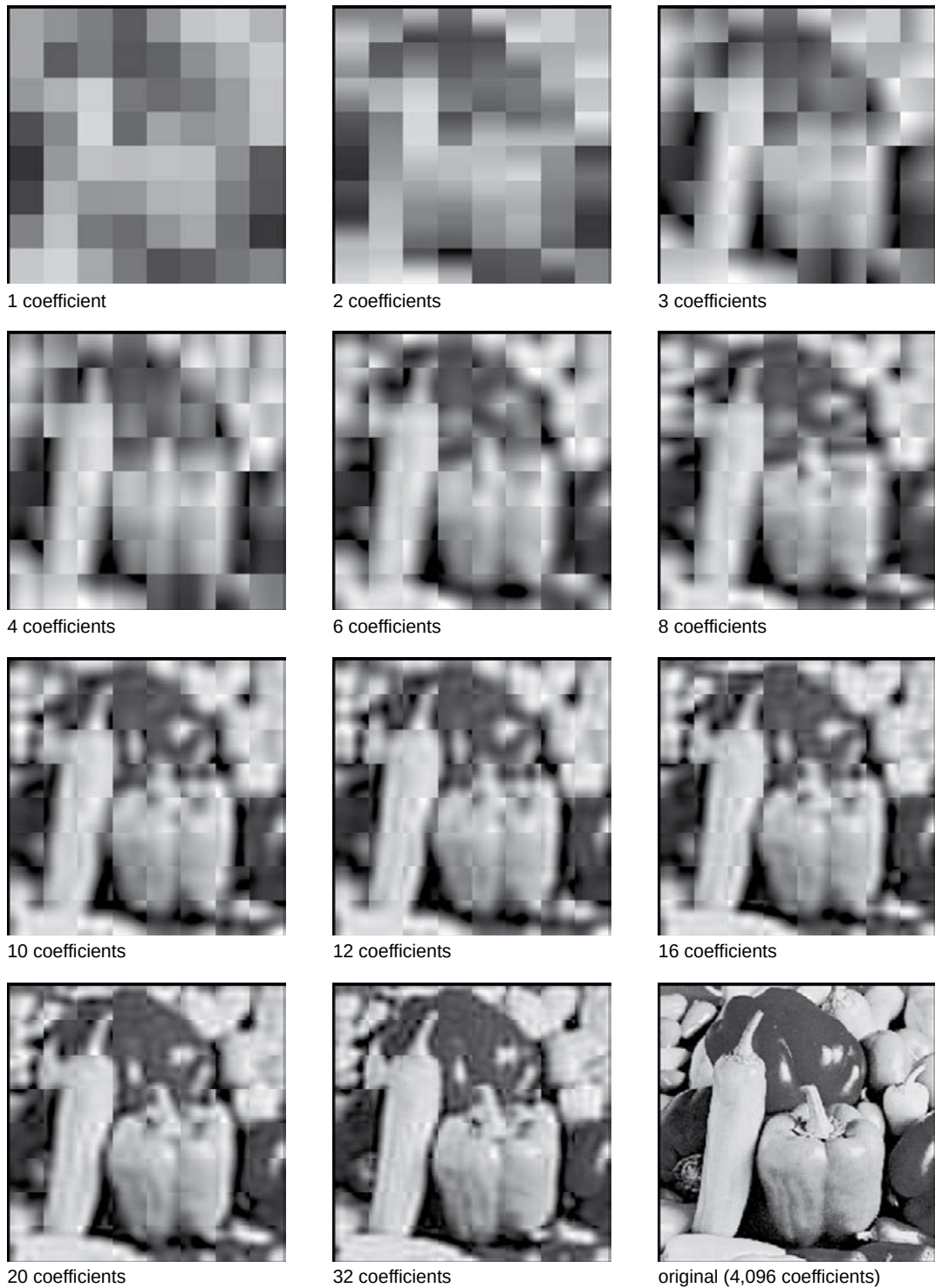


Figure 5.9: Transform-coded images reconstructed from a few coefficients. University of Southern California Image Processing Institute (USC-IPI) image database

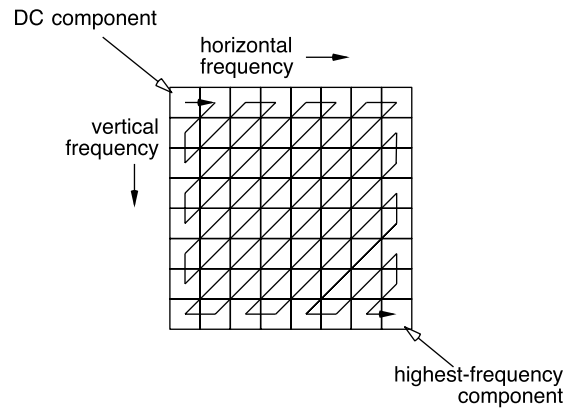


Figure 5.10: Zigzag encoding sequence

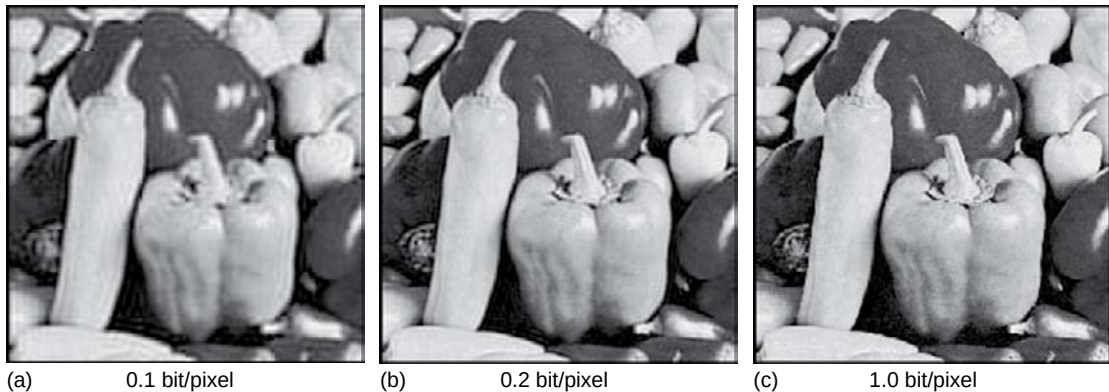


Figure 5.11: Images reconstructed from different numbers of bits (USC-IPI image database)

particularly around the straight edges in the pictures. The JPEG standard describes a postprocessing operation that can be used to suppress blocking artifacts and improve quality at low bit rates; it is not used in these examples.

Although JPEG is a complex format, well-engineered implementations can keep pace with 64 kbit/s communication channels. This makes it suitable for interactive use. For example, compressed to 1 bit/pixel, a 720×576 pixel image takes 6.5 seconds to transmit over the same 64 kbit/s line that is needed to transmit MP3 audio in real time. This is a tolerable delay for occasional image viewing.

As a general guide to the results attainable with a lossy compression method, for color images of moderately complex scenes, various levels of picture quality can be obtained at different compression factors:

0.25–0.5 bit/pixel: moderate to good quality, sufficient for some applications

0.5–0.75 bit/pixel: good to very good quality, sufficient for many applications

0.75–1.5 bit/pixel: excellent quality, sufficient for most applications

1.5–2 bits/pixel: usually indistinguishable from the original, sufficient for the most demanding applications

In many graphical applications that display and manipulate images, when saving in JPEG format, a slider-bar is provided for setting image quality in the range 0 to 100 percent. When choosing 100 percent it is important to realize that this still means the representation is lossy.

The JPEG standard defines many extensions of baseline JPEG. Although the baseline system uses Huffman coding, the standard accommodates the more sophisticated arithmetic coding (also mentioned in [Section 5.1](#)). The Huffman coding method is about twice as fast as arithmetic coding, but—at least in software implementations—throughput is dominated by other parts of the JPEG scheme, so the overall computational penalty is not so great in practice. Arithmetic coding gives about a 10 percent compression advantage over Huffman coding for high-quality reconstructions of the relatively noisy test images used to develop JPEG. For lower bit rates and cleaner images, its benefit is even greater.

Although the baseline system is lossy, JPEG also defines a lossless variant. In the original JPEG definition, this was not a very effective compression scheme, and so a separate JPEG-Lossless standard was developed that provides an excellent scheme for lossless compression of continuous-tone images. However, it is rather complex, and application support for it is limited.

Progressive refinement

Pictures on a conventional TV or printer are drawn in raster order—left to right, top to bottom—across the screen or page. But when transmission is slow, there are advantages to creating the image in a different way. For example, 10 percent of the way through a raster transmission of an ordinary image format over the Web, only the top tenth of the picture can be seen, and this is not usually very informative. Users would generally prefer to see the whole picture at a tenth of the final resolution instead. This technique is called *progressive refinement*.

[Figure 5.12](#) shows a picture that builds up gradually to the full resolution, and contrasts it with ordinary raster transmission. The final picture is hardly distinguishable from the version created from only 10 percent of the information. Progressive refinement is clearly advantageous in an interactive situation where one has an opportunity to stop transmission if the wrong picture appears, or when browsing casually through many pictures. Even if there is no opportunity to cancel transmission, it is much less frustrating to see an initial image that is progressively refined than it is to have it revealed, slowly and tantalizingly, line by line.

A crude but effective form of progressive display called *interlacing* is used in television to reduce flicker by increasing the apparent rate at which successive images appear. The picture formed from the odd-numbered scan-lines is transmitted first, followed by the even-numbered scan-lines. An extension of this technique is used by both GIF and PNG to give a form of progressive display. When compressed, pictures can be stored in “interlaced order” instead of conventional order, so that when the file is read and transmitted over a network, an approximate version of the image is seen initially.



Figure 5.12: Progressive versus raster transmission. USC-IPI image database

In GIF, data is interlaced by row and is stored in four “passes” over the image. Every eighth row is stored first, starting with row 0. Next comes every eighth row starting with row 4. Then every fourth row starting with row 2 is stored, and finally every second row starting with row 1 fills in the missing lines. When the picture is read from this file, one-eighth of it is present by the end of the first pass, one-quarter by the end of the second, one-half by the end of the third, and the full image by the end of the fourth. When the picture is transmitted and displayed, corresponding amounts can be viewed—although just because an image is transmitted in an interlaced format does not mean the application at the receiving end must display it progressively.

PNG supports a two-dimensional interlacing scheme based on the 8×8 template shown in Figure 5.13, which provides a superior form of progressive display. The template is tiled over the whole image. The image is represented in seven pieces that store the values of the correspondingly numbered pixels in Figure 5.13. The first piece, comprising pixels numbered 1 in the figure, stores 1/64 of the information, the second (pixels numbered 2) 1/32, and so on, until all information is complete by the end of the seventh piece (pixels numbered 7). When the picture is transmitted, a reasonable approximation can be drawn as soon as the first piece is received, and the image improves with each new piece. Naturally, this scheme affects compression efficiency, and file size is slightly increased over the noninterlaced version.

One of the extended-feature options in the JPEG standard is a sophisticated form of progressive refinement. The image is approximated by low-frequency terms that are gradually refined to produce the final version. The process usually ends when the image is the same as what would have been transmitted in the normal (lossy) case, but there is a further option of progressive *lossless* coding, which continues until the final picture is almost identical with the original—the only discrepancies being due to the finite-precision arithmetic employed in the encoder and decoder. This final stage involves sending a spatial correction signal that gives the difference between the final lossy image and the original one, pixel by pixel.

1	6	4	6	2	6	4	6
7	7	7	7	7	7	7	7
5	6	5	6	5	6	5	6
7	7	7	7	7	7	7	7
3	6	4	6	3	6	4	6
7	7	7	7	7	7	7	7
5	6	5	6	5	6	5	6
7	7	7	7	7	7	7	7

Figure 5.13: 8×8 tiled template used to generate a PNG interlaced file

Recall that JPEG works by obtaining spatial frequency coefficients using the discrete cosine transform and quantizing them. With progressive refinement, instead of transmitting all coefficients for each block in turn, many passes are made through the picture, and successively higher bands of coefficients are transmitted at each pass. In early passes the low-frequency, or rough-detail, information is sent, leaving the high-frequency, fine-detail information for later. This is called the *spectral selection* method of progressive refinement. An alternative, called *successive approximation*, is also specified in which the coefficients are first sent with reduced precision. Successive approximation yields much better visual quality at low bit rates, but the number of progressive stages is limited. Spectral selection allows more progressive stages and, since it is a simple extension of the sequential mode, is easier to implement. Successive approximation—particularly, it turns out, the Huffman coding version—is more difficult to achieve.

Archiving images: JPEG 2000 and TIFF

JPEG 2000 is a more recent development than JPEG. Its main advance is to use a different transform, the discrete wavelet transform, as the basis for compression. This avoids the tendency toward visible artifacts that can result from dividing an image into discrete blocks, as illustrated in the comparison of the two formats in Figure 5.14. It also typically yields a higher rate of compression (upward of 20 percent for mid-range quality lossy compression), although this is not usually the primary reason for choosing the format.

Rather than lossless compression being a separate add-on as with JPEG, in JPEG 2000 the lossless mode follows on more gracefully from the implementation of the lossy form. Another benefit of JPEG 2000 is that it allows parts of an image to be reconstructed without having to decompress the whole file.



Figure 5.14: Comparison of visual quality of JPEG and JPEG-2000 compression: (a) JPEG baseline (0.125 bit/pixel); (b) JPEG 2000 (0.125 bit/pixel). Reproduced from JPEG-2000 FAQ: <http://www.jpeg.org/.demo/FAQJpeg2k/index.htm>

JPEG 2000 comes in twelve parts. The majority of the parts are complete and have been published as ISO standards. Part I is of most interest here; it defines the core coding system, represented by files with the extension *jp2*. Part II consists of a set of extensions (such as multiple compositing layers and animation) that largely operate independently of each other. Part II files use the extension *jpgx*. Subsequent parts of the standard define features such as security, interactive protocols, and wireless applications.

In addition to the capabilities included in the original JPEG format—such as lossy and lossless forms, and error detection and correction (when transmission is made over a noisy communication medium)—core capabilities include:

- transparency
- random code-stream access
- multiple resolution representation.

Although both GIF and PNG support transparency, JPEG does not. This is rectified in JPEG 2000.

Random code-stream access allows part of an image to be displayed without the need to reconstruct the entire image. (However, the image must have been encoded in a way that exploits this capability.) Multiple resolution provides a progressive transmission facility more powerful than JPEG's: it can display lower-resolution or lower-quality versions as the data is decoded, or even a grayscale version before the full color information is available. These two features used in tandem make a powerful combination.

In terms of representing metadata, both JPEG and JPEG 2000 use a scheme called EXIF tags (see Section 6.3). In JPEG 2000, they are encoded in XML syntax.

The TIFF format dates back to the mid-1980s. It was developed by Aldus, the company that brought desktop publishing to the masses through its PageMaker application, as a standard image format for desktop scanner software. TIFF greatly exceeded this rather modest aspiration, and today it is still a popular choice for a variety of uses, including archiving and medical and scientific imaging. Through mergers and acquisitions, Adobe is now the keeper of the standard.

TIFF began as a format for bilevel images only, but over the years it has been extended to include, among other things:

- grayscale and color images
- different compression schemes
- a variety of color models
- layers and multiple pages within one file
- integer, floating point, and complex number representation for pixel values.

The current version of the standard, TIFF 6.0, dates back to 1992. It consists of a baseline implementation and a series of extensions. (Supplements to the specification were published in 1995 and 2002.) There are also two ISO standards based on version 6.0: one targeted at electronic photography (TIFF/EP), the other for more general image technology use (TIFF/IT).

In the baseline implementation, an image can be either uncompressed or represented in a form of Run Length Encoding (Apple's PackBits scheme), or it can use a variant of the CCITT standard one-dimensional compression scheme for facsimile transmission. The extended version includes CCITT's standard Group 3 and 4 fax compression schemes (which are two-dimensional) as well as LZW, JPEG, and many others. For color models, RGB and an indexed palette are the principal options in the baseline, and CMYK (mentioned above) and YC_bC_r (see [Section 5.4](#)) form part of the extension. Unlike the other image formats discussed above, TIFF can represent pixels as float-point numbers to a high degree of precision, making it a useful format for medical and scientific images.

TIFF, which originally stood for tagged image file format, is based on the notion of *tags*, and is an aspect of the format that is exploited in a digital library by treating them as a form of metadata. The actual set of tags used is described in Chapter 6 (Table 6.5). Tags can include file offsets to other tags in the file (with a few caveats), essentially allowing a nested system of tags. This makes the file structure significantly different from the other image file formats, which, as we shall see, can be both a blessing and a curse.

The baseline defines a fundamental set of tags, such as ImageWidth and ImageLength, that determine how to decode the basic image data—the layout of the pixels. The extension part defines new values for use in existing tags—this is how extra compression schemes are included. It also defines new tags, such as PageName and PageNumber, for use in multi-page images, and JPEGTables, which represents quantization and Huffman table information associated with a JPEG representation. As explained in Section 6.3, private tags can also be used; like the audio formats AIFF and WAV, unrecognized tags are ignored.

Early TIFF software libraries were notorious for being unable to parse all possible variations. Perhaps this is understandable given the number of extensions, but some of the blame can be attributed to poor programming. One large problem concerned how computers represent numbers—the order of the bytes used to represent numbers greater than 256, whether left to right or right to left. The TIFF standard states that implementations are free to choose either form when *writing* a file, but to conform to the baseline, they must be able to *read* both forms. Sadly, this was not always the case, despite the fact that only a few lines of code are required. Other compatibility issues concerned the different forms of compression used: to be universally applicable, a decoder must implement all possible decompressors.

Paradoxically, this led to a situation where, to use TIFF files in, say, a digital library project, they were first read by a comprehensive TIFF decoder and immediately rewritten as TIFF files using a configuration of the standard that was compatible with the other software components in the system. Such issues led wags to aver that TIFF actually stands for “Thousands of Image File Formats.” Today the situation is much improved, and the most common warning seen in TIFF processing software is about unrecognized private tags.

The file-offset mechanism used by tags to access the pixel data creates a potential security vulnerability: if software has not been written carefully enough, buffers can overflow. Indeed, this so-called “jail breaking” is how the iPhone and iPod Touch were first opened up to accept third-party applications. If an iPod owner used the Web browser included on the device to visit a particular URL that

displayed a deliberately mis-configured TIFF image, the iPod executed embedded code that opened it up for use with other software. Writing third-party applications became a cult phenomenon—and even Apple engineers were rumored to be turning up at work with jail-broken iPods because they could get such neat applications for them! Bowing to popular demand, Apple eventually condoned the development of applications outside the company by releasing a Software Development Kit.

Neither JPEG 2000 nor TIFF is widely supported by Web browsers, although Apple's Safari is a notable exception. The image formats supported by all mainstream browsers are GIF, JPEG, and PNG. Plug-ins are available to enable browsers to show other formats, but displaying JPEG 2000 or TIFF images directly is a bad idea for a digital library that is supposed to be universally accessible. As noted earlier, the strength of these formats is in their use for archiving. For example, the Papers Past digital library presented in Chapter 3 uses TIFF as the definitive digitized version of all images and converts them to GIF and PDF as necessary for display. In a similar project by the Singapore National Library, JPEG 2000 was selected as the definitive format, and appropriate screen-resolution images are generated on demand.

A digital library of photographs

We now explore the capabilities of a typical digital library of photographs, the Historic Campus Architecture Project developed by the Council of Independent Colleges (CIC) in the United States. The library's site gives information on 2,000 places of historical significance on private college and university campuses, and it includes 5,000 images, accompanied by rich metadata.

To create the collection, 750 colleges were asked to provide source material; about half actually did. Participating colleges received a questionnaire and details about how to prepare the images. Both questionnaire and black-and-white or color photographs could be returned in hard-copy or electronic form. At least one photo per location was requested, with an upper limit of five.

For the hard-copy photographs:

- Pictures had to be between 4×6 and 8×10 inches.
- Each picture had to be clearly labeled with place (building or site) name followed by institution name.
- If photographs had to be returned to the sender, a self-addressed envelope had to be enclosed.

For digital photographs:

- Images had to be captured at 300 dpi.
- Each image had to be stored as a JPEG file.
- The filenames had to take the form "Current name of place, institution.jpg."
- Data had to be burned to CD-ROM using CD-R format.

In the case of campus maps, participants were encouraged to submit in PDF format rather than JPEG, because campus maps are typically born digital and contain resolution-independent vector graphics and text. A team of ten part-time staff, whose tasks were coordinated by a digital librarian, worked

on the project. Their tasks included bibliographic research, correspondence, data collection, input, editing and evaluation, database development, document scanning, and design development.

The result is shown in [Figure 5.15](#), which walks through an interactive session (for conciseness, not all screenshots are shown). A full-width banner at the top, which helps brand the site, is followed by the principal navigation bar. This prominent display of host information—a reminder to end-users of who did the hard work of constructing the digital library—recurs throughout the examples in this book. Hovering over the *About*, *Browse*, and *Search* buttons brings up menus that offer more detailed choices, while to the right is a search box.

The home page in [Figure 5.15a](#) features a different picture each time it is viewed. Here the user has hovered over *Browse*, and the choices are to list institutions by place or state, or to list places by type, designer, building style, function, or time period. Clicking the time period option brings up a page of hyperlinks from which 1900–1945 has been selected to yield [Figure 5.15b](#). This lists all relevant places alphabetically, in two columns. The alphabet is a further navigational aid: it takes the user directly to places beginning with a selected letter.

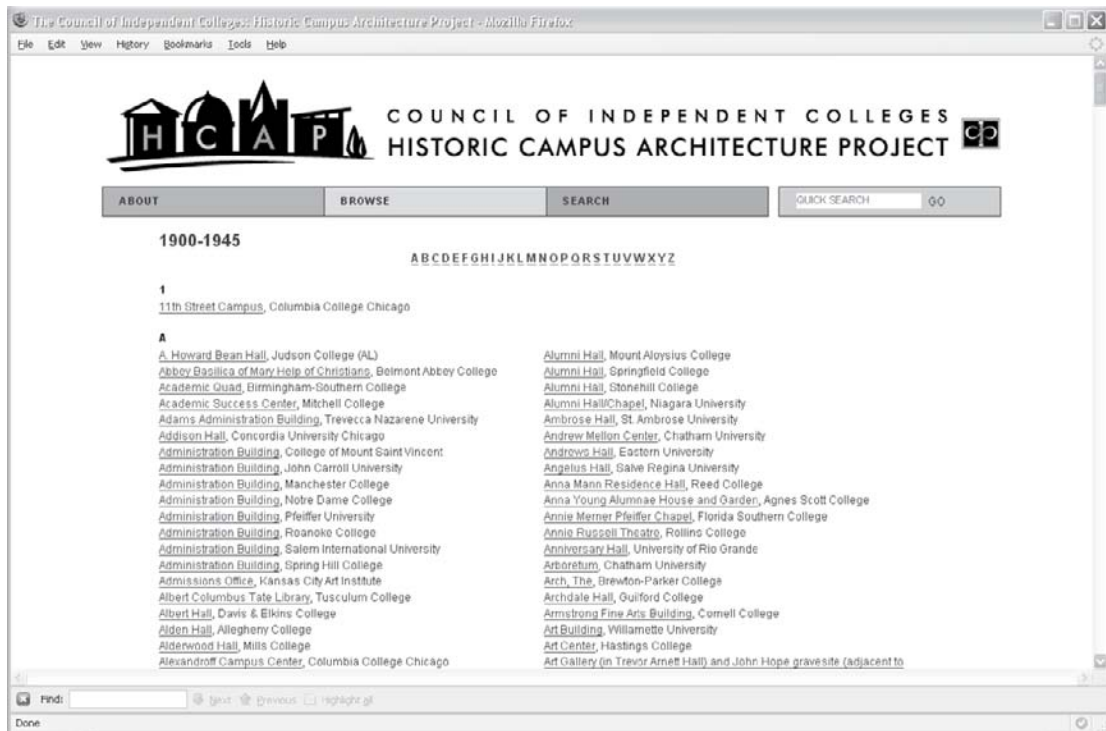
Clicking on Alexandroff Campus Center, part of Columbia College, Chicago, leads to [Figure 5.15c](#), which represents a source document. It displays textual metadata about the location (in this case a building): when it was built, what materials were used, what functions it has served over the years, and so on. Further down is a narrative account of the building (partially shown), references to publications that mention it, and pointers to further information elsewhere. Clicking the image of the building in the upper left corner yields a higher-resolution version.

Hovering over the *Search* button on the ever-present navigation bar gives a menu of options, including advanced search and state/region search. Advanced search (shown in [Figure 5.15d](#)) is a traditional form with dozens of options. In our case, the user backs out and selects state/region search, leading to [Figure 5.15e](#), which shows a map of the United States. Users can add individual states or regions to the search by clicking them. They are automatically highlighted in the textual menu that appears to the right—in the figure the user has selected New York and Pennsylvania. Once satisfied, the user clicks the search button, which produces a page like that of [Figure 5.15b](#)—only this time, instead of letters of the alphabet, the main headings are the selected states, and places relating to that state are listed alphabetically in two columns. As before, clicking on a place leads to a target document that displays photos and detailed metadata.

In the scenario above, the user backed out of the advanced query page (unsure, perhaps, as to what search terms to use or results to expect). This exemplifies a common problem that digital libraries must address. Whereas you can judge the size of a physical collection by simply looking around, this is not the case for the millions (trillions?) of digital bits that constitute a digital library. The relevant cues, some of them subliminal, are simply absent. One partial substitute is a range of different browsing tools—and in this case the user went to a geographical browser that allowed him to see in a few clicks that there are about 30 entries for New York and 40 for Pennsylvania. Another tool is the description of the scope of the collection, and how it was created, that appears on the home page ([Figure 5.15a](#)).



(a)



(b)

Figure 5.15: The Historic Campus Architecture project: (a) home page; (b) browsing by time period 1900–1945;

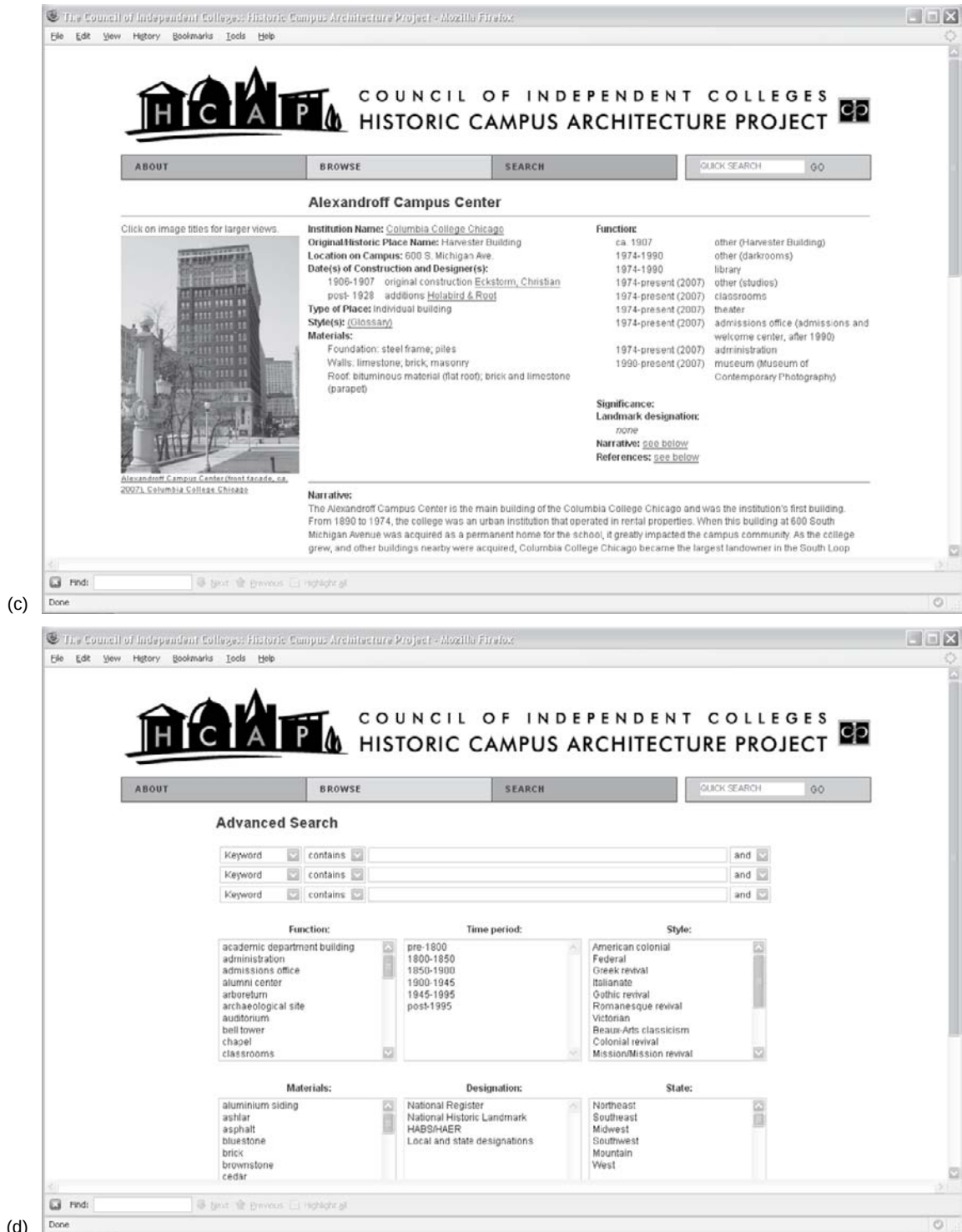


Figure 5.15, cont'd: (c) drilling down to a particular document, the Alexandroff Campus Center; (d) advanced search page;

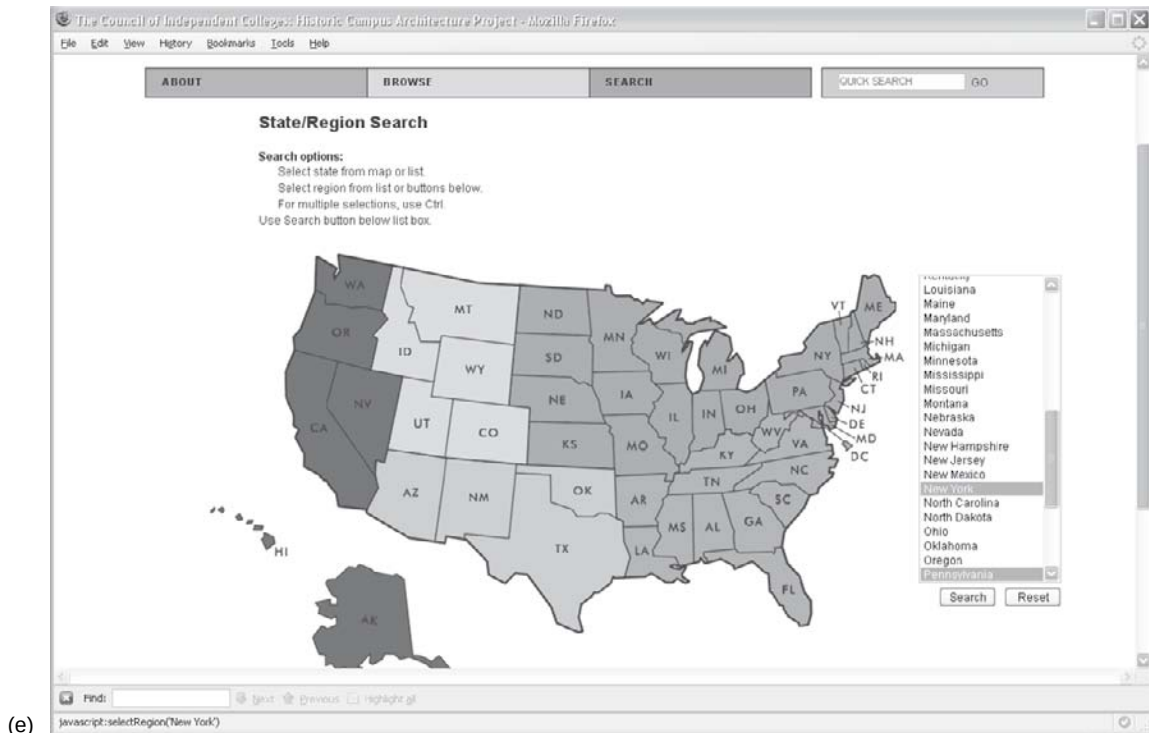


Figure 5.15, cont'd: (e) map-based searching by state

Vector graphics images

Scalable Vector Graphics (SVG) is an emerging standard from the World Wide Web Consortium. We describe it here because what it produces is ultimately rendered as an image, but it could have equally fitted into other sections of the book because it has many aspects in common with other formats as well. Its imaging model resembles that of page description languages like PostScript and PDF (Section 4.5). Unlike these formats, however, SVG is expressed in XML and has close ties with HTML and Cascading Style Sheets (Section 4.3). It also includes audio and video capabilities like the rich media described in Section 5.5. We begin with simple examples that convey the essence of the format, then fill out the description of its capabilities. Following that, we discuss its relevance to digital libraries.

Figure 5.16 recasts the welcome example from Chapter 4 in the SVG format, including both the rendered image (Figure 5.16a) and the XML that produces it (Figure 5.16b). The content is specified in lines 9–13, each of which defines a `<text>` element whose `x` and `y` attributes specify the starting position of the text that appears between the tags. The origin of coordinates is the top left-hand corner: `x` increases from left to right and `y` from top to bottom. Here values are expressed in pixels, but other units can be used: absolute measurements, such as centimeters and inches; relative ones, such as *em* and *ex*, which relate to the size of the font (in printer's points); and percentages. Units are expressed as a suffix—20 mm instead of 20. The *ä* in *Akwäba* is encoded as the XML entity `é`.



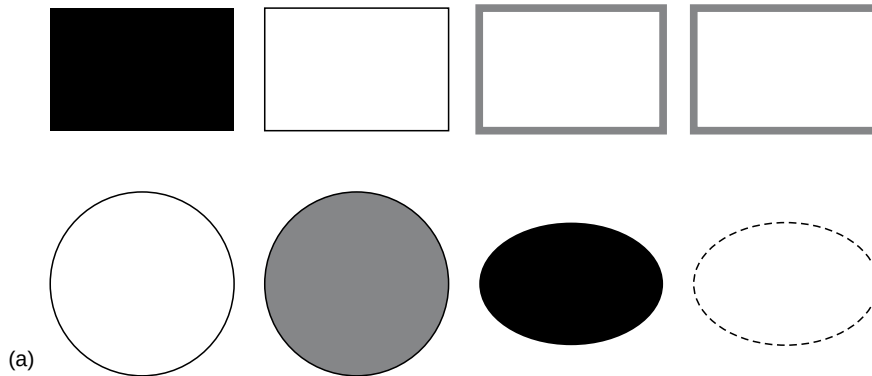
Figure 5.16: The Welcome text example expressed using SVG: (a) rendered; (b) underlying XML

The text is specified as a group using `<g>...</g>` with font and text size attributes, which are inherited by the group members. Top-level metadata is provided by title (`<title>`) and description (`<desc>`) elements; other XML-based metadata schemes can be specified using XML's namespace mechanism. Lines 1–2 are a standard preamble of the kind that we encountered frequently in Section 4.3.

Figure 5.17 illustrates vector shapes and styling. Rectangles, circles, and ellipses have been drawn with different fills and borders: none, single-pixel black border, 5-pixels red border, and dashed. Rectangles are specified by *x* and *y* coordinates of the top left corner and *width* and *height* (lines 8–11 of Figure 5.17b). Circles are given by their center (*cx* and *cy*) and radius *r* (lines 13–14); ellipses include separate radii for *x* and *y* (*rx* and *ry*). Defaults are defined for all attributes, which is why the first rectangle (line 8) comes out solid black with no border. The dashed ellipse is produced by the *stroke-dasharray* attribute (line 17), which takes a comma-separated list of numbers as the length of dashes and gaps. Lines 10 and 11 specify the same border style, but in different ways. Stylistic aspects can be expressed either as individual attributes (line 10) or as a single *style* attribute that gives them in a single string conforming to the Cascading Style Sheets (CSS) standard (line 11). As in CSS, color information can also be specified in different ways.

Figure 5.18 shows some lines. The first row gives single line segments, defined by their beginning and end coordinates as attributes *x1*, *y1* and *x2*, *y2* respectively (Figure 5.18b, lines 12–14). To avoid repeated specifications of width (2 pixels) and color (black), a style *element* called *.plain-line* is defined (lines 5–7)—again conforming to CSS syntax. The first two lines use this prepackaged style through the *class* attribute. The third exhibits rounded caps at the ends, with a width of 10 pixels.

The second row of Figure 5.18 shows multiline segments. The key attribute of a `<polyline>` is *points*, giving a list of vertex coordinates that the line joins up (Figure 5.18b, line 17). The commas in the list are optional, included to emphasize the coordinate pairs. The *class* attribute controls how the line is rendered (black and 2 pixels thick). A `<polygon>` is the same, but is closed by drawing a line from the end point back to the start.



```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
   "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
3 <svg width="100%" height="100%" version="1.1" xmlns="http://www.w3.org/2000/svg">
4
5   <title>Selection of SVG Shapes</title>
6   <desc>Demonstration of rectangle, circle, and ellipse.</desc>
7
8   <rect x="20" y="20" width="120" height="80"/>
9   <rect x="160" y="20" width="120" height="80" fill="none" stroke="black" />
10  <rect x="300" y="20" width="120" height="80" fill="none" stroke="red"
    stroke-width="5"/>
11  <rect x="440" y="20" width="120" height="80" style="fill:none;
    stroke-width:5; stroke:red"/>
12
13  <circle cx="80" cy="200" r="60" fill="none" stroke="black" />
14  <circle cx="220" cy="200" r="60" fill="rgb(255,0,0)" stroke="black" />
15
16  <ellipse cx="360" cy="200" rx="60" ry="40"/>
17  <ellipse cx="500" cy="200" rx="60" ry="40" stroke="black" fill="none"
    stroke-dasharray="5,3"/>
18
19 </svg>

```

(b)

Figure 5.17: Style properties used with basic shapes in SVG: (a) rendered; (b) underlying XML

The `<path>` tag is a very versatile construct. Line 21 uses one to define the same `<polyline>` as before. It specifies a sequence of commands: *M* for move (to the initial point), *L* for line (to the next point). If a command is not specified, the previous one is used again, thus the *L* command can be followed by a sequence of points. The *C* command specifies a smooth curve that passes through the first and last points and bends towards the intermediate ones (a Bézier curve). In fact, *C* stands for cubic, and specifies the degree of the polynomial used in the mathematics that renders the curve; *Q* yields a quadratic one.

The example also demonstrates how the `<g>` tag can be used to apply geometric transformations, such as translation and scaling (lines 16, 20, and 24). To emphasize their similarity, the polylines were drawn with exactly the same local coordinates and then were transformed to the desired final location. Transforms can be nested: on line 9 a scaling operation is used to double the resolution of the entire drawing.

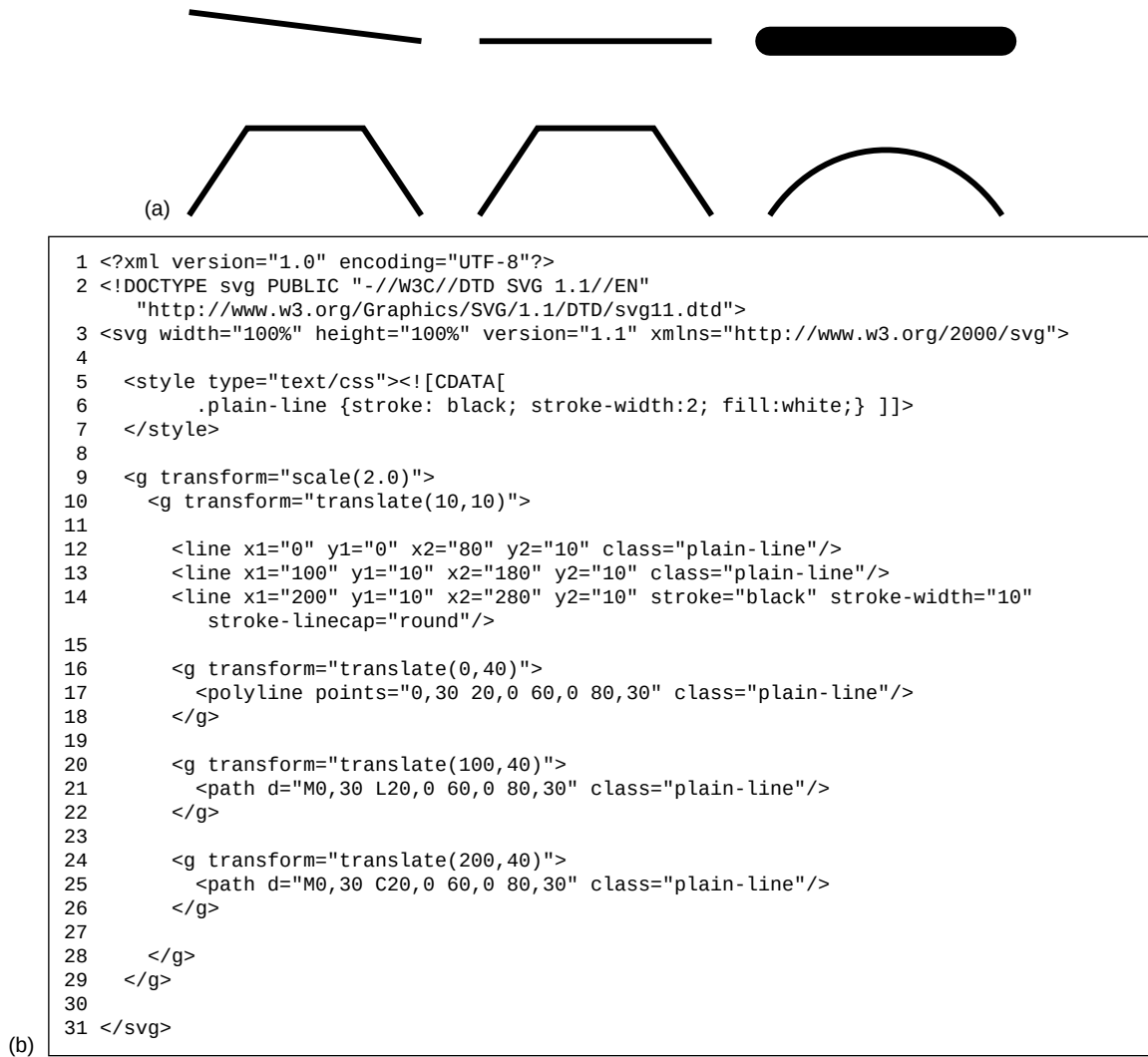
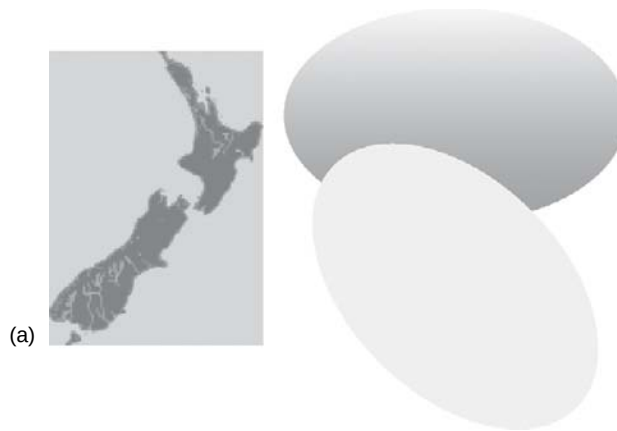


Figure 5.18: SVG line types: (a) rendered; (b) underlying XML

Figure 5.19 shows a raster image (a map of New Zealand that includes major waterways); it also demonstrates how shapes can overlap with controlled opacity and gradient shading and exhibits some interactivity in the form of a hyperlink. Lines 6–9 of Figure 5.19b include the image using the XML XLink specification (briefly mentioned in Section 4.3); line 4 gives its namespace. SVG's `<image>` tag (like HTML's `<img>` tag) specifies the image's width and height and gives the file's location using `href` (in this case scoped by the XLink namespace). The top-left coordinate position is defined by attributes `x` and `y`. Additional textual information is supplied through `<title>` and `<desc>` tags. Neither is displayed in the final graphical image, but a display program could use this information, for example, in tool tips.



(a)

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
   "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
3 <svg width="100%" height="100%" version="1.1"
4     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
5
6   <image x="10" y="30" width="100px" height="138px" xlink:href="nzmap.png">
7     <title>Map of New Zealand</title>
8     <desc>Raster image of New Zealand showing the major waterways</desc>
9   </image>
10
11   <defs>
12     <linearGradient id="mygradient" x1="0%" y1="0%" x2="0%" y2="100%">
13       <stop offset="0%" style="stop-color:yellow;stop-opacity:1"/>
14       <stop offset="100%" style="stop-color:red;stop-opacity:1"/>
15     </linearGradient>
16   </defs>
17
18   <ellipse cx="200" cy="60" rx="80" ry="50" style="fill:url(#mygradient)"/>
19   <ellipse cx="240" cy="100" rx="80" ry="50" style="fill:green;opacity:0.5"/>
20
21   <g transform="translate(200,140) rotate(45)">
22     <a xlink:href="http://www.greenstone.org">
23       <ellipse rx="80" ry="50" style="fill:yellow;" />
24     </a>
25   </g>
26 </svg>

```

(b)

Figure 5.19: SVG advanced features: (a) rendered; (b) underlying XML

The next main section (lines 11–16) uses a `<defs>` element to define a linear gradient labeled *mygradient* that changes from yellow to red. This is used as the fill style of the first ellipse (line 18). The syntax is a little clunky: a URL is constructed that points locally to itself and picks out the element with the *mygradient* identifier, namely `fill:url(#mygradient)`. The second ellipse is given a semi-transparent (opacity 0.5) fill color of green (line 19). The third one is rotated and translated to overlap the others (line 23)—note that objects are drawn in the order in which they appear in the specification. This final ellipse is defined to be hyperlinked (to the home page of the Greenstone

project, www.greenstone.org). When this file is displayed by an SVG-capable Web browser, such as Firefox, the cursor changes shape when moved over the top-most ellipse to indicate the hyperlink, whereupon clicking loads the target Web page.

Many important concepts in SVG are not covered by these examples, such as:

- filters
- markers
- fonts
- animation
- scripting.

Filters are pixel-based raster operations, such as convolution, which can be used to introduce effects like blurring and embossing. This allows Web designers to create decorative images that are rendered with shadows and depth. Filters allow such effects to be achieved within a semantically interpretable structure.

Markers attach symbolic names to sets of drawing instructions and can be used to define repeated shapes, such as icons or arrowheads. Like PDF and PostScript, SVG allows users to define (and augment) font information to achieve greater portability. Animation includes timed events, so that text or shapes can appear or change over time. Interaction can be far more sophisticated than the linking example above: JavaScript can be included that manipulates the Document Object Model (DOM), yielding a rich environment in which to develop interactive features. For example, the arcade game *Space Invaders* has been implemented in SVG!

SVG is a sophisticated format that stores images in a structured way, which can in principle support the kind of searching and browsing operations that mainstream digital libraries provide. However, it is not universally supported by browsers—although the fact that it is published as a W3C specification avoids the anarchic browser wars that plagued HTML's early development. Unfortunately, some major browsers do not provide direct support. Internet Explorer can display SVG only with the help of a plug-in, and Adobe has flagged its intention to stop supporting its SVG application during 2009.

Another complication for digital library software is caused by the proliferation of different versions. At the time of writing there are three versions of the SVG specification. Versions 1.0 and 1.1 are official W3C recommendations, and there are only modest differences between them. Version 1.2, which is in a formative stage, includes a larger set of changes. SVG allows profiles, which are subsets of the standard targeted at particular user communities, to be defined. Alongside the Full profile, the Tiny profile is targeted at devices like cellphones, while the Basic profile addresses more powerful mobile devices, such as PDAs. For version 1.2 the Tiny profile is an official W3C recommendation, whereas the Full profile is still at the stage of a working draft. Because digital libraries can include documents from a wide range of sources, built up over time, they often need to deal with many possible variants. This places greater demands on the software implementation, demands that are not normally difficult to meet in practice provided that the variations are well documented. In the case of SVG, the version number and profile are clearly encoded in the XML.

5.4 Video

Now that we know how to deal with audio and images, let's look at video—which, at least superficially, is a marriage of the two. Three important concepts are:

- containers
- codecs
- streaming.

A multimedia container combines and synchronizes different media streams into a single file. Codecs isolate the compression scheme used from the container format, allowing different codecs—perhaps from different vendors—to be included. Streaming affects how the content is communicated over a network to a Web browser.

The dominant international standard for audiovisual material is MPEG, devised by the ISO Moving Picture Experts Group, and it is the principal format described here. We also touch upon proprietary formats, as well as the emerging royalty-free standard, Ogg Theora from the Xiph Foundation.

Codecs

Video requires at least 24 frames per second for the eye not to perceive flicker. As we have already learned, sample rates for audio range from 8 kHz for telephone-quality speech to 44.1 kHz for CD-quality music. Faced with this volume of data, all practical video formats use compression to maximize the amount that can flow through a given network or be stored in given disk space.

Documents are coded for transmission or storage and decoded upon reception or retrieval. The coder/decoder (equivalently, the compressor/decompressor) is called a *codec*. Audio and video are handled separately. The compression schemes for audio (described in [Section 5.2](#)) form the basis for audio codecs. Video codecs achieve compression by exploiting spatial and temporal coherence in the sequences of images to encode—just as audio codecs exploit acoustic and temporal redundancy. The decoder reverses the process. High compression rates can be achieved, but in practice the process is lossy. Lossless methods could be used, but their gain is small in comparison.

Many codecs are asymmetrical in that they take different times (or other resources) for encoding and decoding. Often, greater compression can be achieved by expending more effort on coding. For example, most video footage is generated once but viewed many times, or broadcast once and viewed at many locations. In both cases, high compression brings great benefits. For online operation, special hardware can assist with encoding (and, less frequently, decoding). Manufacturers produce a wide range of hardware options for desktop computers—right through to video editing studios.

Some multimedia architectures allow encoders to be upgraded to take advantage of new techniques, but remain backward-compatible by ensuring that unmodified decoders can still decompress the data they generate. We previously encountered this idea when describing MP3. The upgraded system produces shorter (or better quality) files that can be played back by existing decoders on other sites.

Multimedia compression: MPEG

The vision of MPEG, formulated in 1988, was an open digital format, comparable to the popular analog home video system VHS, that could be read from storage devices such as CD-ROM. The initiative snowballed, attracting great interest from industry, and its goals expanded. Now several formats constitute the MPEG family. MPEG-1, MPEG-2, and MPEG-4 form the core; MPEG-3 has been abandoned; MPEG-7 incorporates metadata; and MPEG-21 supports the life cycle of multimedia information.

The original format, MPEG-1, was designed for color pictures with 352×240 pixel resolution shown at 30 frames a second, accompanied by audio of near CD quality. Not only did this come within the 1.5 Mbit/s target, but also it could be decompressed in hardware with 512 KB of memory—which corresponded to cost-effective VLSI design in the early 1990s. The frame rate, picture resolution, and audio sampling can be varied to give other bit rates, although they are less common. Data can be unimedia, comprising just sound or just pictures. (MPEG Audio is described in [Section 5.2](#).)

MPEG-2 is designed for higher-quality video representation and forms the basis of DVD video. It can be transported over error-prone networks and supports interlaced images (making it suitable for broadcast TV) and multichannel audio. Decoders are backward-compatible with MPEG-1 and can play back both formats. However, MPEG-2 has not superseded MPEG-1, because the earlier standard's performance is superior at lower bit rates. Users choose whichever version best suits their needs. The next standard, MPEG-3, was intended for high-definition television (HDTV), but it transpired that the MPEG-2 structure scaled well enough, and the new work was merged with the existing standard.

MPEG-4 is designed for low-bandwidth networks, such as mobile communications and the World Wide Web. It is based on objects, rather than signals, and it includes synthetic objects, such as computer-generated text, graphics, and sound. It incorporates interactive capabilities, and the bit rate can be adjusted dynamically to suit the transportation medium.

Beyond MPEG-4, versions are not numbered sequentially. MPEG-7 allows metadata to be defined for multimedia content that is conveyed by MPEG-1, 2, and 4 (see [Section 6.3](#)). MPEG-21 is intended to provide a framework that integrates all elements of the multimedia life cycle, from content creation and production through distribution to end-users. Its designers liaise with other standards bodies, drawing upon existing standards and defining complementary ones to bridge the gaps. In the context of digital libraries, MPEG-21 amounts to a form of interoperability for multimedia documents.

Inside MPEG

Before digging into MPEG's internal details, we briefly review pertinent television standards for color, sample rate, and picture resolution. MPEG-1 was influenced by prior work on visual telephony, by the JPEG format for image data, and by various audio techniques and the video coding standard H.261 by ITU-T. It in turn has influenced proprietary formats, and these commercial decisions have fed back into subsequent MPEG development.

Table 5.2: International television formats and their relationship with CCIR 601

Format	Abbreviation	Transmission rate	Origin	CCIR 601 rate
National Television System Committee	NTSC	525 lines at 60 Hz (interlaced)	United States	720×486 (interlaced)
Phase Alternation Line	PAL	625 lines at 50 Hz (interlaced)	Europe	720×576 (interlaced)
Séquentiel Couleur Avec Memoire (Sequential Color with Memory)	SECAM	625 lines at 50 Hz (interlaced)	Europe	720×576 (interlaced)

MPEG arose out of the same ISO working group as JPEG, but they quickly separated. Although an independent standard, MPEG reuses many JPEG concepts and idioms. Indeed, there is a variant of JPEG called Motion JPEG or Moving JPEG, often used for video editing, which extends this standard to support videos by storing each frame in JPEG format—but this is quite different from MPEG.

Part of MPEG is based on a downsampled version of an international standard used in digital television and video equipment, CCIR 601, which was designed to be compatible with long-established analog formats. Table 5.2 summarizes the principal analog television formats used around the world. In an NTSC environment, for example, CCIR 601 delivers a 720×486 picture that is interlaced, sending 720×243 half-frames 60 times per second. Because analog NTSC video reserves many lines for control information, CCIR 601's effective display area is the same, even though it specifies fewer lines. Analog TV itself is being phased out around the world, in favor of HDTV; however, the legacy it has established is still apparent at times in these newer standards.

CCIR 601 represents color using the YUV model rather than the RGB (red-green-blue) scheme commonly used by computers. (There is a mathematical mapping that converts between the two representations.) The YUV model is motivated by how people perceive color. It encodes brightness (luminance) in the Y parameter, and color difference (chrominance) in the parameters U and V. YUV is preferred over RGB because the resolution for chromatic parameters can be reduced with no perceptible loss of viewing quality, and therefore chromatic parameters can be expressed more compactly.

A closely related color model is $YCbCr$, which is nearly identical but uses different scale factors. Historically, YUV comes from the analog world and $YCbCr$ from the digital one—we have already encountered the latter in the description of the TIFF image format. Increasingly, the distinction between the two has become blurred, especially in informal descriptions on the Web. For any given case, look to the official standard for the precise definition of what is meant.

The three YUV components can be quantized into the same number of bits (e.g., 8 bits, or 10 bits for higher color resolution). Alternatively, the U and V components can be downsampled—that is, transmitted every second (or fourth) sample in the horizontal direction, or every second sample in both horizontal and vertical directions. To reconstruct the image, intermediate values are interpolated.

In MPEG terminology, a picture comparable to analog television is obtained by delivering, noninterlaced, a 352×240 pixel color image at 30 frames/s (NTSC) or 352×288 pixels at 25 frames/s (PAL). Taking into account rounding to the nearest multiple of 8 to simplify implementation, this halves the resolution and sample rate of CCIR 601 and is known as the Standard Interchange Format (SIF). Both settings conveniently yield the same data rate in pixels/second (because $352 \times 240 \times 30 = 352 \times 288 \times 25 = 2,534,400$).

Although PAL and NTSC videotapes are physically the same size, they require different equipment for playback (of course, dual players exist). For MPEG, the NTSC and PAL flavored SIF settings are purely for convenience—they do not pose a problem for playback because the differences can be overcome in software. MPEG data that conforms to one of these variants most likely reflects the capture equipment used.

MPEG-1

We begin by describing MPEG-1. The general ideas carry over to MPEG-2 and MPEG-4; we summarize the differences later. There are five parts to the MPEG-1 standard:

1. systems
2. video
3. audio
4. conformance testing
5. reference software.

Having already described audio in [Section 5.2](#), here we focus on Part 2 of the standard and describe the structure of a video stream. Then we give a brief overview of Part 1, which shows how to combine audio and video streams together.

MPEG represents video as a progression of graphical frames, processed into a bitstream. The frames are not necessarily in the same order as the sequence of pictures in the original video. There are three types: intra frames, predicted frames, and bidirectional frames.

An *intra frame* or *I-frame* provides a fixed point in the image sequence. It exploits spatial but not temporal coherence. In rough outline it follows JPEG: a discrete cosine transform is applied to 8×8 blocks, values are Huffman coded using fixed tables specially designed for two-dimensional data, and the zero-frequency coefficients are differentially encoded.

A *predicted frame* or *P-frame* is derived from the most recent I- or P-frame. It encodes differences between frames and therefore exploits temporal as well as spatial coherence. P-frames are compressed using motion predictors that work with 16×16 macro-blocks in the luminance (Y) channel and can exploit similarity in data translated between one frame and the next.

A *bidirectional frame* or *B-frame* is based on a previous frame (most recent I or P), a future frame (next closest I or P), or the average of the two. The data is compressed along similar lines as in the P-frame.

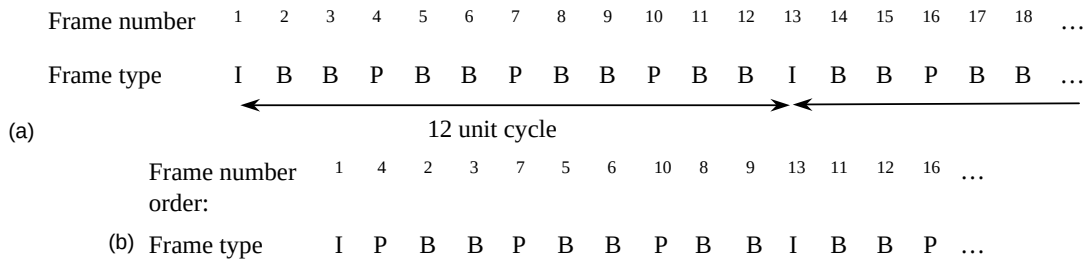


Figure 5.20: (a) Frame sequence for MPEG; (b) reordering for sequential transmission

I-frames allow a form of random access playback. Because they contain all the information necessary to reconstruct that particular image, and subsequent frames (P or B) cannot reference a frame further back in time, a playback application can skip to an I-frame and start playing from that point without needing to access any additional information.

A Group of Pictures (GOP) is defined as the set of frames from one I-frame to the next and is an important structure in a video stream. Figure 5.20a shows a cyclic pattern of 12 I-, P-, and B-frames that works well in practice. Assuming a rate of 30 frames/s, with this configuration the display can be restarted every 0.4 seconds, and this can be used to provide fast-forward or rewind cue-frames. For easy decompression, however, the frames are not encoded in numerical order. Frames 2 and 3, for instance, rely on frames 1 and 4; consequently, they are reordered as shown in Figure 5.20b. Note that this makes the first I-frame cycle different from the next one. The first cycle starts with an I-frame followed by a P-frame, but the second one has two B-frames belonging to the previous cycle slotted between the I-P pair. This second pattern of frame numbers repeats until the final cycle. B-frames complicate life, but they also bring space savings and a degree of noise reduction. Implementers of MPEG encoders can choose not to use them, but decoders must be able to handle them.

The video side of MPEG follows the design philosophy of MP3: while decoding is a well-defined process, there is great scope for variation in encoders—particularly with regard to motion prediction. Implementing a good encoder is a specialized skill that involves far more than the familiar “speed versus compression” tradeoff. Deficiencies in design yield poor-quality video playback that no decoder, however smart, can fix.

Mixing media

Part 1 of the standard (System) defines the media container format. Video and audio bitstreams are combined by breaking each into packets and multiplexing them into a single stream. Packets are time-stamped to ensure correct synchronization.

MPEG-1 can cope with a range of audio sampling rates and a range of image frame sizes and rates. For instance, it is possible to generate video with dimensions $4,096 \times 4,096$ delivered at 60 Hz. However, special significance is given to particular sets of parameter values that provide good tradeoffs between decoding complexity, buffer size, and memory bandwidth while still addressing a

Table 5.3: Upper limits for MPEG-1's constrained parameter bitstream

Parameter	Limit
Pixels per line	704
Lines per frame	576
Pixels per frame	101,376
Frames per second	30
Pixels per second	2,534,400
Bit rate	1.86 Mbit/s
Buffer size	40 Kb

wide range of applications. One configuration is known as the *constrained parameter bitstream* (CPB). The upper limit of each parameter is shown in Table 5.3 (not all limits can be attained simultaneously). The maximum rate in pixels/s corresponds to the rate for the Standard Interchange Format (SIF) mentioned earlier. In a typical configuration, video uses around 1.2 Mbit/s, and audio around 0.3 Mbit/s.

MPEG-2

MPEG-2 incorporates several improvements. Images can be interlaced, audio can have multiple channels, data can be transported over an error-prone network, and delivery is scalable. Also included are low-level implementation enhancements that improve the overall compression factor (however, they restrict picture dimensions to multiples of 16). To achieve all this, the standard is divided into ten parts, the first five parts matching one-for-one with the previous standard.

Interlaced images make MPEG-2 suitable for broadcast television, but the additional complexity hinders compression. There are two audio extensions. One, which is backward-compatible with MPEG-1, provides five main channels and a low-frequency enhancement channel; a typical application is surround sound. We met the other, Advanced Audio Compression (AAC), in Section 5.2. It adds several intricate encoding options to provide superior quality of compression, but it is not backward-compatible.

MPEG-1 assumes that the bitstream is error-free, as when reading and writing to a CD. MPEG-2 extends streams to work in error-prone environments using fixed-size packets, as well as in error-free ones with arbitrary-sized packets. To support this, Part 1 (System) of the standard defines two separate container formats. The Transport Stream tolerates errors but expends more bits overall; the Program Stream does not tolerate errors but has the benefit of being more concise.

Data in MPEG-2 can be prioritized by separating it into constituent bitstreams, which helps support heterogeneous environments. This means a high-definition television (HDTV) signal could be, for example, decomposed into a 720×480 pixel baseline bitstream and a $1,440 \times 960$ pixel bitstream that encodes the differences between the two resolutions. This configuration enables either a standard-resolution (CCIR 601) or a high-definition television to display a suitable picture.

MPEG-2 defines many implementation options, and implementations need not support them all. For instance, part of the standard specifies a real-time interface for video on demand, but particular implementations may not include it. To manage such variations, *profiles* and *levels* are stipulated that define which aspects of the standard need to be supported to conform. Profiles target common patterns of use, such as a video player on a mobile device. Levels are used to establish upper bounds on data rates by restricting the permissible range of frame rate, maximum resolution, and the like. The Simple Profile, for example, does not allow B-frames, and what was called the Constrained Parameters Bitstream (CPB) in MPEG-1 is called Low in MPEG-2, one of four levels.

MPEG-4

Whereas MPEG-2 is an extension of MPEG-1—for example, its decoder can handle MPEG-1 data—MPEG-4 represents a radical departure. It embodies a broader concept of multimedia and is designed for use over the Web and low-bandwidth mobile networks. Video and audio are augmented with still images, synthetically generated graphics and sound, and text. These primitive data types can be composed hierarchically into objects. Objects supplant streams as the key structure, offering more expressive power than before. For example, footage can be created that composes live video (its background masked off to save space) with a two- or three-dimensional computer-generated backdrop and a synthesized music soundtrack.

MPEG-4 is a vast standard. There are over 20 parts! The first five again correspond to the original five in MPEG-1. Part 10 (Advanced Video Coding) pushes the boundary of low bit-rate encoding even further. Depending on the parts an implementation supports, MPEG-4 can deliver a quality of service that adapts to available network bandwidth. It allows users to interact with the composed scene. It lets users implement strategies for identifying and managing intellectual property. Not surprisingly, even more profiles are used to manage this complexity.

High Definition Digital Television

Both MPEG-2 and MPEG-4 are suitable for encoding HDTV. As with NTSC and PAL, different standards have emerged in different geographical areas. Depending on the part of the world you live in, you may encounter terms such as DVB-T, ATSC, or ISDB-T used in connection with this medium. MPEG-4 is increasingly the transmission format of choice, particularly for satellite broadcasters, since it enables them to carry more channels than if they used MPEG-2, for the same available bandwidth. In countries where standards were established early on, they tend to be based on MPEG-2, with subsequent versions shifting to a MPEG-2/MPEG-4 hybrid.

Proprietary formats

MPEG is an open standard, albeit one that includes patents, but with the proviso that they be licensable at a fair rate. Some other multimedia formats are fully proprietary, in particular:

- AVI and ASF from Microsoft
- QuickTime from Apple

- FLV from Adobe
- RealMedia from RealNetworks.

In the age of universal access, proprietary formats seem to defy commercial sense, because to play such material you must possess software from that company. Commercially speaking, however, the separation of multimedia into container formats and codecs changes the rules. Working with only pre-compiled modules of code, so as not to reveal commercially sensitive implementation details, a codec from Company A can be installed that works with a multimedia container format from Company B.

Codecs are where companies vie for dominance—particularly on the compression side—because software that can encode video content is a key source of revenue. Market dominance stems from superior capabilities for encoding speed, memory used, and quality of video reproduction. All the major vendors give away their video players, competing in a secondary battle to be the default player used for a particular format as this makes it easier to hook users into online services and retail opportunities.

The Audio Video Interleave (AVI) format is Microsoft's original media container format. Like WAV, it is based on the RIFF file structure. A file header contains a code that determines which codec to use to process the data that follows. Codecs (there are over 20) range from a simple device-independent bitmap that represents frames using RGB color values and performs no compression whatsoever, to sophisticated approaches like MPEG-4. A proprietary codec for a format derived from QuickTime (see below) is included—a popular choice in practice. AVI files are device independent, and there are players for all major computer platforms.

The long-established QuickTime format from Apple (also available for Windows) comes with a full programming environment. Independent software vendors can use it to parse multimedia files and to develop graphical applications that access the media interactively. A variety of codecs are supported.

Because of the plethora of different codecs, a player that recognizes the container format of a particular file may lack the necessary decoder module, prompting a diagnostic message. To handle this, most multimedia players offer to locate and install the missing codec automatically. In principle this is an ideal solution, with the added benefit of accommodating codecs brought out in the future. Unfortunately, errors often occur in practice. The cause can be as simple as the user having insufficient permission to update the location where codecs are installed on the computer, but the error messages encountered are often mystifying and cryptic—a sure sign that something has happened that the software developer had not anticipated.

FLV, the video format used by Adobe Flash, runs contrary to trends exemplified by other proprietary formats. First, it supports a small, closed list of codecs. This helps avoid the missing codec problem, but it has the potential to stifle innovation. Indeed, for a long time Flash only supported two codecs, both licensed from third parties. However, Adobe recently added support for MPEG-4 video. Second, Adobe does not provide a standalone player, but expects the interaction to be through a Web browser with the Flash plug-in installed. This allows a rich variety of interactive applications. For example, a

browser-based player can be built that is fully customizable in terms of features supported and button style and placement. YouTube is a well-known example of a Flash-based video player.

These differences give Flash distinctive properties with respect to digital library use, which we return to in [Section 5.5](#) when we describe the Flash format more fully. The Flash plug-in is freely available and is well supported (on Linux, MacOS, Solaris, Windows, as well as some mobile platforms), and most video players—such as RealMedia and the open source player VLC—can play FLV files.

Streaming

Application programs for multimedia were originally oriented for work on a single computer. However, today we expect multimedia objects to be available over the Internet, where bandwidth is restricted and unpredictable. Real-time delivery of multimedia over networks is called streaming: the information arrives in a continual stream rather than being received in its entirety before presentation begins. One common sign that video content is being streamed over the Web (rather than reading from a freshly downloaded file) is that the video player appears at once, followed by a message like *buffering 1%, 2%, ... 100%* that appears prior to playback. It then silently continues to download the rest of the video while concurrently playing the part it already has. Depending on the reliability and speed of the connection, playback may stick, and the buffering message reappears.

Against stiff competition from Microsoft and Apple, RealNetworks established itself in the multimedia market by concentrating on developing streaming software. Video, audio, images, and text can be embedded within its multimedia container format. When the software is combined with a RealMedia server, a rich set of features are possible, such as automatically starting the video at a given time offset, controlling how much to play, and displaying text at particular moments in time.

Microsoft provides streaming support in its Advanced Systems Format, ASF (previously called Advanced Streaming Format). ASF is a media container based on serialized objects. Various file-name extensions are associated with the format: beyond *.asf*, which is generic, *.wma* and *.wmv* specifically relate to audio and video, respectively. Such encodings rival MP3 and MPEG-4 in terms of quality at low bit rates. Apple has introduced streaming through successive updates to QuickTime—for example, MPEG-4 and QuickTime 6 are closely related. FLV can be streamed by injecting cue points into the file structure.

Ogg Theora

Ogg, from the Xiph foundation, is a multimedia container format designed to be free of patent issues. We met the lossy and lossless compression schemes Vorbis and FLAC in [Section 5.2](#); Theora is their video counterpart. Like MPEG-4, it is designed for low bit-rate encodings to encourage use over the Web.

It is founded on the same principles as MPEG: a block-based discrete cosine transform exploits spatial redundancy, motion prediction of macro-blocks between frames exploits temporal redundancy, and sub-sampling the YC_bC_r color space exploits perceptual redundancy. It makes use of intra and predictive frames, but not bidirectional ones. Specifically, Theora is based on ON2 Technologies'

patent-protected video codec VP3, which was made open source in 2001. The company decided to relinquish its patent rights through an agreement with Xiph that waives all licensing fees.

Using multimedia in a digital library

Audio, video, and multimedia digital libraries raise many interesting and challenging issues. Here are a few: How is the collection to be searched or browsed? How are users to access different parts of a document? What does it mean to provide a summary of an audio document, or a video document, or a multimedia document? Can summaries be produced accurately? How accurate does the digitization process need to be?

One approach is to build the indexing and browsing structures around textual metadata, and deliver multimedia material—once it has been located—through the Web browser. More ambitious is to combine signal processing algorithms (image, audio, or both) with graphical displays to provide a richer environment in which items can be located, played, or displayed and manipulated more directly. For truly flexible access, it should be possible to search and browse summaries of audio and video and to combine operations on different media. Developing usable tools for this remains a challenging research problem. In the short term, practical digital library systems will take advantage of whatever direct access methods are available, including the use of textual metadata as an intermediary means of access, while in the longer term they will come to grips with the true multimedia nature of the material and also cater to the differing needs of individual users.

It is not easy to digitize physical signals, and considerable expertise and attention to detail are needed. Good results depend on suitable lighting conditions and audio recording levels. Technical parameters include linear resolution, color depth, frame rate, and audio sample rate. The output file format affects how much detail is preserved. When designing a multimedia collection, a critical issue is whether lossy representations are acceptable. For some preservation purposes, no loss can be tolerated, even though this greatly increases storage requirements.

Lossless representations make editing operations easier—and safer. Although there are tools that permit playback and editing of compressed formats, quality inevitably suffers if the process involves lossy decompression followed by lossy recompression. Repeated editing may even render the material unusable. Audio editors commonly use formats like WAV, AIFF, and AU, whose rudimentary codecs suffer less from quality loss. Only when editing is complete is the data converted to a highly compressed format, such as MP3.

Two other issues arise: How widespread is the software that can replay the material, and what are the bandwidth implications for delivery? Early multimedia applications involved individual computers with material stored locally on disk. However, today's users expect cross-platform support and network access.

Rather than imposing a particular fixed solution, it is attractive to allow users to select a format and arrange for the digital library to convert the material on the server. This lets users access it in the most convenient form, taking into account the bandwidth available for downloading. This is the approach adopted by the Internet Archive's audio digital library example in [Section 5.2](#); we will see

it again in the video example that follows. An alternative is to standardize on one format and ensure that all users have the necessary playback software. This is hard to pull off without a locally constrained population of users, except perhaps through the wide cross-platform uptake of Flash, with its controlled use of codecs.

A video digital library

Figure 5.21 shows the Open Video Project at the University of North Carolina at Chapel Hill. The masthead, which is displayed throughout, brands the site and provides top-level links: *Home*, *About*, and *Contribute*. The home page in Figure 5.21a contains news and featured highlights on the right, and searching and browsing on the left. For searching there is a quick search and a hyperlink to detailed search, which yields a fielded search page like those in the audio and photo digital libraries above.

There is a rich selection of browsing options. Users can peruse the collection by

- genre: documentary, educational, ephemeral, historical, lecture, other, or public service
- duration, in a variety of ranges
- color or black/white
- silent or with sound.

The number of videos appears beside each category. Below this area are links to around twenty collections, most having an educational focus; in the example below we use the NASA Science Education Program. Beyond the home page, the main display appears on the right, with navigational items to the left: a search box (Figures 5.21b, c) and related videos (Figure 5.21d).

The user enters *mars* as a quick search term on the home page, searching across all collections to yield Figure 5.21b, from which subsequent pages of matching results are reached by a pull-down menu. The result set can be laid out in different ways: Figure 5.21c shows a thumbnail view with basic metadata; a grid layout can be chosen instead. The number of matches shown per page is controlled by a pull-down menu, and matches can be sorted by relevance to the query or by title, year, and duration.

Figure 5.21d shows the result of selecting the top-most item returned by the query, which is a storyboard format that automatically displays extracted video keyframes; alternatives include a 7-second excerpt that can be played (with audio) and a “fast-forward” gestalt version of the video (no audio). Links to the video in various formats (MPEG, QuickTime, and RealMedia) appear below, along with the download size. Further below (not visible in the figure) is a table of metadata.

Reflection

The digital libraries in Sections 5.2, 5.3, and 5.4 are remarkably similar. Although searching and browsing differ conceptually, they are often implemented by searching a database. To display titles when the user clicks letter A, the strategy is to simply retrieve all items that begin with A and gener-

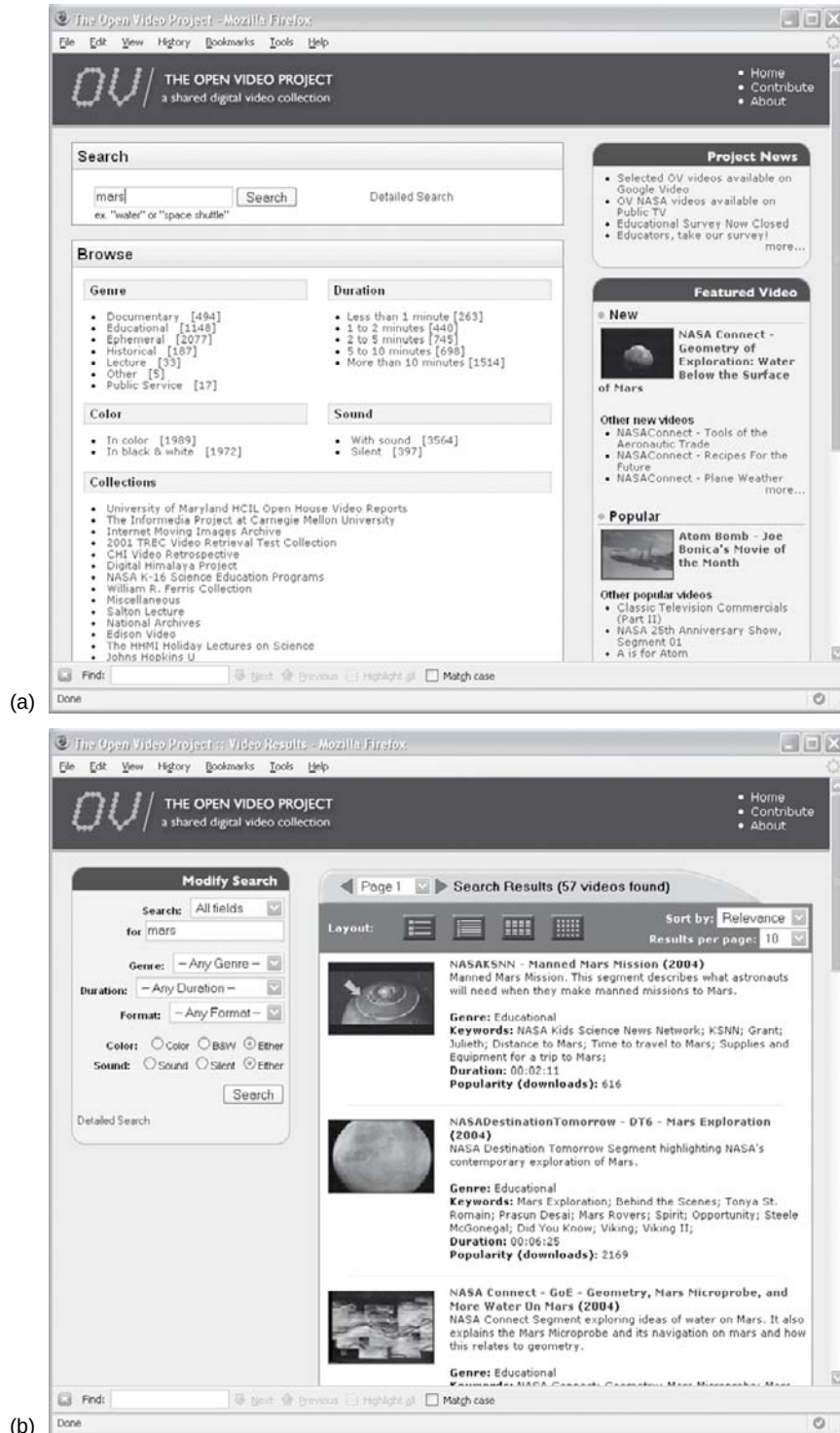


Figure 5.21: Open Video Project: (a) home page; (b) searching across all collections for *mars*;

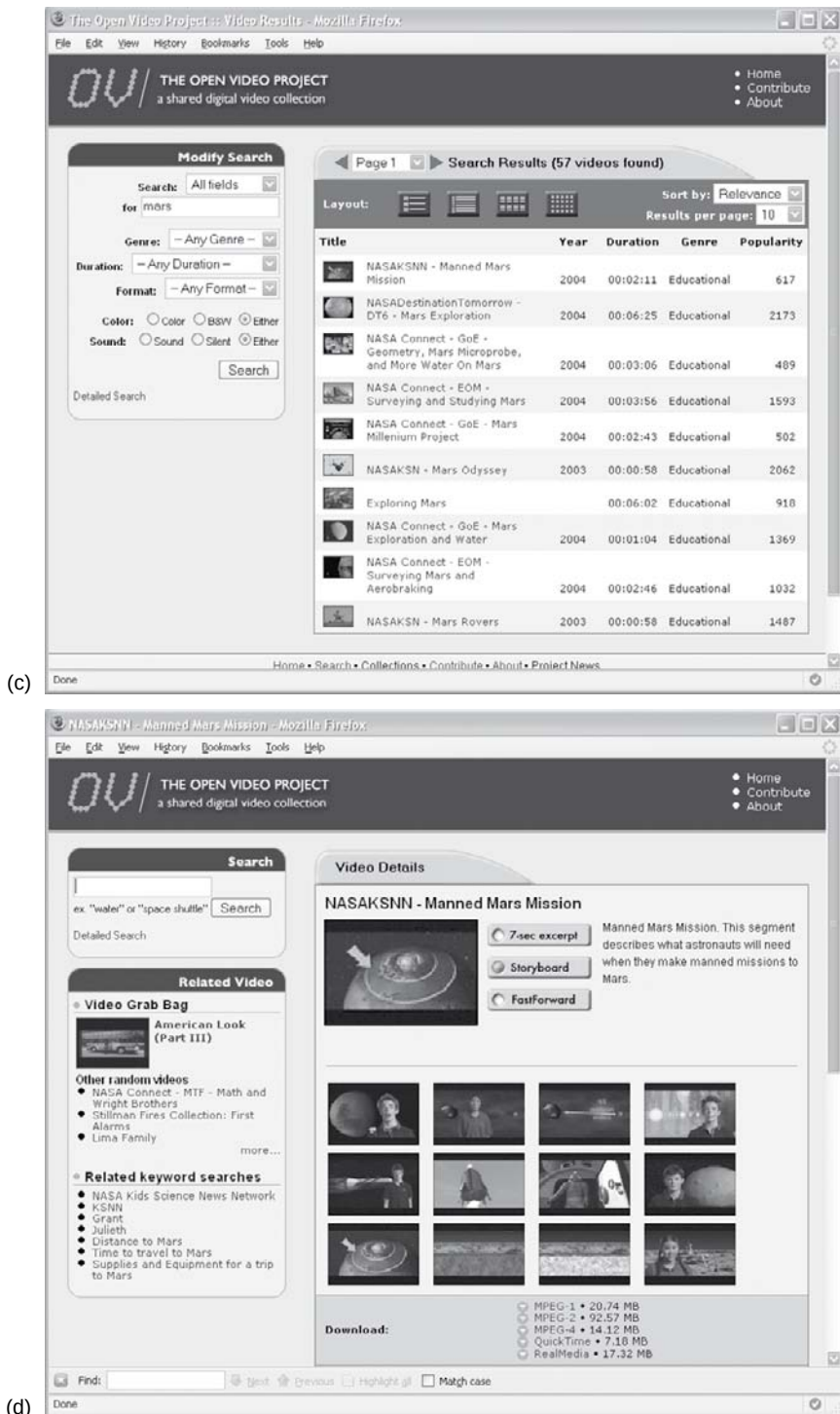


Figure 5.21, cont'd: (c) search result list (miniature thumbnail view); (d) document view, showing storyboard

ate a Web page based on that list. Employing the same strategy for searching simplifies things for users as well: it is easier to learn interfaces in which similar modes of interaction recur.

A note of caution is in order, however. If users are presented with the letters A to Z displayed across the page and some letters do not begin a title, the only way to find out is to click and get an empty page. Offering something and delivering nothing is a strong negative experience that should be avoided where possible. This is why search engines (and digital libraries!) try to avoid turning up empty-handed by offering suggestions like *did you perhaps mean ...* in response to fruitless queries (often ones involving typos). In the case of browsing in a digital library, with a bit of programming effort up front, it is possible to analyze the relevant metadata and display a sequence of letters that better reflects the available content.

5.5 Rich Media

Multimedia in the digital domain embodies concepts that have crossed over from the physical world of photographs, audio recordings, and film, and most of this chapter has been dedicated to detailing relevant representation formats. Rich media, the focus of this section, embodies a concept that follows on logically:

- heterogeneous media elements spatially composed, and
- synchronized over time, with
- the capability to respond interactively to user input.

Rich media has no counterpart in the physical world. Unfortunately it has been surrounded by a lot of hype, and in truth the notion is not particularly well defined. The above characterization is intentionally permissive. In a particular case, all media elements might have the same type (e.g., video), or a mix of images and video might respond to interactive user events rather than being explicitly synchronized over time. It is fair to say, though, that to count as rich media an artifact must exhibit at least two of the above three properties.

We have already encountered an example in [Section 5.3](#): SVG. PowerPoint presentations that incorporate timed or interactive animation also exhibit these properties, although they may not be counted as rich media by some because they are not strongly integrated with Web technologies. Here we focus on two formats that are strongly Web focused: Synchronized Multimedia Integration Language (SMIL) from W3C, and Flash from Adobe. In addition, the constructs in AJAX programming fulfill this definition (see [Section 12.2](#), Design pattern 5), as do Java applets, although strictly speaking they are not file formats *per se* and consequently lie outside the scope of this chapter.

Synchronized Multimedia Integration Language: SMIL

Broadly speaking, SMIL (pronounced *smile*) occupies the same rich media space as SVG. Indeed, W3C, who are responsible for both specifications, has formed a joint committee to coordinate development, the Synchronized Multimedia Working Group. With emphasis on integration, SMIL fulfills the role of an overarching media specification—SVG, in this context, becomes an example of a

multimedia component that can be synchronized with other media elements, such as audio, video, and text, or even other SVG content. The separation is not completely clear-cut, because SVG also overlaps significantly with SMIL in the realm of spatial layout of media elements, interaction, animation, and Document Object Model (DOM) manipulation.

SMIL predates SVG. A working group was formed in 1997, and the first version of the specification emerged a year later. Now in version 3.0, the specification defines profiles for niche uses—as SVG does. It is subdivided into modules from which the profiles are formed. Modules promote reuse in other specifications: for example, the SMIL animation module is wholly subsumed by SVG. Five profiles are defined, and the specification includes a scalability framework for creating others:

- *Language*, the broadest and most capable profile, includes all major capabilities of the specification (this is used in the example below).
- *Tiny* is a minimal specification for devices like MP3 players and low-end mobile phones.
- *UnifiedMobile* is for high-end mobile devices, such as smart phones and PDAs.
- *Daisy* targets a talking book standard designed for people with visual disabilities.
- *smilText* targets simple applications that stream timed text-only events.

Figure 5.22 shows a selection of SMIL’s core capabilities in the form of a sequence of snapshots of the rich media presentation over time. The figure is based on an award recognizing the scientific contributions of the Greenstone team. Initially the logo of the awarding organization, Kudos, appears. After 10 seconds a video commences, containing a short interview with one of the development team. While the video is playing, captions are displayed as text, changing over time. The final caption is hyperlinked to the award organization’s Web site.

Figure 5.22e shows the underlying SMIL specification. Lines 1–2 are the standard XML preamble. This is followed by the root tag, `<smil>`, with attributes specifying version (3.0) and profile (*Language*). Unlike the other illustrations in this book, version 1.1 of XML is specified—because version 3.0 of SMIL requires it. (The changes in XML 1.1 remove possible sources of ambiguity concerning the expression of control characters, such as “Vertical Tab” and “BEL,” and also concern Unicode normalization, mentioned in Section 8.2.) In practical terms, this makes no difference to the XML in our examples.

The overall structure of a SMIL file is encoded in a `<head>` element followed by a `<body>` element, which is intentionally reminiscent of HTML. Within the `<head>` element, lines 5–6 encode top-level document metadata in two `<meta>` tags just like those of HTML. Metadata can also be specified, more richly, using a `<metadata>` element that permits it to be associated with individual parts of the document. Any XML-based metadata scheme, designated using the appropriate namespace, can be encoded inside a `<metadata>` tag. In particular, the SMIL specification promotes the use of the Resource Description Framework, RDF (see Section 6.4).

The `<head>` element defines the characteristics of the canvas used for rendering the synchronized media using a `<layout>` element (lines 7–12). The `<root-layout>` defines the main canvas, with `<region>` tags used to label rectangular areas within it. In this case, the former defines a tiled back-

ground image, and the latter are used to control where media elements in the <body> appear. Three logical areas are defined: *background*, the same size as the main canvas; *video*, which is centrally located; and *text* beneath. A *z-index* attribute is provided to support layers; it controls the height at which the item is placed on the canvas's display stack.

The <body> element gives the actual media content. Our example (lines 15–45) is rudimentary, intended only to convey a sense of how SMIL operates. The temporal presentation of the synchronized media is controlled by two key constructs, <par> and <seq> for parallel and sequential structures, which can be arbitrarily nested. The example nests two parallel structures inside a sequential structure; the second parallel structure contains a sequential structure. Within this hierarchical structure, , <video>, and <smilText> tags are used for external images, video, and in-line text. The



Figure 5.22: SMIL example: (a) rendered snapshot, after 7 seconds; (b) after 13 seconds; (c) after 44 seconds; (d) after 130 seconds;

```

1 <?xml version="1.1" encoding="UTF-8"?>
2 <!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 3.0 Language//EN"
   "http://www.w3.org/2008/SMIL30/SMIL30Language.dtd">
3 <smil xmlns="http://www.w3.org/ns/SMIL" version="3.0" baseProfile="Language">
4   <head>
5     <meta name="title" content="Introductory demonstration of SMIL"/>
6     <meta name="author" content="David Bainbridge"/>
7     <layout>
8       <root-layout xml:id="root" backgroundColor="white"
9         backgroundImage="kudosbackground.gif" backgroundRepeat="repeat"
10        width="460" height="310"/>
11       <region xml:id="background" width="460" height="270" z-index="1"/>
12       <region xml:id="video" left="50" top="10" width="360" height="270"
13        z-index="2"/>
14       <region xml:id="text" left="50" top="290" width="360" height="20"
15        backgroundColor="white" z-index="3"/>
16     </layout>
17   </head>
18   <body>
19     <seq>
20       <par dur="10s">
21         
22         <smilText region="text" begin="5s">
23           Announcing the winner of the ICT Category ...
24         </smilText>
25       </par>
26       <par>
27         <seq>
28           <smilText region="text" begin="2s" dur="20s">
29             The Greenstone Digital Library Research Group
30           </smilText>
31           <smilText region="text" dur="40s">
32             Spreading the practical impact of digital libraries
33           </smilText>
34           <smilText region="text">
35             For further details: www.thekudos.org.nz
36             <area href="http://www.thekudos.org.nz" show="new" external="true"/>
37           </smilText>
38         </seq>
39       </par>
40       <switch>
41         <video src="interview128.flv" systemBitrate="12800" region="video"/>
42         <video src="interview256.flv" region="video"/>
43       </switch>
44     </par>
45   </seq>
46 </body>
47 </smil>

```

(e)

Figure 5.22, cont'd: (e) underlying XML

specification also provides for external audio, text, and streamed text media formats. A catch-all tag, `<ref>`, can link to arbitrary external resources.

Parallel, sequential, and media elements can include timing information, such as start time and duration. The first parallel structure (lines 17–22) occupies 10 seconds: it displays the Kudos logo immediately, positioned on the *background* region, and then after a 5-second delay (*begin*="5s", line 19) adds *Announcing the winner of the ICT category ...* in the *text* region. The second parallel structure (lines 24–42) continues with similar constructs. The `<smilText>` element on lines 32–35 includes an `<area>` element, which has the effect of hyperlinking the final text displayed to the Kudos Web site. The `<switch>` element on lines 38–42 conditions the video file on the available bandwidth (*systemBitrate*). Other properties that can be used in conditions include *systemScreenDepth* for the number of colors the user's display can handle and *systemLanguage* for the user's preferred language.

RealMedia and QuickTime support SMIL well, while Windows Media is more limited, with only the basic `<seq>` structure supported to describe playlists. The snapshots in this section were made with Ambulant, an open source SMIL player.

Adobe Flash

Flash is arguably the best-known example of the rich media formats. Popularized in recent years by YouTube, it is also the underlying technology for the realistic book shown in Figure 3.5. Interactive maps are another common use: in elections, TV stations and national newspapers often turn to Flash to provide an interactive Web-based map that can be clicked to select regions of interest and view data that is updated live as details become available. Such a site consists of a single Web page that embeds, through a hyperlink, a SWF file (see below) detailing how all interaction, display, and communication with an external server should proceed.

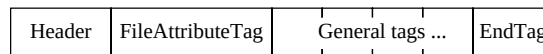
Flash is first and foremost a Web browser plug-in technology. Elsewhere we have cautioned against promoting digital library solutions for widespread use that rely on browsers' being configured with a specific plug-in—users might not even have the necessary access rights to install it. Flash is a notable exception, which has achieved the seemingly impossible: a rate of uptake that, as we learned in Chapter 3, some estimate at 99 percent.

SWF (pronounced *swiff*) is the file format behind Flash applications. It started as a development by FutureWave Software that rivaled Macromedia's Shockwave format. In 1966, FutureWave was acquired by Macromedia, which in turn was bought by Adobe in 2005. The name SWF, for Shockwave Flash, originated in a branding drive by Macromedia, but the format is nevertheless different from the Shockwave format itself—which often creates confusion. Table 5.4 summarizes the differences.

As we shall see below, Flash files can interleave definitions and display instructions (*tags* in Flash terminology), allowing a SWF application to be displayed even before a file is completely downloaded. Compression techniques are built into the format at the individual tag level, increasing the speed at which applications can be downloaded without triggering the need for the whole file to be

Table 5.4: Differences between the Flash and Shockwave formats

Flash	Shockwave
Open format: <i>.swf</i> file extension	Closed format: <i>.dcr</i> file extension
Supported by range of applications	Generated only by Adobe Director
Faster startup time	More powerful and efficient rendering engine
Cross-platform support	Windows and Mac only

**Figure 5.23: Structure of a SWF file**

present before decompression can begin. In contrast, one of the original uses of Shockwave was for interactive content served from CD-ROM, which renders speed of access less important than how files were structured. Technically, Shockwave is a more powerful development tool, including 3D capabilities and even the ability to include SWF content. While it can be used for rich media content, its main use over the years has gravitated towards games development.

The core components for developing a Flash application are:

- the stage
- timelines
- frames
- layers
- events.

We have met most of these components under different names in SVG and SMIL. The *stage* is the rectangular canvas upon which interactive media are displayed. *Timelines* mark when frames appear. *Frames* are constructed component by component out of elements such as vector shapes, and *layers* can be employed to achieve the desired result. User interactions (along with other moments worth reporting, such as when a video clip finishes) fire *events* that can be captured by callback routines written in ActionScript, the Flash scripting language. ActionScript uses the same syntax as JavaScript, but it has a different library of support routines. It can access and manipulate parts of the SWF file in a way similar to DOM manipulation.

The file format is compressed binary, and a complete SWF file is often referred to as a movie—even though interaction with an active frame can alter what frame is shown next. During a decade of Flash development history, the format has evolved significantly, reaching version 10 at the time of writing. [Figure 5.23](#) shows the overall file structure. It begins with a fixed size header that encodes basic information:

- SWF version
- file size
- frame size
- number of frames
- frame rate.

The file's first three characters are either SWF or SWC (actually, for technical reasons they are stored in the file backward to the reading order, as FWS or CWS). The C stands for compression and means that the whole file (starting from byte 4) has been compressed using *zlib*, the algorithm that underlies *gzip*. This layer is separate from the forms of compression built into individual tags, which are fundamental to the format and used whether the file type is SWF or SWC. The SWC option is available from version 6 onward.

A series of tags follow the header, concluding with the EndTag. Like the “chunks” in the Interface File Format (IFF) described in [Section 5.2](#), tags always begin with the same structure to make it easy to decode their type and size. Tags that are not recognized can simply be skipped. There are two types of tag header: short, for tags that are smaller than 63 bytes, and long, for units up to 4 GB. Immediately following this header, the FileAttributeTag records whether or not a MetadataTag is present and which version of ActionScript is used.

Following this (refer to [Figure 5.23](#)) is a mixture of tags. Definition tags define content: shapes, text, bitmaps, sounds, etc., that get stored in a dictionary with a unique character id. Control tags create and manipulate rendered instances of characters in a dictionary and control the flow of the file. Definition tags and control tags can be intermixed—this is how a Flash application can start before it has finished downloading—with the proviso that a tag can only refer to another tag if it has already been defined. For processing a file, there are tags that place, modify, and remove objects from the display list, and when a ShowFrame tag is encountered, the display list is rendered on the screen. SWF has over 60 tag types covering a wide range of capabilities already seen in SVG and SMIL—vector graphics, bitmaps, filter effects, gradient color, text, fonts, audio, and video—as well as additional capabilities like buttons, shape morphing, and sprites.

Like Scalable Vector Graphics, Flash is an attractive proposition for digital libraries because the content is self-contained. Metadata and other items can be extracted for browsing, searching, and document surrogates. The situation for SMIL is similar, but generally only the metadata is self-contained—the software needs to work harder to analyze the media content.

5.6 Music

As discussed in Chapter 3, music comes in many forms: audio (such as recordings made in a studio), images (such as sheet music or album artwork), and video (such as footage of an orchestra's performance, or a rock video). [Sections 5.2–5.4](#) discussed how to work with each of these media. A fourth form is symbolic music representation, which is the focus of this section. We also discuss how digital libraries can allow users to search and browse music in its multiple forms.

Representing music symbolically is a tricky business. Of the dozens of formats that have been proposed, the most widely used is MIDI, which we introduce in this section. MIDI is performance-based. There is a tension in symbolic music representation between synthesizing the information audibly and rendering it as sheet music. Symbolic information that precisely defines the notated music sounds stilted when played, because it lacks the interpretive dimension that musicians bring to performance. The alternative is to capture an actual performance—such as when a key was pressed, at what velocity, and for how long. But automatically generating sheet music from such data is fraught with difficulties and ambiguity, and the printed result differs greatly from how the composer would have scored the music.

Achieving such transformations for music is an active area of research, but we have not reached the point where off-the-shelf solutions are available for mainstream digital libraries. For the present, it is better to think in terms of which delivery mechanism best suits a symbolic representation, and to combine it in the digital library with other forms of representation.

Musical Instrument Digital Interface: MIDI

MIDI is a communication protocol, not a file format. It is based on the notion of timed events that electronic devices can use to control the synthesis of musical sounds (including unpitched percussion). Layered on top of this is the Standard MIDI file format (SMF) maintained by the MIDI Manufacturers Association. The key components of a MIDI file are:

- file header
- tracks
- channels
- events.

A track can carry a maximum of 16 channels, each designated for a particular instrument. The header defines the file as one of three types: a single track, multiple tracks played simultaneously, or multiple tracks played sequentially. The header also establishes how the events specified in the file, whose timing is expressed abstractly, are mapped to real time. This can be specified in one of two ways: ticks per beat or frames per second.

The header and track components create the overall structure of a song. The actual musical content is defined by events. These are preceded by a specification (relative to the last timed event) of the moment in time that they apply, and are further grouped by function:

- channel events
- meta events
- system exclusive events.

Channel Events

Note on and *note off* are the primary events for making sound. They encode the note's pitch (expressed as an integer in the range 0–127, which forms a chromatic scale, middle C being 60),

the channel containing the event (0–15), and the velocity with which the key was struck (again an integer in the range 0–127). More expressivity can be obtained by *note aftertouch* events, which apply changes in pressure to a specified *note on* event—think of a clarinetist modifying the air flow as a note is played—and *pitch bend* events, which alter the pitch of a note—think of a guitarist bending the strings on the fretboard with her fingers. Musical rests are not represented explicitly; they constitute a void between a *note off* event and the next *note on*. A *program change* event changes the instrument sound that is used on a given channel (also called a patch).

Meta Events

Meta events are familiar territory for a digital librarian. Meta events include *copyright notice*, *track name*, and *instrument name*. There is also a *lyric* event—here the time of occurrence is used to synchronize the syllables of a song with its melody—and a generic *text* event. More musically motivated events include *set tempo*, *time signature* and *key signature*, although none of these is required in a MIDI file. *Marker* and *cue point* events can be defined, like those of WAV and AIFF. In total there are over a dozen different types of meta event.

System Exclusive Events

System exclusive events are used to access specific hardware capabilities that are built in to a MIDI device by a manufacturer, such as triggering a memory dump of an instrument patch. This means that further standardization is required for reliable playback of a MIDI file. A standard called General MIDI (or GM) specifies a minimal feature set for a device like a sound card to be compliant and constrains the meaning of certain parameters—such as the 128 instrument sounds that can be specified by *program change* events. Channel 10 is reserved for unpitched percussion.

MIDI was published as an industry standard in 1983 and its header and track file syntax conforms to the 8-byte chunk prefix used in the Interchange File Format (IFF). (*MThd* and *MTrk* are the 4-byte tags used for headers and tracks.) However, it does not fully conform to IFF: events are encapsulated differently.

MIDI files deal exclusively in symbolic information and contain no raw audio. This makes them an order of magnitude smaller than regular compressed audio files. In addition, MIDI's symbolic nature sets it apart from other formats. For digital libraries, it constitutes an important category of document, because full musical information about the songs—such as tempo, key signature, and lyrics—can (if represented in the file) be easily extracted.

Digital music libraries

In Chapter 3 we give two examples of digital music libraries, one based on a symbolic representation derived from scanned sheet music and the other on MIDI files. Here is another: the Chopin Early Editions from the University of Chicago, illustrated in [Figure 5.24](#). Different editions of Chopin's music have been published under different titles by different publishers, so the librarian has assigned each work a uniform title. In [Figure 5.24a](#), the user is browsing by title and has focused on *Ballades, piano, no. 1, op. 23, G minor*—of which there are five versions. Clicking the first one brings up the bibliographic view of the work (in [Figure 5.24b](#)), which gives details such as publisher, approximate

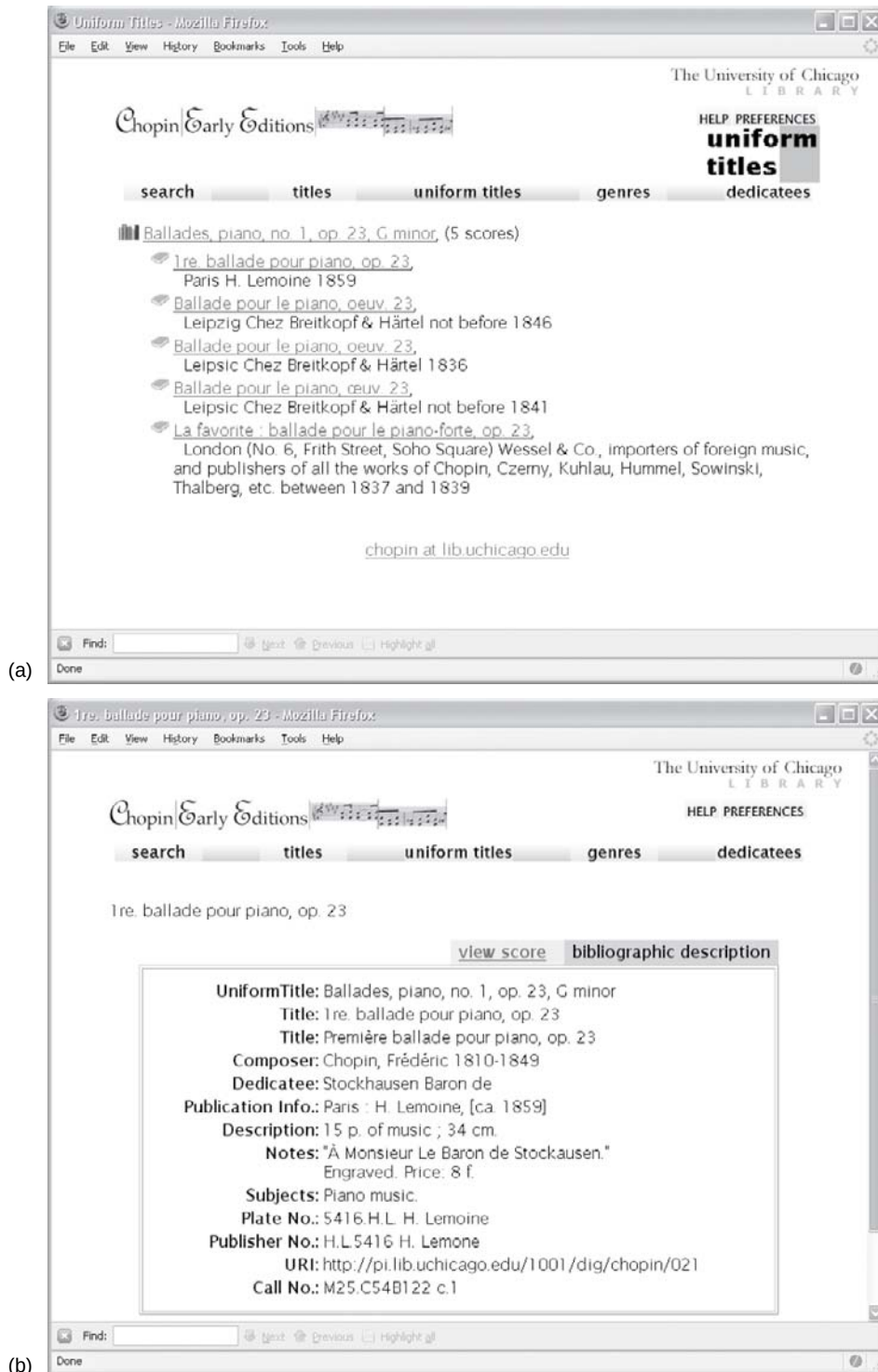


Figure 5.24: Chopin digital library from the University of Chicago: (a) browsing by title (*Ballades, piano, no. 1, op. 23, G minor*); (b) bibliographic view of *Première ballade pour piano, op. 23*;



Figure 5.24, cont'd: (c) page 2 of *Première ballade pour piano, op. 23*

publication date, and number of pages. The work can be viewed by clicking the *view score* link. One page is shown at a time, with links that navigate forward and back and a pull-down menu for going directly to any given page. In Figure 5.24c the user is on page 2 of the publication, which corresponds to the first page of the score itself.

Given additional time and resources, one could go further. Digital libraries begin with documents in a particular format, but it may be possible to locate other forms and make those available to the user as well—or perform automatic conversion, as in the Papers Past collection, which offers users PDF versions of individual articles and complete newspaper pages. For example, the Chopin collection would be greatly enhanced by including recordings of the music in it.

In Section 5.2 we have presented an example of an audio digital library that is organized around metadata attached to audio resources. But if a printed version is also available, as in the Chopin collection, it might be useful to synchronize audio and symbolic forms of document delivery. Flash and SMIL are possible implementation platforms; alternatively, JavaScript could be used to manipulate the page display. In Chapter 12 we present a sheet music example based on PDF files, with animated page turns.

5.7 Notes and Sources

For an introduction to the general area of multimedia systems (image, sound, and video), we recommend Chapman and Chapman's (2000) *Digital Multimedia*. This useful text also discusses topics that appeared in Chapter 4: HTML, XML, and cascading style sheets. Murray and van Ryper's (1996) *Encyclopedia of Graphics File Formats* covers an extensive range of image formats and also includes broad-brush descriptions of PostScript, RTF, MPEG, QuickTime, and AVI—which all qualify as graphics formats at some level.

The compression scheme used for the GIF image format is called LZW, and is the work of Welch (1984). The earlier compression scheme LZ77, which underlies PNG and *gzip* and has always been patent free, is the classic work of Ziv and Lempel (1977).

Audio

The fundamental theoretical result referred to in [Section 5.1](#), that the minimum sampling rate required to represent an analog signal is twice the highest frequency it contains, was developed by Harry Nyquist and Claude Shannon and is described in a classic book called *The Mathematical Theory of Communication* (Shannon and Weaver, 1949); see Jerri (1977) for a tutorial review.

The acronym of the popular mobile phone standard GSM stands for Global System for Mobile Communications, or Groupe Spécial Mobile (its original name). Because it is better known by its abbreviation, that's what we use in the chapter.

The Interchange File Format IFF, which serves as a basis for WAV and AIFF, was designed in 1985 to facilitate the transfer of data between software applications by Electronic Arts, a computer games pioneer that has grown into one of the world's largest software companies.

One detail that plagues many of the formats described in this chapter is the order of bytes used to represent integers internally (we mentioned it in passing when describing the TIFF image format). For example, the RIFF family for audio differs from the original IFF in that it represents numbers in “little-endian” instead of “big-endian” form. The processors in IBM PCs (and their clones), which were central to Microsoft's market at the time (1991), were little-endian, and the change was made for expediency because these computers can perform arithmetic on the audio files more quickly in their native representation. Today they are fast enough to make the difference immaterial, which is why WAV, AIFF, and AU are now considered platform independent.

Pohlmann's (2000) *Principles of Digital Audio* is a comprehensive resource for audio that moves from introductory material, through hardware and formats (including streaming), to signal processing techniques. For a text that is more focused on formats and practical programming, see Kientzle's (1997) *A Programmer's Guide to Sound*. Another useful reference is Whitaker's (2002) *Master Handbook of Audio Production*, which covers the full range of modern audio techniques, including digital sound recording in the studio. Sound banks for different instruments can be found at, for example, SampleSwap (www.sampleswap.org) and the Freesound Project (www.freesound.org).

The unequal frequency bands used in MP3 are derived from the Bark scale proposed by Zwicker (1961), which is motivated by the critical bands of human hearing. The Fraunhofer Institute was

instrumental in developing the MP3 format and is a key patent holder. This has been very profitable; for example, in 2005 Fraunhofer was believed reputed to have received € 100M (U.S. \$140M) in revenue from licensing fees.

Images

For a comprehensive explanation of the JPEG standard, see Pennebaker and Mitchell (1993). Animated GIFs were introduced in a revision to the file format in 1989; the two file types are distinguished by the first 6 characters of the file—GIF87a or GIF89a.

The Aldus Corporation, who originally conceived TIFF, is named after the 15th-century Italian printer Aldus Manutius. The company was ultimately merged with Adobe. The tongue-in-cheek acronym “Thousands of Image File Formats” is from Trauth (2006, p.198).

The Council of Independent Colleges, who developed the digital library of photographs described in [Section 5.3](#), is an association of independent liberal arts colleges and universities and higher education affiliates and organizations. The system uses the Greenstone digital library software (described in Part II), customized by Digital Library Consulting, who specialize in Greenstone work. The installation runs on a Sun Microsystems server under Solaris (<http://hcap.artstor.org/cgi-bin/library>).

ImageMagick (www.imagemagick.org) is a very useful open source software tool for image conversion that encapsulates many image software libraries, including *libjpeg*, *libpng*, and *libtiff*. Despite the similarity in naming, each of these libraries—and ImageMagick incorporates dozens of such libraries—is an independent project. JPEG 2000 support is not included by default but is easy enough to add in. If Ghostscript is available on the host computer, ImageMagick can also convert PostScript (or PDF) files to or from a raster format.

Video

Video Demystified (Jack, 2007), written with an engineering bias, explains the labyrinth of video and TV standards, with separate chapters on MPEG-1 and MPEG-2. The International Telecommunication Union (ITU) is now responsible for both CCIR and CCITT standards, since these two bodies joined forces. CCIR 601 is now officially known as ITU-R Recommendation BT.601, often shortened to Rec. 601. The definitive references for MPEG are the standards themselves: MPEG-1 is ISO/IEC 11172; MPEG-2 is ISO/IEC 13818; and MPEG-4 is ISO/IEC 14496.

Microsoft’s Audio Video Interleave (AVI) format is subsumed by its fully fledged Advanced Systems Format (ASF) for streaming audio and video over the Internet. This, in turn, is an integral part of its media technologies software development kit, which also runs on the Apple Macintosh.

For more information on the Open Video Digital Library described at the end of [Section 5.4](#), see Marchionini and Geisler (2002) and Wildemuth et al. (2003). The Open Video Digital Library pre-dates the popular ascent of Flash and delivers video using QuickTime (mentioned in [Section 5.4](#)). The Internet Archive audio and video development, being more recent, uses a Flash-based player, the open source FlowPlayer (www.flowplayer.org). Surprisingly, perhaps, there are many open source

tools for Flash. Carnegie Mellon's InforMedia digital video library (www.informedia.cs.cmu.edu), a substantial project that has run for many years, is another excellent and mature example of a multi-media digital library.

Rich Media

More details of the differences between Flash and Shockwave can be found at the Adobe Web site (http://www.adobe.com/go/tn_13971). Naturally, the Adobe account tends to accentuate the positive aspects. Independent descriptions and discussions of the differences can be located by searching on such terms such as *flash shockwave differences* using your favorite search engine.

Music

The largest known collection of sheet music (over 29,000 pieces) created from print sources by Optical Music Recognition technology is the Lester S. Levy collection at Johns Hopkins University (<http://levysheetmusic.mse.jhu.edu>). Resulting from this work is GAMERA, an open source OMR toolkit based on the techniques pioneered by Ichiro Fujinaga at McGill University (Droettboom et al., 2003).

For research that investigates how to transform music in the other direction (i.e., putting expression into a computer's rendition of a score) see, for example, the Intelligent Music Processing and Machine Learning Group, led by Gerhard Widmer at the Austrian Research Institute for Artificial Intelligence (Blashill, 2001). This is a long-standing research group that has studied this problem and a variety of related issues.

Metadata: Elements of organization

If thorough and consistent metadata has been created, it is possible to conceive of its use in an almost infinite number of new ways ...

Metadata, often described as *data about data*, is critical to all forms of organized digital content. It is the means by which all organization in a digital library is achieved, and much of what you have read in this book so far implicitly assumes its existence. Whether in the creation of surrogates (Section 3.3), categories of video browsing (Section 5.4), or usage information (Section 2.2), it is metadata that yields collections that are *organized*, rather than digital piles of unstructured and unconnected objects.

Specifically, metadata enables almost all the examples of digital library technology that we have seen in this book, including

- the item displays and surrogates in the Pergamos Digital Library in Figure 1.10;
- the browsing structures of *subjects*, *titles a-z*, *organization*, and *how to* of the *Village Brickmaking* collection in Figure 3.1;
- the table of contents of the *Otago Witness* newspaper in Figure 3.4d;
- the colored tabs marking the chapters in the realistic book *Farming Snails I* in Figure 3.5b;
- the entire record display in Figure 3.9a;
- the structured form searching interface in Figure 3.15;
- the searching interface in Figure 3.17;
- the structured subject browsing of Figure 3.20;
- the user log graphs in Figure 2.4.

Metadata is generally taken to be structured information about a particular information resource. Information is “structured” if it can be meaningfully manipulated without understanding its content. For example, given a collection of source documents, bibliographic information about each document would be metadata for the collection—the structure is made plain, in terms of which pieces of text represent author names, which represent titles, and so on. But given a collection of bibliographic information, metadata might also comprise some information about each bibliographic item, such as who compiled it and when.

The role of metadata in your digital library can be clarified by considering such questions as:

- Where does your metadata come from? Is it automatically extracted from digital objects, manually assigned, or imported from an external source?
- How will the metadata affect document display, browsing, searching, and maintenance of the digital library?
- Does the metadata need any extra processing before use? For example, do different versions of people's names need to be harmonized?
- Is the metadata in your digital library affected by the activities of the end-users?
- Can you monitor the metadata in your library to continually assess its quality?
- Is the metadata private to your library or can it be shared with others?
- Can you migrate your metadata to another software application?

This chapter starts with a description of different perspectives on metadata and how these perspectives frame the answers to the questions above. Then we consider the evolution of metadata in library catalog systems, as this has had a lasting influence on digital library systems. Following this, we describe how metadata represents material like multimedia content ([Section 6.3](#)) and complex compound objects ([Section 6.4](#)). We then consider how to process and assess metadata within the workflow of a digital library. Finally, in [Section 6.6](#) we examine how metadata can be extracted from source documents—a key process in automating the construction of large-scale collections.

6.1 Characteristics of Metadata

Imagine when electronic books are cheap and high-quality enough to begin displacing printed books. Every time a student highlights or annotates a page, that information will be used—with permission—to enhance the public metadata about the book. Even how long it takes people to get through pages or how often they go back to particular pages.... We'll be able to ask our books to highlight the passages most often read by poets, A [grade] students, professors of literature, or Buddhist priests.

Some of the earliest organized external descriptions for physical objects are found in Sumerian collections of clay tablets used for recording commercial transactions. As the number of items in a collection increased, the need for efficient mechanisms to find particular items and to simply manage the collection grew accordingly. Gradually, physical descriptions grew very complex and very large, as illustrated by the card catalogs of the 20th century ([Figure 6.1](#)). As with many other data-intensive applications, computers evolved as the appropriate tool for managing these descriptions. Some argue that descriptions only formally became *metadata* when they were moved to an electronic context, but we prefer the inclusive approach that emphasizes the continuity of resource description.

The objects of description have also evolved, to include varied media, and, in museums, a panoply of diverse physical objects. As the described objects become digital, we move from the library to the digital library—but the principles of description for access and management remain largely the same. The topic of metadata for *digital* libraries inherits a vast amount of work performed by librarians in creating, sharing, applying, and managing metadata for library collections.

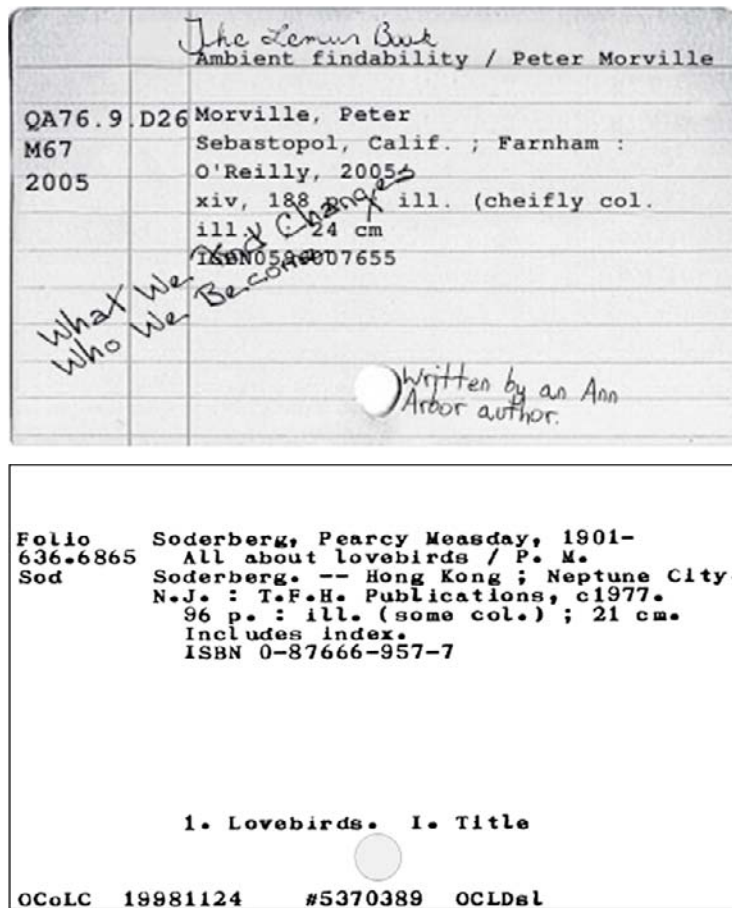


Figure 6.1: Library cards

There are several approaches to classifying metadata, depending on its source, purpose, audience, and format. In building a digital library, a practical concern is the source of the metadata: Where do the metadata values come from? How do we know that the author of the Word file on our hard disc is *John Smith*? Broadly speaking, there are two possibilities: either a human has declared that Smith is the author, or a computer program has determined the author's identity.

In the case of *human-assigned* metadata, a human being examines the digital document (or its physical counterpart) and assigns a particular value (*John Smith*) to a particular metadata element (*the author*). This person (often a librarian) may consult other documents or other people in the process of determining the metadata value, but the ultimate source of the metadata assignment is the human brain.

In the case of *computationally assigned* metadata, a computer program, after processing the digital document (by comparing it with other documents or using remote online resources), outputs a value for the metadata element. If the document was *born digital* (such as a word-processed document or

an image from a digital camera), metadata may have been embedded within the file at the moment of its creation. If the document was digitized from an analog source (using OCR, for example), the software applications used often will have embedded metadata within the file (e.g., TIFF tags, see [Section 6.3](#)). Such embedded metadata is usually easily extracted from digital objects, although it is often of less use than metadata derived from human judgments.

For a specific library system, metadata values may be imported from an external source, but if you follow the metadata creation process back far enough, eventually you will reach either a human being or a computer program.

Irrespective of its source, where does the metadata actually reside? Is it embedded within a digital document, as in HTML or XML files, for example, or is it stored separately in library catalog records or some remote metadata repository? Many file formats contain embedded metadata that needs to be *extracted* from the file before it can be used to organize items in a digital library. The central issue is whether the file format is clearly documented and available for inspection. Where formats are publicly specified (e.g., PNG, PDF, ODF), it is relatively straightforward to extract metadata. However, when file formats are proprietary, programmers often have to resort to reverse engineering the format, which is time consuming and less reliable.

An alternative perspective is to consider the function of different types of metadata, such as

- *administrative* metadata for managing resources, such as rights information;
- *descriptive* metadata for describing resources, such as the cards in [Figure 6.1](#);
- *preservation* metadata for describing resources, such as recording preservation actions;
- *technical* metadata related to low-level system information, such as data formats and any data compression used;
- *usage* metadata related to system use, such as tracking user behavior.

One important aspect of this categorization is that it highlights the fact that the end user's view of metadata ([Figure 6.1](#)) is only the tip of the iceberg. Much of the metadata is not intended for public display, and in the case of *usage* metadata, the ALA Code of Ethics ([Figure 2.2](#)) reminds us that some parts may be confidential. The categorization further illustrates that metadata is not *static*.

Although we might expect that data like author and publisher might not change over the lifetime of the object, metadata related to preservation and use is likely to change, perhaps frequently. We would also expect metadata to be generated by user enhancements such as annotations (see [Section 2.5](#)).

6.2 Bibliographic Metadata

Anyone working with digital libraries needs to know about the different standard methods for representing document metadata. Much of the work on metadata in digital libraries is based on practices that have been adopted in library cataloging, particularly the MARC (*machine-readable cataloging*) format that has been used by libraries internationally for decades. The spread of electronic resources in the 1990s led to the creation of a new scheme called Dublin Core, which is an intentionally minimalist standard intended to be applied to a wide range of digital library materials by people who are

not trained in library cataloging. These two schemes are of interest not only for their practical value, but also for highlighting diametrically opposed underlying philosophies. We also include descriptions of two bibliographic metadata formats: BibTeX and EndNote. The former is common in scientific and technical fields (and a companion to the LaTeX format described in Section 4.6), while the latter is in widespread general use (and integrates with Microsoft Word, for example).

MARC

[The MARC standard] forever changed the relationship of a library to its users, and the relationship of geography to information.

Managing metadata with software tools has a long history in libraries, mostly deriving from the necessity to move from card catalogs to computer-based records. The core work was the development of the MARC standard in the late 1960s by Henriette Avram at the Library of Congress. Once metadata was expressed in computer-based MARC records, it could be copied, distributed, and shared among libraries. MARC is a comprehensive and detailed standard whose use is carefully controlled and transmitted to budding librarians in library science courses. Here we have space to present only an outline of all its richness.

Figure 6.2a shows an entry that was obtained from the Library of Congress online catalog giving metadata for the first edition of this book, *How to Build a Digital Library*. It includes information about authorship, topic coverage, details about the physical book itself, the publisher, various identification numbers, and links to more detailed information. This is an electronic record that describes a physical object, presented in a classic, familiar, library catalog style.

Producing a MARC record for a particular publication is an onerous undertaking that is governed by a detailed set of rules and guidelines called the *Anglo-American Cataloging Rules*, familiarly referred to by librarians as AACR2R (the 2 stands for second edition, the final R for revised). These rules, inscribed in a formidable handbook, are divided into two parts: Part 1 applies mostly to the description of documents, Part 2 to the description of works. Part 2, for example, treats *Headings, Uniform titles, and References* (i.e., entries starting with “See ...” that capture relationships between works). Under *Headings* there are sections on how to write people’s names, geographic names, and corporate bodies. Appendices describe rules for capitalization, abbreviations, and numerals.

The rules in AACR2R are highly detailed, almost persnickety. It is hard to convey their flavor in a few words. Here is one example: How should you name a local church? Rules under *corporate bodies* give the answer. The first choice of name is that “of the person(s), object(s), place(s), or event(s) to which the local church ... is dedicated or after which it is named.” The second is “a name beginning with a word or phrase descriptive of a type of local church.” The third is “a name beginning with the name of the place in which the local church ... is situated.” Now you know. If rules like this interest you, there are thousands more in AACR2R.

Internally, MARC records are stored as a collection of tagged fields in a fairly complex format.

Figure 6.2b gives something close to the internal representation of the catalog record, while Table 6.1 lists some of the field codes and their meaning. Many of the fields contain identification codes. For example, field 008 contains fixed-length data elements, such as the source of the cataloging for the

The Library of Congress > LCCN Permalink



LIBRARY OF CONGRESS CATALOG RECORD



View LC holdings for this title in the: [LC Online Catalog](#) View this record in: [MARCXML](#) | [MODS](#) | [Dublin Core](#)

How to build a digital library

LC Control No.: 2002107327

Type of Material: Book (Print, Microform, Electronic, etc.)

Personal Name: Witten, Ian H. » [More like this](#)

Main Title: How to build a digital library / Ian H. Witten, David Bainbridge.

Published/Created: San Francisco, CA : Morgan Kaufmann Publishers, c2003.

Description: 518 p. ; 24 cm.

ISBN: 1558607900 (alk. paper)

Series: Morgan Kaufmann series in multimedia information and systems » [More like this](#)

Other System No.: (DLC) 2002107327

Quality Code: pcc

Links: Publisher description: <http://www.loc.gov/catdir/description/els031/2002107327.html>

Table of contents: <http://www.loc.gov/catdir/toc/els031/2002107327.html>

LCCN Permalink: A Service of the Library of Congress

More information: [LCCN Permalink FAQ](#)

[About](#) | [Site Map](#) | [Contact](#) | [Accessibility](#) | [Legal](#) | [USA.gov](#)

(a)

How to build a digital library / Ian H. Witten, David Bainbridge.

LC Control No.: 2002107327

LCCN Permalink: <http://lcn.loc.gov/2002107327>

000 00965cam a22002414a 450

001 12791256

005 20071130191532.0

008 020529s2003 cau 000 0 eng

035 _ |a (DLC) 2002107327

906 _ |a 0 |b ibc |c orignew |d 2 |e epcn |f 20 |g y-gencatlg

925 0_ |a acquire |b 2 shelf copies |x policy default

955 _ |a pc21 2002-05-29 |a pv19 2002-12-10 to HLCDC (2 copies) |a lh39 2007-11-30

010 _ |a 2002107327

020 _ |a 1558607900 (alk. paper)

040 _ |a DLC |c DLC |d DLC

042 _ |a pcc

100 1_ |a Witten, Ian H.

245 10 |a How to build a digital library / |c Ian H. Witten, David Bainbridge.

260 _ |a San Francisco, CA : |b Morgan Kaufmann Publishers, |c c2003.

300 _ |a 518 p. ; |c 24 cm.

440 _0 |a Morgan Kaufmann series in multimedia information and systems

856 42 |3 Publisher description |u <http://www.loc.gov/catdir/description/els031/2002107327.html>

856 41 |3 Table of contents |u <http://www.loc.gov/catdir/toc/els031/2002107327.html>

(b)

Figure 6.2: Metadata for a book: (a) metadata display; (b) MARC data;

(c)

```

<?xml version="1.0" encoding="UTF-8"?>
<record xmlns="http://www.loc.gov/MARC21/slim" ...>
  <leader>00965cam a2200241a 4500</leader>
  <controlfield tag="001">12791256</controlfield>
  <controlfield tag="005">20071130191532.0</controlfield>
  <controlfield tag="008">020529s2003    cau          000 0 eng  </controlfield>
  <datafield tag="035" ind1=" " ind2=" ">
    <subfield code="a">(DLC) 2002107327</subfield> </datafield>
  ...
  <datafield tag="010" ind1=" " ind2=" ">
    <subfield code="a">2002107327</subfield> </datafield>
  <datafield tag="020" ind1=" " ind2=" ">
    <subfield code="a">1558607900 (alk. paper)</subfield> </datafield>
  <datafield tag="042" ind1=" " ind2=" ">
    <subfield code="a">pcc</subfield> </datafield>
  <datafield tag="100" ind1="1" ind2=" ">
    <subfield code="a">Witten, Ian H.</subfield> </datafield>
  <datafield tag="245" ind1="1" ind2="0">
    <subfield code="a">How to build a digital library </subfield>
    <subfield code="c">Ian H. Witten, David Bainbridge.</subfield> </datafield>
  <datafield tag="260" ind1=" " ind2=" ">
    <subfield code="a">San Francisco, CA :</subfield>
    <subfield code="b">Morgan Kaufmann Publishers,</subfield>
    <subfield code="c">c2003.</subfield> </datafield>
  ...
</record>

```

(d)

```

<?xml version="1.0" encoding="UTF-8"?>
<srw_dc:dc xmlns:srw_dc="info:srw/schema/1/dc-schema"
  xmlns="http://purl.org/dc/elements/1.1/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="info:srw/schema/1/dc-schema
    http://www.loc.gov/z3950/agency/zing/srw/dc-schema.xsd">
  <title>How to build a digital library </title>
  <creator>Witten, Ian H.</creator>
  <type>text</type>
  <publisher>San Francisco, CA : Morgan Kaufmann Publishers,</publisher>
  <date>c2003.</date>
  <language>eng</language>
  <identifier>http://www.loc.gov/catdir/description/els031/2002107327.html</identifier>
  <identifier>http://www.loc.gov/catdir/toc/els031/2002107327.html</identifier>
  <identifier>URN:ISBN:1558607900 (alk. paper)</identifier>
</srw_dc:dc>

```

Figure 6.2, cont'd: (c) MARCXML version; (d) Dublin Core metadata record;

item and the language in which the book is written. Many of the variable-length fields contain subfields, which are labeled *a*, *b*, *c*, and so on, each with their own distinct meaning (in the computer file they are separated by a special subfield delimiter character). For example, field 100 is the personal name of the author, with subfields indicating the standard form of the name, full forenames, and dates. Field 260 gives the imprint, with subfields indicating the place of publication, publisher, and date. The information in the more legible representation of Figure 6.2a is evident in the coded form of Figure 6.2b. Some fields can occur more than once, such as the subject headings stored in field 650.

```

<?xml version="1.0" encoding="UTF-8"?>
<mods xmlns="http://www.loc.gov/mods/v3" ...>
  <titleInfo>
    <title>How to build a digital library</title></titleInfo>
  <name type="personal">
    <namePart>Witten, Ian H.</namePart>
    <role><roleTerm type="text" authority="marcrelator">creator</roleTerm></role>
  </name>
  <typeOfResource>text</typeOfResource>
  <originInfo>
    <place><placeTerm type="code" authority="marccountry">cau</placeTerm></place>
    <place><placeTerm type="text">San Francisco, CA</placeTerm></place>
    <publisher>Morgan Kaufmann Publishers</publisher>
    <dateIssued>c2003</dateIssued><dateIssued encoding="marc">2003</dateIssued>
    <issuance>monographic</issuance></originInfo>
  <language>
    <languageTerm type="code" authority="iso639-2b">eng</languageTerm></language>
  <physicalDescription>
    <form authority="marcform">print</form>
    <extent>518 p. ; 24 cm.</extent> </physicalDescription>
  <relatedItem type="series">
    <titleInfo><title>Morgan Kaufmann series in multimedia information and systems</title>
    </titleInfo></relatedItem>
  <identifier type="isbn">1558607900 (alk. paper)</identifier>
  <identifier type="lccn">2002107327</identifier>
  <identifier type="uri">http://www.loc.gov/catdir/toc/els031/2002107327.html</identifier>
  <relatedItem type="otherVersion">
    <location>
      <url displayLabel="Table of contents">
        http://www.loc.gov/catdir/toc/els031/2002107327.html</url></location>
    </relatedItem>
  <recordInfo>
    <descriptionStandard>aacr2</descriptionStandard>
    <recordContentSource authority="marcorg">DLC</recordContentSource>
    <recordCreationDate encoding="marc">020529</recordCreationDate>
    <recordChangeDate encoding="iso8601">20071130191532.0</recordChangeDate>
    <recordIdentifier>12791256</recordIdentifier></recordInfo>
</mods>

```

(e)

```

@book{863700,
  author = {Ian H. Witten and Bainbridge, David},
  title = {How to Build a Digital Library},
  year = {2003},
  isbn = {1558607900},
  publisher = {Elsevier Science Inc.},
  address = {New York, NY, USA},
}

```

(f)

```

%0 Book
%1 863700
%A Ian H. Witten
%A David Bainbridge
%T How to Build a Digital Library
%D 2003
%@ 1558607900
%P 552
%I Elsevier Science Inc.

```

(g)

Figure 6.2, cont'd: (e) MODS metadata record; (f) BibTeX metadata record; (g) EndNote metadata record

Table 6.1: Meaning of some MARC fields

001	Control number uniquely identifying the record
005	Date and time that the record was last modified
008	Fixed fields
010	Library of Congress control number
015	National Bibliographic number
035	System control number
040	Cataloging source
050	Library of Congress classification
082	Dewey classification
100	Main entry—personal name
260	Imprint: place of publication, publisher, date
300	Physical description
500	General note
504	Bibliography note
650	Subject entry
650	Subject entry
906	Tags in the 900 range are reserved for local use, and are used by vendors, systems, or individual
985	libraries to exchange additional data
991	

The MARC format covers more than just bibliographic records. It is also used to represent authority records—that is, standardized forms that are part of the librarian’s controlled vocabulary (see [Section 6.5](#)).

The rules and detailed formatting of the MARC standard are what allows these records to be exchanged between different library software systems. Rather than each library creating a new record for a book, it can be created once and then shared. The use of standardized records allows for great efficiencies in metadata creation and also facilitates the construction of so-called union catalogs that combine metadata from several libraries. The WorldCat union catalog at the Online Computer Library Center, a U.S. organization that helps libraries locate, acquire, catalog, and lend library materials, holds more than 125 million records from 112 countries. Cooperation on this scale is based on a consortium approach common in the library community and made practical through the widespread use of standards like MARC.

MARCXML

The Library of Congress guided the development of a standard way of representing MARC data in the XML language, called MARCXML. [Figure 6.2c](#) shows an example that contains an abbreviated version of the record in [Figure 6.2b](#). Field codes are represented as the values of attributes within the opening tag of *datafield* elements. Within these elements, *subfield* elements represent the MARC subfields, with *code* attributes to distinguish them. For example, the last few lines of [Figure 6.2c](#) give the publisher information in MARC field 260, including place of publication, publisher name, and

publication date. The simplicity of this transformation emphasizes one of the design goals of MARCXML: lossless transformation between it and MARC in both directions. This round-trip compatibility is useful because there are many digital library applications where data is transformed or migrated between different systems.

MARCXML uses XML in a different way from other XML-based metadata standards, where element names usually convey semantic information (e.g., <title>, <publisher>, etc.). This characteristic makes it hard to read for a human unfamiliar with MARC. For comparison, look at the MODS representation of the same data in [Figure 6.2e](#), also expressed in XML, which you will probably find much easier to understand. However, MARCXML provides a useful mechanism for connecting the wealth of library data in MARC format with XML processing technologies, such as XSL (Section 4.4).

Dublin Core: DC

The Dublin Core is a set of metadata elements that are designed specifically for non-specialist use. It is intended for the description of electronic materials—such as a Web page or site—which will almost certainly not receive a full MARC catalog entry. The result of a collaborative effort by a large group of people, Dublin Core is named for Dublin, Ohio, where the first meeting was held in 1995. It received the approval of ANSI, the American National Standards Institute, in 2001.

Compared with the MARC format, Dublin Core has a refreshing simplicity. [Table 6.2](#) summarizes the metadata elements it contains: just 15, rather than the several hundred used by MARC. As the name implies, the elements are intended to form a “core” set that may be augmented by additional

Table 6.2: Dublin Core metadata standard

Metadata	Definition
Title	The name given to the resource by the creator or publisher
Creator	The person or organization primarily responsible for the intellectual content of the resource
Subject	The topic of the resource
Description	A textual description of the content of the resource
Publisher	The entity responsible for making the resource available
Contributor	A person or organization (other than the Creator) who is responsible for making significant contributions to the intellectual content of the resource
Date	A date associated with the creation or availability of the resource
Type	The nature or genre of the content of the resource
Format	The physical or digital manifestation of the resource
Identifier	An unambiguous reference that uniquely identifies the resource within a given context
Source	A reference to a second resource from which the present resource is derived
Language	The language of the intellectual content of the resource
Relation	A reference to a related resource, and the nature of its relationship
Coverage	Spatial locations and temporal durations characteristic of the content of the resource
Rights	Information about rights held in the resource

elements for local purposes. In addition, the existing elements can be refined through the use of qualifiers. All elements can be repeated where it is appropriate. Dublin Core uses the general term *resource* for what is being described—the term subsumes pictures, illustrations, movies, animations, simulations, and even virtual reality artifacts, as well as textual documents. Indeed, *resource* has been defined in Dublin Core documents as “anything that has identity.” Figure 6.2d shows a Dublin Core metadata record (in an XML format) for the book in Figure 6.2a; naturally, it contains only a subset of the full information.

The *Creator* might be a photographer, an illustrator, or an author. The *Subject* is typically expressed as a keyword or phrase that describes the topic or the content of the resource. The *Description* might be an abstract of a textual document, or a textual account of a non-textual resource, such as a picture or an animation. The *Publisher* is generally a publishing house, a university department, or a corporation. A *Contributor* could be an editor, a translator, or an illustrator. The *Date* is the date of resource creation, not the date or dates covered by its contents. For example, a history book will have an associated *Coverage* date range that defines the historical time period it covers, as well as a publication *Date*. Alternatively (or in addition), *Coverage* might be defined in terms of geographical locations that pertain to the content of the resource. The *Type* might indicate a home page, research report, working paper, poem, or any of the media types listed above. The *Format* is used to help identify software systems needed to access the resource.

Qualified Dublin Core

An extension of the basic 15-element Dublin Core described above was introduced to add greater expressivity and to extend the range of uses. Two forms of qualification were added: *element refinement* and *encoding schemes*.

Any element can be refined or qualified. For example, the *Date* element can be refined as *date created*, *date valid*, *date available*, *date issued*, or *date modified*. The key principle in element refinement is “dumbing down”; the qualifier can be safely removed and the element value interpreted as a simple element. So the *date created*, for instance, can be safely interpreted as a *Date*. This principle allows qualified and simple Dublin Core to co-exist easily inside digital libraries.

Encoding schemes address the permissible ranges of element values. For example, a controlled vocabulary for an element could be expressed as an encoding scheme—that is, the Library of Congress Subject Headings could be used for the Dublin Core element *Subject*. Given such an encoding scheme, digital library software can expect that only valid subject headings will appear as values in the *Subject* element. Alternatives to the Library of Congress Subject Headings include the Dewey Decimal Classification and the Medical Subject Headings (MeSH) designed by the U.S. National Library of Medicine. A different form of encoding scheme can specify that a value should follow a precise structural rule, such as a date format, URL, or language-encoding scheme. The restrictions imposed by encoding schemes are valuable, because it is easier to write software to deal with constrained values than it is to write software for free text.

Table 6.3 gives details of the qualifiers and encoding schemes that are specified in qualified Dublin Core, including some notes on encoding schemes suggested by the official Dublin Core Metadata

Table 6.3: Qualified Dublin Core**(a) Sample elements, qualifiers, and encoding schemes**

Element	Qualifier	Encoding schemes
Audience	Education Level, Mediator	
Coverage	Spatial, Temporal	DCMI Point, ISO 3166, DCMI Box, TGN
Date	Created, Valid, Available, Issued, Modified, Date Accepted, Date Modified, Date Submitted	DCMI Period, W3CDTF
Description	Table of Contents Abstract	
Format	Extent, Medium	IMT
Identifier	Bibliographic Citation	URI
Language		ISO 639-2, RFC4646
Relation	13 different relations are specified as qualifiers, including Is Part Of, Has Version, and Is Referenced By	
Rights	Access Rights, License	
Source		URI
Subject		LCSH, MeSH, DDC, LCC, UDC
Title	Alternative	
Type		DCMIType

(b) Notes on encoding schemes

Scheme	Type	What it encodes	Example
DCMI Box	Syntax	Region of space	The Western Hemisphere: westlimit=180; eastlimit=0
DCMI Period	Syntax	Period of time	World War II: name=World War II; start=1939; end=1945
DCMI Point	Syntax	A particular location	Perth, Western Australia: name=Perth, W.A.; east=115.85717; north=-31.95301
DCMIType	Vocabulary	Type of resource	Quicktime video file: MovingImage
DDC	Vocabulary	Dewey Decimal Classification	599 Mammalia. Mammals
IMT	Vocabulary	Internet Media Type—also known as MIME types	Quicktime video file: video/quicktime
ISO 639-2	Syntax	Three-letter codes for languages	Arabic: ara
LCC	Vocabulary	Library of Congress Classification	QL700-739.8
LCSH	Vocabulary	Library of Congress Subject Headings	Animal Psychology
MeSH	Vocabulary	Medical Subject Headings	Animals, Wild
RFC4646	Syntax	Languages	English as used in the U.S.: en-US
TGN	Vocabulary	Getty Thesaurus of Geographic Names	Perth in Western Australia: Perth (inhabited place) ID: 7001977
UDC	Vocabulary	Universal Decimal Classification	599 Mammalia. Mammals
URI	Syntax	Uniform Resource Identifiers	http://www.ietf.org/rfc/rfc3986.txt
W3CDTF	Syntax	Date and time	November 5, 1994, 8:15:30 a.m., U.S. Eastern Standard Time: 1994-11-05T08:15:30-05:00

Accrual Method, Accrual Periodicity, Accrual Policy, Contributor, Creator, Instructional Method, Provenance, Publisher, and Rights Holder do not have refinements or encoding schemes.

Initiative Usage Board. [Table 6.3a](#) includes qualifiers for *Audience*, *Coverage*, *Date*, *Description*, *Format*, *Identifier*, *Relation*, and *Rights* metadata fields (among others), and encoding schemes for both spatial and temporal coverage, date, identifiers, languages, and subjects. [Table 6.3b](#) gives suggested standards for encoding such things as regions of space, periods of time, particular locations, subjects, and languages.

Metadata Object Description Schema: MODS

The Metadata Object Description Schema (MODS) is a bibliographic XML-based metadata format that can be used to represent a subset of MARC (specifically, the MARC21 variant that is a combination of the United States and Canadian MARC formats). Because it uses a subset of a particular MARC variant, general MARC records cannot be converted losslessly to MODS, and so round-trip compatibility is sacrificed.

[Figure 6.2e](#) shows a MODS record that describes the first edition of this book. The MARC field 260 publisher information we looked at in the last few lines of [Figure 6.2c](#) can be found starting at the tenth line of [Figure 6.2e](#), within the *originInfo* element:

```
<originInfo>
  <place>
    <placeTerm type="code" authority="marccountry">ca</placeTerm>
  </place>
  <place>
    <placeTerm type="text">San Francisco, CA</placeTerm>
  </place>
  <publisher>Morgan Kaufmann Publishers</publisher>
  <dateIssued>c2003</dateIssued>
  <dateIssued encoding="marc">2003</dateIssued>
  <issuance>monographic</issuance>
</originInfo>
```

This extract illustrates a key quality of well-designed XML-based representations: it is easily readable by people as well as by computers. As you can see, MODS allows richer descriptions than Dublin Core and is a useful alternative when qualified Dublin Core is inadequate for describing your resources.

Along with Dublin Core, MODS can be used within the METS standard described in [Section 6.4](#). It does not in itself address issues of authority control, but there is an analog to MARC authority data in the MADS format described in [Section 6.5](#).

BibTeX

Scientific and technical authors, particularly those using mathematical notation, often favor a widely used generalized document-processing system called TeX (pronounced *tech*), or a customized version called LaTeX (see [Section 4.6](#)). This freely available package contains a subsystem called BibTeX that manages bibliographic data and references within documents.

Figure 6.2f shows a record in BibTeX format. Records are grouped into files, and files can be brought together to form a database. Each field can flow freely over line boundaries—extra white space is ignored. Records begin with the @ symbol followed by a keyword naming the record type: article, book, and so forth. The content follows in braces and starts with an alphanumeric string that acts as a key for the record. Keys in a BibTeX database must be unique. Within a record, individual fields take the form *name=value*, with a comma separating entries. Names specify bibliographic entities such as author, publisher, address, and year of publication. Each item type can be included only once, and values are typically enclosed in braces (or double quotation marks) to protect spaces. Certain standard abbreviations, such as abbreviated month names, can be used, and users can define their own abbreviations.

Two items deserve explanation. First, the *author* field is used for multiple authors, and names are separated by the word *and*, rather than by commas, with a final *and* as in ordinary English prose. This is because the tools that process BibTeX files incorporate bibliographic standards for presenting names. The fact that the two names in Figure 6.2f use different conventions is of no consequence: both will be presented correctly in whatever style has been chosen for the bibliography. Second, titles are also presented in whatever style has been chosen for the bibliography—for example, only the first word may be capitalized, or all content words may be capitalized—regardless of how they appear in the BibTeX file. Braces override this and preserve the original capitalization, so that the proper names can appear correctly capitalized in the document.

Unlike in other bibliographic standards, in BibTeX the set of attribute names is determined by a style file named in the source document and is used to formulate citations in the text as well as to format the references as footnotes or in a list of references. The style file is couched in a full programming language and can support any vocabulary. However, there is general consensus in the TeX community about the keywords to use. Advantage can be taken of TeX's programmability to generate XML syntax instead; alternatively, there are many standalone applications that simply parse BibTeX source files.

Academic authors often create BibTeX bibliographic collections for publications in their area of expertise, accumulating references over the years into large and authoritative repositories of meta-data. Aided by software heuristics to identify duplicate entries, the repositories constitute a useful resource for digital libraries in scientific and technical areas.

EndNote

A format promoted by the popular bibliographic tool EndNote is illustrated in Figure 6.2g, which shows the same basic bibliographic record. The syntax is based on an earlier format that, like BibTeX, was designed for use by scientific and technical researchers. Today, many systems for maintaining bibliographic databases can export databases in this format. It is formatted line by line, and records are separated by a blank line. Each line starts with a key character, introduced by a percent symbol, that signals the kind of information the line contains. The rest of the line gives the data itself.

Table 6.4: EndNote keywords

Tag	Description
%0	Type of record (e.g., Book, Journal Article)
%1	Record identification key
%@	ISBN number
%A	Author
%B	Book title (for an article that is part of a book)
%C	Place (city) of publication
%D	Date of publication
%E	Editor (for an article that is part of a book)
%F	Caption
%G	Government ordering number (United States)
%I	Publisher (issuer)
%J	Journal name (for an article in a journal)
%K	Keywords
%L	Label
%N	Number of journal issue
%O	Other information (usually printed at the end of the reference)
%P	Page number—A range of pages can be specified as m–n
%Q	The name of the author, if the author is not a person
%R	Technical report number
%S	Series name
%T	Title
%U	URL
%V	Volume number of the journal or book
%X	Annotation
\$\$	Price
%*	Copyright information

The format has a fixed set of keywords, listed in [Table 6.4](#). The keyword %0, or percent zero, appears as the first line of a record to make the type explicit—Book, in our example. Unlike BibTeX, EndNote repeats the author field (%A) for multiple authors; the ordering reflects the document’s authorship.

6.3 Metadata for Multimedia

Metadata is by no means confined to textual documents. In fact, because it is much harder to search the content of image, audio, or multimedia data than to search full text, flexible ways of specifying metadata become even more important for locating these resources.

The image file formats described in Chapter 5 (Section 5.3) incorporate some limited ways of specifying image-related metadata. For example, GIF and PNG files include the height and width of the

image (in pixels), and the number of bits per pixel (up to 8 for GIF, 48 for PNG). PNG specifies the color representation (palletized, grayscale, or true color) and includes the ability to store text strings representing metadata. JPEG also specifies the horizontal and vertical resolution. But these formats do not include provision for other kinds of structured metadata, and when they are used in a digital library, image metadata is usually put elsewhere.

We describe several metadata formats for images and multimedia. The Tagged Image File Format, or TIFF, is a practical scheme for associating metadata with image files that has been in widespread use for well over a decade. TIFF is often used storing for images—including document images—in today’s digital libraries. EXIF, XMP/IPTC, and MIX are alternative approaches for image metadata: embedding metadata in the image file itself or separating the metadata from the image data. Temporal multimedia, such as audio and video, offer limited facilities for storing metadata within the files themselves. MPEG-7 is a far more sophisticated and ambitious scheme for defining and storing metadata associated with any multimedia information.

The following sections describe TIFF in specific detail, briefly cover metadata for temporal formats, and proceed to outline the facilities that MPEG-7 provides. Finally, we outline the usage-based framework of the MPEG-21 standard.

Image metadata: TIFF

The Tagged Image File Format, TIFF, described in Section 5.3, incorporates extensive facilities for descriptive metadata. It is used to describe image data that comes from scanners, frame-grabbers, paint programs, and photo-retouching programs. It is a rich format that can take advantage of many image requirements but is not tied to particular input or output devices. It provides numerous options—for example, several different compression schemes and comprehensive information for color calibration. It is designed so that private and special-purpose information can be included.

A single TIFF file can include several images, each of which is characterized by tags whose values define particular properties of the image. Most tags contain integers, but some contain ASCII text—and provision is made for tags containing floating-point and rational numbers. Baseline TIFF has a dozen or so mandatory tags that give physical characteristics and features of images: their dimensions, compression, various metrics associated with the color specification, and information about where they are stored in the file.

Table 6.5 shows some TIFF tags, all of which except the last group are mandatory. The first group specifies the image dimensions in pixels, along with enough information to allow conversion to physical units where possible. All images are rectangular. The second group gives color information. For bilevel images, the color group defines whether they are standard black-on-white or reversed; for grayscale it gives the number of bits per pixel; for palette images, it specifies the color palette. The third group specifies the compression method—baseline TIFF allows only extremely simple schemes. Finally, a TIFF image can be broken into strips for efficient input/output buffering, and the last group of mandatory tags specifies their location and size.

Additional features go far beyond the baseline illustrated in Table 6.5 and users can define new TIFF tags and compression. This makes the TIFF format even more flexible than the official list of exten-

Table 6.5: TIFF tags

Dimensions	Image width (in pixels) Image length (in pixels) Resolution unit: none, inch, cm X resolution: pixels per resolution unit Y resolution: pixels per resolution unit
Color	Photometric interpretation (black-on-white or white-on-black) Bits per sample (1 for bilevel, 4 or 8 for grayscale) Samples per pixel (RGB only): normally 3 for RGB images Color map (palette-color only): specifies a color table for the image
Compression	Bilevel: uncompressed, packed into bytes as tightly as possible, CCITT fax compression standard, byte-oriented run-length coding Others: uncompressed, byte-oriented run-length coding
Location of the data	Rows per strip Strip offsets Strip byte counts
Optional fields	Document name Page name (typically used for page number) Artist or creator Image description: free-form text Software program that generated the image Date and time when it was generated Host computer on which it was generated Make and model of the equipment used for scanning

sions given in the standard, although naturally care needs to be taken to ensure that the software used to read and write image files is conversant with the tags they use. To avoid conflict, a registration process is provided for allocating private tags. There are over 70 such tag sets, including support for EXIF (see next subsection). More radical extensions include GeoTIFF, which permits the addition of geographic information associated with cartographic raster data and remote sensing applications, such as projections and datum reference points. Many digital cameras produce TIFF files, and Kodak has a PhotoCD file format based on TIFF with proprietary color space and compression methods.

Most digital library projects of images use the TIFF format to store and archive the original captured images, even though they may convert them to other formats for display. At the bottom of [Table 6.5](#) are some optional fields that are widely used in digital library work. Some, such as the name of the program that generated the image and the date and time it was generated, are usually filled in automatically by scanner programs and other image-creation software. Digital library projects often establish conventions for the use of the other fields. For example, in a digitization project, the *Document name* field might contain the catalog ID of the original document. These fields are coded in ASCII, but there is no reason why they should not contain data that is further structured. For example, the *Image description* field might contain an XML specification that itself includes several subfields.

Image metadata: EXIF, XMP, IPTC, and MIX

The exchangeable image file format (EXIF) is a standard for embedding technical metadata in image files that many camera manufacturers use and many image-processing programs support. EXIF metadata can be embedded in TIFF and JPEG images. Table 6.6 shows a metadata record extracted from a JPEG image produced by a digital camera: it includes the exposure, resolution, focal length, and whether flash was used. The record shows that the GIMP image-editing program has been used to process the file, which illustrates that embedded metadata expressed in EXIF is only as accurate as the last application that saved the file and may have been altered by several programs before it is added to your digital library. EXIF metadata can include a thumbnail image—in this case the thumbnail is a 196×147 pixel JPEG that occupies 3.6 KB, compared with the original $2,048 \times 1,536$ (262 KB) image. Including the thumbnail, the metadata itself occupies 4.4 KB.

The DIG35 standard provides an extensible XML framework that can be used to embed metadata in the private tags of image formats like TIFF and JPEG. DIG35 groups elements into four main categories: image creation, content description, provenance, and intellectual property rights. Many elements are optional. The JPEG 2000 file format has an XML *Box* container that can include any XML data, although DIG35 is suggested.

In 2001, Adobe introduced the *Extensible Metadata Platform* (XMP) to provide a general solution for embedding metadata within files like images and PDF documents. XMP values are expressed in XML using a subset of RDF (see Section 6.4), allowing great flexibility in metadata expression. XMP's model is flexible enough to allow structural metadata to be expressed—so that, for example, different pages in a document can be described independently.

Within XMP, metadata can be expressed using different schemas and can be extended by adding new schemas. Schemas can be based on external standards (e.g., Dublin Core and EXIF) or internal schemas (e.g., XMP Rights Management) and can be associated with specific file types (e.g., PDF) or applications (e.g., Adobe Photoshop). Many applications now provide custom dialog windows to simplify the manual entry and editing of XMP metadata.

The XMP metadata itself is embedded in files in a serialized *Packet*, optionally surrounded by a wrapper consisting of a Header, Padding, and Trailer sections. The suggested size for the Padding section is 2–4 KB, allowing space for editing and expanding the metadata without overwriting application data. The actual mechanism for embedding the XMP packet varies depending on the particular file format. In PDF, for example, the packet is embedded in a metadata stream stored within a PDF object (see Section 4.5). Using custom embedding techniques allows XMP to be applied to a wide range of file types, including GIF, PNG, JPEG, JPEG 2000, Photoshop, MP3, MPEG-4, HTML, and WAV. When XMP cannot reliably be embedded in the native file format, as with MPEG-2, the packet is placed in a small “sidecar” file with the same filename and the suffix *.xmp*.

In 2008 the International Press Telecommunications Council (IPTC) released the *IPTC Photo Metadata 2008* standard, based on requirements gathered from device manufacturers and photographers, which is implemented as a core XMP schema. Many image-processing applications provide support for XMP metadata, and it is likely that support for IPTC embedded metadata will grow quickly.

Table 6.6: EXIF metadata record

Color space	sRGB
Components configuration	YCbCr
Compressed bits per pixel	0.533203125
Compression	JPEG (old-style)
Contrast	High
Create date	2003:02:17 20:39:36
Custom rendered	Normal
Date/time original	2003:02:17 20:39:36
Digital zoom ratio	inf
Exif image size	2,048 × 1,536
Exif version	0220
Exposure compensation	+1
Exposure mode	Manual
Exposure program	Program AE
Exposure time	1/40
F number	2.6
File source	Digital camera
Flash	Off
Flashpix version	0100
Focal length	5.8 mm
Focal length in 35 mm format	35 mm
Gain control	High gain up
Interoperability index	R98—DCF basic file (sRGB)
Interoperability version	0100
Light source	Unknown
Make	Casio Computer Co. Ltd
Max aperture value	2.6
Metering mode	Center-weighted average
Camera model name	EX-Z3
Modify date	2009:10:06 10:50:29
Orientation	Horizontal (normal)
Print image matching	(40 bytes binary data)
Resolution	72 pixels/inch
Saturation	Normal
Scene capture type	Standard
Sharpness	Hard
Software	GIMP 2.4.4
Thumbnail length	3,672
White balance	Auto
YCbCr positioning	Centered

Metadata for Images in XML (MIX) is a new standard for image metadata that, in contrast to the *de facto* use of EXIF, is being coordinated and promoted by the Library of Congress. Like EXIF, it addresses technical rather than descriptive metadata. Its elements represent scanner resolutions, details of camera exposure, and processing software—and nothing about the image content or who took a particular photograph. Its flavor is illustrated by Figure 6.3, which shows a fragment of a MIX document for a photograph.

Both EXIF and MIX-style metadata address the technical and preservation aspects of the five-part metadata classification in Section 6.1. EXIF is almost universal in consumer electronics but is restricted to certain image types, whereas MIX, being relatively new, is not yet widely used. The IPTC Photo Metadata standard is very new but has strong growth potential: it is open, has an extensible metadata model, and is based on the well-known XMP approach.

Audio metadata

Although audio resources are often described using external metadata schemes, such as Dublin Core, several audio formats contain embedded metadata. The ubiquitous MP3 format is an interesting case.

```
<?xml version="1.0"?>
<CameraCaptureSettings>
  <ImageData>
    <fNumber>3</fNumber>
    <exposureTime>3.1</exposureTime>
    <exposureProgram>1</exposureProgram>
    <spectralSensitivity>0.015</spectralSensitivity>
    <isoSpeedRatings>100</isoSpeedRatings>
    <oECF use="system">
      <numerator>2</numerator>
    </oECF>
    <exifVersion>0220</exifVersion>
    <shutterSpeedValue use="system">
      <numerator>1</numerator>
      <denominator>8</denominator>
    </shutterSpeedValue>
    <apertureValue>
      <numerator>1</numerator>
      <denominator>3</denominator>
    </apertureValue>
    <brightnessValue use="system">
      <numerator>99.99</numerator>
    </brightnessValue>
    <exposureBiasValue>
      <numerator>3.0</numerator>
    </exposureBiasValue>
    <maxApertureValue>
      <numerator>0.00</numerator>
    </maxApertureValue>
    <subjectDistance>5.3</subjectDistance>
    <meteringMode>Multispot</meteringMode>
    <lightSource>Daylight</lightSource>
    <flash>Flash fired</flash>
  </ImageData>
</CameraCaptureSettings>
```

Figure 6.3: Fragment of a MIX document representing a photograph

Initially it had virtually no internal metadata, apart from the technical audio information needed to play the file.

The first set of metadata tags for MP3 was developed in 1996 and is known as ID3 (for “IDentify an MP3”). ID3 tags were placed at the end of the audio data (in an attempt to minimize disruption to software). They were limited to 128 bytes: 3 for the starting string “TAG,” which identified the metadata block, 30 bytes for the title, 30 for the artist, 30 for the album, 4 for the year, 1 for track number, 1 for genre, and 30 for comments. The genre byte was an index into a list of 80 predefined genres. These limits were found to be too restrictive for describing music metadata and the ID3 format has been evolving ever since to provide more flexible metadata containers.

The current ID3v2 tag format allows many *frames* of data of variable size (up to 16 MB each) that can contain text, images, and technical metadata about the audio data. They are normally located at the beginning of the file to assist media streaming software. There are 39 predefined text frames, including composer, length, copyright, publisher, and title. Other frames provide scope for URLs, images (such as CD cover art), synchronized lyrics, and, in a “commercial” frame, for advertising. All frames are optional, and users can also define their own. MP3 ID3 is not a formal standard, so metadata extracted from MP3 collections is likely to be incomplete and inconsistent.

The Ogg container format that wraps Vorbis audio (Section 5.2, Post-MP3 formats) does not include a structured metadata component, although there is a comment facility for simple textual values. Vorbis comments take the form *fieldname=data*, where the *fieldname* is a case-insensitive ASCII string and *data* is a UTF-8 encoded string. A proposed set of field names includes TITLE, COPYRIGHT, and GENRE, but, as with MP3, none is mandatory and usage is variable. FLAC files support Vorbis comments along with a PICTURE block for storing an image and a CUESHEET block that uses track and index points compatible with the CD audio standard.

WAV audio is a form of RIFF file consisting of *chunks* (Section 5.2, Early formats). A LIST chunk with a type of INFO provides a limited facility for embedded metadata. A variety of text chunks are available, including genre, source, and copyright, and there is a display chunk that allows text or images to be associated with the audio data. AIFF files have optional chunks for name, author, copyright, annotation, and comments. Table 5.1 lists some of the chunk types for WAV and AIFF files. The AU format has a single unstructured text field.

The WAV *cue* chunk (or the comparable AIFF *marker*) allows points of interest and text to be associated with particular points in an audio track internally. These cue points can provide useful structural metadata, and the text can be used to supplement the listening experience for users. Alternatively, the Continuous Media Markup Language (CMML) is a generalized approach to linking metadata with a specific point in temporal media. CMML is an XML format for linking text to time periods through *clip* elements, such as

```
<clip id="painting" start="npt:3.5" end="npt:5:5.9">
  
  <desc>A close up of the painting</desc>
  <meta name="Subject" content="painting"/>
</clip>
```

which defines a description, subject metadata, and image—and associates it with a period of audio. CMML can be used to synchronize lyrics with music, to allow metadata values to be assigned with temporal precision, and to provide subtitling for video. CMML can be serialized into data that can be inserted into container formats such as Ogg.

Neither AAC nor MPEG-4 audio includes a standard metadata format. However, Apple's tagging format has become a *de facto* standard for AAC because of its use in the popular iTunes music store. It exploits the generic user-data ("udta") *atom* (or *box*), and metadata is stored as tag-value pairs. The values themselves are flexible, textual metadata (e.g., the *aART* tag for album artist, *cpri* for copy-right, and *©nam* for title). They are stored as UTF-8 strings. iTunes also sets a *covr* value for a cover art image (typically JPEG or PNG).

Video metadata

Much video data is delivered within container formats like AVI, MPEG-4, Flash, or Ogg, and these containers often provide embedded metadata support. Apple's QuickTime format became the basis for the MPEG-4 container format, so MPEG-4 handles metadata in a similar way to the iTunes AAC audio discussed above. AVI is a form of RIFF file, so it handles metadata in a similar way to WAV. Ogg Theora video uses the same container format as Ogg Vorbis audio, so the same comment structure is supported. However, software support for these video comments is currently limited.

MPEG-2 video has no textual metadata support. However, with some extensions and restrictions, it is used to add subtitles and menu information to DVD video. The added information is put on the DVD as images rather than text—thus avoiding the need for DVD players to include the entire set of Unicode characters as fonts. The lack of metadata support in MPEG-2 led to the MPEG-7 format described in the next section.

The basic idea of CMML—to associate external resources with time points—is used by several subtitling/captioning technologies, such as MPEG-4 Timed Text (supported in Apple's Quicktime tools). Most subtitles are simply plain text rather than structured metadata, although they could be used to enable limited text searching of video collections. The larger idea of coordinating temporal multimedia content is embodied in QuickTime, SMIL, and Flash (Sections 5.4 and 5.5).

The relative lack of structured metadata embedded in video files has been addressed in two ways: expanded container formats with explicit metadata support, and placing the metadata in external storage, such as linked XML files.

The Material Exchange Format (MXF) is a new container format that can hold many types of video data in a platform-independent manner. It is managed by the Society of Motion Picture and Television Engineers (SMPTE), who maintain central lists of metadata labels, metadata dictionaries, and the basic Descriptive Metadata Schema (known as DMS-1). MXF metadata is based on keys (e.g., *Titles:MainTitle*) and values and can be defined either globally or for specific time intervals.

MPEG-7 has an alternative but complementary approach, where external metadata is expanded to include a low-level description of content as well as the familiar Dublin Core-like high-level descriptions.

It is early days for video content, at least in terms of digital library usage, and few tools support metadata editing. Despite the availability of methods for splicing sophisticated metadata into video container file formats, in practice the video material that is actually supplied is often inadequate for the needs of digital libraries. As we have seen, collections like the Open Video project (Section 5.4) augment the video material with external metadata that meets their needs.

Multimedia metadata: MPEG-7

We learned about the MPEG family of standards in Section 5.4. MPEG-7, which is formally called the *multimedia content description interface*, is intended to provide a set of tools for describing multimedia content. The aim is to allow the user to search for audiovisual material that has associated MPEG-7 metadata. The material can include still pictures, graphics, 3D models, audio, speech, video, and any combination of these elements in a multimedia presentation. It can be information that is stored or that is streamed from an online real-time source. The MPEG-7 standard was developed by broadcasters, electronics manufacturers, content creators and managers, publishers, and telecommunication service providers under the aegis of the International Organization for Standardization (ISO).

MPEG-7 has exceptionally wide scope. For example, it foresees that metadata may be used to answer queries like:

- The user plays a few notes on a keyboard and retrieves musical pieces with similar melodies, rhythms, or emotions.
- The user draws a few lines on a screen and retrieves images containing similar graphics, logos, or ideograms.
- The user defines objects, including color or texture patches, and retrieves similar examples.
- The user describes movements and relations between a set of given multimedia objects and retrieves animations that exhibit them.
- The user describes actions and retrieves scenarios containing them.
- Using an excerpt of Pavarotti's voice, the user obtains a list of his records and video clips and photographic material portraying him.

We encountered systems that embody the first and third of these examples in Chapter 3. However, it should be noted that the way in which MPEG-7 metadata is used to support such queries is beyond the scope of the standard.

MPEG-7 is based on four components: *Descriptors*, *Description Schemes*, *Description Definition Language*, and *Systems Tools*. Descriptors represent low-level features, the fundamental qualities of audiovisual content. They range from statistical models of signal amplitude to the fundamental frequency of a signal, from emotional content to parameters of an explicit sound-effect model. Description Schemes specify the types of Descriptors that can be used and the relationships between them or between other Description Schemes. The Description Definition Language (DDL) allows users to extend the predefined description capabilities of Descriptors and Description Schemes. DDL uses XML syntax and is a form of XML Schema (more details appear in an appendix at the book's Web

site). In this case, XML Schema alone is not flexible enough to handle low-level audiovisual forms, so DDL was formulated to address these needs. Finally, the Systems Tools are a suite of basic tools to synchronize metadata descriptions with content encoding and transmission.

A Description Scheme, together with instantiated values for a Descriptor, produces an actual Description, such as the one shown in Figure 6.4. The three AttributeValuePairs in the lower half (un-indented) give a trio of values of features (called *HSV_lin-4x4x4-G*, *tamuraS-2-A*, and *gabor-6-4*) that were derived by analyzing the image whose URL is specified in the *MediaUri* element. The actual values appear in the three lists of numbers (the *FloatMatrixValue* elements). In fact, this example is from the system we described under Content-Based Image Search in Section 3.4 (illus-

```
<?xml version="1.0" encoding="UTF-8"?>
<Mpeg7 xsi:schemaLocation="urn:mpeg:mpeg7:schema:2001 ./davp-2005.xsd" ...>
  <Description xsi:type="ContentEntityType">
    <MultimediaContent xsi:type="AudioVisualType">
      <AudioVisual>
        <StructuralUnit href="urn:x-mpeg-7-pharos:cs:AudioVisualSegmentationCS:root"/>
        <MediaSourceDecomposition criteria="kmi image lowlevel features">
          <StillRegion>
            <MediaLocator>
              <MediaUri href="http://www.kmi.open.ac.uk/people/helen/cat.jpg"/>
            </MediaLocator>
            <StructuralUnit href="urn:x-mpeg-7-pharos:cs:SegmentationCS:image"/>
            <Semantic>
              <Label/>
              <SemanticBase xsi:type="SemanticStateType">
                <Label/>
              </SemanticBase>
            </Semantic>
            <AttributeValuePair>
              <Attribute href="urn:x-mpeg-7-pharos:cs:FeatureCS:image:kmi:HSV_lin-4x4x4-G"/>
              <FloatMatrixValue mpeg7:dim="52">0.0050 0.0049 0.0084 0.0005 0.0000 0.0018 0.0013
0.0 002 0.0001 0.0093 0.0003 0.0005 0.0010 0.0204 0.0089 0.0492 0.0037 0.0003 0.0077
0.0020 0.0000 0.0004 0.0400 0.0000 0.0000 0.0005 0.0188 0.0112 0.0360 0.0339 0.0002
0.0118 0.0000 0.0000 0.0001 0.0134 0.0000 0.0000 0.0000 0.2596 0.0506 0.0313 0.3667
0.0000 0.0002 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000</FloatMatrixValue>
            </AttributeValuePair>
            <AttributeValuePair>
              <Attribute href="urn:x-mpeg-7-pharos:cs:FeatureCS:image:kmi:tamuraS-2-A"/>
              <FloatMatrixValue mpeg7:dim="3">3.2319 46.1946 3.8093</FloatMatrixValue>
            </AttributeValuePair>
            <AttributeValuePair>
              <Attribute href="urn:x-mpeg-7-pharos:cs:FeatureCS:image:kmi:gabor-6-4"/>
              <FloatMatrixValue mpeg7:dim="48">1.3049 1.9345 1.1704 2.5407 1.4041 2.3992 1.2248
2.6 779 0.9925 1.5276 0.9085 1.9340 1.0974 1.9481 0.9352 1.9238 0.8344 1.1988 0.7632
1.8182 0.9451 1.9347 0.7535 1.2778 0.7291 0.9900 0.6408 1.2625 0.7580 1.2591 0.6345
0.8528 0.6011 0.7880 0.4867 0.9261 0.5491 0.8229 0.4702 0.6047 0.3985 0.5865 0.2590
0.5236 0.3186 0.4729 0.2500 0.3690</FloatMatrixValue>
            </AttributeValuePair>
          </StillRegion>
        </MediaSourceDecomposition>
      </AudioVisual>
    </MultimediaContent>
  </Description>
</Mpeg7>
```

Figure 6.4: Sample MPEG-7 description

trated in Figure 3.17), which, as we learned there, derives several mysteriously named features from the image content.

The MPEG-7 DDL is able to express spatial, temporal, structural, cardinal and data-type relationships between Descriptors and Description Schemas. For example, structural constraints specify the rules that a valid Description should obey: what child elements must be present for each node, or what attributes must be associated with elements. Cardinality constraints specify the number of times an element may occur. Data-type constraints specify the type and the possible values for data or Descriptors within the Description.

There are different Descriptors for audio, visual, and multimedia data. The audio description framework operates in both the temporal and spectral dimensions, the former for sequences of sounds or sound samples, and the latter for frequency spectra that comprise different components. At the lowest level, you can represent such things as instantaneous waveform and power values, various features of frequency spectra, fundamental frequency of quasi-periodic signals, a measure of spectral flatness, and so on. There is a way to construct a temporal *series* of values from a set of individual samples, and a spectral *vector* of values, such as a sampled frequency spectrum. At a higher level, description tools are employed for sound effects, instrumental timbre, spoken content, and melodic descriptors to facilitate query-by-humming.

In the visual domain, basic features include color, texture, region-based and contour-based shapes, and camera and object motion. Another basic feature is a notion of *localization* in both time and space, and these dimensions can be combined in a space-time trajectory. These basic features can be built into structures like a grid of pixels or a time series of video frames.

Multimedia features include low-level audiovisual attributes like color, texture, motion, and audio energy; high-level features of objects, events, and abstract concepts; and information about compression and storage media. Browsing and retrieval of audiovisual content can be facilitated by defining summaries, partitions, and decompositions. Summaries, for example, allow an audiovisual object to be navigated either hierarchically or sequentially. For hierarchical navigation, material is organized into successive levels that describe it at different levels of detail, from coarse to fine. For sequential navigation, images or video frames can be arranged in sequences and possibly synchronized with audio and text so that they compose a slide show or audiovisual synopsis.

MPEG-7 descriptions can be entered by hand or extracted automatically from the signal (as in Figure 6.4). Some features (color, texture) can best be extracted automatically, while others (“this scene contains three shoes,” “that music was recorded in 1995”) cannot be extracted automatically.

The application areas envisaged for MPEG-7 are many and varied and stretch well beyond the ambit of what most people mean by digital libraries. They include education, journalism, tourist information, cultural services, entertainment, geographical information systems, remote sensing, surveillance, biomedical applications, shopping, architecture, real estate, interior design, film, video and radio archives, and even dating services.

Multimedia application metadata: MPEG-21

MPEG-7 provides a description framework for multimedia content that supports many digital library functions, such as structured browsing and surrogate creation. However, in reality, multimedia like

music and video participate in complex international flows of content that are worth vast amounts of money. As we all know, some of this content is illegally distributed at various points in its lifecycle, due to the simplicity and fidelity of digital copying. MPEG-21 is a broad and ambitious standard that links multimedia producers and consumers to address issues of content control. According to the official overview,

MPEG-21 aims at defining a normative open framework for multimedia delivery and consumption for use by all the players in the delivery and consumption chain. This open framework will provide content creators, producers, distributors and service providers with equal opportunities in the MPEG-21 enabled open market. This will also be to the benefit of the content consumer, providing them access to a large variety of content in an interoperable manner.

MPEG-21 is founded upon two concepts: a *Digital Item* (such as a video or music album) and the *User's Interaction* with it. Because it is concerned with the access, consumption, and exchange of multimedia content, MPEG-21 emphasizes aspects of user interaction that are not considered by other metadata frameworks for digital content. Technically, Digital Items are compound objects that include multimedia content and metadata. Each Digital Item is defined by a Digital Item Declaration (DID) expressed in the Digital Item Declaration Language (DIDL) and written as an XML Schema. Another key part of the MPEG-21 framework is the Rights Expression Language, also written using XML Schema. The Rights Expression Language is based on licenses that grant principals (the users) the right to use a resource (such as a video) under certain conditions (such as a time limit). The intention is that MPEG-21 can link a multimedia resource with a machine-readable statement, such as *Alice permits Bob to play the video "Stupid Pet Tricks vol 26.mpg" during November 2011*.

The development of MPEG-21 illustrates two important factors in the future of digital libraries: 1) the computational specification of rights management and 2) increasingly complex compound digital objects.

6.4 Metadata for Compound Objects

As we have seen, the descriptions of digital objects are becoming richer, and they are ever more interlinked with other objects or online resources—and the metadata needed to represent these connections is itself a source of complexity. In this section we provide examples of technologies that allow complex *compound objects* to be rigorously defined.

Resource Description Framework: RDF

The Resource Description Framework (RDF) is designed to facilitate the interoperability of metadata, particularly in the realm of the World Wide Web. Because metadata covers too great a variety of information to specify exhaustively and categorically, RDF follows the lead of XML: rather than providing a set of possibilities, it supplies a means for describing a valid system. It is expected that communities of users will assemble to establish RDF systems suited to their collective needs. They will have to agree on a vocabulary, its meaning, and the structures that can be formed from it. This is done by specifying an RDF Schema, just as DTDs and XML Schemas are used to control XML vocabulary and structure.

RDF is a way of modeling as a resource anything that can be represented as a Universal Resource Identifier. What is a URI? Well, officially, according to the W3C standard (RFC 3986), a Universal Resource *Identifier* is anything expressed in the appropriate format that serves to identify a Web resource, and thus a URI includes both locators—URLs—and names—URNs. Universal Resource *Locators* are a subset of URIs that, in addition to identifying the resource, describe its location on the network. Universal Resource *Name* refers to any URI that is guaranteed to remain globally unique and persistent even when the resource is deleted or becomes unavailable. A Web page, part of a document, an FTP site, or an e-mail address are all examples of the former, and the International Standard Book Number (ISBN) is an example of the latter.

RDF resources are described in a machine-readable fashion through a framework for specifying—in digital library terminology—metadata. The framework is compositional: new resources can be built from existing ones.

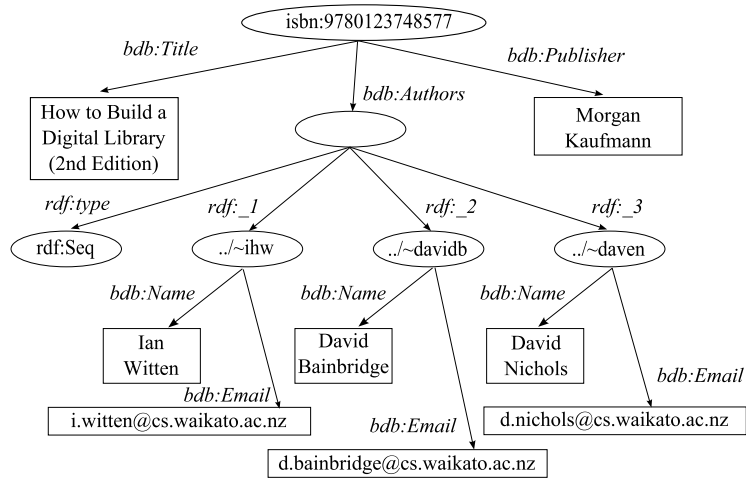
Figure 6.5 uses RDF to give a graphical description of the book you are holding in your hands. The book is represented by its ISBN (International Standard Book Number) in URI syntax. The top-level description comprises a title, three authors, and a publisher. The authors in turn are characterized by a name and an e-mail address.

Mapping the graphical description into a character stream is a process of serialization, and for RDF the language of choice is (again) XML. Figure 6.5b shows the description of the example graph. This representation obscures some of the abstract aspects of RDF and makes it seem like just another example of an XML language—especially since we have recently seen so many of them! When considering RDF, you should keep in mind that it is a sister format to XML, not a subsidiary. However, this medium bestows practical benefits: software support for parsing and editing, transparent handling of international characters, and so on.

The basic construction in RDF (indeed as it is in English) is to connect a subject to an object using a predicate. This is known as a statement. Subjects and objects are nodes in the graph. The directed arc that joins them shows the connection; a label on the arc identifies the predicate used. To take an example from Figure 6.5a, *How to Build a Digital Library* (object) is the *Title* (predicate) of the resource *ISBN: 9780123748577* (subject). The subject is a resource, represented in the diagram as an oval node. An object may be either a resource or a string literal. In our case it is a string, represented as a rectangular node.

Now let's match up the graphical representation with the serialized form in Figure 6.5b. It begins with the familiar processing instruction, and a root node that declares some namespaces. In this case there are two: one for RDF and another for "Book Database," denoted by the prefix *bdb*. This represents a hypothetical organization that has developed an XML schema for representing metadata about books in terms of titles, authors, names, e-mail addresses, and publishers.

The first child of the root node is an *rdf:Description* element that includes an *about* attribute to specify the resource as a URI that gives the ISBN. The *bdb>Title* tag that follows sets the predicate, and its content represents the string literal object that forms the title itself. Two other predicates connected to the ISBN resource are *Authors* and *Publisher*. The latter, like *Title*, declares a string literal



```

<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:bdb="http://bookdatabase.org/schema/publications">

  <rdf:Description about="urn:isbn: 9780123748577">
    <bdb:Title>How to build a digital library</bdb:Title>
    <bdb:Authors>
      <rdf:Seq>
        <rdf:li>
          <rdf:Description about="http://www.cs.waikato.ac.nz/~ihw">
            <bdb:Name>Ian Witten</bdb:Name>
            <bdb:Email>i.witten@cs.waikato.ac.nz</bdb:Email>
          </rdf:Description>
        </rdf:li>
        <rdf:li rdf:resource="http://www.cs.waikato.ac.nz/~davidb"/>
        <rdf:li rdf:resource="http://www.cs.waikato.ac.nz/~daven"/>
      </rdf:Seq>
    </bdb:Authors>
    <bdb:Publisher>Morgan Kaufmann</bdb:Publisher>
  </rdf:Description>

  <rdf:Description about="http://www.cs.waikato.ac.nz/~davidb">
    <bdb:Name>David Bainbridge</bdb:Name>
    <bdb:Email>d.bainbridge@cs.waikato.ac.nz</bdb:Email>
  </rdf:Description>

  <rdf:Description about="http://www.cs.waikato.ac.nz/~daven">
    <bdb:Name>David Nichols</bdb:Name>
    <bdb:Email>d.nichols@cs.waikato.ac.nz</bdb:Email>
  </rdf:Description>

</rdf:RDF>

```

(b)

Figure 6.5: Modeling this book using RDF: (a) graphical model; (b) XML serialization

object (the string “Morgan Kaufmann”); the former, however, is more complex. Thus we see how the top level of the tree in Figure 6.5 is constructed.

Not only is *bdb:Authors* the first object encountered in our explanation that is itself a resource, but also it is an example of an *anonymous resource*—an intermediary node that has no specific resource name but is itself the subject of further qualifying statements. The counterpart in the graphical version of the model in Figure 6.5a is the node at the end of the *Authors* predicate, which is nameless.

The anonymous resource also demonstrates a new structure called a *container*, which is used to group resources together. RDF has three types of container: *bag*, *sequence*, and *alternative*. A *bag* is an unordered list of resources or string literals; a *sequence* is an ordered list; and an *alternative* represents the selection of just one item from the list. Each item in a *container* is denoted by an *rdf:li* tag. The example uses *rdf:Seq* to represent a sequence of authors because the order is significant. The container type is represented by the *rdf:type* predicate, and its contents are numbered *rdf:_1*, *rdf:_2*, *rdf:_3*. These are implicitly inferred from the XML description (Figure 6.5b) but are shown explicitly in the pictorial version (Figure 6.5a).

There are three list items in the example, one for each author, and they happen to be compound resources. They are introduced by the RDF list item tag *rdf:li*, and for illustrative purposes they are specified in different ways. The first is embedded in-line by starting a new *rdf:Description* tag. The second and third receive a more compact *rdf:li* tag that defers the resource’s definition through the *rdf:resource* attribute. The missing detail is filled in when a resource description is encountered whose *rdf:about* attribute matches the named list item. This occurs in the lower third of the example.

RDF is designed to be created and read by machines, rather than people, so its verbosity is less important than its descriptive power and flexibility as a format. Figure 6.6 shows an automatically generated RDF file that describes a Web resource, the home page of the World Wide Web Consortium. The page has been downloaded and analyzed to extract metadata values, which have been expressed as Dublin Core metadata and included in the RDF file by importing the Dublin Core namespace in the *rdf* element (the value of attribute *xmlns:dc*, near the beginning). This example shows how RDF can interoperate with other metadata schemas to contribute to the Semantic Web. An extension of URIs called Internationalized Resource Identifiers (IRIs) has been proposed to extend RDF to resources that use Chinese, Korean, and Cyrillic characters.

RDF is a rich framework whose design draws upon structured documents, entity relationships, object orientation, and knowledge representation. We cannot illustrate all aspects here, but the above examples convey the flavor of what it does.

Metadata Encoding and Transmission Standard: METS

The Metadata Encoding and Transmission Standard (METS) is designed to permit the representation, maintenance, and exchange of the increasingly complex digital objects that make up digital libraries. Library catalogs have for years used MARC records to fulfill these functions for their materials. The METS initiative aims to do the same for digital collections.

```

<?xml version="1.0"?>
<!DOCTYPE rdf:RDF SYSTEM
    "http://dublincore.org/documents/2002/07/31/dcmes-xml/dcmes-xml-dtd.dtd">
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/">
<rdf:Description rdf:about="http://www.w3.org/">
  <dc:title> World Wide Web Consortium - Web Standards </dc:title>
  <dc:subject>
    W3C; World Wide Web; Web; WWW; Consortium; computer; access; accessibility;
    semantic; worldwide; W3; XHTML; XML; standard; language; technology; link;
    CSS; RDF; XSL; Berners-Lee; Berners; Lee; style sheet; cascading; schema;
    XHTML; mobile; SVG; PNG; PICS; DOM; SMIL; MathML; markup; Amaya; Jigsaw;
    free; open source; software
  </dc:subject>
  <dc:description>
    The World Wide Web Consortium (W3C) is an international consortium where
    Member organizations, a full-time staff, and the public work together to
    develop Web standards. W3C primarily pursues its mission through the
    creation of Web standards and guidelines designed to ensure long-term
    growth for the Web. Over 400 organizations are Members of the Consortium.
    W3C is jointly run by the MIT Computer Science and Artificial Intelligence
    Laboratory (MIT CSAIL) in the USA, the European Research Consortium for
    Informatics and Mathematics (ERCIM) headquartered in France, Keio
    University in Japan, and has additional Offices worldwide.
  </dc:description>
  <dc:date> 2009-01-03 </dc:date>
  <dc:type> Text </dc:type>
  <dc:format> text/html; charset=utf-8 </dc:format>
  <dc:format> 41316 bytes </dc:format>
</rdf:Description>
</rdf:RDF>

```

Figure 6.6: RDF representation of the W3C home page

METS is an open standard implemented in XML that encodes three of the metadata types identified in [Section 6.1](#): descriptive, structural, and administrative. It also contains an extension mechanism, through which it can encompass further areas. A METS document consists of six major sections: *Structural Map*, *Structural Links*, *File*, *Descriptive Metadata*, *Administrative Metadata*, and *Behaviors*. The Structural Map provides the structure that links content and metadata together; it is the only compulsory element. Structural Links allow METS creators to record the existence of hyperlinks between nodes in the Structural Map. The File section lists the actual files that make up the digital object. Descriptive Metadata and Administrative Metadata come in two possible forms: external metadata, such as a MARC record in a library catalog, and internal metadata, such as a Dublin Core or MODS record for Descriptive Metadata or a MIX or PREMIS (see below) record for Administrative Metadata. Finally, the Behaviors section is used to associate executable behavior with content in a METS object.

METS is a complex format and can be used in different ways—for example, as the internal format of a digital library, or as an exchange format. [Figure 6.7](#) shows an example document. The simplest part is the File section in the middle (`<mets:fileSec>` element), which identifies the content files. The *Flocat* elements all use a URL and an XLink pointer (see the appendix at the book's Web site) to

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<mets:mets xmlns:mets="http://www.loc.gov/METS/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:gsdl3="http://www.greenstone.org/namespace/gsdldata/1.0/"
  xmlns:xlink="http://www.w3.org/TR/xlink"
  xsi:schemaLocation="http://www.loc.gov/METS/
    http://www.loc.gov/standards/mets/mets.xsd
    http://www.greenstone.org/namespace/gsdldata/1.0/
    http://www.greenstone.org/namespace/gsdldata/1.0/gsdldata.xsd"
  OBJID="HASH01957a39eda286c59471aa53:2">
<mets:dmdSec ID="DM1" GROUPID="1">
  <mets:mdWrap MDTYPE="OTHER" OTHERMDTYPE="gsdl3" ID="gsdl1">
    <mets:xmlData>
      <gsdl3:Metadata name="Language">en</gsdl3:Metadata>
      <gsdl3:Metadata name="Encoding">iso_8859_1</gsdl3:Metadata>
      <gsdl3:Metadata name="FileSize">9312</gsdl3:Metadata>
      <gsdl3:Metadata name="Title">HEY JUDE!</gsdl3:Metadata>
      <gsdl3:Metadata name="FileFormat">HTML</gsdl3:Metadata>
      <gsdl3:Metadata name="Identifier">HASH01957a39eda286c59471aa53</gsdl3:Metadata>
      ...
    </mets:xmlData>
  </mets:mdWrap>
</mets:dmdSec>
...
<mets:fileSec>
  <mets:fileGrp ID="FILEGROUP_PRELUDE1">
    <mets:file MIMETYPE="text/xml" ID="FILE1">
      <mets:FLocat LOCTYPE="URL"
        xlink:href="file:doctxt.xml#xpointer(/Section[1]/text())" />
    </mets:file>
  </mets:fileGrp>
  <mets:fileGrp ID="default">
    <mets:file MIMETYPE="text/xml" ID="default.1">
      <mets:FLocat LOCTYPE="URL" xlink:href="heyjudeUS.htm" />
    </mets:file>
    <mets:file MIMETYPE="image/gif" ID="default.2">
      <mets:FLocat LOCTYPE="URL" xlink:href="title.gif" />
    </mets:file>
    <mets:file MIMETYPE="image/jpeg" ID="default.3">
      <mets:FLocat LOCTYPE="URL" xlink:href="70USheyjudeBIG.jpg" />
    </mets:file>
  </mets:fileGrp>
</mets:fileSec>
...
<mets:structMap ID="Section" TYPE="Section" LABEL="Section">
  <mets:div ID="DS1" TYPE="Section" ORDER="0" ORDERLABEL="1" LABEL="1" DMDID="DM1">
    <mets:fptra FILEID="FILEGROUP_PRELUDE1" />
  </mets:div>
</mets:structMap>
<mets:structMap ID="All" TYPE="Whole Document" LABEL="All">
  <mets:div ID="DSAll" TYPE="Document" ORDER="0" ORDERLABEL="All"
    LABEL="Whole Document" DMDID="DM1">
    <mets:fptra FILEID="default.1" />
    <mets:fptra FILEID="default.2" />
    <mets:fptra FILEID="default.3" />
  </mets:div>
</mets:structMap>
</mets:mets>

```

Figure 6.7: Sample METS document

locate a file, together with a MIME type and an identifier, so that other parts of the description can refer to it.

The Structural Map parts at the end (`<mets:structMap>`) use *div* elements to indicate divisions within the document (following a convention established in HTML), and `<mets:fptr>` elements to indicate which content files contribute to each division (using the ID values from the *Flocat* elements in the File section). There are two Structural Maps in Figure 6.7, each with one *div*; the second consists of three content files. Structural Maps could reflect the physical structure of a scanned book in pages or its logical structure in chapters; the indirect way in which they are expressed allows multiple perspectives to be described for the same basic content.

The Descriptive Metadata section (`<mets:dmdSec>`) continues the theme of flexibility. Figure 6.7 has embedded metadata near the top (indicated by the `<mdWrap>` element) of type *Other*—in this case, a specific Greenstone style of metadata. More standard metadata types, such as MODS or Dublin Core, could be included instead (or as well), either embedded within a `<mdWrap>` element or linked externally through a `<mdRef>` element.

Figure 6.7 does not show the Administrative Metadata section (`<mets:amdSec>`), which encodes technical metadata (such as MIX), intellectual property rights, information about a physical source document, and preservation metadata. A typical component for the last is PREMIS, which describes information in terms of Intellectual Entities, Objects, Events, Rights, and Agents. Practically, such metadata will be split into groups to match METS sub-elements of `<mets:amdSec>`. The use of PREMIS highlights the redundancy that often occurs when multiple metadata schemes are combined—file sizes and checksums can be part of an *objectCharacteristic* element in PREMIS and an attribute of a *file* element in METS. Redundancy is not necessarily a bad thing, and here it is reasonable to use both.

Finally, as noted above, the Structural Links section of a METS document includes hyperlinks between parts of the Structural Map, in order to model Web content, while the Behavior section associates executable behavior with METS objects.

One of the consequences of the complexity and richness of METS is that its use requires extensive support in the form of software tools. Even for simple objects, descriptions can easily be 50 lines of XML, and complex objects can produce documents 50 KB or more in size.

A useful component of the METS framework is the ability to express a profile. Profiles represent constraints and requirements that METS documents must fulfill to conform to the profile. Restrictions on the document values make it easier for software tools to process the documents and simplify exchange and aggregation. For example, a profile can require that there is no Behavior section, that the images referenced from the document must be in TIFF v6.0 format, or that subject headings in an embedded MODS section must use Library of Congress Subject Headings. Profile requirements are expressed in natural language, so conformance to a profile cannot be checked automatically.

Collection-level metadata

The examples of metadata seen so far have been for individual items. Typically, many items are grouped together in collections, and it becomes useful to consider metadata at the collection level.

Such metadata could include the people who collated the collection, background on why it was created, and who to contact for maintenance requests. Collection-level metadata can also include information derived automatically from the items, such as their total number, the earliest item, or recent additions.

Collection-level metadata helps to contextualize item-level metadata that has been divorced from its collection. The need for this is exemplified by the so-called “on a horse” problem, where a collection of material about *Theodore Roosevelt* (collection-level metadata) contains a photograph with the description *on a horse* (item-level metadata). The description makes sense within the bounds of the collection, but the item-level metadata will be incomprehensible if it is separated from the collection (e.g., through harvesting via OAI-PMH, as described in Chapter 7).

This problem shows that collection-level metadata can be used to supplement item-level metadata to enhance user understanding, shareability, and searching and browsing. It also allows users to start their information-gathering process one step earlier by interrogating a set of collections to determine which best suits their need. For this idea to work across different digital library systems, standard terms and meanings are required—exactly what RDF, for example, was designed for.

The Research Support Libraries Programme (RSLP) Collection Description project is an example of collection-level metadata. The project uses a model that represents *Collections*, *Locations*, and *Agents* as RDF resources. There are three types of agent—*Collector*, *Owner*, and *Administrator*—reflecting the roles that people or organizations play in providing and maintaining a collection. The bulk of the detail is contained in the collection resource, which may in turn reference further resources, such as a location and a collector.

Figure 6.8 shows an abridged version of a description for the Morrison Collection of Chinese Books housed at the School of Oriental and African Studies Library in London, England. Four existing namespaces provide relevant elements and attributes: RDF (naturally), unqualified Dublin Core, qualified Dublin Core, and vCard, which is a namespace devised by the Internet Mail Consortium to represent fax numbers, phone numbers, and so on for electronic business cards. An additional namespace for RSLP covers attributes and elements not defined elsewhere (prefix *cl*d).

The example contains four top-level resource descriptions: *Collection*, *Collector*, *Owner*, and *Location* (marked with XML comments). There is no *Administrator* resource in this example. Most of the information supplied in the collection resource description is through Dublin Core. In particular, the `<dc:subject>` element shown uses the anonymous resource mechanism mentioned earlier to embed another resource, which is a Library of Congress Subject Heading, expressed using qualified Dublin Core.

About halfway down the collection resource description, the Dublin Core `<dc:type>` element is used to give the collection’s type. The RSLP collection description defines an enumerated list of possible types, starting with broad classifications, such as *Catalog* and *Index*, and then proceeding to more specific items, such as *Library*, *Museum*, and *Archive*, and finally defining even more specific items, such as *Text*, *Image*, and *Sound*. A collection can be given more than one of these types by separating the terms by a period (.), as can be seen in the figure.

Elements `<dc:hacsPart>`, `<dcq:isPartOf>`, and `<cl:hasDescription>` are examples of external relationships. They identify or name other resources that have a bearing on the collection being

```

<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf  ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc   ="http://purl.org/dc/elements/1.1/"
  xmlns:dcq  ="http://purl.org/dc/qualifiers/1.0/"
  xmlns:vcad ="http://www.imc.org/vcard/3.0/"
  xmlns:cld  ="http://www.ukoln.ac.uk/metadata/rslp/1.0/">

  <!-- Collection resource -->
  <rdf:Description about="urn:x-rslpcd:967715792-47835">
    <dc:title>Morrison Collection of Chinese Books</dc:title>
    <dc:description>
      This collection comprises the Chinese books accumulated by Dr.
      Robert Morrison (1782 - 1834), the first Protestant missionary to
      China.
    </dc:description>

    <dc:subject>
      <rdf:Description>
        <dcq:scheme>LCSH</dcq:scheme>
        <rdf:value>Missionaries -- China</rdf:value>
      </rdf:Description>
    </dc:subject>
    <!-- additional subject entries for "Rare Books -- China -- Bibliography
      -- Catalogs" and "Chinese Imprints -- Catalogs" -->

    <cld:agentName>Morrison, Robert, 1782-1834.</cld:agentName>
    <!-- additional entries for cld:agent, dcq:place, dcq:time, cld:strength,
      dc:language, and dc:format -->

    <dc:type>Collection.Library.Special</dc:type>
    <cld:accumulationDateRange>1807-1823</cld:accumulationDateRange>
    <cld:contentsDateRange>1650-1825</cld:contentsDateRange>

    <dcq:hasPart>
      Literature collection within Morrison Collection of Chinese Books
    </dcq:hasPart>
    <!-- additional entries for dcq:hasPart, dcq:isPartOf, cld:hasDescription,
      cld:accrualStatus, cld:accessControl, cld:note -->

    <dc:creator resource="urn:x-rslpcd:967715792-32366"/>
    <cld:owner resource="urn:x-rslpcd:967715792-62789"/>
    <cld:hasLocation resource="urn:x-rslpcd:967715792-16277"/>
  </rdf:Description>

  <!-- Collector resource -->
  <rdf:Description about="urn:x-rslpcd:967715792-32366">
    <vcad:fn>Morrison, Robert, 1782-1834.</vcad:fn>
  </rdf:Description>

  <!-- Owner resource -->
  <rdf:Description about="urn:x-rslpcd:967715792-62789">
    <vcad:org>School of Oriental and African Studies Library</vcad:org>
    <vcad:voice>+44 207 898 4163</vcad:voice>
    <vcad:fax>+44 207 898 4159</vcad:fax>
    <vcad:email>libenquiry@soas.ac.uk</vcad:email>
  </rdf:Description>

  <!-- Location resource -->
  <rdf:Description about="urn:x-rslpcd:967715792-16277">
    <dc:title>School of Oriental and African Studies Library</dc:title>
    <!-- additional entries for cld:address, cld:postcode, cld:country,
      cld:accessConditions, cld:seeAlso -->
    <cld:isLocationOf resource="urn:x-rslpcd:967715792-47835"/>
  </rdf:Description>
</rdf:RDF>

```

Figure 6.8: RSLP description of the Morrison Collection of Chinese Books

described. There are seven external relationships in all. The ones appearing here are used to name subcollections (*hasPart*), the larger library this collection fits into (*isPartOf*), and where a description of the collection appears (*hasDescription*).

The lower part of the collection resource description contains references to the collector, owner, and location resources through *dc:creator*, *cld:owner*, and *cld:hasLocation*, respectively. The first two are examples of agents and make use of *vcard* elements to supply the necessary information. A location can be electronic or physical, and most of its elements are defined by RSLP namespace elements.

Open Archives Initiative Object Reuse and Exchange: OAI-ORE

The Open Archives Initiative Object Reuse and Exchange (OAI-ORE) is a standard designed expressly for representing aggregations of digital objects. An extension of the OAI project (described in Chapter 7), it can be applied when separate (perhaps disparate) objects need to be treated as a group. Grouping objects is a natural human activity: photos are grouped into albums, music tracks are grouped into playlists, documents are grouped into collections.

OAI-ORE introduces a specific resource, called an *Aggregation*, that refers to a set of objects. An aggregation is described by a *Resource Map* that associates metadata with it. For example, [Figure 6.9](#) shows an OAI-ORE document that describes this book. The aggregation contains four items: TIFF images of the front and back covers and PDF documents for the two parts of the book. The first `<rdfDescription>` element is declared to be an OAI-ORE Aggregation by reference to the term `http://www.openarchives.org/ore/terms/Aggregation`. This element also contains Dublin Core metadata *dc:creator* and *dc:title*. The second `<rdfDescription>` element is a resource map: it declares that the RDF file at `http://www.cs.waikato.ac.nz/~ihw/rdf/htbadl2/` is a description (or resource map) of the aggregation. The remainder of the file attaches Dublin Core metadata to the individual items as item-level metadata.

OAI-ORE is a new framework with many potential uses in representing resources and metadata. It can link together different versions of the same document (e.g., PDF, PostScript, and Word). It can express parallel aggregations like a photograph collection containing themed groups (holiday in France, New Year party, graduation ceremony), all defined without affecting the base collection. The resources are URIs, so items can be defined across collections as well, from anywhere on the Internet that has a URI; thus, virtual collections can be built from remote resources. Because aggregations are themselves resources, they can themselves be aggregated into groups of collections, which can then be further aggregated.

The design of OAI-ORE is reminiscent of the METS Structural Map, which illustrates one of the virtues of the digital world: the ability to break apart, recombine, and restructure content in new and creative ways.

Metadata for education: LOM and SCORM

Libraries have long supported a variety of educational activities, and digital libraries for education have driven both content creation and library technology. Educational technologists talk of *Learning*


```

<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ore="http://www.openarchives.org/ore/terms/"
  xmlns:dc="http://purl.org/dc/elements/1.1/">

  <rdf:Description rdf:about="http://www.cs.waikato.ac.nz/~ihw/htbadl2/">
    <rdf:type rdf:resource="http://www.openarchives.org/ore/terms/Aggregation"/>
    <ore:aggregates rdf:resource="http://www.cs.waikato.ac.nz/~ihw/htbadl2/front.tiff"/>
    <ore:aggregates rdf:resource="http://www.cs.waikato.ac.nz/~ihw/htbadl2/part-1.pdf"/>
    <ore:aggregates rdf:resource="http://www.cs.waikato.ac.nz/~ihw/htbadl2/part-2.pdf"/>
    <ore:aggregates rdf:resource="http://www.cs.waikato.ac.nz/~ihw/htbadl2/back.tiff"/>
    <dc:title>How to Build a Digital Library (Second edition)</dc:title>
    <dc:creator>Ian Witten</dc:creator>
    <dc:creator>David Bainbridge</dc:creator>
    <dc:creator>David Nichols</dc:creator>
  </rdf:Description>

  <rdf:Description rdf:about="http://www.cs.waikato.ac.nz/~ihw/rdf/htbadl2/">
    <rdf:type rdf:resource="http://www.openarchives.org/ore/terms/ResourceMap"/>
    <ore:describes rdf:resource="http://www.cs.waikato.ac.nz/~ihw/htbadl2/">
  </rdf:Description>

  <rdf:Description rdf:about="http://www.cs.waikato.ac.nz/~ihw/htbadl2/front.tiff">
    <dc:format>image/tiff</dc:format>
    <dc:title>How to Build a Digital Library: front cover image</dc:title>
  </rdf:Description>

  <rdf:Description rdf:about="http://www.cs.waikato.ac.nz/~ihw/htbadl2/back.tiff">
    <dc:format>image/tiff</dc:format>
    <dc:title>How to Build a Digital Library: back cover image</dc:title>
  </rdf:Description>

  <rdf:Description rdf:about="http://www.cs.waikato.ac.nz/~ihw/htbadl2/part-1.pdf">
    <dc:format>application/pdf</dc:format>
    <dc:language>en</dc:language>
    <dc:title>How to Build a Digital Library (Second edition) Part 1</dc:title>
  </rdf:Description>

  <rdf:Description rdf:about="http://www.cs.waikato.ac.nz/~ihw/htbadl2/part-2.pdf">
    <dc:format>application/pdf</dc:format>
    <dc:language>en</dc:language>
    <dc:title>How to Build a Digital Library (Second edition) Part 2</dc:title>
  </rdf:Description>

</rdf:RDF>

```

Figure 6.9: OAI_ORE description of this book

Objects and have devised a metadata scheme called Learning Object Metadata (LOM) to describe them. According to the IEEE Learning Technology Standards Committee,

the LOM standard will specify the syntax and semantics of Learning Object Metadata, defined as the attributes required to fully/adequately describe a Learning Object. Learning Objects are defined here as any entity, digital or non-digital, which can be used, reused or referenced during technology supported learning.

As you can see, the scope of Learning Objects is very wide. Figure 6.10 shows a sample LOM record; it is expressed in XML, is quite readable, and resembles other XML metadata representations, such as MODS.

The resource that this metadata record describes is titled *A Teddy Bear Picnic*, which is stated in the `<general>` section as the `<title>` tag. The previous section gives its subject classification as *Arts*. The LOM standard allows information to be provided in more than one language if desired. In this case, the overall language of the resource is set to English through the `<language>` tag. Individual `<langstring>` tags could override this by including an attribute specification. Further down the `<general>` section is a description of the resource: *This film is about a teddy bear trying to enjoy his picnic. Until his sandwich runs away ... now he has to get it back!*

The resource that this particular LOM record describes is a film, a fact that is prescribed in machine-readable form in the `<technical>` section, which shows that it is a QuickTime video that is 1 minute 38 seconds long, requiring 2,053,707 bytes (2 MB) of file space. LOM allows a URL to be specified that points to an online version of the resource. The `<lifecycle>`, `<educational>`, and `<rights>` sections of this example present details such as the name of the contributor, Dalvin Chung Trieu; the copyright owner, the Vancouver Film School; and the status of the work, *Finished*.

The LOM standard is designed to cover a variety of situations. It is possible to use a subset of LOM and still remain compliant with the standard. Subsets are examples of *application profiles*. Although LOM itself gives no directive about elements that are compulsory or optional for describing an educational resource, application profiles often stipulate such requirements.

LOM, which in isolation is a relatively straightforward metadata framework, is also an important component of the Sharable Content Object Reference Model (SCORM). This ambitious educational initiative aims to foster the creation of reusable learning content as “instructional objects” within a common technical framework for computer and Web-based learning. SCORM was created as a collaborative effort among government, industry, and academia to establish a new distributed learning environment that permits the interoperability of learning tools and course content on a global scale. In essence, SCORM acts as a wrapper for various components specified using existing standards—one of which is a particular profile of the LOM standard. A properly configured Sharable Content Object (SCO) can be shared and reused because it can be executed by any compliant learning management system (such as Moodle).

A SCORM package can contain many other resources. It contains a manifest file, an inventory of all the individual files that constitute it, along with how they are pulled together to form the learning object. Three key elements are *resources*, which are the building blocks out of which the learning object is formed; *items*, which draw upon the resources and stipulate their organizational structure; and *object metadata*, expressed using the IEEE LOM standard.

Metadata for eResearch

Digital libraries can store any type of digital object and are now being used to record the process and results of research. The aim is not just to store the process of research, but to further the advance-

```

<?xml version="1.0" encoding="utf-8"?>
<lom xmlns="http://linux2.commonsworld.org/rob_text/ims1_2_1"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://linux2.commonsworld.org/rob_text/ims1_2_1.xsd">
  <classification>
    <keyword> Arts </keyword>
    <purpose> Discipline </purpose>
  </classification>
  <general>
    <language> English </language>
    <title> A Teddy Bear Picnic </title>
    <keyword> teddy bear </keyword>
    <!-- ... more keywords specified here ... -->
    <description>
      This film is about a teddy bear trying to enjoy his picnic. Until his
      sandwich runs away ... now he has to get it back!
    </description>
  </general>
  <lifecycle>
    <status> Finished </status>
    <contribute>
      <centity><vcard> Dalvin Chung Trieu </vcard></centity>
      <date>
        <datetime> 2002-02-01 </datetime>
        <description>
          Born and raised in B.C. Dalvin grew up having a knack and interest in
          art since grade 3. He went to study Fine Arts <!-- ... -->
        </description>
      </date>
      <role> Director/Creator </role>
    </contribute>
  </lifecycle>
  <educational>
    <context>
      <value> VFSdept </value>
      <source> 3D Animation/Maya </source>
    </context>
    <language> English </language>
  </educational>
  <rights>
    <description> Copyright 2002 Vancouver Film School </description>
    <copyrightandotherrestrictions> yes </copyrightandotherrestrictions>
  </rights>
  <technical>
    <size> 2053707 </size>
    <duration> 1:38 </duration>
    <location> http://tree.vfs.com:8080/aloha/uploads/rbrayer/DalvinT.mov </location>
    <format> video/quicktime </format>
  </technical>
</lom>

```

Figure 6.10: Sample LOM document

ment of knowledge by ensuring that experiments are reproducible. Traditional academic research articles appear in physical journals in paper form. Many journals now have online versions, and some are entirely electronic. As storage costs fall and network speeds rise, online journals can store supplementary resources as well. These research products include data sets, intermediate results, high-definition images, and computer code. This raises interesting questions about metadata. What are appropriate metadata schemes for describing, say, an archive of tree-ring width measurements?

Along with the ability to publish more of the background context of research papers, researchers seek access to data sets and source code in order to independently replicate or reproduce results.

Reproducibility calls for data sets and software to be made available for 1) verifying published findings, 2) conducting alternative analyses of the same data, 3) eliminating uninformed criticisms that do not stand up to existing data, and 4) expediting the interchange of ideas among investigators. Ultimately, all scientific evidence should be held to the standard of full replication and the confirmation of important findings by independent investigators.

A statement like this implies that the scope of traditional research papers is insufficient, and a panoply of research artifacts need to be preserved as well—along with metadata for searching, browsing, and understanding diverse objects. It is also likely that access to some artifacts will be restricted, making rights metadata critically important.

Researchers who commit to reproducibility as a principle in their own laboratories see positive effects within their groups. Students quickly get up to speed with previous work, and since the process has been archived, it is less critical if researchers forget small details. An alternative perspective is to view the problem as one of the *provenance* of research results. What do we know about the history of data points as they pass through various processing stages?

Tools for recording and reproducing work come together in the form of workflow environments for eScience. For example, Taverna is a software tool for defining and executing workflows. Workflows contain actual executable code, as well as references to external Web services and a sequence of input/output operations. When a workflow is executed on a Taverna platform, the operations and results are reproduced. Soon researchers will need to be able to search, browse, and reuse such workflows in a variety of disciplines and digital libraries will need to support preservation of both the workflows and their products.

6.5 Metadata Quality

Supporting the development of quality metadata is perhaps one of the most important roles for Library and Information Science professionals.

The applications of metadata outlined earlier rely on the accuracy of the information. The quality of the available metadata constrains its usefulness in any digital library. Metadata quality can be considered across these seven dimensions:

- completeness
- accuracy
- provenance
- conformance to expectations
- logical consistency and coherence
- timeliness
- accessibility.

Table 6.7: Spelling variants of the name *Muammar Qaddafi*

Qaddafi, Muammar	Muammar al-Qadhafi	Qathafi, Muammar
Gadhafi, Mo ammar	Mu ammar al-Qadhdhafi	Gheddafi, Muammar
Kaddafi, Muammar	Qadafi, Mu ammar	Muammar Gaddafy
Qadhafi, Muammar	El Kazzafi, Moamer	Muammar Ghadafi
El Kadhafi, Moammar	Gaddafi, Moamar	Muammar Ghaddafi
Kadhafi, Moammar	Al Qathafi, Mu ammar	Muammar Al-Kaddafi
Moammar Kadhafi	Al Qathafi, Muammar	Muammar Qathafi
Gadafi, Muammar	Qadhdhafi, Mu ammar	Muammar Gheddafi
Mu ammar al-Qadafi	Kaddafi, Muammar	Khadafy, Moammar
Moamer El Kazzafi	Muammar al-Khaddafi	Qudhafi, Moammar
Moamar al-Gaddafi	Muamar al-Kad'afi	Qathafi, Mu'Ammar
Mu ammar Al Qathafi	Kad'afi, Mu amar al-	El Qathafi, Mu'Ammar
Muammar Al Qathafi	Gadaffy, Muammar	Kadaffi, Momar
Mo ammar el-Gadhafi	Gadafi, Muammar	Ed Gaddafi, Moamar
Muammar Kaddafi	Gaddafi, Muammar	Moamar el Gaddafi
Moamar El Kadhafi	Kaddafi, Muamar	

Consider the accuracy of metadata for personal names. [Table 6.7](#) shows variations in the name of the Libyan leader *Mummar Qaddafi* that appear in official documents. Some derive from the transliteration from Arabic into English, but they all refer to the same person. How is the software supposed to know about these connections? Librarians faced this problem many years ago, and the answer they devised is known as *authority control*.

Authority control: Names

Authority control can apply to any set of values, but in practice it is most commonly used for personal names. The Library of Congress maintains millions of authority records in a Name Authority File as part of the MARC standard. Each record contains three sections:

- an authorized version of the name
- a list of variant versions (including other names or aliases)
- documentation and notes to provide the rationale for the authorized version.

[Figure 6.11](#) shows an excerpt from the name authority record for *Mummar Qaddafi*. The structure follows the MARC format of numbered fields: field 100 contains the authorized version, and 400 fields, many of which have been omitted from the figure, show the numerous variants. The 670 fields contain notes that help explain the authorized/variant split. Libraries can subscribe to the Name Authority File and receive updates as names are added and revised.

Name authority data can be used in diverse ways. For example, in a browsing structure of author names, all variants may be collapsed into the authorized version. Or search terms can be expanded. For example, a search for *Mark Twain* can be expanded to include his other pseudonyms *Quintus*

The Library of Congress >> Go to Library of Congress Online Catalog



LIBRARY OF CONGRESS AUTHORITIES



[Help](#) | [New Search](#) | [Search History](#) | [Headings List](#) | [Start Over](#)

[< Previous](#) [Next >](#)

[MARC Display](#) [Labelled Display](#)

LC Control Number: n 81068638

HEADING: Qaddafi, Muammar

000 04366cz a2201117n 450

001 60188

005 20081014054259.0

008 810824n| acannaabn |b aaa

010 __ |a n 81068638

035 __ |a (OCoLC)oca00613985

040 __ |a DLC |b eng |c DLC |d DLC |d InU |d DLC |d MIU |d NJP |d DLC |d OCoLC |d DLC |d UPB |d DLC |d VIU |d OCoLC

053 _0 |a PJ7858.A2595

100 1_ |a Qaddafi, Muammar

400 1_ |a Qadhafi, Mo'amar

400 1_ |a Kaddafi, Muammar

400 1_ |a Qadhafi, Muammar

400 1_ |a El Kadhafi, Moammar

400 1_ |a Kadhafi, Moammar

400 0_ |a Moammar Kadhafi

400 1_ |a Gadafi, Muammar

400 0_ |a Mu'amar al-Qaddafi

400 1_ |a قذافي، معمر

400 1_ |a قذافي، مؤامر

400 1_ |a قذافي، محمد

400 1_ |a معمر القذافي

667 __ |a Machine-derived non-Latin script reference project.

667 __ |a Non-Latin script references not evaluated.

670 __ |a His Evacuation day of American troops ... 1971.

670 __ |a Calderón, H. Khaddafi, la Operación "Jerusalén," 1981: |b t.p. (Khaddafi) p. 4 of cover (Muammar Khaddafi)

670 __ |a Enc. Brit., 1977 |b (Mu'amar al-Qaddafi)

670 __ |a Gera, G. Derekh Kad'afi be-Luv, 1983: |b front matter (Mu'amar al-Qad'afi)

670 __ |a Comm. for the Defence of Human Rights & Polit. Prisoners in Libya. Selected documents, 1981: |b p. 1, etc. (Muammar Ghaddafi; Muammar Ghadafi; Muammar Ghaddafi; Muamar Al-Kaddafi; Muammar Quathafi; Muammar Ghaddafi)

670 __ |a His 'Arā' jadidah fi al-sūq wa-al-ta'biyah wa-mabādī' al-ḥarb, 1981: |b t.p. (Mulāzīm Awwal Mu'amar Muḥammad Abū Minyār al-Qadhāfi)

670 __ |a Swisher, R.K. Kill Kadaffi, 1986: |b p. 4 of cover (Colonel Mornar Kadaffi)

670 __ |a Muscat, F. September one, c1981: |b p. v (mu'Ammar el Qadhafi)

670 __ |a Charvin, R. Le syndrome Kadhafi, 1988, c1987: |b p. 7 (Moammar Kadhafi)

670 __ |a Brill, H. Libyens Aussen- und Sicherheitspolitik, 1988: |b t.p. (Moamar el Gaddafis ...)

670 __ |a His Opinions on military command, 19--: |b t.p. (Moammar Kazzafi)

670 __ |a Muḥammad Gheddafi et la nuova Libia, 2001.

670 __ |a His Green book, 1980?--: |b t.p. (Muammar al-Qadhafi)

670 __ |a International Symposium on the Thought of Muammar Al-Gathafi "The Green Book" (1st : 1983 : Banghāzi, Libya). al-Ta'lim, 1984.

670 __ |a Dans le concert des nations, c2004: |b t.p. (Mouammar Kadhafi) p. 4 of cover (b. 1942)

670 __ |a The Green book, 2005: |b t.p. (M. Al Gathafi)

675 __ |a Collier's, 1973; |a Americana, 1975.

952 __ |a RETRO

953 __ |a xx00 |b lf07

Figure 6.11: Part of the MARC name authority record for *Qaddafi, Muammar*

```

<?xml version="1.0" encoding="UTF-8"?>
<mads xmlns="http://www.loc.gov/mads/"
      xmlns:mods="http://www.loc.gov/mods/v3"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.loc.gov/mads/ mads.xsd">
  <authority>
    <name>
      <namePart>Smith, John</namePart>
      <namePart type="date">1995-</namePart>
    </name>
  </authority>
  <variant type="other">
    <name>
      <namePart>Smith, J</namePart>
    </name>
  </variant>
  <variant type="other">
    <name>
      <namePart>Smith, John J</namePart>
    </name>
  </variant>
  <note type="history">Biographical note about John Smith.</note>
  <affiliation><organization>Lawrence Livermore Laboratory</organization>
    <dateValid>1987</dateValid>
  </affiliation>
</mads>

```

Figure 6.12: MADS record

Curtius Snodgrass and *Louis de Conte*, as well as his actual name, *Samuel Langhorne Clemens*. Such alternatives are recorded in the authority record's 500 field, whereas variants of the authorized version are in the 400 field. A system could expand a user's search automatically, or present the user with interactive expansion options. Although authority files are designed for metadata values, they could be used for full-text search or other applications.

Earlier we described MODS, an XML-based format for metadata that performs an analogous role to MARC. Accompanying it is an XML-based authority format, the Metadata Authority Description Schema (MADS). Figure 6.12 shows a sample MADS record with one authorized name (*Smith, John*) and two variant versions (*Smith, J* and *Smith, John J*).

Through the relationships expressed in authority records, digital libraries can ensure a one-to-one relationship between people and conceptual items in the interface. They can also inform users about relevant relationships to other authors (as with Mark Twain). The alternative makes for an unpleasant user experience: authors that seem different but are in fact the same person, and, as a result of poor navigational support, browsing by author yields a messy list of uncontrolled names, which confuses users and fails to inspire confidence in the service.

The naïve user of an institutional repository, for example, will swiftly find that the absence of name authority control inhibits retrieval of items by a single author. Should a user arrive at a specific item and desire to see more items by the same author, clicking on the author's name will lead only to results for that particular name spelling or variant.

In these respects, authority control contributes to one of the seven quality dimensions, conformance to users' expectations. Most digital library systems do a poor job, resulting in considerable manual work:

The institutional repository manager hoping for proper name-authority control currently has little choice but to clean up records by hand after the fact.

Rectifying this deficiency should be a priority for digital library software developers, so that collection managers can manage and display their information appropriately.

Authority control: Subjects

A widely used form of authority control for subjects is the Library of Congress Subject Headings (LCSH), a comprehensive controlled vocabulary for assigning subject descriptors. The print versions occupy five large volumes, amounting to about 6,000 pages each, commonly referred to by librarians as “the big red books.” Users generally encounter the subject headings in online library catalogs, and librarians encounter them in the system they use to manage and maintain the catalog. Once headings have been assigned to items in the library, they can be used for subject-based searching and browsing and to help inform users of common terms used to classify items in the collection.

The aim of the LCSH is to provide a standardized vocabulary for all categories of knowledge, descending to quite a specific level, so that books—on any subject, in any language—can be described in a way that allows all books on a given subject to be easily retrieved. There are a total of around two million different Subject Headings. Perhaps 60 percent of them are full entries like the one for *Agricultural machinery* in the first row of [Table 6.8](#). This entry indicates that *Agricultural machinery* is a preferred term, and should be used instead of the three terms *Agriculture—Equipment and supplies*, *Crops—Machinery*, and *Farm machinery*. *UF* stands for “use for.” Each of the three deprecated terms has its own one-line entry that indicates (with a *USE* link) that *Agricultural machinery* should be used instead; deprecated terms account for the remaining 40 percent of entries in the red books. The *UF/USE* links, which are inverses, capture the relationship of *equivalence* between terms. One of each group of equivalent terms is singled out as the preferred one, not because it is intrinsically special, but purely as a matter of convention.

Another relationship captured by the subject headings is the *hierarchical* relationship of broader and narrower topics, which are expressed by *BT* (broader topic) and *NT* (narrower topic), respectively (also inverses). The *Agricultural machinery* example of [Table 6.8](#) stands between the broader topic *Machinery* and narrower topics, such as *Agricultural implements* and *Agricultural instruments*—there are many more not shown in the table. Each of these narrower topics will have links back to the broader topic *Agricultural machinery*; and *Agricultural machinery* will appear in the (long) list of specializations under the term *Machinery*.

The abbreviation *RT* stands for “related topics” and gives an *associative* relationship between a group of topics that are associated but neither equivalent nor hierarchically related. The fact that *Farm equipment* is listed as a related topic under *Agricultural machinery* indicates that the converse is also true: *Agricultural machinery* will be listed as a related topic under *Farm equipment*.

Table 6.8: Library of Congress Subject Heading entries

Agricultural machinery	
UF	Agriculture—Equipment and supplies
	Crops—Machinery
	Farm machinery
BT	Machinery
RT	Farm equipment
	Farm mechanization
	Machine-tractor stations
SA	subdivision Machinery under individual crops, e.g. Corn—Machinery
NT	Agricultural implements
	Agricultural instruments
	...
Agriculture—Equipment and supplies	
USE	Agricultural machinery
Crops—Machinery	
USE	Agricultural machinery
Farm machinery	
USE	Agricultural machinery
Machinery	
UF	...
BT	...
RT	...
SA	...
NT	...
	Agricultural machinery
	...
Agricultural implements	
UF	...
BT	Agricultural machinery
	...
Farm equipment	
UF	...
BT	...
RT	Agricultural machinery
	...
	...

Finally, the SA (“see also”) entry indicates a whole group of subject headings, often specified by example—such as *Corn—Machinery* under *Agricultural machinery* in Table 6.8. The dash in this example indicates that there is a subheading *Machinery* under the main entry for *Corn*. However, there is no back reference from this entry to *Agricultural machinery*.

The Library of Congress maintains a Subject Authority File, which is similar to the Name Authority File. Other organizations also maintain parallel subject authorities, such as the National Library of Medicine’s Medical Subject Headings (MeSH).

Unfortunately, the current state of authority control implementation in digital library software systems is well behind that of traditional library catalogs. When choosing digital library software, you should consider how you intend to address issues of consistency, because you may find that there is little system support available.

Controlling metadata values

Name authority control and subject authority control improve the user experience and simplify the task of maintaining complex information structures. Before we examine other ways of controlling metadata values, we first consider some of the errors that can creep into metadata.

We are all familiar with typographical errors in text we read and write; of course, similar errors also occur in any human-assigned metadata. Common errors include:

- omissions: *Managment* instead of *Management*
- insertions: *Economics* instead of *Economics*
- transpositions: *Phsyics* instead of *Physics*
- substitutions: *Sosiology* instead of *Sociology*

Typographical errors abound whenever text is created by humans, and they occur in document text as well as metadata. In the main text, they reduce the accuracy of full-text searching, while in metadata they affect all kinds of metadata-enabled facilities. Errors also arise when metadata is transferred between different schemes and when it is migrated between different software systems. They also arise from automatic metadata extraction processes, which are often relatively noisy.

The controlled vocabularies of authority control are one way to address these issues. For example, when Library of Congress Subject Headings are used for subject metadata, we can expect our software system to flag incorrect entries (such as *Phsyics*). Even better, it should prevent the entry of incorrect terms in the first place. The best place to stop errors from propagating is at the time of creation. Library catalog systems that implement the MARC/AACR2 standard contain numerous validation checks on the tags and subfields that prevent many forms of entry errors. The encoding schemes of qualified Dublin Core function in the same way—if implemented in the software.

Other common forms of metadata control include requirements that certain elements be present (for example, every item must have a title) or that the values of elements are constrained in appropriate ways. Such constraints include:

- Adopting the ISO 8601 international standard for date and time representation, which specifies patterns of date expression (for example, 1997-07-16T19:20+01:00 and 1997-07-16 are legal values, but 97-07-16 is not, because years must be expressed in four digits).
- Using similar pattern restrictions for URLs, MIME types (e.g., *image/tiff*, *video/quicktime*), and digital object identifiers (see Section 7.3).
- Choosing values from restricted sets of externally defined possibilities, such as language codes (e.g., *en-us*, *fr*, *es*) and Library of Congress Subject Headings.

- Choosing values from a restricted set of locally defined possibilities (for example, ensuring that each item in an institutional repository is associated with a Department or Faculty identifier chosen from an approved list).

In practice, the ability to express constraints in your digital library metadata depends on the underlying software support. As with authority control, current digital libraries lag behind library catalog systems. This has several causes, including limited software functionality and inadequate resourcing—but fundamentally it reflects the fact that the digital library community has not yet evolved the same consortial approaches to metadata control.

Metadata tools

Managers of digital libraries should monitor the quality and characteristics of their metadata as their collections grow. However, their ability to assess the metadata is limited by the tools available: it is not feasible to hunt down metadata errors in large repositories manually. Although some of the quality dimensions listed earlier cannot be evaluated mechanically, others are easy to assess.

Completeness measures whether the value of a particular metadata element is defined across the entire collection. The requirement that all items should have a title is tantamount to saying that the completeness of the title element should be 100 percent. Analysis tools can report the completeness of each element to the collection manager and relate the results to any applicable constraints.

Figure 6.13 illustrates a tool that visualizes a collection's metadata for a subset of the Dublin Core elements by showing all the records in a scrolling table. Each column represents a metadata element and each row a document. The presence of a value is indicated by a filled rectangle; white areas indicate undefined items. The completeness percentage for each element appears at the bottom of the column.

Errors should be corrected as soon as they are detected. However, if they derive from automated processes, such as converting metadata from one scheme to another or extracting it from documents, they may occur on a large scale, and manual fixes are likely to be inordinately time-consuming. What is needed is the ability to change many values at once in a precise manner; in other words, batch editing of metadata—also known as *metadata cleaning*. This can occur at any point in the workflow of a digital library. The edits or cleaning operations may be expressed in a pattern language (such as regular expressions) or a programming language. Again, most current digital library systems do not readily support such operations.

6.6 Extracting Metadata

We now turn to the business of extracting metadata automatically from a document's contents. Automatic extraction of information from text—*text mining*, as it is often called—is a hot research topic. The ready availability of huge amounts of textual information on the Web has placed a high premium on automatic extraction techniques. In this area there is hardly any underlying theory, and existing methods use heuristics that are complex, detailed, and difficult to replicate and evaluate.

Structured markup languages like XML help make key aspects of documents accessible to computers and people alike. They encode certain kinds of information explicitly in such a way that it can be extracted easily by parsing the document structure. Of course, except for the simplest of documents, this information falls far short of that conveyed by a complete and comprehensive understanding of the text.

Relatively few documents today contain explicitly encoded metadata. The balance will shift as authors recognize the added value of metadata, standards for its encoding become widespread, and improved interfaces reduce the mechanical effort required to supply it. However, although their role may diminish, schemes for extracting metadata from raw text will never be completely replaced by explicit provision of metadata.

Fortunately, it is often unnecessary to understand a document in order to extract useful metadata from it. In the following discussion we give several examples that indicate the breadth of what can be done, although we do not describe the techniques in full detail because they usually require considerable tuning to the problem at hand. The first three sections, extracting document metadata, generic entities, and bibliography entries, describe useful general techniques. The last four sections, language identification, extracting acronyms and key phrases, and generating phrase hierarchies, are facilities that are included in the Greenstone software described in Part II. This material pushes beyond the boundaries of what is conventionally meant by “metadata”—our focus is on extracting information that is generally of use in digital libraries rather than on any narrow interpretation of the term.

Extracting document metadata

Basic metadata about a document—its title, author, publisher, date of publication, keywords, and abstract—is often present on the first page for all to see. Moreover, it is frequently presented fairly uniformly: the title first, centered, followed by some white space; then the authors’ names and affiliations, also centered; followed by the publication date; keywords preceded by the word *Keywords*; and an abstract, preceded by the word *Abstract* or *Summary*. Document families bear family resemblances. Different type sizes or typefaces may be used for different elements of the title page.

Such structure is easy to spot for a well-defined and tightly controlled family of documents. However, doing the same in a general way is not so easy. Furthermore, practical document collections often contain exceptions that human readers do not even notice as anomalies, but that confound automated extraction. Giving a full account of the techniques used for generic entity extraction is beyond the scope of this book. Instead we give some samplers. Some of the techniques described in the next subsection may be applicable in particular situations.

Generic entity extraction

Some information is easy to extract from plain text documents because it is expressed in a fixed syntax that is easy to recognize automatically. E-mail addresses and URLs are good examples, but, of course, they are both products of the Internet era: they are explicitly designed for automatic recognition and processing.

Other artificial entities are also readily recognized, although less reliably. Sums of money, times of day, and dates are good examples. But even these entities have variants. Dates can be expressed in several different ways that sometimes cause ambiguity, such as the date format 9/9/99, and time can be expressed in both the 12-hour (a.m. and p.m.) system and the 24-hour system (“1700 hours”). Some differences are cultural—for example, when sums of money are written in decimal currency, Americans and Europeans use the comma and period in opposite ways.

Names of people, places, and companies are an important kind of semistructured data, and it is often useful to identify them as metadata. Names can be recognized partly by intrinsic and partly by extrinsic properties. They almost always begin with capital letters. (But not always, as in the class of archaic English surnames, such as ffoulkes, the avant-garde poet e e cummings, and the contemporary singer k d lang. Also, capitalization as a name identifier loses much of its practical value in languages like German that capitalize all nouns.) However, in Western documents the statistical patterns in which letters appear differ between names and ordinary language in general, a trend that is accentuated by globalization, with the increasing incidence of foreign names. People’s names are commonly preceded by forenames and may contain initials. There are numerous honorific prefixes, such as Mr., Ms., Dr., and Prof. Names may also include baronial prefixes, such as *von*, *van*, or *de*, and other miscellaneous qualifiers, such as *Jr.* or *Sr.*

Extrinsic properties constrain the contexts in which names occur in text. People’s names are often recognizable because they are introduced by phrases like “according to ...” or “... said.” Similar stock phrases characterize company names and place names. Indeed, sometimes entity names can be distinguished from people’s names only by the surrounding words. When a string like *Norman* occurs, context provides the only clue as to whether it is a place (Norman, Oklahoma), a person (such as Don Norman), or a race (the Normans). (Notice that this last sentence itself uses context to make the referents clear!)

Subsequent references to an already mentioned entity provide a further dimension of richness. Once a document has mentioned a person, place, or company, subsequent references may be abbreviated. Here, for example, it should be clear who cummings or lang is.

The task of identifying entities like times, dates, sums of money, and names in running text is called *generic entity extraction*. Techniques can be broadly classed into three categories: those that use pre-programmed heuristics; those that can accommodate different textual conventions using manually tagged training data; and those that can self-adapt using untagged data.

When heuristics are preprogrammed, human intelligence is used to refine the extraction scheme manually to account for the vagaries of natural language usage. However, such heuristic entity extraction systems are never finished, and modifying such systems manually to account for newly discovered problems can be daunting—such systems quickly become practically unmanageable.

Some adaptive systems use training data in which the entities in question have been tagged manually. These systems embody a predefined generic structure that can adapt to the kinds of textual patterns that are actually encountered. Adaptation consists of adjusting parameters in accordance with pre-tagged training documents. This gives an easy way of catering to newly discovered variants: add some appropriate examples, manually tagged, to the training data.

Tagging training data is a boring, laborious, and error-prone task—particularly since large amounts of training data are often needed for such systems to work adequately. Therefore, some current research focuses on *self-adaptive* techniques that, once primed, can work autonomously through large volumes of untagged documents to improve their performance. This approach promises good performance with a minimum of manual effort. However, its inherent limitations are not yet known.

Bibliographic references

Many documents contain bibliographic references, which constitute an important kind of metadata—although now we are beginning to stretch the conventional meaning of the term—that is extremely useful both for characterizing the topic of the article and for linking it to related articles. Traditional citation indexes identify the citations that a document makes and link them with the cited works. An advantage of citation indexes is the capacity for navigation forward, by listing articles that cite the current one, as well as backward, through the list of previously cited articles. Scholars find citation indexes useful for many purposes, including locating related literature, placing given articles in context, evaluating their scientific influence, and analyzing general research trends to identify emerging areas.

It is not hard to automatically locate lists of references in the plain text of a document with a reasonable degree of accuracy. Then each reference is parsed individually to extract its title, author, year of publication, page numbers, and so on, and the tag that is used to cite it in the body of the document (e.g., [1]). The special structure of references makes this easier than the general problem of generic entity extraction. Fields with relatively little variation in syntax and position should be identified first; then the relative position of not-yet-identified fields can be exploited to predict where they occur (if they are present at all). For example, author information often precedes the title, and publisher information often comes after it. Databases of author's names, journal names, and so on can be used to help identify the fields.

The power of a citation index depends on the ability to identify the article that is being referenced and to recognize different references to the same article. Thus, references must be normalized and heuristics must be used to identify when they refer to the same article.

Language identification

Two important pieces of metadata that can be readily and reliably derived from a document's content are the language in which it is written and the character encoding scheme used. Assigning such values is an example of text categorization, in which an incoming document is assigned to some pre-existing category. Because category boundaries are almost never clear-cut, it is necessary to be able to recognize when a given document does not match any category, or when it falls between two categories. Also, to be useful, the process must be robust to spelling and grammatical errors in text, and to character recognition errors in OCR'd documents.

A standard technique for text categorization computes a profile that consists of the “*n*-grams,” or sequences of *n* consecutive letters, that appear in each document. From a training set containing several documents in each language or encoding scheme, a profile is obtained for each category.

Then, given an unknown document, a distance measure between that document's profile and each category profile is calculated. The category whose profile is closest is selected—if none is close enough, no category is selected.

It is sufficient for successful language identification to consider just the individual words that make up the documents—the effects of word sequences can be neglected. Documents are preprocessed by splitting the text into separate word tokens consisting only of letters and apostrophes (the usage of digits and punctuation is not especially language-dependent). The tokens are padded with a sufficient number of spaces, and then all possible n -grams of length 1 to 5 are generated for each word in the document. These n -grams are counted and sorted into frequency order to yield the document profile.

In document profiles the most frequent 300 or so n -grams are highly correlated with the language. The highest-ranking n -grams are mostly unigrams consisting of one character only, and simply reflect the distribution of the letters of the alphabet in the document's language. Starting around rank 300 or so, the frequency profile begins to be more specific to the topic of the document.

A simple metric for comparing a document profile to a category profile is to calculate, for each document n -gram, the difference between its positions in the two profiles—how far “out of place” it is. Document n -grams that do not appear in the category profile are given some maximum value. The total of these “out of place” figures gives a measure of the overall difference. An unknown document is assigned to the category with the closest profile.

A small experiment demonstrates the accuracy of this method. About 3,500 articles in Internet news-groups were used, written in eight different languages—Dutch, English, French, German, Italian, Polish, Portuguese, and Spanish. An independent training set contained a modest sample of each category, ranging from 20 to 120 KB in length. Only the most frequent 400 n -grams in each category profile were used. The system misclassified only seven articles, yielding an overall accuracy of 99.8 percent, an accurate way, therefore, of automatically assigning language metadata to documents.

Acronym extraction

Technical, commercial, and political documents make extensive use of acronyms. With a list of acronyms and their definitions, users can click acronyms to see their expansion and check whether acronyms are being used consistently in the collection. The dictionary definition of *acronym* is

a word formed from the first (or first few) letters of a series of words, such as SCUBA for Self-Contained Underwater Breathing Apparatus.

Acronyms are often defined at their first use by a preceding or following textual explanation, as in this example. Finding them, along with their definitions, in technical documents is a problem that can be tackled using heuristics. The information desired—acronyms and their definitions—is relational, which distinguishes their extraction from many other information extraction problems.

One way of identifying acronyms is to encode potential acronyms with respect to the initial letters of neighboring words. With a well-chosen coding scheme, this should achieve a more efficient representation that can be detected by measuring the compression achieved. The criterion is whether a candidate acronym can be coded more efficiently using a special model than it is using a regular text

compression scheme. A phrase is declared to be an acronym definition if the discrepancy between the number of bits required to code it using a general-purpose compressor and to code it using the acronym model exceeds a certain threshold.

The first step is to identify candidates for acronyms—for example, all words that are expressed in uppercase only. The definition may precede or follow the acronym itself and invariably occurs within a window of a fixed number of preceding or following words—say 16 words.

Candidate acronyms are expressed in terms of the leading letters of the words on either side; we omit the technical details. Then all legal encodings for each acronym are compared and the one that compresses best is selected, and this is then compared with the version compressed using a generic text model. Incidentally, it is a curious fact that, using a standard text model, longer acronyms tend to compress into fewer bits than do shorter ones. The reason is that, whereas short acronyms are often spelled out, long ones tend to be pronounced as words. This affects the choice of letters: longer acronyms more closely resemble “natural” words.

Needless to say, acronym extraction is not entirely perfect. However, experiments on a sizable sample of technical reports show that the scheme we describe here performs well and provides a viable basis for extracting acronyms and their definitions from plain text.

Keyphrase metadata

In the scientific and technical literature, keywords and phrases are often attached to documents to provide a brief synopsis of what they are about. (Henceforth we use the term *key phrase* and regard keywords as one-word key phrases.) They condense documents into a few pithy phrases that can be interpreted individually and in isolation. Their brevity and precision make them useful in a variety of information-retrieval tasks: as document surrogates, for search indexes, as a means of browsing an information collection, and as a document clustering technique. Key phrases can be used to help users get a feel for the content of an information collection, to provide sensible entry points into it, to show how queries can sensibly be extended, to support novel browsing techniques with appropriate linkage structures, and to facilitate document skimming by emphasizing important phrases visually. They also provide a way of measuring similarity among documents.

The key phrases that accompany articles are chosen manually. In academia, authors assign key phrases to documents they have written. Also, librarians often select phrases from a controlled vocabulary that is predefined for the domain at hand (e.g., the Library of Congress Subject Headings; see [Section 6.5](#)). However, the great majority of documents, particularly those on the Web, don’t have key phrases, and assigning them manually is a laborious process that requires careful study of the document and a thorough knowledge of the subject matter.

Surprisingly, perhaps, keyphrase metadata can be obtained automatically from documents with a considerable degree of success. There are two fundamentally different approaches: keyphrase *indexing*, where a controlled vocabulary is used, and keyphrase *extraction*, where key phrases are identified without a controlled vocabulary. Both approaches perform best when a training algorithm is defined that uses a set of documents to which key phrases have already been attached manually.

Key-phrase extraction

Extracting key phrases involves first identifying candidate phrases that occur in the document and then using heuristics to select those that characterize it best. Most key phrases are noun phrases, and syntactic parsing can identify the candidates. The features used for selection range from simple ones, such as the position of the phrase's first occurrence in the document, to more complex ones, such as its occurrence frequency in the document versus its occurrence frequency in a corpus of other documents in the subject area. The training set is used to tune the parameters that balance these different factors. This process uses methods of machine learning, and it would take us too far off track to describe them here.

How accurate are such methods? Suppose a well-tuned extraction algorithm is used to select half a dozen key phrases for a particular document, and these key phrases are compared with phrases selected manually by the document's author. Generally speaking, one might expect one or two of the key phrases to match exactly or almost exactly, one or two others to be plausible phrases that the document author happened not to choose, and the remaining two or three to be less satisfactory key phrases.

As an example, [Table 6.9](#) shows sample output for three research papers in computer science. For each paper the title is shown, along with two sets of key phrases: one set assigned by its author, the other assigned by an automatic procedure. The phrases are aligned to make them easier to compare. It is not hard to guess that the ones on the left are the author's. Although many of the automatically extracted key phrases on the right are plausible, some are strange. Examples are *gauge* and *smooth* for the second paper; *smooth* is not even a noun phrase. The key phrase *garbage* for the third paper is a complete giveaway—while that word may be used repeatedly in a computer science paper, and even displayed prominently in the title, no author is likely to choose it as a keyword for a paper! Although automatically extracted key-phrase metadata may not reflect exactly what the author might have chosen, it is useful for many purposes.

Key-phrase indexing

Key-phrase indexing depends on a controlled vocabulary or thesaurus that contains legitimate index terms. Thesauri include relations between terms, such as those in the Library of Congress Subject Headings: *USE* and *UF* (use for) for synonyms; *BT* and *NT* for broader and narrower terms; and *RT* for related terms. These relational links can provide valuable assistance to the algorithm in choosing the most appropriate index terms for documents.

The indexing procedure begins by identifying candidate phrases in the document, just like key-phrase extraction. However, in indexing, the identified phrases are then sought in the controlled vocabulary, and ones that cannot be found are discarded. First, though, they are normalized by removing stop-words such as *the* and *a*, stemming all words to their grammatical roots, and disregarding the order of words in the comparison with phrases in the controlled vocabulary. (Word order is rarely significant in actual key phrases.) Thus, phrases like *an efficient algorithm* and even *these algorithms are very efficient* would be matched to the index phrase *algorithm efficiency*. So, of course, would *these algorithms are not very efficient*—but, when you think about what index terms are used for, that is perfectly appropriate.

Table 6.9: Titles and key phrases (author- and machine-assigned) for three papers**Protocols for secure, atomic transaction execution in electronic commerce**

anonymity	
atomicity	Atomicity
auction	Auction
	customer
electronic commerce	electronic commerce
	intruder
	merchant
privacy	
	protocol
real-time	
security	security
	third party
transaction	transaction

Neural multigrid for gauge theories and other disordered systems

disordered systems	disordered
	gauge
gauge fields	gauge fields
	interpolation kernels
	length scale
	multigrid
multigrid	
neural multigrid	
neural networks	

Proof nets, garbage, and computations

	smooth
cut-elimination	cut
	cut elimination
	garbage
linear logic	
proof nets	proof net
typed lambda-calculus	
	Weakening

Terms are also conflated by replacing synonymous alternatives (as indicated by USE/UF links) with the preferred form. This greatly extends the approach of conflation based on word-stem matching and also (unlike key-phrase extraction) allows terms that do not actually appear in a document to be assigned to it. The result is a much more reliable set of candidate terms than for key-phrase extraction.

As before, the next step is to use heuristics to select the candidates that characterize the documents best. The same features mentioned above—such as position of first occurrence, and relative frequency—can be used. Moreover, the thesaurus permits the addition of new features, such as the frequency of individual terms and their degree of connectedness to other candidate terms for the

document. Again, a training set is used to tune the parameters that balance the different factors using methods of machine learning.

Results of key-phrase indexing depend on the documents and the thesaurus, but they are often surprisingly good. In [Table 6.9](#), for example, rogue words like *smooth* would be eliminated because they are absent from the controlled vocabulary; more specific terms like *gauge fields* would be preferred to general ones like *gauge* because of their greater connectivity with other terms in the document. The only key phrases that can be assigned are ones that occur in the thesaurus, ensuring that all key phrases are well formed, but at the same time barring novel topics from consideration. In contrast, key-phrase extraction is open-ended: it selects phrases from the document text itself.

Experiments have shown that key-phrase indexing techniques rival the work of professional indexers in terms of consistency if a focused domain-specific thesaurus is employed. Wikipedia article names can serve as a large and extremely general thesaurus for index terms and inter-relationships among the terms can be inferred from hyperlinks and other information in Wikipedia itself. The size of the resultant “Wikipedia thesaurus” has several million terms and exceeds that of the Library of Congress Subject Headings. While admittedly Wikipedia terminology is nowhere near so carefully controlled, proponents point out that it reflects contemporary usage of living language, which real users are likely to employ when searching and browsing. Again, experiments have shown that automatic techniques can outperform subject specialists, in terms of consistency, when assigning relevant Wikipedia articles to documents.

6.7 Notes and Sources

The quotation at the beginning of this chapter is from Gilliland-Swetland (1998), from which the five-part categorization of different functions of metadata in [Section 6.1](#) is also taken. The quotation at the beginning of [Section 6.1](#) is from Weinberger (2007, p. 222).

Chapter 1 of Intner et al. (2006) provides a good overview of different metadata categorizations. Lagoze and Payette (2000) give an interesting and enlightening discussion of metadata. They point out that the term is meaningful only in a context that makes clear what the data itself is, and they discuss the different kinds of metadata identified at the start of this chapter.

The quotation about MARC near the beginning of [Section 6.2](#) is from Avram’s obituary (Australian Library and Information Association, 2006). Details of the MARC record structure are presented in books on library automation—for example, Cooper (1996). The AACR2R cataloging rules are published by the American Library Association (Gorman and Winkler, 1988). Svenonius (2000) gives a readable account of them, and she quoted the extract about naming a local church. The information about the size of the WorldCat union catalog is from the Online Computer Library Center Web site. Information about MARCXML can be found at www.loc.gov/standards/marcxml.

The Dublin Core metadata initiative is described by Weibel (1999). Thiele (1998) gives a review of related topics. More recent information, including qualified Dublin Core and current developments, is available on the official Dublin Core Web site, <http://dublincore.org> (also <http://purl.org/dc>).

BibTeX, part of the TeX system invented by Knuth (1986), is covered by Lamport (1994). The EndNote bibliographic system, a commercial reference database and bibliographic software program, can be obtained from www.endnote.com. A description of Refer, on which the EndNote format is based, is included in the Unix manual pages.

The TIFF specification is provided online by Adobe and can be found at <http://partners.adobe.com/asn/developer/technotes/prepress.html>. The EXIF format is published by the Japan Electronics and Information Technology Industries Association (JEITA); a useful unofficial site is <http://exif.org>. The XMP home page is at www.adobe.com/products/xmp; IPTC is at www.iptc.org. The MIX fragment in Figure 6.3 is from the Library of Congress, who maintain the standard at www.loc.gov/standards/mix.

MP3 ID3 tags are described at www.id3.org. Vorbis comments are described at www.xiph.org/vorbis/doc/. The FLAC format is covered at <http://flac.sourceforge.net>. Apple's *iTunes* metadata appears not to be publicly documented but various Web sites present the results of reverse engineering the files—with all the accompanying caveats about accuracy. Wells et al. (2006) cover MXF in depth.

Comprehensive information about MPEG-7 and MPEG-21 is available from the MPEG home page, www.chiariglione.org/mpeg. A readable early account of the MPEG-7 standard is given by Nack and Lindsay (1999a, 1999b). Wang (2004) gives a brief overview of the MPEG-21 rights expression language, while Bekaert et al. (2003) give an example of using it in a digital library context.

Cundiff (2004) gives an overview of METS, while Dappert and Enders (2008) provide a good example of using several metadata schemes within it. PREMIS, like MPEG, is the name of a working group, PREservation Metadata—Implementation Strategies, and is located at www.loc.gov/standards/premis. The “on a horse” problem is from Wendler (2004). Freire et al. (2008) give a useful survey of provenance.

The quotation about reproducibility and eResearch is from Peng et al. (2006). Calls for replication are not confined to the hard sciences: see King (1995) in the area of politics. Schwab et al. (2000) discuss the positive experiences that stem from using reproducibility as a principle when running a laboratory. The Taverna workflow system can be found at <http://taverna.sourceforge.net>.

The quotation about metadata quality at the beginning of Section 6.5 is from Robertson (2005); Beall (2006) gives an overview of metadata quality problems. The 30th edition of the LCSH was published in 2007 (Library of Congress, 2007). The Library of Congress authority file can be searched online at authorities.loc.gov. The MADS example is from the Library of Congress, who maintain the standard at www.loc.gov/standards/mads. The quotation bemoaning the lack of name-authority control in institutional repositories is from Salo (2009). The National Library of Medicine's Medical Subject Headings (MeSH) can be found at their Web site, www.nlm.nih.gov/mesh. The visualization tool in Figure 6.13 that shows the completeness of metadata elements in an institutional repository is described by Nichols et al. (2008).

An account of an excellent and widely used automatic citation indexing system that embodies the techniques mentioned for extracting bibliographic references is given by Giles et al. (1998). The

method described for language identification was developed and investigated by Cavnar and Trenkle (1994) and implemented in a system called TextCat. The acronym extraction algorithm was developed and evaluated by Yeates et al. (2000). Different systems for browsing using key phrases are described by Gutwin et al. (1999) and Jones and Paynter (1999). Dumais et al. (1998) describe state-of-the-art text classification techniques for key-phrase assignment, while Frank et al. (1999) and Turney (2000) describe key-phrase extraction. Medelyan and Witten (2008) describe the process of key-phrase indexing and report experiments that show that automatic techniques can rival the work of professional indexers.

Interoperability: Protocols and services

Interoperability is the name of the game for libraries. An important part of traditional library culture is the ability to locate copies of information in other libraries and receive them on loan—interlibrary loan. Libraries work together to provide a truly universal international information service. The degree of cooperation is enormous and laudable. What other large organizations cooperate in this way?

Digital libraries, like library catalogs before them, are a product of a networked environment. Information can flow between different systems, between clients and servers, and between people and institutions. The examples we have seen so far have assumed the existence of a network and of cooperating systems. However, these interconnected systems did not come about by accident; they required the development of common understandings about the nature of data formats. It is these communication protocols that allow the reach of digital libraries to extend across our networked world and to interoperate.

This chapter begins by examining two key protocols in the digital library world: the relatively recent Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) and the long established Z39.50 standard. These protocols are separated historically by a decade and a half, culturally by the communities in which they arose, and ideologically by the approach taken. Z39.50 has been developed by standards committees whose members come from widely disparate backgrounds and who have very different agendas. Z39.50 supports a rich set of commands but it is large and complex to implement. OAI-PMH, on the other hand, is simpler and more focused on particular requirements. However, the protocols share the same aim, communicating with digital repositories, and common elements and themes recur between them.

As digital libraries pervade the workplace, the collective needs of users mushroom. The standard *modus operandi* of Web usage is to reach a new document by clicking on a button or hyperlink—and this is the standard method for a digital library, too. How can we step outside this model in a flexible way that does not involve a heavy commitment to a particular alternative? The answer is to provide fine-grained interaction with the content of a digital library through a defined protocol that others can use. A Web Service is defined as a software system designed to support interoperable machine-to-machine interaction over a network, and Web Services are becoming a popular way for digital libraries to supply their content to other systems and interfaces that are designed to satisfy particular user needs. For example, Search/Retrieval via URL (SRU) is a more recent standard (derived from Z39.50) that recasts Z39.50-like requests in the form of Web Services.

Protocols for spreading content and metadata address some of the questions we posed in Chapter 1 (Table 1.1) concerning the sharing of resources—in particular, questions about exporting material

from the library and interoperating with other libraries. When institutions hosting digital collections agree to share, it is these protocols that make it happen. Another enabling technology for sharing on the Web is the availability of standard methods that reliably identify digital resources. We have seen how stable URLs are central to the Web (Section 2.4)—imagine how inconvenient it would be if all the URLs you used were only valid for a few days! In this chapter we describe technological support for long-term stable URLs. [Section 7.5](#) returns to some of the access issues discussed in Chapter 2 and outlines techniques for authenticating users. Finally, [Section 7.6](#) presents two widely used open-source digital library systems, DSpace and Fedora, and discusses how they operate and interoperate.

7.1 Z39.50 Protocol

The Z39.50 standard defines a wide-ranging protocol for information retrieval between a client and a database server. It is administered by the Library of Congress, and progressive versions of the specification were ratified by standards committees in 1988, 1992, 1995, and 2003.

Z39.50 is an example of an application layer of the Open System Interconnection (OSI) Reference Model, a comprehensive standard for networked computer environments. Message formats are specified using Abstract Syntax Notation One (ASN.1) and serialized for transmission over the OSI transport layer using Basic Encoding Rules (BER). The Transmission Control Protocol (TCP) is typically used for the actual information communication.

Accessing and retrieving heterogeneous data through a protocol in a way that promotes interoperability is a challenging problem. To address the broad spectrum of different domains where it might be used—such as bibliographic data, museum collection information, and geospatial metadata—Z39.50 includes a set of classes, called *registries*, that provide each domain with an agreed-upon structure and attributes. Registries cover query syntax, attribute fields, content retrieval formats, and diagnostic messages. For example, content retrieval formats include Simple Unstructured Text Record Syntax (SUTRS) and the various MARC formats.

The Z39.50 protocol is divided into eleven logical sections (called *facilities*) that each provide a broad set of services. We cannot do justice to the myriad details here, but instead convey some idea of the functionality supported. [Table 7.1](#) gives a high-level summary of each category.

The protocol is predominantly client-driven; that is to say, a client initiates a request and the server responds. Only in a few places does the server demand information from the client—for example, the Access Control Facility might require the client to authenticate itself before a particular operation can be performed. Any server that implements the protocol must retain information about the client's state and apportion resources so that it can respond sensibly to clients using the Initialization Facility, which sets resource limits. Mandatory search capabilities include fielded Boolean queries, which yield result sets that can be further processed by the Sort and Browse Facilities or canceled by the Result-set-delete Facility. Results themselves are returned through the Retrieval Facility. At any stage the response to a request might be an error diagnostic.

Establishing which of the many Z39.50 options, registries, and domain-specific attributes are supported by a particular server is accomplished through the Explain Facility. The Extended Services

Table 7.1: Facilities provided by Z39.50

Z39.50 Facility	Client-side description
Initialization	Establish connection with server and set/request resource limits.
Search	Initiate search using a registered query syntax, generating a result set server-side.
Retrieval	Retrieve a set of records from a specified result set; a large record may be segmented and transmitted piecemeal.
Result-set-delete	Request deletion of server-side result set or sets.
Access Control	Server-initiated authentication check.
Accounting & Resource Control	Request status reports of committed server resources and dictate if server is allowed to contact client when agreed limits are reached.
Sort	Specify how a result set should be sorted.
Browse	Access ordered lists, such as title and subject metadata.
Explain	Interrogate server to discover supported services, registries, and so on.
Extended Services	Access services that continue beyond the life of this client-server exchange, such as persistent queries and database update.
Termination	Abruptly end client-server session: initiated by either client or server.

Facility is a mechanism for accessing server functionality that persists beyond the duration of a given client-server session—for example, periodic search schedules and updating the database. The client-server session can be canceled immediately by either side through the Termination Facility.

A particular Z39.50 system need not implement all parts of the protocol. Indeed, the protocol is so complex that full implementation is a daunting undertaking and may in any case be inappropriate for a particular digital library site. For this reason, the standard specifies a minimal implementation, which comprises the Initialize Facility, the Search Facility, the Present Service (part of the Retrieval Facility), and Type 1 Queries (part of the registry).

Using this baseline implementation, a typical client-server exchange works as follows. First the client uses the Initialization Facility to establish contact with the server and negotiate values for certain resource limits. This puts the client in a position to transmit a Type 1 query using the Search Facility. The number of matching documents is returned, and the client then interacts with the Present Service to access the contents of desired documents.

7.2 Open Archives Initiative

The Open Archives Initiative (OAI) was motivated by the electronic preprint community, which has a strong desire to increase the availability of scholarly repositories and to enhance access to them. Part of the initiative has been to devise a protocol for the efficient dissemination of content, the OAI Protocol for Metadata Harvesting, OAI-PMH, which is often abbreviated to just OAI. This protocol is intentionally broad and independent of content, making it useful to a wide range of areas, not just scholarly information. It is “open” in the sense that its definition is freely available and its use is

encouraged. The term *archive* should not be taken to imply only the compilation of digital material for historical purposes—although this is certainly one use.

OAI Protocol for Metadata Harvesting: OAI-PMH

OAI-PMH is intended as a standard way to move metadata from one point to another within the virtual information space of the World Wide Web. It supports interaction between a *data provider* and a *service provider*—a renaming of the client-server model that emphasizes that interaction is driven by the client (the service provider) and that the client alone has the onus of deciding what services are offered to users. Data providers, in contrast, are in the business of managing repositories. They export on-demand data records in a standardized form, unencumbered by any consideration of how the information will be used. If other operations, such as text searching, are to be supported, they must be performed by the service provider, not the data provider. Of course, a site may choose to be both a service provider and a data provider. It may also manage more than one repository.

A key technical goal of the protocol is that it be easily implemented using readily available software tools and support. The protocol provides a framework, using HTTP, in which requests are encoded in URLs and executed by programs on the server. Results are returned as XML records. [Figure 7.1](#) shows the basic form of interaction using an example that requests a record pertaining to a particular document identifier and expressed using Dublin Core metadata. First the arguments are encoded into a URL (characters like `:` and `/` must be specially coded) and dispatched over HTTP. The response is an XML document that draws heavily on XML namespaces and schemas.

XML namespaces are a way of ensuring that markup tags intended for a particular purpose do not conflict with other tag sets, and XML Schema is a way of extending the idea of predefined tag sets and document structure with typed content. For example, *responseDate* in OAI is defined to be the complete date, including hours, minutes, and seconds, using the schema type *xsd:timeInstant*.

The top-level element in the example record is *GetRecord*. Through its attributes this sets up a suitable namespace using a URL in the Open Archives Web site and specifies the defining XML Schema stored at the same site. Nested deeper in the *GetRecord* structure is a *metadata* element that uses the Dublin Core namespace and a schema specifically for this metadata standard that is also defined at the Open Archives site.

Within the *metadata* element is the main content—expressed, as requested, using Dublin Core metadata—for the digital item that matches document identifier *oai:nzdl:hdl/018cf2f4256b4c8827e747b8*. The book is titled *Farming Snails* and was written for the Food and Agriculture Organization of the United Nations (and part of the Humanity Development Library featured in Chapters 1 and 3). Further information supplies a brief description of its content, gives the year of publication, declares that it is principally text, and states its format as HTML. This particular record is rich in metadata, but the protocol does not require this level of richness.

The protocol supports six services through the *verb* argument, and [Table 7.2](#) summarizes them. *Identify* and *ListSets* are services that are typically called early on in a client's interchange with a server to establish a broad picture of the repository. [Figure 7.2](#) shows an example of a response to an

```

Request Arguments:
  verb=GetRecord
  identifier=oai:nzdl:hdl/018cf2f4256b4c8827e747b8
  metadataPrefix=oai_dc

Encoded URL:
  http://www.nzdl.org/cgi-bin/oai/request?verb=GetRecord
    &identifier=oai%3Anzdl%3Ahd%2F018cf2f4256b4c8827e747b8
    &metadataPrefix=oai_dc

Response:
<?xml version="1.0" encoding="UTF-8"?>
<GetRecord
  xmlns="http://www.openarchives.org/OAI/1.0/OAI_GetRecord"
  xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xsi:schemaLocation="http://www.openarchives.org/OAI/1.0/OAI_GetRecord
    http://www.openarchives.org/OAI/1.0/OAI_GetRecord.xsd">
  <responseDate>2001-06-16T13:15:42+00:00</responseDate>
  <requestURL>http://www.nzdl.org/cgi-bin/oai/request?verb=GetRecord
    &identifier=oai%3Anzdl%3Ahd%2F018cf2f4256b4c8827e747b8
    &metadataPrefix=oai_dc</requestURL>
  <record>
    <header>
      <identifier>oai:nzdl:hdl/018cf2f4256b4c8827e747b8</identifier>
      <datestamp>1986-01-01</datestamp>
    </header>
    <metadata>
      <dc xmlns="http://purl.org/dc/elements/1.1/"
        xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
        xsi:schemaLocation="http://purl.org/dc/elements/1.1/
          http://www.openarchives.org/OAI/dc.xsd">
        <title>Farming Snails</title>
        <description>Learning about snails; Building a pen; Food
          and shelter plants</description>
        <creator>Food and Agriculture Organization of the United Nations</creator>
        <type>Text</type>
        <format>text/html</format>
        <date>1986-01-01</date>
        <identifier>
          http://www.nzdl.org/cgi-bin/library?&a=d&cl=search&d=HASH018cf2f4256b4c8827e747b8
        </identifier>
      </dc>
    </metadata>
  </record>
</GetRecord>

```

Figure 7.1: OAI *GetRecord* request and XML response

Identify request, which contains administrative information about the server: in this case, the OCLC's OAI Cat Repository Framework, whose repository name is *IDEALS @ UIUC*. *ListIdentifiers* is a way of receiving all the document identifiers or a group that matches a stipulated set name. *ListMetadataFormats* can be applied to the repository as a whole or to a particular document within it to establish which metadata formats are supported. Dublin Core is mandatory, but other formats, such as MARC, which has the capacity to export a greater volume of metadata per record, may also be supported. We have already seen *GetRecord* in action in the example just given—*ListRecords* is similar except that

Table 7.2: Open Initiative Archive protocol requests

OAI protocol request	Description
<i>Identify</i>	Returns both fixed-format and domain-specific descriptions.
<i>ListSets</i>	Returns the repository's classification hierarchy.
<i>ListIdentifiers</i>	Returns a list of document identifiers.
<i>ListMetadataFormats</i>	Returns the metadata formats supported by the repository in general or for a specific document.
<i>GetRecord</i>	Returns the repository item specified by the document identifier in the requested format.
<i>ListRecords</i>	Returns a list of repository items in the requested format.

more than one record can be returned, and group selection is possible with the same set-naming technique used by *ListIdentifiers*.

These are the general facilities supported by the protocol. Greater flexibility can be achieved using the input arguments and the set and resumption mechanisms. First, repository items can be categorized into sets—multiple hierarchies, where an item can be in more than one hierarchy or in none at all. Each node in the hierarchy has a set name, such as *beekeeping*, and a descriptive name, such as *Beekeeping and honey extraction*. Its hierarchical position is specified by concatenating set names separated by colons—like *husbandry:beekeeping*. *ListSets* returns this information for the complete repository, marked up as XML.

The set mechanism means that instead of retrieving all the document identifiers, a service provider can restrict the information returned by *ListIdentifiers* to a particular set. (The same applies to *ListRecords*.) Because records in the repository are date-stamped, the information returned can be restricted to a particular range of dates using *from* and *until* arguments. Even so, the list of data returned may still be excessively long. If so, a data provider might transmit part of the data and include a *resumption token* in the record. This mechanism enables the service provider to contact the data provider again and request the next installment (which might include another resumption token).

The OAI protocol makes use of the HTTP status code mechanism to indicate the success or failure of a request. Status code 400, in this context, indicates a syntactic error in the request, such as an invalid input argument. Other forms of failure, such as a repository item being unavailable in the requested metadata format, produce null data in the XML record that is returned.

Serving

OAI-PMH imposes no requirements on how metadata is represented on the server of a data provider. The metadata can be stored in a database, as simple XML files, or even embedded within other documents. All that matters is that the data provider responds to protocol requests with appropriate responses; in the case of metadata embedded in documents, the metadata would first have to be extracted and then packaged up as an OAI-PMH response.

```

Request Arguments:
    verb=Identify

Encoded URL:
    http://www.ideals.uiuc.edu/dspace-oai/request?verb=Identify

Response:

<?xml version="1.0" encoding="UTF-8" ?>
<OAI-PMH xmlns="http://www.openarchives.org/OAI/2.0/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/
    http://www.openarchives.org/OAI/2.0/OAI-PMH.xsd">
  <responseDate>2009-02-04T20:01:57Z</responseDate>
  <request verb="Identify">http://www.ideals.uiuc.edu/dspace-oai/request</request>
  <Identify
    <repositoryName>IDEALS @ UIUC</repositoryName>
    <baseUrl>http://www.ideals.uiuc.edu/dspace-oai/request</baseUrl>
    <protocolVersion>2.0</protocolVersion>

    <adminEmail>ideals-gen@uiuc.edu</adminEmail>
    <earliestDatestamp>2001-01-01T00:00:00Z</earliestDatestamp>
    <deletedRecord>persistent</deletedRecord>

    <granularity>YYYY-MM-DDThh:mm:ssZ</granularity>
    <compression>gzip</compression><compression>deflate</compression>
    <description>
      <toolkit xsi:schemaLocation="http://oai.dlib.vt.edu/OAI/metadata/toolkit
        http://oai.dlib.vt.edu/OAI/metadata/toolkit.xsd"
        xmlns="http://oai.dlib.vt.edu/OAI/metadata/toolkit">
        <title>OCLC's OAIcat Repository Framework</title>
        <author>
          <name>Jeffrey A. Young</name>
          <email>jyoung@oclc.org</email>
          <institution>OCLC</institution></author>
        <version>1.5.48</version>
        <toolkitIcon>http://alcme.oclc.org/oaicat/oaicat_icon.gif</toolkitIcon>
        <URL>http://www.oclc.org/research/software/oai/cat.shtm</URL></toolkit>
      </description>
    </Identify>
  </OAI-PMH>

```

Figure 7.2: OAI *Identify* request and response

It is even possible to have a static metadata repository. A static repository is a single XML file that contains all the metadata a data provider wishes to share. This file needs to be registered with a Static Repository Gateway, which then performs the appropriate transformations to allow service providers to access the metadata. Static repositories are only really appropriate for small collections that do not change frequently. Most digital library systems come with OAI-PMH support built in, although they may only support basic Dublin Core metadata. Support for qualified Dublin Core (or MODS, METS, etc.) is rarer; as a result, although a digital collection may have detailed internal metadata, only a subset may be available via OAI-PMH. Naturally, such restrictions limit the applications that OAI service providers can create.

Harvesting

OAI-PMH was designed to allow structured retrieval of metadata on the Web. We can usefully contrast this with the unstructured approach of Web harvesting, or Web mirroring. When the remote resources are as varied as Web content, then most attempts to gather metadata will produce uneven results. There is no standard metadata scheme for Web content, and although metadata like Dublin Core is sometimes embedded within HTML, it is not present in sufficient quantity or with enough reliability to be of much use. OAI-PMH, on the other hand, provides structure through a defined protocol, which is a necessary condition for effective metadata sharing. However, a protocol by itself does not guarantee that the metadata available is of good quality.

Creating and sharing quality metadata is not a straightforward task. It requires appropriate tools, trained staff, and a long-term commitment to resource availability. Although the technologies may be relatively simple, this is only a *necessary* condition for success, and without the associated human support it will not be sufficient. One example of a large-scale effort to harvest OAI metadata has been the U.S. National Science Digital Library (NSDL), which began in 2003 to implement a digital library based on metadata aggregation using Dublin Core and OAI-PMH. However, a retrospective analysis in 2006 found that these standards, which are supposed to be easy to adopt with a low human cost, were harder to deploy than expected.

Our expectation was that Dublin Core and OAI-PMH were relatively simple and that surely every collection provider would be able to implement them and be integrated into the National Science Digital Library. In fact, reality fell far short of our expectations. We discovered that the WorldCat paradigm, which works so well in the shared professional culture of the library, is less effective in a context of widely varied commitment and expertise. Few collections were willing or able to allocate sufficient human resources to provide quality metadata.

The comparison here is with WorldCat, the world's largest bibliographic database. WorldCat combines the collections of more than 10,000 libraries from over 100 countries that participate in the Online Computer Library Center global cooperative. It comprises 125 million different records pointing to physical and digital assets in more than 500 languages. (These statistics quickly become outdated, however, because a new record is added to the database on average every 10 seconds.)

Overall, OAI-PMH has been a successful method of sharing metadata. Its widespread adoption is due to the simplicity of the protocol and its building on top of other existing standards: HTTP, XML, and Dublin Core.

7.3 Object Identification

The Web scheme of Uniform Resource Locators (URLs) has been a great success. It has passed from the world of computer networking to common usage. URLs enable users to refer to resources, whether simple HTML documents or services, and be fairly confident that if they use those URLs again the URLs will still refer to the same things. Furthermore, computer programs can refer to external resources in the same way.

However, every Web user has clicked a link and received a message indicating that the target was not found (in raw form this message cryptically states “404 not found,” although Web servers often

translate it into something more informative and helpful). Sometimes, the explanation is that the link was incorrectly written—there was a typo in the HTML file—but often the target resource has been moved. The one-way link structure that underlies the Web means that when a resource is moved to a different URL, a Web browser that finds the old link on some other HTML page cannot ascertain the new destination. Dead links are particularly frustrating when you think you have just found the document you were looking for only to be foiled at the last step. It is even worse for Web-based software services, which, being computer programs, may simply stop working altogether, leaving their users completely stymied. Consequently, it is very useful for URLs, or, more generally, digital object identifiers, to be robust.

Many schemes have been devised to make identifiers persist over time. The basic idea is to register names with ultra-reliable servers that keep track of their actual location (URL). Any reference to the resource goes through a server, which knows where the resource actually is. When a resource (such as a document in a digital library) is moved, this server must be notified. Without these long-life approaches, *link rot* sets in—links to the content start to fail, and users become frustrated that (in this case) the digital library is unreliable and hard to use. Two generic approaches for naming Internet objects are the Uniform Resource Name (URN) and the Uniform Resource Identifier (URI). Both are specifications, not implementations, and both are the subject of considerable debate. We previously encountered these in Section 6.4 in connection with the Resource Description Framework (RDF).

Handles

An early technical solution to the problem of persistence was the Persistent URL (PURL), operated by the Online Computer Library Center (OCLC). Instead of describing the location of the resource to be retrieved, a PURL gives an intermediate, persistent, location which, when retrieved, is automatically redirected to the current location of the resource. A similar technical solution called Handle, from the Communications Network Research Institute (CNRI), is implemented as a Java-based program for assigning, managing, and resolving persistent identifiers for digital objects on the Internet.

The PURL and Handle systems (and the DOIs discussed below) are similar in that they maintain a database of the relationship between a name and an actual URL. When a user or a piece of software has the name, it uses a *resolution service* to convert the name into an actual URL. If the actual URLs need to be changed, typically because the content has moved to a new server, only the database is updated.

Handles can be resolved by typing into any Web browser the string

```
http://hdl.handle.net/
```

followed by the Handle itself. For example, a paper called *Building digital library collections with Greenstone* can be found in the University of Waikato Research Commons at <http://hdl.handle.net/10289/1825>. The resolution service (at <http://hdl.handle.net>) takes the object's Handle (10289/1825) and returns a URL (at the time of writing, <http://waikato.researchgateway.ac.nz/handle/10289/1825>). When used in a Web browser, the effect is to seamlessly navigate to the actual URL. However, Handles can be used just as easily by software applications outside of a browser, provided they use the HTTP protocol.

Digital object identifiers: DOIs

Achieving persistence is not really a technical problem, or even a naming problem. It's an organizational problem. To make URLs persistent, all you really need is a mechanism for assigning names, and an organization that administers them and records their locations—an organization that is expected to persist over time and is scalable enough to meet the world's needs. A scheme that is actually in use is the Digital Object Identifier (DOI), which is a *de facto* practical implementation of the URN and URI specifications using the Handle system. In fact, DOIs are Handles with the prefix "10." (The Handle system also incorporates other namespaces.)

The International DOI Foundation defines a Digital Object Identifier as a permanent identifier for any object of intellectual property and explains that it is used to identify a piece of intellectual property on a digital network and associate related data with it in an extensible way. A typical use is to give a scholarly article a unique identification number that anyone can use to obtain information about its location on a digital network. DOIs can easily be resolved: just type in the address bar of any Web browser the string

`http://dx.doi.org/`

followed by the DOI of the document. For example, the paper mentioned above has the DOI 10.1145/1065385.1065530, and you can see it at <http://dx.doi.org/10.1145/1065385.1065530>. The resolution service (at <http://dx.doi.org/>) takes the object's name (10.1145/1065385.1065530) and returns a URL (<http://portal.acm.org/citation.cfm?doid=1065385.1065530>). This identifies a copy of the document in the ACM Digital Library, rather than the one mentioned above at the University of Waikato.

DOIs are administered by the International DOI Foundation, a consortium that includes both commercial and non-commercial partners, and are currently being standardized through ISO. The structure of the DOI conveys these pieces of information:

- The prefix 10.1145 identifies the naming authority who assigned the DOI (10 is the directory code; 1145 is the publisher ID).
- The suffix 1065385.1065530 is the item ID or local name.

As noted earlier, DOIs build upon the Handle system, and DOIs are Handles with the directory code "10," which refers to the International DOI Foundation. The publisher ID 1145 indicates the Association for Computing Machinery (ACM). Other examples include 1000, which is used by the International DOI Foundation itself, and 1007, which is used by the Springer publishing house.

The prefixes are guaranteed unique because they are registered with the global Handle Registry, and the Handle syntax ensures that suffixes are unique within the namespace defined by the prefix. The DOI directory code (10 above) is a prefix in the Handle system, and the DOI prefix 10.1145 is known as a *derived prefix*. Because DOIs are themselves Handles, the same document that we discussed above can also be found in the ACM Digital Library by resolving the DOI through the central Handle server: <http://hdl.handle.net/10.1145/1065385.1065530>.

DOIs are administered by agencies recognized by the International DOI Foundation. They register identifiers, allocate prefixes, and allow registrants to declare and maintain metadata. The agencies are commercial: they levy a small charge to establish each new identifier. Practically, a publisher using this scheme uploads a file to a DOI Registration Agency, which links a DOI with the actual URL. If the URL changes, the publisher uploads the revised DOI-URL association.

OpenURLs

The purpose of persistent URLs is to create reliable identifiers that can be used to link to a document, no matter where it is stored. A different problem is to discover a link to a document when all you have is its metadata. This is accomplished by a scheme called OpenURL, which is often run as a service by libraries. An OpenURL is generated by a bibliographic citation or bibliographic record; its target is a resource that satisfies the user's information need.

An OpenURL consists of a base URL that usually addresses an institutional link server and a query string that specifies metadata in the form of key-value pairs. For example, <http://www.oxfordjournals.org/content?genre=article&issn=0006-8950&volume=126&issue=2&spage=413> specifies a particular article at the base URL www.oxfordjournals.org by giving its serial number, volume, issue, and page number. In the case of this particular OpenURL resolver, the first four parameters—genre, ISSN (International Standard Serial Number), volume, and issue—are mandatory, but metadata like title and author can be given as an alternative to the page number.

The idea of OpenURLs is that a user can query an online bibliographic database and end up with a pointer to the full text of the article he wants to read. Whether the user is able to download the article depends on whether his institution has an appropriate subscription. To determine this, the link needs to be resolved by the user's institution to point to the appropriate institutional copy. An OpenURL can be made to resolve to a copy of a resource in a different library simply by changing the base URL. Thus, the same OpenURL can easily be adjusted by either library to provide access to its own copy of the resource.

OpenURLs work well for documents that have standard metadata associated with them. In order to refer to general documents on the Web, a scheme involving associative links has been proposed. Pages can be identified not just by a name or location, but also by their content—the words they contain. A few well-chosen words or phrases are sufficient to identify almost every Web page exactly, with high reliability. These snippets—called the page's *signature*—could be added to the URL. Browsers would locate pages in the normal way, but if they encountered a broken-link error, they would pass the signature to a search engine to find the page's new location. The same technique identifies all copies of the same page on the Web—an important facility for search engines, so they need not bother to index duplicates. The scheme relies on the uniqueness of the signature: there is a trade-off between signature size and reliability.

Persistence

As noted above, persistence is really an organizational problem, and any solutions require the organizations involved to be perceived as stable and long-lasting. Some organizations, such as the Library

of Congress or the International Organization for Standardization (ISO), have obvious longevity and national/international stature. Newer organizations, such as the Communications Network Research Institute that runs the Handle system and the International DOI Foundation that runs the DOI system, rely on technical competence and their wide user base.

7.4 Web Services

In order for an institutional repository to successfully meet the needs of its various user communities, it should provide not only preservation and access to digital objects but also a range of services that make these objects useful.

Many of the examples of digital collections we have seen so far allow users to access digital objects of various types. Z39.50 and OAI-PMH provide mechanisms for programmatic access to metadata, with indirect access to the actual objects. Eclipsing bespoke solutions such as these, the Semantic Web looks forward to a networked world with ubiquitous access to a wide variety of both resources *and* services, a world of Web Services. For digital libraries, the Web Services paradigm addresses the issue of how your library holdings can be shared in a more general way with other users or programs.

Web Services are a broad and rapidly evolving topic, with many supporting technologies and countless applications; we give only a brief outline here. There is general agreement that Web Services are:

- modular
- self-describing
- self-advertising
- standards-based
- platform and programming-language independent
- XML-based.

In practice, a typical service receives an incoming request, processes it, and returns a response. Both the request and response are XML documents. The Web contains many services, each providing small chunks of functionality, and in processing a request, a service may call on several other services. The idea is that many organizations, including digital libraries, will open up their resources in a controlled manner to encourage new value-added services.

Technically, a server that processes Web Service requests resembles a Web server responding to ordinary requests from users who click links in their Web browsers. However, Web Service requests are more likely to originate with software and need to be used in more structured ways than clicking a link and returning a page. In order to inform other programs how it is to be used, a Web Service is self-describing: the Web Services Description Language (WSDL, also pronounced *wiz-del* by some), which is based upon XML, is a common choice.

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions ...>
  <!--WSDL created by Apache Axis version: 1.2.1. Built on Jun 14, 2005-->
  <wsdl:types>
    <schema targetNamespace="http://..." xmlns="http://...">
      <element name="getUsername" type="xsd:anyType"/>
    </schema>
    <schema targetNamespace="...">
      <element name="getUsernameReturn" type="xsd:anyType"/>
    </schema>
  </wsdl:types>
  <wsdl:message name="getUsernameResponse">
    <wsdl:part element="impl:getUsernameReturn" name="getUsernameReturn"/>
  </wsdl:message>
  <wsdl:message name="getUsernameRequest">
    <wsdl:part element="tns1:getUsername" name="part"/>
  </wsdl:message>
  <wsdl:portType name="Username">
    <wsdl:operation name="getUsername">
      <wsdl:input message="impl:getUsernameRequest" name="getUsernameRequest"/>
      <wsdl:output message="impl:getUsernameResponse" name="getUsernameResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="UsernameQuerySoapBinding" type="impl:Username">
    <wsdlsoap:binding style="document" transport="http://...">
    <wsdl:operation name="getUsername">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="getUsernameRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="getUsernameResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="UsernameService">
    <wsdl:port binding="impl:UsernameQuerySoapBinding" name="UsernameQuery">
      <wsdlsoap:address location="http://...">
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>

```

Figure 7.3: Web Service description expressed in WSDL

A WSDL file provides enough information for a program to be able to interact with a Web Service without any human intervention. Figure 7.3 shows a WSDL file that describes a service called *UsernameService* (defined near the end) that processes usernames, possibly as part of a larger account management service. The WSDL specification describes where to find the service in the *wsdlsoap:address* element (also near the end). It specifies that it expects a string *literal* as input (in the *<wsdl:input name="getUsernameRequest">* element) and produces another string *literal* as output (in the *<wsdl:output name="getUsernameResponse">* element).

The input and output described in the WSDL of Figure 7.3 are both *wsdlsoap* elements, indicating that the request and response to the *UsernameService* should be packaged in the XML Simple Object Access Protocol (SOAP). Figure 7.4 shows examples of the input and output SOAP messages to

(a)

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope"
  xmlns:xsd="http://www3.org/2001/XMLSchema"
  xmlns:xsi="http://www3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <message>
      <username>js</username>
    </message>
  </soapenv:Body>
</soapenv:Envelope>
```

(b)

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope ... as above>
  <soapenv:Body>
    <message>John Smith</message>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 7.4: Web Services request and response pair using SOAP: (a) request; (b) response

UsernameService. As you can see, a SOAP message consists of an *envelope* that surrounds the XML content or *payload*. Typically this payload will be a structured XML document: in [Figure 7.4a](#) it is `<username>js</username>`. The output comes in the same form. It could be a result list from a query, but might be as simple as a plain string (*John Smith* in [Figure 7.4b](#)) or a URL to a further resource. SOAP is typically sent over HTTP, although it can be used with many different transport mechanisms.

Many Web Services will be deployed at specific known URLs. If you want a name processed by the *UsernameService*, you can simply use its Web address. However, the Web Services environment contains a technology for self-advertising. The Universal Description, Discovery and Integration (UDDI) protocol provides a standard service for the discovery of Web Services, in effect acting as a registry of suppliers. UDDI is yet another XML format that allows various aspects of services to be expressed, such as *businessServices* and *discoveryURL*. UDDI allows programs to dynamically discover Web Services, although user interfaces for people to browse the services are usually available as well.

Web Service requests can be sent as SOAP messages, but they can also be encoded within a standard URL as HTTP GET requests. Just as an OAI request can be expressed as a URL (see [Figures 7.1](#) and [7.2](#)), many Web Service requests can be simplified from a SOAP message to a plain URL. For example, the request in [Figure 7.4](#) to look up the user associated with the username *js* might be expressed as `http://www.example.com/username/js`. If a Web Service can be expressed using URLs and simple stateless requests, then it is sometimes described as RESTful (Representational State Transfer). The Web itself is a RESTful system, and the HTTP methods GET and POST are examples of stateless requests, ones that are independent of previous requests. REST is not a standard itself—though it builds on familiar Web standards like HTTP, URL, and XML—but more a software architectural style that can simplify Web Service development.

All this technological infrastructure allows managers of digital collections to provide value-added services that build on the objects and metadata they already possess—or enable others to do so. Services for a repository that could be realized through the Web Services architecture include:

- annotation services
- automatic document alignment
- automatic map and timeline generation
- citation linking
- dynamic/ad hoc collection building
- gazetteer lookup
- named entity identification
- personal publication and aggregation
- social tagging and bookmarking
- text chunking and alignment
- vocabulary lookup.

Search/Retrieval via URL: SRU

As described in [Section 7.1](#), Z39.50 is a complex protocol that was developed before the Web really took off. More recently, its basic ideas have been re-engineered for the Web environment, in the form of the Search/Retrieval via URL (SRU) protocol. SRU can operate in two flavors, with requests in HTTP (either GET or POST requests) or as a SOAP Web Service. The latter used to be called the Search/Retrieval Web Service (SRW).

Whereas OAI has six verbs, SRU has three main operations:

- *searchRetrieve* sends queries and receives responses,
- *explain* allows client programs to learn about the server’s capabilities, and
- *scan* retrieves lists of terms used in creating browsing structures.

[Figure 7.5](#) shows an SRU response to a *searchRetrieve* query. The response returns a metadata record, and in this case the metadata is couched as a MODS record ([Section 6.2](#)). The *recordData* contains origin information (publisher, place and date issued), title information (*Five Songs*), personal name and role (*Mary Lane Bayliff*, creator), language, physical description, and information about local library holdings. The *explain* operation mirrors Z39.50’s Explain Facility mentioned in [Section 7.1](#). The *scan* operation allows the client to request a range of the available terms at a given point within a list of indexed terms. It enables clients to present users with an ordered list of values and to show how many hits there would be for each one. This operation is often used to select terms for subsequent searching or to verify a negative search result.

```

<?xml version="1.0"?>
<zs:searchRetrieveResponse xmlns:zs="http://www.loc.gov/zing/srw/">
  <zs:version>1.1</zs:version>
  <zs:numberOfRecords>71</zs:numberOfRecords>
  <zs:records>
    <zs:record>
      <zs:recordSchema>info:srw/schema/1/mods-v3.0</zs:recordSchema>
      <zs:recordPacking>xml</zs:recordPacking>
      <zs:recordData>
        <mods xmlns:xlink="http://www.w3.org/1999/xlink"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns="http://www.loc.gov/mods/v3"
              xmlns:h="http://copac.ac.uk/schemas/holdings/v1"
              xmlns:cpc="http://copac.ac.uk/schemas/mods-copac/v1"
              xsi:schemaLocation="...">
          <recordInfo>
            <recordIdentifier source="copac">72004200472</recordIdentifier>
            <recordContentSource>Copac</recordContentSource>
            <recordCreationDate>20060525</recordCreationDate>
          </recordInfo>
          <originInfo>
            <dateIssued encoding="marc">1904</dateIssued>
          </originInfo>
          <originInfo>
            <place><placeTerm type="text">London :</placeTerm></place>
            <publisher>Weekes & Co,</publisher>
            <dateIssued>[1904]</dateIssued>
          </originInfo>
          <titleInfo>
            <title>
              Five Songs, words by L. Shoobridge. No. 1. Ucelli che sono in mare!
              No. 2. A Day in March. No. 3. In the Breeze ...
            </title>
          </titleInfo>
          <name type="personal">
            <namePart>Bayliff, Mary Lane.</namePart>
            <role><roleTerm type="text">creator</roleTerm></role>
          </name>
          <language>
            <languageTerm type="text">Undetermined</languageTerm>
          </language>
          <physicalDescription><extent>4.&#x30A;</extent></physicalDescription>
          <extension>
            <h:localHolds>
              <h:org type="CURL" displayName="British Library">BLI</h:org>
              <h:objId>72004200472</h:objId>
              <h:holds>
                <h:item>
                  <h:loc displayName="Printed Music, St Pancras Reading Rooms">
                    MUSIC
                  </h:loc>
                  <h:shelfmark>G.385.bb.(4.) G</h:shelfmark>
                </h:item>
              </h:holds>
            </h:localHolds>
          </extension>
        </mods></zs:recordData>
        <zs:recordPosition>1</zs:recordPosition>
      </zs:record>
    </zs:records>
  </zs:searchRetrieveResponse>

```

Figure 7.5: SRU response wrapping MODS metadata

SRU services are more fine-grained than those of OAI-PMH. They allow clients to retrieve records with greater precision. It is perfectly reasonable for a digital library to provide access to its resources through both protocols and to allow clients to choose the one that best suits their needs.

7.5 Authentication and Security

As discussed in Section 2.2, it is sometimes necessary to restrict access to parts of a digital library. Threats to the integrity of a digital library arise from human hackers (or crackers) and from the destructive programs they create, such as viruses and worms. Library administrators will almost always need to restrict access to the administrative functions of the software (the only possible exceptions might be public-editing open-content systems like Wikipedia). In some cases they will also need to restrict access to the content. Some of these restrictions are inevitable once a collection is placed on a network. Collection maintenance relies on the security of the underlying library software and of the associated applications, such as Web servers.

The next general step is to ensure that the content that is placed on your network-facing server is content that you *really* want to share. Many users place private documents on Web servers where they think they are concealed, but simple searches for terms like *confidential* and *not for distribution* show that Web servers and Web-indexing robots often conspire to reveal such documents. Once you have organized your content, arranged for regular and reliable backups, and changed any default passwords in your software, you are ready to consider how to configure access to your particular services.

Computer users are accustomed to providing usernames and passwords to log in to systems or gain access to resources. The username and password pair authenticates a person to the system, which then grants some privileges to the person. The password is an *authentication factor*. Authentication factors can be classified into three groups:

- *something you know*: a password or personal identification number (PIN);
- *something you have*: a token, such as bank card;
- *something you are*: biometrics, such as fingerprints and voice recognition.

Sometimes the factors are combined. Using a bank card in an ATM is two-factor authentication: the card and the PIN. The more authentication factors used, the harder it is to bypass the authentication. But greater security is accompanied by increasingly complex access. This complexity is often desirable: we *want* it to be difficult to launch a nuclear missile, so we accept that launch systems should have complex multi-factor authentication. For users of Web-based systems, the one-factor pair of username and password is the *de facto* norm, but the appropriate level of authentication depends on the nature of the content or service. For example, the administrative functions of a digital library could require that access be from a particular network (e.g., a university campus) as well as be password-protected.

For *something you know* authentication factors to be effective, the *something* has to remain private—if others know your password, they can pretend to be you. When private data is used to authenticate over a network, it must be transmitted using a protocol that hides that data. On the Web, URLs that

begin with *https://* initiate a connection using the Hypertext Transfer Protocol Secure (HTTPS). HTTPS takes normal HTTP data and encrypts it to prevent third parties from observing the requests and responses using a cryptographic protocol such as Secure Sockets Layer (SSL) or its successor Transport Layer Security (TLS). Many Web browsers show a padlock icon when a HTTPS session is in progress, and most users are now generally aware that this indicates a secure connection.

Once user name and password have been securely transmitted to the digital library software, they need to be checked against a register, or directory, of valid usernames and passwords. A digital library system could manage this process itself or use an external module that provides a *directory service*. Many directory services support the widely used Lightweight Directory Access Protocol (LDAP) to manage user data; LDAP can be accessed by e-mail clients as well as other applications. Implementations of LDAP, including encryption, are available for many platforms and are often integrated into other applications, such as the Apache Web server.

The appropriate level of authentication for a digital library needs to balance the value of identifying users with the costs imposed by the authentication mechanisms. If restrictions are imposed on the content of passwords, they become burdensome for users to remember. If users are forced to choose a new password for every service they use, they are likely to simply re-use the same password. Some users react to password restrictions by writing them down on sticky notes attached to their displays, negating the security benefits. Because many services ask users to create usernames and passwords, an approach for reducing username/password overload has been devised, called *single sign-on*.

True single sign-on allows one set of authentication details to be used by many different services; users enter their credentials just once. Single sign-on is often adopted within organizations to reduce password overload and to lower barriers to access. There is no widely used single sign-on technology on the Internet. A related idea, sometimes also referred to as single sign-on, allows users to have one username/password pair for many different services. Although users still have to log in many times, they can do so with a single identity.

The OpenID standard is a mechanism for using one digital identity on the Web. A user first registers with an OpenID *provider* and generates an OpenID *identifier* (which can be a URL). A Web site that adopts this authentication process asks users for their *identifier* and redirects them to the provider to enter their password. The provider checks the password and sends a message back to the site to confirm the login. When many sites support OpenID, users can simply re-use their identifier without needing to remember many usernames and passwords. At the time of writing, OpenID claims tens of thousands of Web sites and hundreds of millions of users.

An alternative solution to password overload is the Shibboleth system, which is based on the Security Assertion Markup Language (SAML). SAML is a standard built on top of HTTP, XML Schema, and SOAP that uses XML to express statements about authentication and authorization. Shibboleth is broadly similar to OpenID but in practice it is usually institutionally oriented, because the server that authenticates your identity is managed by your organization. For example, a university might vouch for your online identity—a situation that is particularly useful for access to high-value resources where institutional weight enhances trust. Whereas your Shibboleth access may continue only so long as you remain with the same institution, OpenID is potentially a user-focused lifelong identity solution. However, given the speed with which the Web changes, reliable predictions are difficult to make.

Most digital library software comes with either a built-in authentication infrastructure or the ability to connect to existing external systems. Here are some practical suggestions for designing authentication for digital library collections:

- Ensure that the system uses HTTPS connections on the Web to reassure users who are familiar with this idea.
- Use organizational single sign-on systems wherever possible.
- Consider cross-domain systems, such as Shibboleth or OpenID, to reduce barriers to access for external users.

Once everything has been set up, do not forget the ongoing maintenance element, which requires:

- updating software with security patches and newer versions,
- maintaining the user database as people leave and arrive,
- checking that backups are really working, and
- regularly checking your system's log files (and those of any associated Web servers) to monitor usage.

The security and integrity of a digital library system and its content are an ongoing concern, not just a one-off task. Backups are particularly important: The very term “library” encourages users to think that, as in a physical library, every precaution is taken not to lose material, and backups are insurance policies—people usually need backups only in a time of crisis, which is not a good moment to discover that something has gone wrong.

7.6 DSpace and Fedora

This chapter concludes by looking at two open source digital library software systems: DSpace and Fedora. We met Fedora in Chapter 1, although its name was not mentioned: it powers the Pergamos digital library at the University of Athens (Section 1.3, Search for Sophocles). We met DSpace in Chapter 3 when looking at MIT's institutional repository (Section 3.6, Putting it all together). Here we describe these software systems in detail and discuss how they relate to the themes of this book, and in particular to the topics discussed in this chapter.

DSpace

In the year 2000, Hewlett-Packard teamed up with MIT's library to develop a system to meet the latter's need for an institutional repository. The result was called DSpace, for “Document space.” When the repository was launched in 2002, the software was simultaneously released as open source for others to use. Chapter 3 illustrated DSpace from the end-user's perspective for searching and browsing. We return to one of the documents viewed there, Kofi Annan's Master's thesis, and consider the steps required to enter it into the system.

Figure 7.6 shows snapshots from the document submission process. To have reached this point, the digital librarian (or end-user, since institutional repositories often allow registered users to submit documents) has first had to authenticate herself using a facility built into DSpace. Stages in the

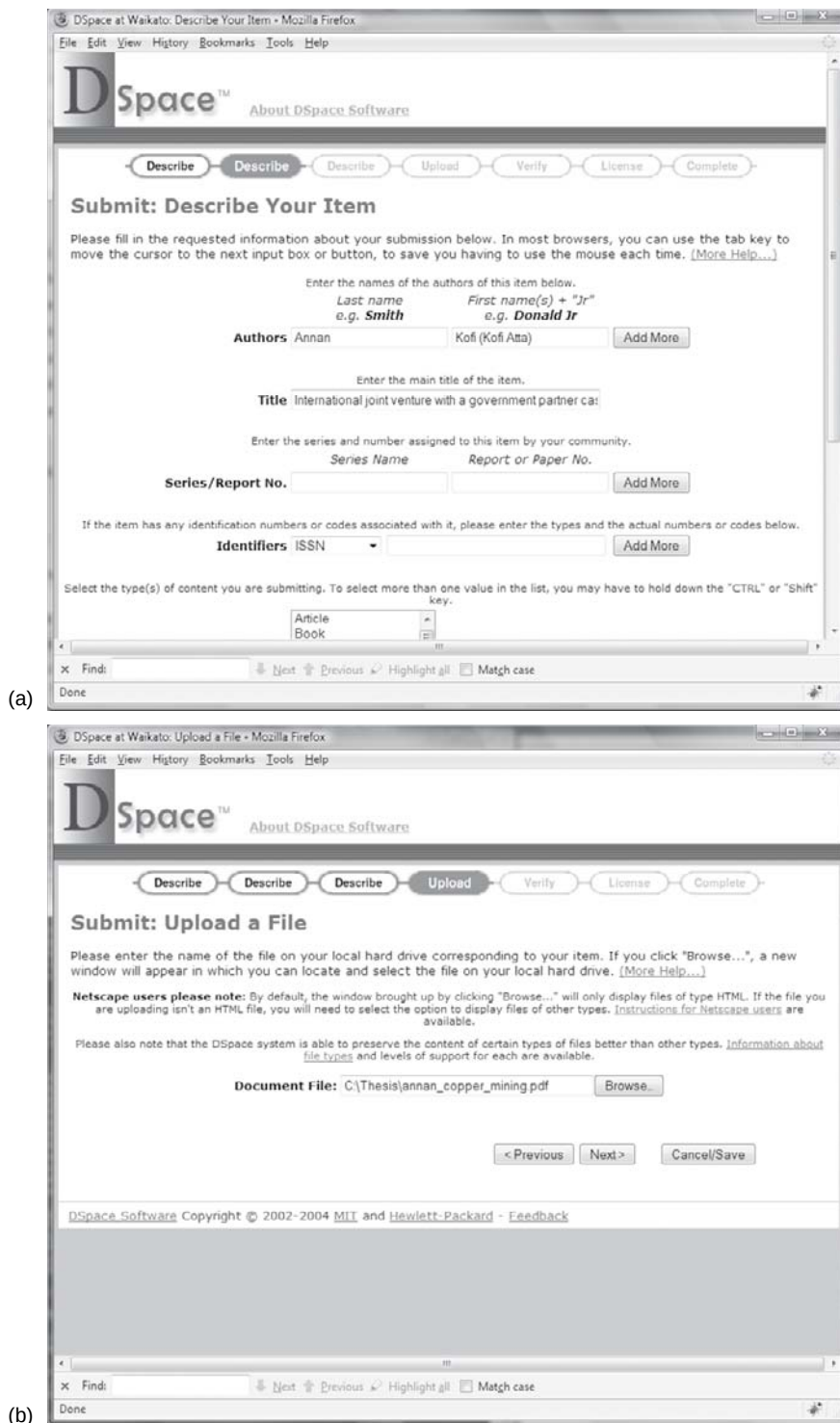


Figure 7.6: Submitting a document in DSpace: (a) entering author, title, and related metadata; (b) uploading the PDF version of the document;



(c)



(d)

Figure 7.6, cont'd: (c) checking the result of uploading; (d) verifying the metadata and file information

upload sequence are displayed along the top of each screenshot. The first three stages, labeled “describe,” gather metadata from the user. [Figure 7.6a](#) shows the second step, soliciting the title, author, and type of document (article, book, thesis, etc.).

Next, the librarian selects the file to upload. In [Figure 7.6b](#) she has pressed the browse button and located the file—in this case a PDF document—in a pop-up window. Pressing Next produces [Figure 7.6c](#) (after an uploading delay), where she can check the uploaded data—for example, its file size (roughly 6 MB in this case) and type. Clicking on the hyperlinked filename shows the uploaded version; through this page the librarian can also rectify errors, such as uploading the wrong file or its type being incorrectly assigned. The *Show checksums* button below computes the MD5 checksum of the uploaded file, which allows the librarian an opportunity to verify the integrity of the bytes that have been transferred. To do this, the librarian computes the checksum for the original file on her computer and then checks that the value she arrived at matches the value displayed in DSpace.

[Figure 7.6d](#), the next page, shows a summary of the information entered (and derived). Each section has a button that allows the librarian to edit the information. An additional step (not shown) displays the license that is to be used and requires the librarian to confirm that permission is granted, whereupon the item is finally submitted to DSpace and (ultimately) can be viewed as illustrated in Chapter 3 ([Figure 3.22h](#)). The exact procedure depends on how DSpace is configured. Often there is a further round of checking and quality control before the item is published in the repository. If the librarian leaves the submission sequence partway through, the information she has entered so far is stored under her DSpace username and the next time she logs in she can pick up the process where she left off.

The metadata values entered in DSpace are matched to Dublin Core elements, and the software includes an OAI-PMH server. Because it’s written in Java, it is Unicode compliant, and the interface has been translated into different languages. DSpace is servlet-based and makes extensive use of JavaServer Pages (JSP). Its developers recommend that it be used in conjunction with the Tomcat Web server, using PostgreSQL or Oracle as the database management system.

DSpace treats documents as black boxes to which metadata is attached. It is therefore important to identify each document’s MIME type correctly, because this dictates which helper application a Web browser will use to view it when it is opened. However, DSpace does recognize Word, PDF, HTML, and plain text documents and can be configured to support full-text indexing using the open source indexing package Lucene. Documents can be submitted by librarians or registered users individually over the Web or may be ingested as a batch by command-line scripts that run on the DSpace server.

Fedora

Fedora (Flexible Extensible Digital Object Repository Architecture) began in 1997 at Cornell University as a conceptual design backed up by a reference implementation. In 1999, the University of Virginia library used this reference implementation as a foundation for developing a digital library tailored to their needs. The two groups (note, incidentally, another pairing of a technology research group with a large-scale library, as for DSpace) ended up working together to produce an open source digital library toolkit for others to use.

Fedora is based on a powerful digital object model, encapsulated as a repository that is extremely flexible and configurable. Moreover, a repository stores all kinds of objects, not just the documents placed in it for presentation to the end-user (we shall see examples shortly). In contrast to DSpace, Fedora is not ready to use out of the box, a deliberate design decision by its developers that means Fedora can be used to develop a wide variety of digital libraries with radically different functionality. However, there is a price to be paid: you cannot use Fedora without programming support by IT staff.

Figure 7.7 shows the default Web interface for a repository that contains a collection of historic maps. This interface is not intended for end-users, but rather to indicate the possibilities and to allow developers to explore the system's capabilities. In Figure 7.7a, the user has called up the search page and sought documents that include the word *Africa* (searching across all indexed fields). The array of check-boxes determine which fields are displayed in the result list, in this case the document's identifier (persistent identifier, or pid) and title. In Figure 7.7b, the user has clicked on the third item, *Africae tabula nova*, and is viewing its associated information.

A *datastream* in Fedora is MIME-encoded data associated with an object: for instance, a source file, such as an image or PDF file, or XML data. *Disseminators* are methods that act upon an object, and they are linked to Web Services that provide dynamic capabilities that access the object's datastreams. Figure 7.7b, called the Item Index view, is the result of running the default disseminator, which lists the object's datastreams. Datastream names are hyperlinked, and clicking them serves up the underlying MIME-encoded data source. For instance, clicking DC yields an XML datastream representing the object's Dublin Core metadata. In this example we would see that <dc:Subject> is set to *Africa*, which explains why the item was found despite the fact that its title (*Africae*) is a Latin derivative of the stem (*Africa*). Clicking on the *url* datastream brings up Figure 7.7c, which displays a high-resolution version of the digitized image.

Figure 7.7d shows the disseminators associated with the object. This image is associated with the content model UVA_STD_IMAGE, which is why these disseminators appear. This content model, provided by Fedora, supports a wide range of image manipulation operations—resizing, zooming, brightening, watermarking, grayscale conversion, cropping, and so forth—through the RELS-EXT datastream (listed in Figure 7.7c). The default Web interface lists the disseminators along with input components (text boxes, radio buttons, and the like) that match the disseminators' parameters. To crop the image, for example, the user enters the *x*, *y*, *width*, and *height* parameters and presses the *run* button to generate a cropped excerpt.

Ingesting, modifying, and presenting information from this rich repository of objects is accomplished through a set of Web Services. The services fall into two core groups, one for access and the other for management, and both RESTful and full SOAP-based versions of the services (Section 7.4) are provided. A further protocol exists for expressing relationships between objects in terms of the Resource Description Framework (RDF, Section 6.4). These protocols have allowed a variety of digital libraries to be developed, including Pergamos (Section 1.3) and the U.S. National Science Digital Library mentioned in Section 7.2, which contains over 4 million items.

The capabilities provided by these protocols can be investigated using the Fedora Administration application illustrated in Figure 7.7e. The Administration application is another aspect of Fedora that

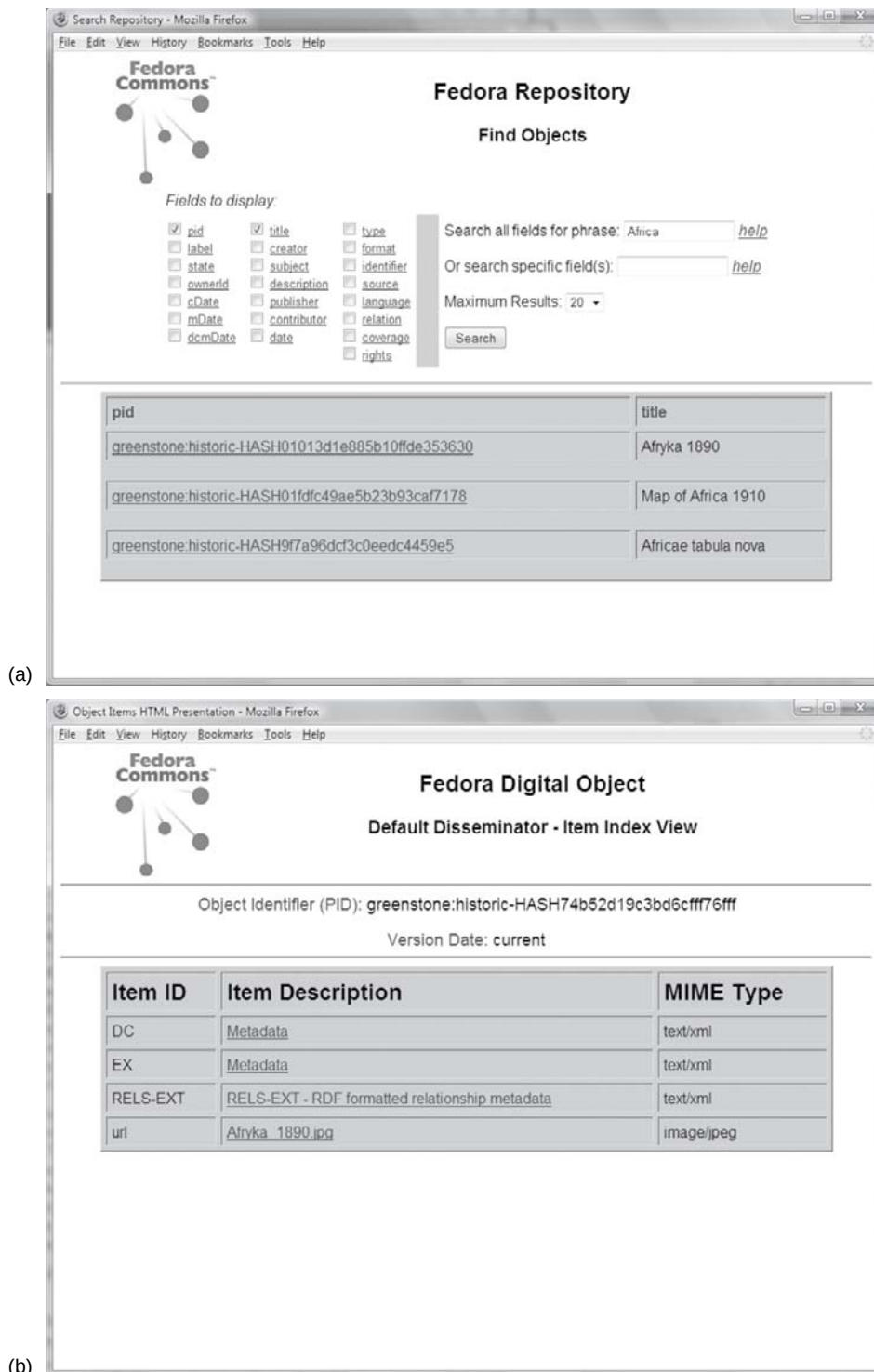


Figure 7.7: Fedora: (a) searching for metadata fields that include the word *Africa*; (b) viewing the item index for *Africae tabula nova*;

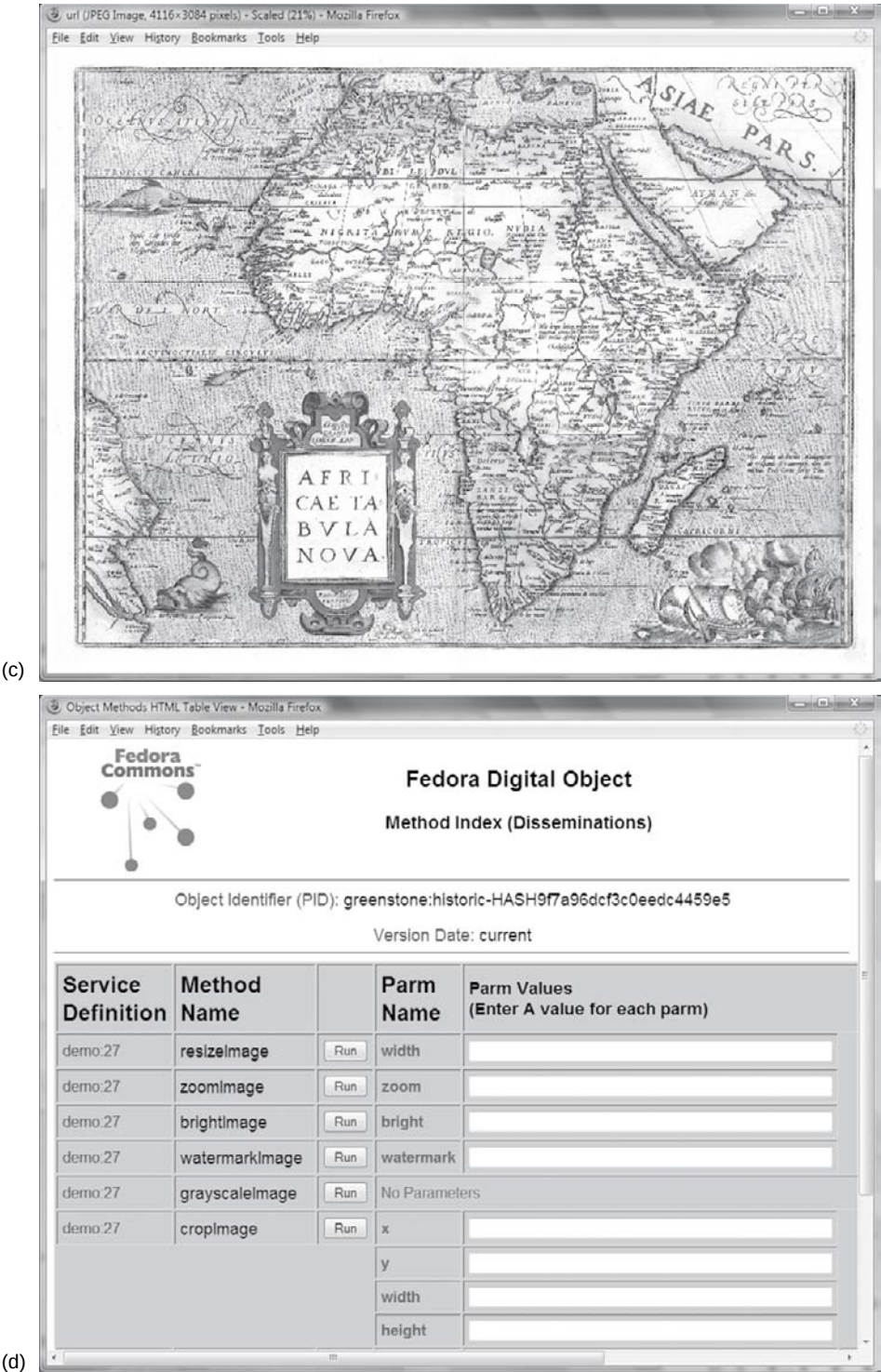


Figure 7.7, cont'd: (c) viewing the *url* datastream; (d) viewing the disseminators;

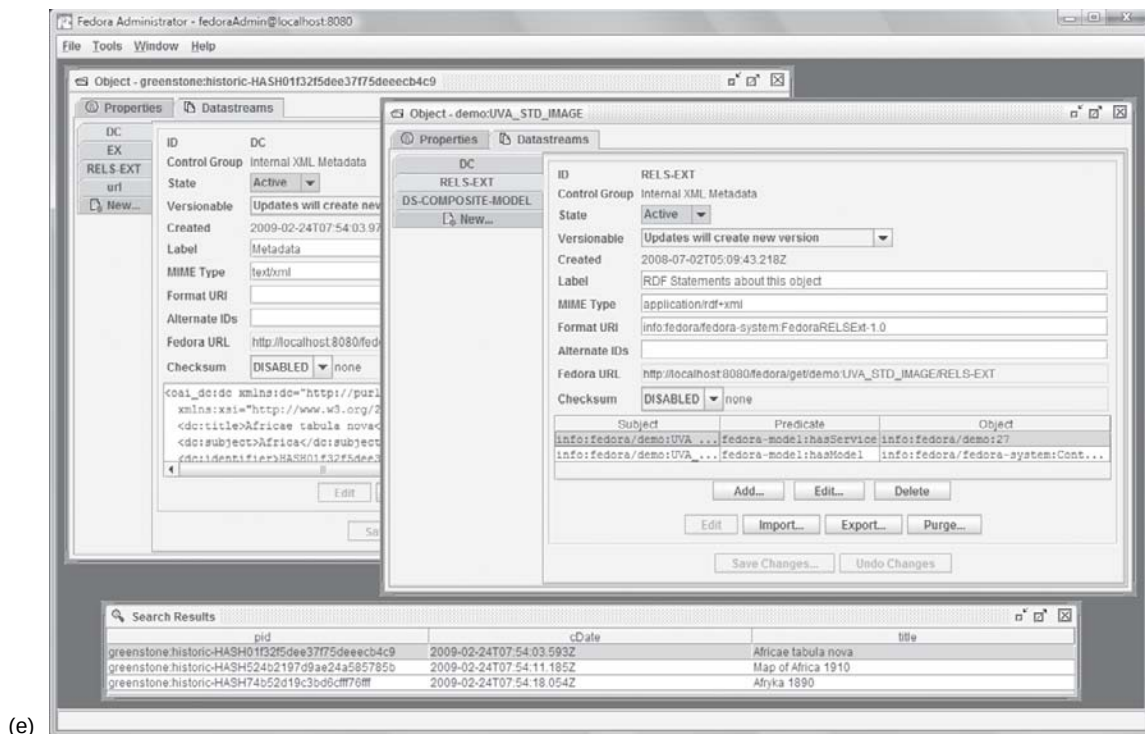


Figure 7.7, cont'd: (e) Fedora Administration tool

is targeted toward IT developers rather than end-users or digital librarians. After they log in, users are presented with a blank virtual desktop upon which windows appear as they interact. For example, new objects can be created and existing ones can be edited or purged from the repository. The repository can be searched, and items can be selected and manipulated. Sets of files can be ingested or modified; they are streamed to the Fedora server using the protocol. Files must be in either FOXML, a native Fedora XML format; Fedora METS, a METS extension defined by the project; or the Atom Syndication Format, an XML extension used for Web feeds. Authentication is provided by the Extensible Access Control Markup Language. The Administration application also gives access to consoles that connect directly with Fedora's management and access facilities.

In Figure 7.7e the user has repeated the query for *Africa*: the result list is displayed near the bottom. Double-clicking the first item produces the partially obscured window at the top left, through which datastreams and disseminators can be viewed and altered. Here the DC datastream is selected, which presents XML metadata that can be edited directly within the text box or using an external XML editor. Exploring the datastreams for this object, the user views the RELS-EXT datastream (not shown) and finds that the associated content model is UVA_STD_IMAGE, which means that disseminators for cropping, watermarking, and so on are available for this image as well.

Fedora's Content Model Architecture is a way of establishing properties that are shared by a set of digital objects. It is built upon Service Definitions and Service Deployments, which, like the content

model itself, are merely additional objects stored in the repository. The former provide abstract methods; the latter, concrete implementations. For example, a Service Definition tailored for images might provide a method for generating thumbnails at runtime suitable for previewing in a Web browser. Further definitions would be provided to handle different image types: perhaps one for JPEG, GIF, and PNG formats (since Java provides standard support for these) and a separate one for JPEG 2000 files, which require additional library support and may not mesh well with the built-in ones. Since the mechanism is accessed through Web Services, it might be implemented in an entirely different programming language. Regardless of which mechanism is used, the fact that images can have associated thumbnails can be included in the content model and shared by a variety of source documents.

Seeking to understand more about the content model associated with the map, our user calls up the UVA_STD_IMAGE content model object, displayed on the right of [Figure 7.7e](#). The RELS-EXT datastream in this object shows the RDF statements that are currently assigned, such as the *hasService* predicate. Through the interface, additional statements can be added or existing ones can be edited or deleted. Alternatively, the displayed identifier information can be used as the source of new retrieval requests to continue the user's exploration of the repository.

Like DSpace, Fedora is written in Java and makes use of servlets. It ships with the McKoi relational database, a light-weight Java implementation, but can be configured to use other database management systems. Full-text indexing is achieved through a third-party package (GSearch) into which the Lucene, Solr (built on top of Lucene), or Zebra indexing packages can be plugged.

Because Fedora is not a turnkey software solution, additional development is necessary to shape it into a usable end product. However, following the open source philosophy, such modifications can be packaged up and made available to others in the form of ready-to-run software tools. Fez is a Web interface to Fedora that provides an institutional repository. It uses PHP and MySQL and includes notes for users migrating from DSpace or other repository systems. Muradora provides a Web front-end to a Fedora repository that re-factors authentication and authorization into pluggable middleware components. It provides a Shibboleth authentication module ([Section 7.5](#)) and also extends the above-mentioned Extensible Access Control Markup Language; in addition, it has rich user-oriented search and browse capabilities. For many of these systems, installation is quite complex. For example, in addition to a Fedora installation, Muradora requires software support for LDAP, DB XML, and XForms.

7.7 Notes and Sources

The home page for Z39.50 is located at the Library of Congress Web site, www.loc.gov/z3950/ agency. An active group of online developers and implementers known as ZIG (for Z39.50 Implementers' Group) maintain an e-mail discussion list and hold regular meetings.

Cole and Foulonneau (2007) give a very readable yet detailed treatment of the Open Archives Initiative Protocol for Metadata Handling (OAI-PMH). Further information on the Open Archives Initiative can be found in an article by Lagoze and Van de Sompel (2001) and on the Web site www.openarchives.org. The National Science Digital Library can be found at <http://nsdl.org>; the

quote near the end of [Section 7.2](#) about using OAI-PMH to harvest its metadata is from Lagoze et al. (2006).

The quotation at the beginning of [Section 7.4](#) on the importance of services to institutional repositories is from Chavez et al. (2007) and applies to digital libraries in general. The list of potential Web Services in [Section 7.4](#) is from the same source. Gurugé (2004) provides a good overview of the Web Services field. The Search/Retrieval via URL service is fully documented at <http://www.loc.gov/standards/sru>; Morgan (2004) provides a nice introduction. Sanderson et al. (2005) compare and contrast SRU and OAI and discuss how they can work together.

For more on authentication, see Smith's (2002) *Authentication: From Passwords to Public Keys*. Restricting access to a particular network is really a restriction on the networking configuration of your computer. Desktop computers are usually locked in rooms that require a key (something you have) for physical access, while laptops are often operated within a Virtual Private Network configured with another username and password. Stamp (2006) provides detailed coverage of both the theory and practice of information security, including cryptographic theory and real world protocols. OpenID is at <http://openid.net> and Shibboleth is at <http://shibboleth.internet2.edu>. Powell and Recordon (2007) provide an outline of OpenID and compare it to Shibboleth. SAML is described at <http://saml.xml.org>.

Further details about DSpace can be found in Smith et al. (2003). The original paper describing Fedora's architecture is by Payette and Lagoze (1998), with more recent developments summarized in Lagoze et al. (2006). Kortekaas (2007) gives details of Fez, while Muradora can be found at www.muradora.org. Going to press, a new open source repository initiative was announced, DuraSpace. This project sees the teams behind Fedora and DSpace join forces, with a coordinated strategy for growing the areas in which open repositories are used, while retaining their commitment to already established communities. More details can be found at www.duraspace.org. The historical map of Africa depicted in the Fedora example ([Figure 7.7](#)) was penned by the renowned Belgian cartographer Abraham Ortelius and was first published in 1570 as part of his *Theatrum Orbis Terrarum* (Theatre of the World), considered by many to be the first modern atlas.

Internationalization: The global challenge

Internationalization is a central component of the vision of digital libraries that motivates this book. It is all too easy from the native English speaker's perspective (which includes most of our readers and all three of this book's authors), to sweep under the carpet the many challenging problems of digitally representing and working with text in other languages.

Of course, we cannot possibly hope to cover all the issues raised by all the world's languages—and in a general text like this one, it would be inappropriate to attempt to do so. Instead, this chapter highlights some issues that are involved when making digital libraries available in languages other than English. We prefer to tackle a few specific issues in detail, rather than to give a general high-level sketch of what needs to be done for the world in general.

We have already introduced a foreign-language collection: the Pergamos system described in Section 1.3 has a user interface in Greek, English, and French; we set the stage for this chapter by showing some further interfaces and collections in other languages. Then we give a comprehensive account of Unicode. This standard aims to represent all the characters used in all the world's languages. We discuss it briefly in Section 4.1, but to work globally you need to know much more. As mentioned in Chapter 4, the same character can have multiple representations. To understand the complications this causes, imagine searching for an accented word like *détente* in French text, where the non-ASCII character *é* is sometimes represented as a single character in extended ASCII, sometimes in regular 7-bit ASCII as *e* followed by a backspace followed by *'*, and sometimes by the HTML incantation *é*.

Prior to the establishment of Unicode, ASCII had been extended in many ad hoc ways, including a standard for Hindi and related languages called ISCII, where the initial *I* stands for “Indian.” It turns out that Hindi and other Indic languages present many interesting problems for character coding that have not yet been entirely solved in Unicode-compliant applications. These languages raise subtle problems that are difficult for people of a European background to appreciate, and [Section 8.3](#) gives a glimpse of the complexities involved. We also discuss the development of ISCII and practical problems with the adoption of Unicode for such languages.

Finally, we look at some issues with Chinese and other Asian languages: the so-called CJK (Chinese-Japanese-Korean) family of languages. As discussed in Chapter 4, languages like Chinese are written without using any spaces or other word delimiters (except for punctuation marks)—indeed, the Western notion of a *word boundary* is literally alien. Nevertheless, the CJK languages do contain words. Most Chinese words comprise several characters: two, three, or four. Five-character words also exist, but they are rare. Many characters can stand alone as words in themselves, while on other occasions the same character is the first or second component of a two-character word, and on still others it participates as a component of a three- or four-character word.

Another issue is that CJK languages are not alphabetic, and—although it will seem surprising to Western readers—there is no single universally used way of ordering text strings analogous to alphabetic ordering in European languages. This makes the creation of browsing lists in digital libraries, for example, problematic. These are international problems, and they call for international solutions. Digital library systems must be globalized, not created by a single group of specialists.

8.1 Multilingual Interfaces and Documents

Figure 8.1 is from a collection of historical New Zealand Māori newspapers and shows the same page in image and text form. As you can see, the tabular information at the top of the page (just beneath the masthead and issue details), which represents sums of money (although it is difficult to make out), is missing from the text version. The text version is what supports searching, so this information is invisible to searches. On the other hand, the word *Rotorua*, which was the search term for this page, is highlighted in the text but not in the page image. The collection provides both page images and text because of the advantage of being able to locate search terms in the text. Readers can choose which to view and can move quickly from one to the other. A magnified version of page images is also available.

The Māori newspapers record fascinating historical information that is useful from many points of view. The newspapers were published from 1842 to 1933, a formative period in the development of New Zealand, which, being so far from Europe, was colonized quite late. Far-reaching political developments took place during these years, and the collection is a significant resource for historians. It is also an interesting corpus for linguists, because changes in the Māori language were still taking place and can be tracked through the newspapers.

The collection contains 40 different newspaper titles that were published during the period. They are mostly written in Māori, although some are in English and a few have parallel translations. There are a total of 12,000 images, varying from A4 to double-page tabloid spreads—these images contain a total of around 20,000 newspaper pages. The images had previously been collected on microfiche. They are in a variety of conditions: some are crisp, others are yellowish, and still others are badly water-stained.

The enormous effort of constructing the microfiche was undertaken to make this national treasure available to scholars. However, the material was entirely unindexed. Although a research project is producing manual summaries of the articles, these will take many years to complete. The digital library collection completely transforms access to the material. First, the documents are available on the Web, which makes them readily and universally accessible. Second, they can be searched, which makes almost any kind of research immeasurably easier. Third, the collection does not require scholarly research skills to use, allowing a wider group of people to discover things that they would never otherwise know about their heritage, ancestry, or home town. This collection is the largest single online Māori resource in the country.

Figure 8.2 shows digital libraries with interfaces in French and Portuguese. The French illustration is from a UNESCO collection called Sahel Point Doc that contains information about the Sahel region of Sub-Saharan Africa. Everything in this collection is in French: all target documents, the entire user

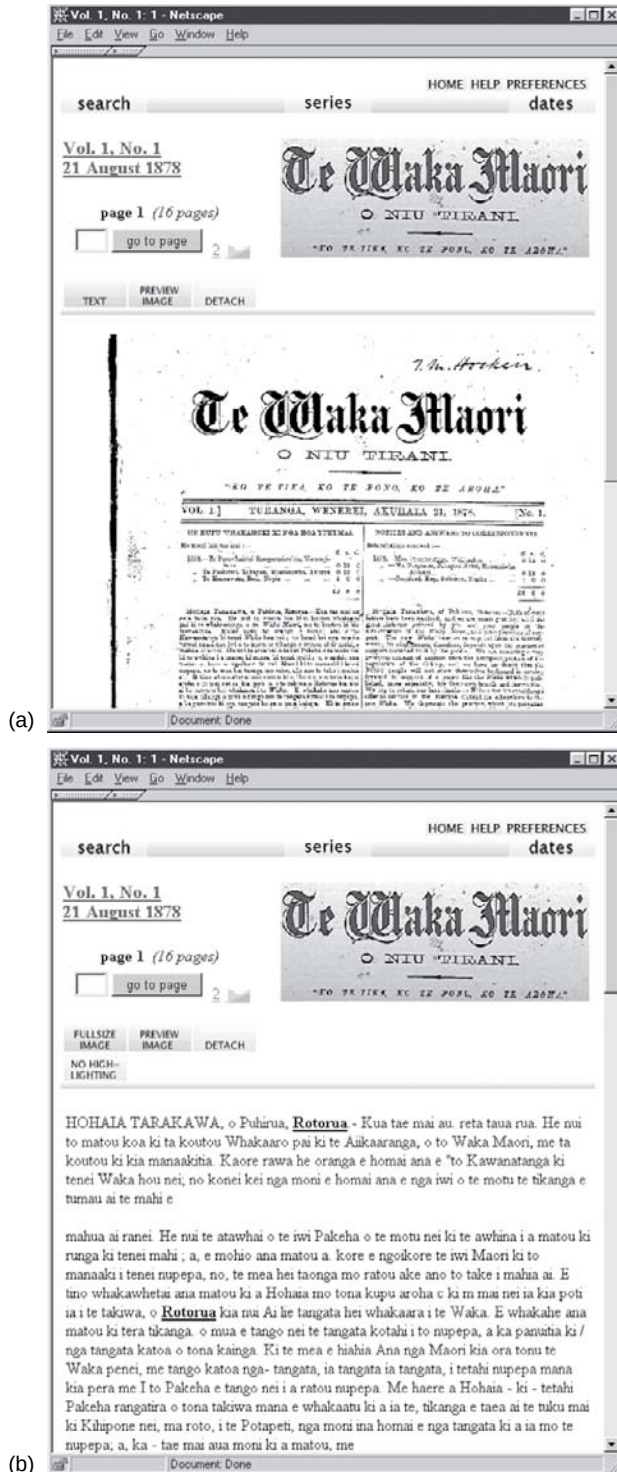


Figure 8.1: An historic Māori newspaper: (a) page image; (b) extracted text

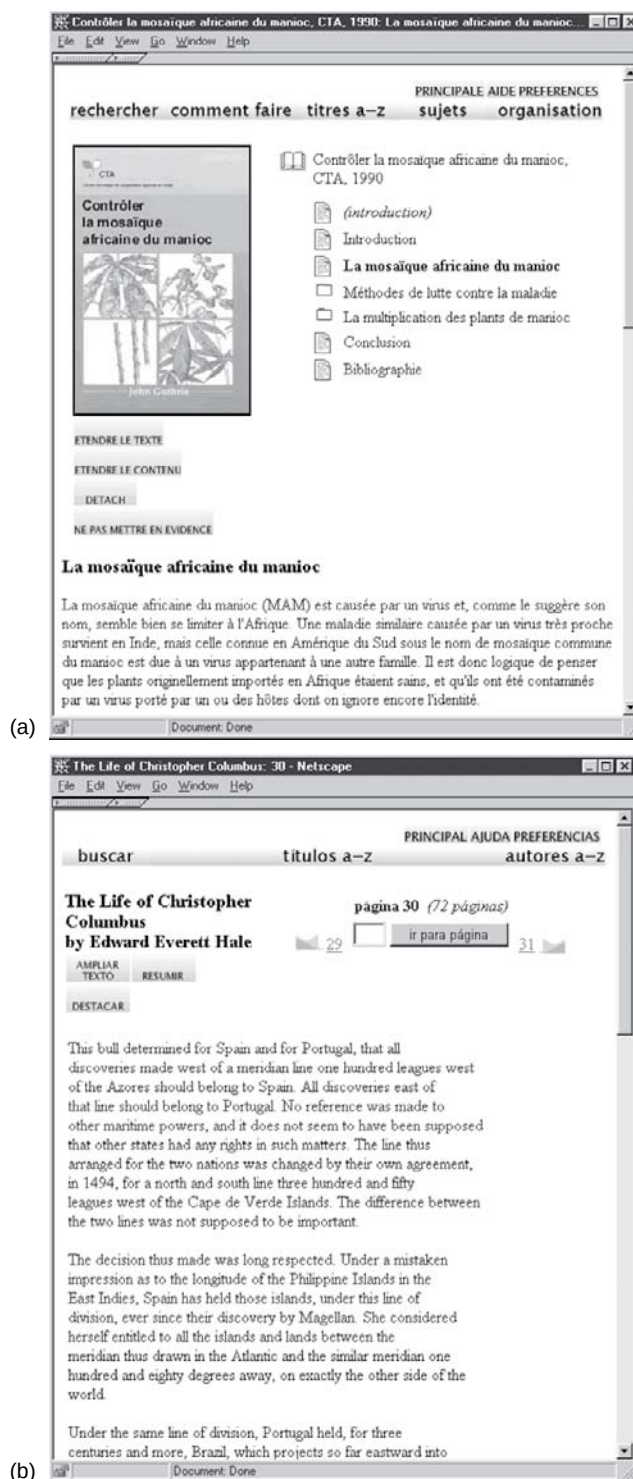


Figure 8.2: Foreign-language collections: (a) French (*Contrôler la mosaïque africaine du manioc*, by J. Guthrie, CTA, 1990); (b) Portuguese interface to an English collection (*The Life of Christopher Columbus*, by Edward Everett Hale, Electric Umbrella Publishing)

interface, and the help text. [Figure 8.2b](#) shows a Portuguese interface to an English-language collection—the same Gutenberg collection that we examined earlier. Again, the entire user interface (and help text) has been translated into Portuguese, but in this case the target documents are in English. In fact, the user interface language is a user-selectable option on a Preferences page: you can instantaneously switch to languages like German, Dutch, Spanish, and Māori, too.

[Figure 8.3](#) shows documents from two different Chinese collections. The first comes from a collection of rubbings of Tang poetry—not unlike the stone steles described in Chapter 1, the world’s oldest surviving library. These documents are images, not machine-readable text. There are machine-readable versions of each document—although in this case they were entered manually rather than by OCR—but the user never sees them: they are used only for searching.

The second Chinese example is from a small collection of classic literature. Here, books are represented in the same way as they are in the Humanity Development Library (see Section 3.1), with chapters shown by folders. The work illustrated here is *The Plum in the Golden Vase*, an anonymous early 17th-century satirical novel that recounts the domestic life of a corrupt merchant with six wives and concubines who slowly destroys himself with conspicuous consumption, political imbroglios, and sexual escapades. One of the three most famous Ming Dynasty novels, it reflects the debaucheries of society at the time—and is still banned in China. The literature collection is textual, so matching search terms are highlighted in boldface. Boldface characters (and italics) are used in Chinese just as they are in Western languages.

It’s easy to display documents in any language within Web browsers. In the early days you had to download a special plug-in for the character set being used, but today’s browsers incorporate support for many languages. [Figure 8.4](#) shows pages from an Arabic collection of information on famous mosques, displayed using an ordinary browser.

As [Figures 8.3a](#) and [8.4b](#) imply, the text of both the Chinese and Arabic collections (as well as text in all other languages) is fully searchable. Again, the browser does the hard part, facilitating character entry and ensuring that Arabic text is composed from right to left, not from left to right as in other languages. To enter ideographic languages like Chinese, which go beyond the normal keyboard, you need special software. All documents in this digital library system are represented internally in Unicode and the system converts between this and the representation supported by the browser (which can differ from one browser to another). The next section discusses the complexities of different character sets, and the approach Unicode uses to bring them together.

8.2 Unicode

The Unicode standard is massive. It has two parts (ISO 10646-1 and ISO 10646-2) and specifies a total of 94,000 characters. The first part focuses on commonly used living languages and is called (for reasons to be explained shortly) the *Basic Multilingual Plane*. It weighs in at 1,000 pages and contains 49,000 characters.

[Table 8.1](#) breaks down the ISO 10646-1 code space into different scripts, showing how many codes are allocated to each (unassigned codes are shown in parentheses). Unicode’s scope is impressive: it

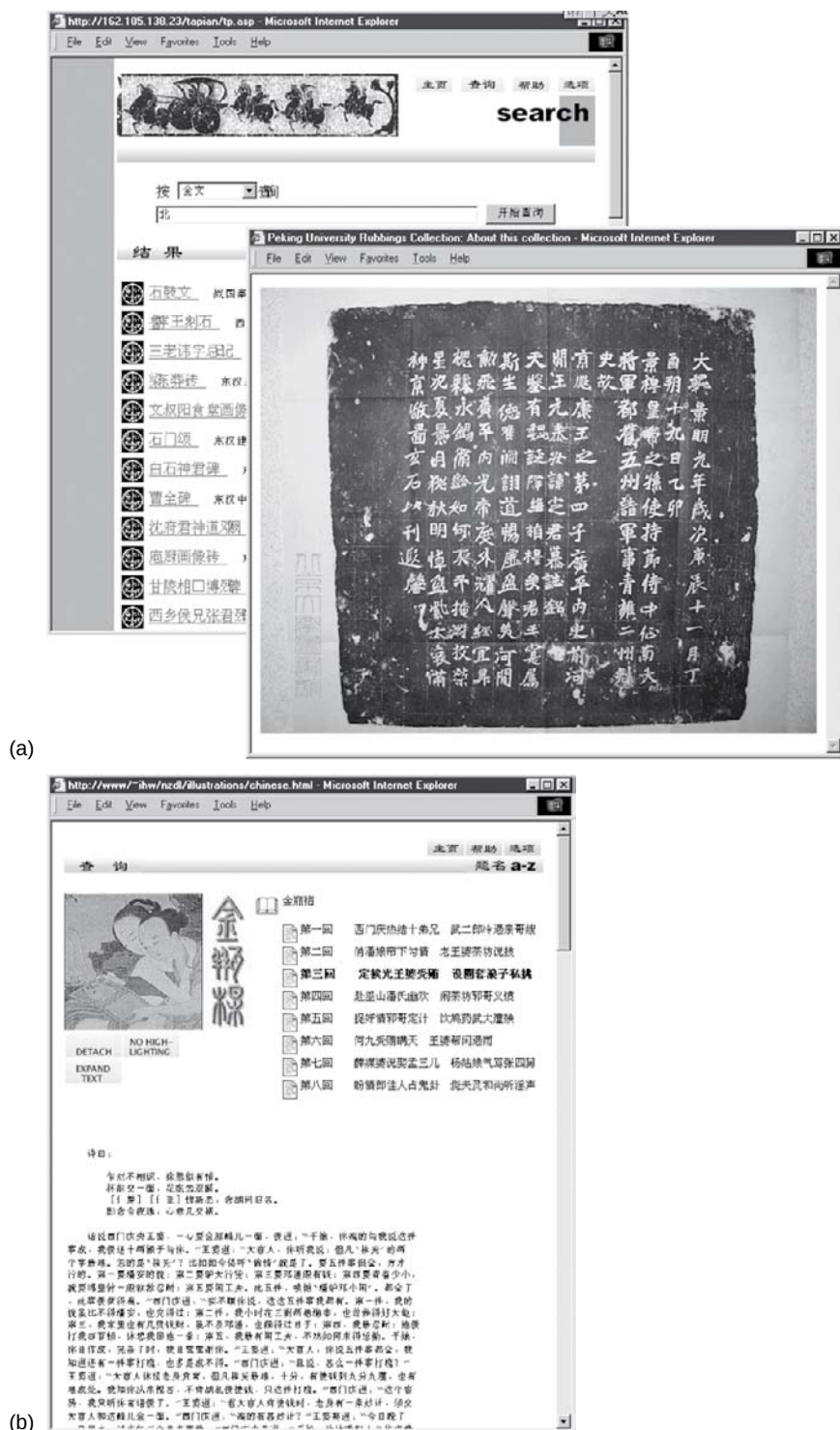


Figure 8.3: Documents from two Chinese collections: (a) rubbings of Tang poetry; (b) classic literature



Figure 8.4: An Arabic collection: (a) a document; (b) searching

Table 8.1: Unicode Part 1: The basic multilingual plane

Zone	Area	Code	Script	Number of codes
Alphabetic	General scripts	0000	Basic Latin (US-ASCII)	128
		0080	Latin-1 (ISO-8859-1)	128
		0100	Latin Extended	336
		0250	IPA Extensions	96
		02B0	Spacing Modifier Letters	80
		0300	Combining Diacritical Marks	112
		0370	Greek	144
		0400	Cyrillic	256
		—	—	(48)
		0530	Armenian	96
		0590	Hebrew	112
		0600	Arabic	256
		0700	Syriac	78
		—	—	(50)
		0780	Thaana	50
		—	—	(334)
		0900	ISCII Indic Scripts	
		0900	Devanagari	128
		0980	Bengali	128
		0A00	Gurmukhi	128
		0A80	Gujarati	128
		0B00	Oriya	128
		0B80	Tamil	128
		0C00	Telugu	128
		0C80	Kannada	128
		0D00	Malayalam	128
		0D80	Sinhalese	128
		0E00	Thai	128
		0E80	Lao	128
		0F00	Tibetan	192
		—	—	(64)
		1000	Mongolian	160
		10A0	Georgian	96
		1100	Hangul Jamo	256
		1200	Ethiopic	384
		—	—	(32)
		13A0	Cherokee	96
		1400	Canadian Syllabics	640
		1680	Ogham	32
		16A0	Runic	96
		1700	Burmese	90
		—	—	(38)
		1780	Khmer	106
		—	—	(1558)

Table 8.1: Unicode Part 1: The basic multilingual plane—cont'd

Zone	Area	Code	Script	Number of codes
Ideographic	Symbols	1E00	Latin Extended Additional	256
		1F00	Greek Extended	256
		2000	General Punctuation	112
		2070	Superscripts and Subscripts	48
		20A0	Currency Symbols	48
		20D0	Combining Marks for Symbols	48
		2100	Letterlike Symbols	80
		2150	Number Forms	64
		2190	Arrows	112
		2200	Mathematical Operators	256
		2300	Miscellaneous Technical	256
		2400	Control Pictures	64
		2440	Optical Character Recognition	32
		2460	Enclosed Alphanumerics	160
		2500	Box Drawing	128
		2580	Block Elements	32
		25A0	Geometric Shapes	96
		2600	Miscellaneous Symbols	256
		2700	Dingbats	192
		—	—	(64)
		—	Braille Pattern Symbols	256
		—	—	(1536)
	CJK phonetics and symbols	2F00	KangXi radicals	214
		—	—	(42)
		3000	CJK Symbols and Punctuation	64
		3040	Hiragana	96
		30A0	Katakana	96
		3100	Bopomofo	48
		3130	Hangul Compatibility Jamo	96
		3190	Kanbun	16
		—	—	(96)
		3200	Enclosed CJK Letters and Months	256
		3300	CJK Compatibility	256
		3400	CJK Unified Ideographs, Extension A	6656
		4E00	CJK Unified Ideographs	20902
	Other	—	—	(90)
		A000	Yi	1225
		—	—	(1847)

Continued

Table 8.1: Unicode Part 1: The basic multilingual plane—cont’d

Zone	Area	Code	Script	Number of codes
Reserved	Private use Compatibility and specials	AC00	Hangul symbols	11172
			—	(92)
		D800	High Surrogates	1024
		DC00	Low Surrogates	1024
		E000		6400
		F900	CJK Compatibility Ideographs	512
		FB00	Alphabetic Presentation Forms	80
		FB50	Arabic Presentation Forms-A	688
			—	(32)
		FE20	Combining Half Marks	16
		FE30	CJK Compatibility Forms	32
		FE50	Small Form Variants	32
		FE70	Arabic Presentation Forms-B	144
		FF00	Halfwidth and Fullwidth Forms	240
		FFF0	Specials	16

covers Western and Middle Eastern languages, such as Latin, Greek, Cyrillic, Hebrew, and Arabic; Chinese, Japanese, and Korean Hangul ideographs; as well as other scripts, such as Bengali, Thai, and Ethiopic—to name just a few. Also included are Braille, mathematical symbols, and a host of other shapes.

[Table 8.1](#) divides the Unicode code space into five zones: alphabetic scripts, ideographic scripts, other characters, surrogates, and reserved codes. Falling within the first zone are the broad areas of general scripts, symbols, and CJK (Chinese-Japanese-Korean) phonetics and symbols. The reserved blocks are intended to help Unicode’s designers respond to unforeseen circumstances.

Unicode distinguishes characters by script, and those pertaining to a distinct script are blocked together in contiguous numeric sequences: Greek, Cyrillic, Hebrew, and so on. However, characters are not distinguished by language. For example, the excerpt from the Unicode standard in [Figure 8.5](#) shows the Basic Latin (or standard ASCII) and Latin-1 Supplement (an ASCII extension) characters, which are used for most European languages. Capital A in French is the same character as in English. Only the accented letters that are used in European languages are included in the Latin-1 Supplement; the basic character forms are not redefined.

Punctuation is also shared among different scripts. The ASCII period (i.e., full stop) in [Figure 8.5](#) is used in Greek and Cyrillic text too. However, periods in languages like Armenian, Arabic, Ethiopic, Chinese, and Korean are shaped differently and have their own Unicode representations. The multi-function ASCII hyphen is retained for the purpose of round-trip compatibility (see [Section 4.1](#)), but

000	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
010	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
020	SP	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
030	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
040	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
050	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
060	`	A	b	c	d	e	f	g	h	i	j	k	l	m	n	o
070	p	Q	r	s	T	u	v	w	x	y	z	{		}	~	DEL
080	XXX	XXX	BPH	NBH	XXX	NEL	SSA	ESA	HTS	HTJ	VTS	PLD	PLU	R	SS2	SS3
090	DCS	PU1	PU2	STS	CC1	MM	SPA	EPA	SOS	XXX	SC	CSI	ST	OSC	PM	APC
0a0	NB/SP	i	ç	£	¤	¥	¦	§	¨	©	ª	«	¬	SHY	®	—
0b0	°	±	²	³	´	µ	¶	·	,	¹	º	»	¼	½	¾	¿
0c0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
0d0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
0e0	à	Á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
0f0	ð	Ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Figure 8.5: Unicode excerpt: Basic Latin and Latin-1 Supplement (U+0000–U+00FF)

new codes are defined under “general punctuation” (codes 2000 and up in Table 8.1) to distinguish among the hyphen (-), en-dash (–), and em-dash (—). The minus sign, which is not usually distinguished typographically from the en-dash, has its own symbol as a “mathematical operator” (codes 2200 and up in Table 8.1).

Unicode distinguishes letters from different scripts even though they may look identical. For example, the Greek capital alpha looks just like the Roman capital A, but capital alpha receives its own code in the Greek block (codes 0370 and up in Table 8.1). This allows you to lower-case capital alpha to α , and capital A to *a*, without worrying about exceptions.

The characters at the core of Unicode are called the *universal character set*, and the standard is fundamentally a suite of lookup tables that specify which character is displayed for a given numeric code. What are all these characters, and what do they look like? Every one tells a story: its historical origin, how it changed along the way, the languages it is used in, how it relates to other characters, and how it can be transliterated or transcribed. We cannot tell these stories here: you will have to refer to other sources.

As an example of diversity, Figure 8.6a shows some of the Extended Latin characters, while Figure 8.6b shows part of the Cyrillic section. As you can see, some Cyrillic letters duplicate identical-looking Latin equivalents.

Composite and combining characters

In ordinary usage, the word *character* refers to various things: a letter of an alphabet, a particular mark on a page, a symbol in a certain language, and so on. In Unicode, the term refers to the abstract form of a letter, but in a broad sense, so that it can encompass the fabulous diversity of the world’s writing systems. More precise terminology is employed to cover particular forms of use.

100	Ā	Ā	Ā	ā	Ą	ą	Ć	ć	Ĉ	ĉ	Č	č	Ċ	ċ	Đ	đ
110	Ð	Ð	Ē	ē	Ė	ė	Ě	ě	Ɛ	ɛ	Ė	ė	Ğ	ğ	Ġ	ġ
120	Ĝ	Ĝ	Ĝ	ĝ	Ĥ	ĥ	Ħ	ħ	Ĩ	ĩ	Ĭ	ĭ	Į	į	İ	ı
130	Í	í	Ĳ	ĳ	Ĵ	ĵ	Ķ	ķ	Ľ	ĺ	Ł	ł	Ł	ł	ŀ	ŀ
140	Œ	œ	Œ	œ	Œ	œ	Œ	œ	Œ	œ	Œ	œ	Œ	œ	Œ	œ
150	Œ	œ	Œ	œ	Œ	œ	Œ	œ	Œ	œ	Œ	œ	Œ	œ	Œ	œ
160	Š	š	Ť	ť	Ŧ	ŧ	Ũ	ũ	Ū	ū	Ŭ	ŭ	Ů	ů	Ű	ű
(a) 170	Ű	ű	Ų	ų	Ŵ	ŵ	Ŷ	ŷ	Ž	ž	Ż	ż	Ž	ž	Ž	ž
400	Ё	ѐ	ѐ	ѐ	ѐ	ѐ	ѐ	ѐ	ѐ	ѐ	ѐ	ѐ	ѐ	ѐ	ѐ	ѐ
410	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
420	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
430	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
440	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
(b) 450	ё	ѐ	ѐ	ѐ	ѐ	ѐ	ѐ	ѐ	ѐ	ѐ	ѐ	ѐ	ѐ	ѐ	ѐ	ѐ

Figure 8.6: Unicode excerpts: (a) Latin Extended A (U+0100–U+017F); (b) Cyrillic (U+0400–U+045F)

A *glyph* refers to a particular rendition of a character (or composite character) on a page or screen. Different fonts create different glyphs. For example, the character “a” in 12-point Helvetica is one glyph; in 12-point Times it is another. Unicode does not distinguish between different glyphs. It treats characters as abstract members of linguistic scripts, not as graphic entities.

A *code point* is a Unicode value, specified by prefixing *U+* to the numeric value given in hexadecimal. LATIN CAPITAL LETTER G (as the Unicode description goes) is U+0047. A *code range* gives a range of values: for example, the characters corresponding to ASCII are located at U+0000–U+007F and are called Basic Latin.

A code point does not necessarily represent an individual character. Some code points correspond to more than one character. For example, the code point U+FB01, which is called LATIN SMALL LIGATURE FI, represents the sequence *f* followed by *i*, which in most printing is joined together into a single symbol that is technically called a *ligature*, *fi*. Other code points specify part of a character. For example, U+0308 is called COMBINING DIAERESIS, and—at least in normal language—is always accompanied by another symbol to form a single unit, such as *ü*, *ë*, or *ï*.

The diaeresis is an example of what Unicode calls a *combining character*. To produce the single unit *ü*, the Latin small letter *u* (which is U+0075) is directly followed by the code for the diaeresis (which is U+0308) to form the sequence U+0075 U+0308. This is the handwriting sequence: first draw the base letter, then place the accent. Combining characters occupy no space of their own: they share the space of the character they combine with. They also may alter the base character’s shape: when *i* is followed by the same combining diaeresis, the result looks like *ï*—omitting the original dot in the base letter. Drawing Unicode characters is not straightforward: it is necessary to be able to place accents over, under, and even through arbitrary characters—which may already be accented—and still produce acceptable spacing and appearance.

Because of the requirement to be round-trip compatible with existing character sets, Unicode can also have single code points that correspond to precisely the same units as character combinations—in

fact, we have seen examples of these in the Latin-1 supplement shown in [Figure 8.6a](#). This means that certain characters have more than one representation. For example, the middle letter of *naïve* could be represented using the combining character approach by U+0069 U+0308, or as a single, precomposed unit using the already prepared character U+00EF. Around 500 precomposed Latin letters, for instance, in the Unicode standard are superfluous, in that they can be represented using combining character sequences.

Combining characters allow many character shapes to be represented within a limited code range. They also help compensate for omissions. For example, the Guaraní Latin small *g* with tilde can be expressed without embarking upon a lengthy standardization process to add this previously overlooked character to Unicode. Combining characters are an important mechanism in intricate writing systems like Hangul, the Korean syllabic script. In Unicode, Hangul is covered by 11,172 precomposed symbols (codes AC00 and up in [Table 8.1](#)), each of which represents a syllable. Syllables are made up of *Jamo*, which is the Korean name for a single element of the Hangul script. Each Unicode Hangul syllable has an alternative representation as a composition of Jamo, and these are also represented in Unicode (codes 1100 and up in [Table 8.1](#)). The rules for combining Jamo are complex, however—which is why the precomposed symbols are included. But to type medieval Hangul even more combinations are needed, and these are not available in precomposed form.

The existence of composite and combining characters complicates the processing of Unicode text. When searching for a particular word, alternate forms must be considered. String comparison with regular expressions presents a knottier challenge. Even sorting text into lexicographic order becomes nontrivial. Algorithmically, all these problems can be reduced to comparing two strings of text—which is far easier if the text is represented in some kind of normalized form.

Unicode defines four normalized forms using two orthogonal notions: *canonical* and *compatibility* equivalence. Canonical equivalence relates code points to sequences of code points that produce the same character—as in the case discussed earlier, where the combination U+0069 U+0308 and the single precomposed character U+00EF both represent *i*. *Canonical composition* is the process of turning combining character sequences into their precomposed counterparts; *canonical decomposition* is the inverse.

Compatibility equivalence relates ligatures to their constituents, such as the ligature *fi* (U+FB01) and its components *f* (U+0066) and *i* (U+0069). Decomposing ligatures simplifies string comparison, but the new representation no longer maintains a one-to-one character-by-character mapping with the original encoding.

The four normalized forms defined by the Unicode standard are

- canonical decomposition
- canonical decomposition followed by canonical composition
- compatibility decomposition
- compatibility decomposition followed by canonical composition.

Certain Unicode characters are officially *deprecated*, which means that although they are present in the standard, they are not supposed to be used. This is how mistakes are dealt with. Once a character has been defined, it cannot be removed from the standard (because that would sacrifice backward compatibility); the solution is to deprecate it. For example, there are two different Unicode characters that generate the symbol Å, LATIN CAPITAL LETTER A WITH RING ABOVE (U+00C5) and ÅNGSTRÖM SIGN (U+212B); the latter is officially deprecated. Deprecated characters are avoided in all normalized forms.

Further complications, which we only mention in passing, are caused by directionality of writing. Hebrew and Arabic are written right to left, but when numbers or foreign words appear, they flow in the opposite direction; thus bidirectional processing may be required within each horizontal line. The Mongolian script can only be written in vertical rows. Another issue is the fact that complex scripts, such as Arabic and Indic scripts, include a plethora of ligatures, and contextual analysis is needed to select the correct glyph.

Because of the complexity of Unicode, particularly with regard to composite characters, three implementation levels are defined:

- Level 1 forms the base implementation, excluding combining characters and the Korean Hangul Jamo characters.
- Level 2 permits a fixed list of combining characters, adding, for example, the capability to express Hebrew, Arabic, Bengali, Tamil, Thai, and Lao.
- Level 3 is the full implementation.

Unicode character encodings

With future expansion in mind, the ISO standard formally specifies that Unicode characters are represented by 32 bits each. However, all characters so far envisaged fit into the first 21 bits. In fact, the Unicode consortium differs from the ISO standard by limiting the range of values prescribed to the 21-bit range U+000000–10FFFF. The discrepancy is of minor importance; we follow the Unicode route.

So-called planes of 65,536 characters are defined; there are 16 of them in the 21-bit address space. All the examples discussed above lie in the Basic Multilingual Plane, which represents the range U+0000–U+FFFF and contains virtually all characters used in living languages. The Supplementary Multilingual Plane, which ranges from U+10000–U+1FFFF, contains historic scripts, special alphabets designed for use in mathematics, and musical symbols. Next is the Supplementary Ideographic Plane (U+20000–U+2FFFF), which contains 40,000-odd additional Chinese ideographs that were used in ancient times but have fallen out of current use. The Supplementary Special-Purpose Plane (U+E0000–U+EFFFF), or Plane 14, contains a set of tag characters for language identification, to be used with special protocols.

Given the ISO 32-bit upper bound, the obvious encoding of Unicode uses 4-byte values for each character. This scheme is known as UTF-32. A complication arises from the different byte ordering that computers use to store integers: *big-endian* (where the 4 bytes in each word are ordered from

most significant to least significant) versus *little-endian* (where they are stored in the reverse order), and the standard includes a mechanism to disambiguate the two.

The Basic Multilingual Plane is a 16-bit representation, so a restricted version of Unicode can use 2-byte characters. In fact, a special escape mechanism called *surrogate characters* (explained below) is used to extend the basic 2-byte representation to accommodate the full 21 bits. This scheme is known as UTF-16. It too is complicated by the endian question, and resolved in the same way. For almost all text, the UTF-16 encoding requires half the space needed for UTF-32.

We mention UTF-8 in Chapter 4. It is a variant of Unicode that extends ASCII—which, as we have seen, is a 7-bit representation where each character occupies an individual byte—in a straightforward way. UTF-8 is a variable-length encoding scheme where the basic entity is a byte. Being byte-oriented, this scheme avoids the endian issue that complicates UTF-32 and UTF-16. We explain the UTF methods more fully in the following sections.

UTF-32

In [Figure 8.7](#) the word *Welcome* in five different languages has been converted to Unicode ([Figure 8.7a](#)) and encoded in the three different UTF methods ([Figure 8.7b](#)). UTF-32 maps the universal character set to 4-byte integers, and Unicode is turned into UTF-32 by dropping the U+ prefix.

Byte order is irrelevant when data is generated and handled internally in memory; it only becomes an issue when serializing information for transfer to a disk or transmission over a byte-oriented protocol, such as the hypertext transfer protocol (http). In the Unicode standard, the big-endian format is preferred, bytes being ordered from most significant to least significant—just as they are when writing out the number in hexadecimal, left to right. This is what is shown in [Figure 8.7b](#). However, two variants are defined: UTF-32BE and UTF-32LE for big-endian and little-endian, respectively.

		Unicode			
(a)	Welcome	(English)	U+0057	U+0065	U+006C U+0063 ...
	Haere mai	(Māori)	U+0048	U+0061	U+0065 U+0072 ...
	Wilkommen	(German)	U+0057	U+0069	U+006C U+006B ...
	Bienvenue	(French)	U+0042	U+0069	U+0065 U+006E ...
	Akwäba	(Fante from Ghana)	U+0041	U+006B	U+0077 U+00E4 ...
		UTF-32	UTF-16	UTF-8	
(b)	Welcome	00000057000000650000006C00000063 ...	00570065006C0063 ...	57656C63 ...	
	Haere mai	00000048000000610000006500000072 ...	0048006100650072 ...	48616572 ...	
	Wilkommen	00000057000000690000006C0000006B ...	00570069006C006B ...	57696C6B ...	
	Bienvenue	0000004200000069000000650000006E ...	004200690065006E ...	4269656E ...	
	Akwäba	000000410000006B00000077000000E4 ...	0041006B007700E4 ...	416B77C3A4...	

Figure 8.7: Encoding *Welcome*: (a) in Unicode; (b) in UTF-32, UTF-16, and UTF-8

A UTF-32 file can be either. Working within a single computer system, it does not matter which ordering is used—the system will be internally consistent. If necessary, however, UTF-32 encoding can include a *byte-order mark* to differentiate the two cases. This character, from the Reserved zone in [Table 8.1](#), is defined as “zero-width no-break space.” Its byte-for-byte transposition is intentionally defined to be an invalid character. This means that software that does not expect a byte-order mark will be fail-safe, since a correctly matched endian system displays the byte-order mark as a zero-width space, and a mismatched one immediately detects an incompatibility through the presence of the invalid character.

In practice it is rare to encounter UTF-32 data, because it makes inefficient use of storage. If text is constrained to the basic multilingual plane (and it usually is), the top two bytes of every character are zero.

UTF-16

In UTF-16, characters within the Basic Multilingual Plane are stored as 2-byte integers, as can easily be discerned in [Figure 8.7b](#). The same byte-order mark as above is used to distinguish the UTF-16BE and UTF-16LE variants.

To represent code points outside the Basic Multilingual Plane, specially reserved values called *surrogate characters* are used. Two 16-bit values, both from the Surrogates zone in [Table 8.1](#), are used to specify a 21-bit value in the range U+10000–U+10FFFF (a subset of the full 21-bit range, which goes up to 1FFFFFF). Here are the details: the 21-bit number is divided into 11 and 10 bits, respectively, and to each is added a predetermined offset to make them fall into the appropriate region of the surrogate zone. This is a kind of “escape” mechanism, where a special code is used to signify that the following value should be treated differently. However, the surrogate approach is more robust than regular escaping: it allows recovery from errors in a corrupted file, because it does not overload values from the nonescaped range, and thus the meaning cannot be confused even if the first surrogate character is corrupted. Inevitably, robustness comes at a cost—fewer values can be encoded than with conventional escaping.

UTF-8

UTF-8 is a byte-based, variable-length scheme that encodes the same 21-bit character range as UTF-16 and UTF-32. Code lengths vary from one byte for ASCII values through four bytes for values outside the Basic Multilingual Plane. If the top bit of a UTF-8 byte is 0, that byte stands alone as a Unicode character, making the encoding backward-compatible with 7-bit ASCII. Otherwise, when the byte begins with a 1 bit, the leading 1 bits in it are counted, and their number signals the number of bytes in the code—11 for two bytes, 111 for three, and 1111 for four. The four possibilities are illustrated in [Table 8.2](#). Subsequent bytes in that character’s code set their two top bits to 10—so that they can be recognized as continuation bytes even out of context—and use the remaining six bits to encode the value.

[Figure 8.7b](#) shows the *Welcome* example in UTF-8. Of interest is the last line, which is one byte longer than the others. This is because the encoding for *ä* falls outside the ASCII range: Unicode U+00E4 is represented as the two bytes C3 A4, in accordance with the second entry of [Table 8.2](#).

Table 8.2: Encoding the Unicode character set as UTF-8

Unicode value	21-bit binary code	UTF-8 code			
U+00000000 – U+0000007F	00000000000000www	0www			
U+00000080 – U+000007FF	000000000wwwxxxx	110www	10xxxx		
U+00000800 – U+0000FFFF	0000wwwxxxxxyyyyy	1110www	10xxxx	10yyyy	
U+00010000 – U+001FFFFF	Wwwwxxxxxyyyyyzzzzz	11110www	10xxxx	10yyyy	10zzzzz

To eliminate ambiguity, the Unicode standard states that UTF-8 must use the shortest possible encoding. For example, writing UTF-8 F0 80 80 C7 encodes U+0047 in a way that satisfies the rules, but it is invalid because 47 is a shorter representation.

Using Unicode in a digital library

Unicode encoding may seem complex, but it is straightforward enough to use in digital library software. Internally, every string can be declared as an array of 16-bit integers and used to hold the UTF-16 encoding of the characters. In Java this is exactly what the built-in *String* type does. In C and C++ the data type *short* is a 16-bit integer, so UTF-16 can be supported by declaring all strings to be *unsigned short*. Readability is improved by building a new type that encapsulates this. In C++ a new class can be built that provides appropriate functionality. Alternatively, a support library can be used, such as the type *wchar_t* defined in the ANSI/ISO standard to represent “wide” characters.

Further work is necessary to treat surrogate characters properly. But usually operation can be restricted to the Basic Multilingual Plane, which covers all living languages—more than enough for most applications. A further practical restriction is to avoid combining characters and to work with Unicode level 1, which makes it far easier to implement string matching operations.

Take the problem of accent folding. Speakers of languages like French and Spanish generally type words in the properly accented form, because without accents words look as strange to them as misspellings do to us. But in some circumstances it may be helpful to arrange for queries to match terms in documents whether or not accents are used—either in the document or the query. For example, in a digital library based on the publications of famous 17th-century mathematicians, documents containing the word *Hôpital*—a French mathematician, as well as the word for hospital—would be returned for the query *Hopital*. Equally, documents containing the unaccented term (perhaps caused by erroneous optical character recognition) would be returned for the query *Hôpital*, as well as ones containing the correct spelling.

If the input is Unicode level 1, the software need only check each character in turn, looking up its Unicode value in a table containing the fixed list of accented characters to see whether it should be

elided. The situation is a little more complex when working with higher levels of Unicode representation—because combining characters must be checked too—but it is not overly complex.

Sorting Unicode strings into order is a more involved problem. English can be sorted simply and efficiently by numerically ordering a given list of strings by their ASCII values (ignoring, for now, complications like case difference), but things are more complex in an international setting. Although the code pages in Unicode present characters in logical groups that have a systematic ordering, extending the simple approach to sort strings according to each character's Unicode value quickly runs into difficulties. Corrections in later versions of the standard are necessarily out of order. In German, the letter *ö* precedes *z* in the alphabet, but the opposite convention is used in Swedish. In Slovak, *ch* is treated as a single letter and positioned after *h*. Even within a language, the rules for ordering can differ depending on the context. (In certain English-speaking countries, telephone directories treat names beginning with *Mc* and *Mac* as though they were the same, with *MacAdam* and *McAdam* appearing together and *Maguire* later on.) The situation is more complex for non-European languages, and for ideographic scripts like Chinese characters, the notion of ordering is not well defined—we return to this at the end of Section 8.4.

The good news is that Unicode defines an approach that takes all these issues into account: the so-called *collation algorithm*. This algorithm first transforms each string into the first of the four Unicode normal forms—the process of canonical decomposition. Then a lookup table is used to map the characters to a set of numeric values known as *levels*. Different levels are used to achieve different types of ordering, such as ignoring case or accents. The standard defines a 4-level scheme as the default. At the first level is the base character: such as the difference between an *A* and a *B*. Accent differences are taken into account in the second level, case differences at the third, and punctuation differences at the fourth, as illustrated in Table 8.3. The default table is then combined with a mechanism known as *tailoring* that allows for language-specific variations. The scheme is powerful enough to support the ordering described in Section 8.4, such as by the number of strokes in an ideograph.

Most programming languages provide library support for the Unicode collation algorithm. Although efficiency is a primary objective, generality comes at a price, and ordering strings this way carries a significant computation cost—far greater than accent folding, for example. This will become less of a concern over time as technological advances improve computer performance.

When writing Unicode characters to disk in a digital library, the 16-bit character representation is usually converted to UTF-8; the process is reversed when the disk is read. This normally reduces file

Table 8.3: Different levels of sorting in Unicode

Level	Description	Examples
1	Base characters	role < roles < rule
2	Accents	role < rôle < roles
3	Case	role < Role < rôle
4	Punctuation	role < “role” < Role

size greatly and increases portability, because the files do not depend on the order in which the bytes are written.

Care is necessary to display and print Unicode characters properly. Most Unicode-enabled applications incorporate an arsenal of fonts and use a lookup table to map each character to a displayable glyph in a known font. Complications include composite Unicode characters and the direction in which the character sequence is displayed. Digital library software benefits immensely from the Unicode support already present in modern Web browsers.

8.3 Hindi and Indic Scripts

Unicode is advertised as a uniform way of representing all the characters used in all the world's languages. Unicode fonts exist and are installed as standard with most commercial word processors and Web browsers. It is natural for people—particularly people from Western linguistic backgrounds—to assume that all problems associated with representing different languages on computers have been solved. Unfortunately, today's Unicode-compliant applications fall far short of providing a satisfactory solution for languages with intricate scripts.

We use Hindi and related Indic scripts as an example. As [Table 8.1](#) shows, the Unicode space from 0900 to 0DFF is reserved for ten Indic scripts. Although many hundreds of different languages are spoken in India, the principal officially recognized ones are Hindi, Marathi, Sanskrit, Punjabi, Bengali, Gujarati, Oriya, Assamese, Tamil, Telugu, Kannada, Malayalam, Urdu, Sindhi, and Kashmiri. The first 12 of these are written in one of nine writing systems that have evolved from the ancient Brahmi script. The remaining three, Urdu, Sindhi, and Kashmiri, are primarily written in Persian Arabic scripts, but can be written in Devanagari, too (Sindhi is also written in the Gujarati script). The nine scripts are Devanagari, Bengali, Gujarati, Oriya, and Gurmukhi (northern or *Aryan* scripts), and Tamil, Telugu, Kannada, and Malayalam (southern or *Dravidian* ones). [Figure 8.8](#) gives some characters in each of these scripts. As you can see, the characters are beautiful—and the scripts differ radically from each other. Unicode also includes a script for Sinhalese, the official language of Sri Lanka.

Hindi, the official language of India, is written in Devanagari (pronounced *Dayv'nagri*, with the accent on the second *a*), which is used for writing Marathi and Sanskrit as well. (It is also the official script of Nepal.) The Punjabi language is written in Gurmukhi. Assamese is written in a script that is very similar to Bengali, but it has one additional glyph and another glyph that is different. In Unicode, the two scripts are merged, with distinctive code points for the two Assamese glyphs. Thus the Unicode scripts cover all 12 of the official Indian languages that are not written in Persian Arabic. All these scripts derive from Brahmi, and all are phonetically based. In fact the printing press did not reach the Indian subcontinent until missionaries arrived from Europe. The languages had a long time to evolve before they were fixed in print, which contributes to their diversity.

ISCII: Indian Script Code for Information Interchange

During the 1970s the Indian Department of Official Languages began working on devising codes that catered to all official Indic scripts. A standard keyboard layout was developed that provides a

Devanagari	अ आ इ ई उ ऊ ऋ लृ एँ ऐ ओ औ क ख ग घ ङ च छ ज झ
Bengali	অ আ ই ঐ উ ঊ খা ঞ এ ঔ ও ভে ক থ গ ঘ ঙ চ ছ জ ঝ ণ ট ঠ ড
Gurmukhi	ਅ ਆ ਇ ਈ ਉ ਊ ਏ ਐ ਓ ਕ ਖ ਗ ਘ ਙ ਚ ਛ ਜ ਝ ਵ ਟ ਠ ਡ ਢ ਤ ਥ
Gujarati	અ આ ઇ ઈ ઉ ઊ ઋ એ ઐ ઒ ઑ ઔ ક ખ ગ ઘ ઙ ળ જ ઝ ણ ટ ઠ ડ દ ન
Oriya	ଅ ଆ ଇ ଲି ଉ ଊ ଋ ୌ ଏ ଐ ଓ ଓଁ କ ଖ ଗ ଘ ଙ ଚ ଛ ଜ ଭ ଟ ଠ ଡ ଢ ତ ଥ
Tamil	அ ஆ இ ஈ உ ஊ எ ஏ ஐ ஒ ஓ ஔ க கா ச சா ஞ ன த ந
Telugu	అ ఆ ఇ ఈ ఉ ఊ యు న వ ఏ వి ఐ ఒ ఓ ఔ క ఖ గ ఘ జ చ ఛ జ ర్ము
Kannada	ಅ ಆ ಇ ಈ ಉ ಊ ಯು ನ ವ ಎ ಏ ಒ ಓ ಔ ಕ ಖ ಗ ಘ ಜ ಚ ಛ ಜ ರ್ಲು
Malayalam	അ അ ഇ ഹു ഉ ഊ ലി ണ ഫ്ല ഒ ഓ ഔ ക വ ഗ ഘ

Figure 8.8: Examples of characters in Indic scripts

uniform way of entering them all. Despite the very different scripts, the alphabets are phonetic and have a common Brahmi root that was used for ancient Sanskrit. The simultaneous availability of multiple Indic languages was intended to accelerate technological development and to facilitate national integration in India.

The result was ISCII, the Indian Script Code for Information Interchange. Announced in 1983 (and revised in 1988), it is an extension of ASCII that places new characters in the upper region of the code space. The code table supplies all the characters required in the Brahmi-based Indic scripts. [Figure 8.9a](#) shows the ISCII code table for the Devanagari script. Tables for the other scripts in [Figure 8.8](#) are similar but contain differently shaped characters (and some entries are missing because there is no equivalent character in that script). The code table contains 56 characters, 10 digits (in the last line of [Figure 8.9a](#)), and 18 accents and combining characters. There are also three special escape codes, but we will not delve into their meaning here.

Unicode for Indic scripts

The Unicode developers adopted ISCII lock, stock, and barrel—they had to, because of their policy of round-trip compatibility with existing codes. They used different parts of the code space for the various scripts, which means that (in contrast to ISCII) documents containing multiple scripts can easily be represented. However, they also included some extra characters—about 10 of them—that in the original ISCII design were supposed to be formed from combinations of other keystrokes. [Figure 8.9b](#) shows the Unicode code table for the Devanagari script.

Most of the extra characters give a shorthand for frequently used characters, and they differ from one language to another. An example in Devanagari is the character Om, a Hindu religious symbol:

ॐ (Unicode U+0950).

A0		ॐ	:	अ	आ	इ	ई	उ	ऊ	ऋ	ऐ	ए	ऐ	ँ	ओ
B0	ओ	औ	ऑ	क	ख	ग	घ	ङ	च	छ	ज	झ	ञ	ट	ठ
C0	ड	ण	त	थ	द	ध	न	त	प	फ	ब	भ	म	य	र
D0	ॠ	ॡ	ॢ	ॣ	।	॥	०	१	२	३	४	५	६	७	८
E0	०	१	२	३	४	५	६	७	८	९					
F0	EXT	०	१	२	३	४	५	६	७	८	९				

(a)

0900		ॐ	:		अ	आ	इ	ई	उ	ऊ	ऋ	ॠ	ऐ	ए	
0910	ऐ	ऑ	ओ	औ	क	ख	ग	घ	ङ	च	छ	ज	झ	ञ	ट
0920	ठ	ड	ण	त	थ	द	ध	न	त	प	फ	ब	भ	म	य
0930	र	ॠ	ॡ	ॢ	ॣ	।	॥	०	१	२	३	४	५	६	७
0940	०	१	२	३	४	५	६	७	८	९					
0950	०	१	२	३	४	५	६	७	८	९					
0960	०	१	२	३	४	५	६	७	८	९					
0970	०	१	२	३	४	५	६	७	८	९					

(b)

20		!	'	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
30	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
40	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
50	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
60	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
70	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
80															
90															
A0		ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
B0	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
C0	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
D0	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
E0	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
F0															

(c)

Figure 8.9: Devanagari script: (a) ISCII; (b) Unicode (U+0900-U+0970); (c) code table for the Surekh font

Although it is not part of the ISCII set, it can be created from the keyboard by typing the sequence of characters

उँ (ISCII A8 A1 E9).

The third character (ISCII E9) is a special diacritic sign called the *Nukta* (which phonetically represents nasalization of the preceding vowel). ISCII defines Nukta as an operator used to derive some

little-used Sanskrit characters that are not otherwise available from the keyboard, such as Om. However, Unicode includes these lesser-used characters as part of the character set (U+0950 and U+0958 through U+095F).

Although the Unicode solution is designed to adequately represent all the Indic scripts, it has not yet found widespread acceptance. A practical problem is that these scripts contain numerous clusters of two to four consonants without any intervening vowels, called *conjuncts*. Conjuncts are similar to the ligatures discussed earlier, characters represented by a single glyph whose shape differs from the shapes of the constituents. Indic scripts contain far more of these, and there is a greater variation in shape. For example, the conjunct

लृ

is equivalent to the two-character combination

लृ (Unicode U+0932 U+0943).

In this particular case, the conjunct happens to be defined as a separate code in Unicode (U+090C)—just as the ligature *fi* has its own code (U+FB01). The problem is that this is not always the case. In the ISCII design, *all* conjuncts are formed by placing a special character between the constituent consonants, in accordance with the design goal of a uniform representation for input of all Indic languages on a single keyboard. In Unicode, some conjuncts are given their own code—like the one above—but others are not.

Problems with the adoption of Unicode

Figure 8.9c shows the code table for a particular commercially available Devanagari font, Surekh (a member of the ISFOC family of fonts). Although there is much overlap, there is certainly not a one-to-one correspondence between the Unicode and Surekh characters, as can be seen in Figures 8.9b and c. Some conjuncts, represented by two to four Unicode codes, correspond to a single glyph that does not have a separate Unicode representation but does have a corresponding entry in the font. And in fact the converse is true: there are single glyphs in the Unicode table that are produced by generating pairs of characters. For example, the Unicode symbol

ओ (Unicode U+0912)

is drawn by specifying a sequence of three codes in the Surekh font.

We cannot give a more detailed explanation of why such choices have been made—it is a controversial subject, and a full discussion would require a book in itself. However, the fact is that the adoption of Unicode in India has been delayed because some people feel that it represents an uncomfortable compromise between the clear but spare design principles of ISCII and the practical requirements of actual fonts. They prefer to represent their texts in the original ISCII, because they regard it as conceptually cleaner.

The problem is compounded by the fact that today's word processors and Web browsers take a simplistic view of fonts. In reality, combination rules are required—and were foreseen by the designers

of Unicode—that take a sequence of Unicode characters and produce the corresponding single glyph from [Figure 8.9c](#). Such rules can be embodied in the “composite fonts” that were described in Section 4.5. But ligatures in English, such as *fi*, have their own Unicode entry, which makes things much easier. For example, the “insert-symbol” function of word processors implements a one-to-one correspondence between Unicode codes and the glyphs on the page.

The upshot is that languages like Hindi are not properly supported by current Unicode-compliant applications. A table of glyphs, one for each Unicode value, is insufficient to depict text in Hindi script. To make matters worse, in practice some Hindi documents are represented using ISCII while others are represented using raw font codes like that of [Figure 8.9c](#), which are specific to the particular font manufacturer. Different practices have grown up for different scripts. For example, the majority of documents in the Kannada language on the Web seem to be represented using ISCII codes, whereas in the Malayalam language, diverse font-specific codes are used. Often, to read a new Malayalam newspaper in your Web browser you have to download a new font!

To accommodate Indic documents in a digital library that represents documents internally in Unicode, it is necessary to implement several mappings:

- from ISCII to Unicode, so that ISCII documents can be incorporated;
- from various different font representations (such as ISFOC, used for the Surekh font) to Unicode, so that documents in other formats can be accommodated;
- from Unicode to various different font representations (such as ISFOC), so that the documents can be displayed on computer systems with different fonts.

The first is a simple transliteration, because Unicode was designed for round-trip compatibility. However, both of the other mappings involve translating sequences of codes in one space into corresponding sequences in the other space (although all sequences involved are very short). [Figure 8.10](#) shows a page produced by such a scheme.



Figure 8.10: Page produced by a digital library in Devanagari script

8.4 Word Segmentation and Sorting

Spaces are a relatively recent invention in Western languages. Although ancient Hebrew and Arabic used spaces to separate words, partly to compensate for the lack of vowels, they were not used in Latin until 600 to 800 A.D. When the Latin alphabet was adopted for English, it was written *scripta continua*, without any word separators. Later, centered dots were added to make reading easier, and subsequently the dots were replaced with spaces. Today, languages in the CJK (Chinese-Japanese-Korean) family are written without using any spaces or other word delimiters; so are the Thai, Khmer (Cambodian), and Lao languages. This introduces the need for segmentation algorithms to separate words for indexing.

Segmenting words

Finding word boundaries in the absence of spaces is a non-trivial problem, and ambiguities often arise. To help you appreciate the problem, Figure 8.11a shows two interpretations of the same Chinese characters. The text is a play on the ambiguity of phrasing. Once upon a time, the story goes, a man embarked on a long journey. Before he could return home the rainy season began, and he took shelter at a friend's house. As the rains continued he overstayed his welcome, and his friend wrote him a note: the first line in Figure 8.11a. As shown in the second line, it reads “It is raining, the god would like the guest to stay. Although the god wants you to stay, I do not!” But before taking the hint and leaving, the visitor added the punctuation shown in the third line, making three sentences whose meaning is totally different—“The rainy day, the staying day. Would you like me to stay? Sure!”

This is an example of ambiguity related to phrasing, but ambiguity also can arise with word segmentation. Figure 8.11b shows a more prosaic example. For the ordinary sentence on the first line, there are two different interpretations, depending on the context: “I like New Zealand flowers” and “I like fresh broccoli.”

Written Chinese documents are unsegmented, and readers are accustomed to inferring the corresponding sequence of words almost unconsciously. Accordingly, machine-readable text is usually

(a)	下雨天留客天留我不留	Unpunctuated Chinese sentence
	下雨、天留客。天留、我不留！	<i>It is raining, the god would like the guest to stay. Although the god wants you to stay, I do not!</i>
	下雨天、留客天。留我不？留！	<i>The rainy day, the staying day. Would you like me to stay? Sure!</i>
(b)	我喜欢新西兰花	Unsegmented Chinese sentence
	我 喜欢 新西兰 花	<i>I like New Zealand flowers</i>
	我 喜欢 新 西兰花	<i>I like fresh broccoli</i>

Figure 8.11: Alternative interpretations of two Chinese sentences: (a) ambiguity caused by phrasing; (b) ambiguity caused by word boundaries

unsegmented. To render them suitable for full-text retrieval, a segmentation scheme should be used to insert word boundaries at appropriate positions prior to indexing.

One segmentation method is to use a language dictionary. Boundaries are inserted to maximize the number of the words in the text that are also present in the dictionary. Of course, there may be multiple valid segmentations, and heuristics are needed to resolve ambiguities.

Another method is based on the fact that text divided into words is more compressible than text that lacks word boundaries. You can demonstrate this with a simple experiment. Take a text file, compress it with any standard compression utility (such as gzip), and measure the compression ratio. Then remove all the spaces from the file, making it considerably smaller (about 17 percent smaller, because in English approximately one letter in six is a space). When you compress this smaller file, the compression ratio is noticeably worse than for the original file. Inserting word boundaries improves compressibility.

This fact can be used to divide text into words, based on a large corpus of hand-segmented training data. Between every two characters lies a potential space. A text compression model can be trained on presegmented text, and coupled with a search algorithm to interpolate spaces to maximize the overall compression. [Section 8.5](#) (“Notes and sources”) at the end of this chapter points to a fuller explanation of the technique.

For non-Chinese readers, the success of the space-insertion technique can be illustrated by applying it to English. [Table 8.4](#) shows original text at the top, complete with spaces. Below is the input to the segmentation procedure. Underneath that is the output of two segmentation schemes: one dictionary-based and the other compression-based. The training text was a substantial sample of English, although far smaller than the corpus used to produce the word dictionary.

Word-based segmentation fails badly when the words are not in the dictionary. In this case both *crocidolite* and *Micronite* are segmented incorrectly. In addition, *inits* is treated as a single word because it occurred that way in the text from which the dictionary was created, and in cases of ambiguity the algorithm prefers longer words. The strength of the compression-based method is that it performs well on unknown words. Although *Micronite* does not occur in the training corpus, it is correctly segmented. The compression-based method makes two errors, however. First, a space was not

Table 8.4: Segmenting words in English text

Original text	the unit of New York-based Loews Corp that makes Kent cigarettes stopped using crocidolite in its Micronite cigarette filters in 1956.
Without spaces	theunitofNewYork-basedLoewsCorpthatmakesKentcigarettesstoppedusingcrocidoliteinitsMicronitecigarettefiltersin1956.
Word-based segmentation	the unit of New York-based Loews Corp that makes Kent cigarettes stopped using c roc id o lite inits Micron it e cigarette filters in 1956.
Character-based segmentation	the unit of New York-based LoewsCorp that makes Kent cigarettes stopped using croc idolite in its Micronite cigarette filters in 1956.

inserted into *LoewsCorp* because it happens to require fewer bits to encode than *Loews Corp.* Second, an extra space was added to *crocidolite* because that also reduced the number of bits required.

Segmenting words in Thai/Khmer/Lao

Thai, Khmer, and Lao are other languages that do not use spaces between the words in a sentence, although they do include spaces between phrases and sentences. Unlike the CJK family, which uses ideographs, they are alphabetic languages. However, they are easier to read than English would be if spaces were omitted, because English provides fewer clues about word breaks.

In Thai, for example, there are many rules that govern where words can begin and end. Thai includes a “silence marker” called *gaaran*—the little symbol that appears above the last letter in the word การ์นต์—which indicates that the letter or letters underneath are not pronounced in the usual Thai pronunciation. Here are some of the rules:

- *gaaran* ends a word (except for some European loan words, such as กอล์ฟ for *golf*);
- ะ ends a word (except in the rare cases when it is followed by a consonant with the *gaaran* symbol, as in เคราะห์);
- ั ends a word;
- the vowels แ อ โ อี start a word (these are called “preposed” vowels, and are written before their accompanying consonant).

These rules help to determine some word boundaries, but not all. Sometimes ambiguities arise. For example, depending on how it is segmented,

ตากลม

means (roughly) *breath of fresh air* (ตาก ลม) or *round eyes* (ตา กลม). As with Chinese, there are several possible approaches to determining word boundaries for searching.

Sorting Chinese text

Several different schemes underlie printed Chinese dictionaries and telephone directories. Characters can be ordered according to the number of strokes they contain; or they can be ordered according to their *radical*, which is a core symbol on which they are built; or they can be ordered according to a standard alphabetical representation called Pinyin, where each ideograph is given a one- to six-letter equivalent. Stroke ordering is probably the most natural way of ordering character strings for Chinese users, although many educated people prefer Pinyin (not all Chinese people know Pinyin). This presents a problem when creating lists of Chinese text that are intended for browsing.

To help you appreciate the issues involved, [Figure 8.12](#) shows title browsers for a large collection of Chinese documents. The rightmost button on the green access bar near the top invokes [Figure 8.12a](#). Here, titles are ordered by the number of strokes in their first character, which is given across the top: the user has selected six. In all the titles that follow, the first character has six strokes. This is probably not obvious from the display, because you can only count the strokes in a character if you



Figure 8.12: Browsing a list of titles in Chinese: (a) stroke-based browsing; (b) Pinyin browsing

know how to write it. To illustrate this, the initial characters for the first and seventh titles are singled out and their writing sequence is superimposed on the screen shot: the first stroke, the first two strokes, the first three strokes, and so on, ending with the complete character, which is circled. All people who read Chinese know immediately how many strokes are needed to write any particular character.

There are generally more than 200 characters corresponding to a given number of strokes, almost any of which could occur as the first character of a title. Hence the titles in each group are displayed in a particular conventional order, again determined by their first character. This ordering is more complex. Each character has an associated *radical*, the basic structure that underlies it. For example, the radical in the upper example singled out in [Figure 8.12a](#) (the first title) is the pattern corresponding to the initial two strokes, which in this case form the left-hand part of the character. Radicals have a conventional ordering that is well known to educated Chinese; this particular one is number 9 in the Unicode sequence. Because this character requires four more strokes than its radical, it is designated by the two-part code 9.4. In the lower example singled out in [Figure 8.12a](#) (the seventh title), the radical corresponds to the initial three strokes, which form the top part of the character, and is number 40; thus this character receives the designation 40.3. These codes are shown to the right of [Figure 8.12a](#) but would not form part of the final display.

The codes form the key on which titles are sorted. Characters are grouped first by radical number, then by how many strokes are added to the radical to make the character. Ambiguity occasionally arises: the code 86.2, for example, appears twice. In such situations, the tie is broken randomly.

Stroke-based ordering is quite complex, and Chinese readers have to work harder than we do to identify an item in an ordered list. It is easy to decide on the number of strokes, but once a page like [Figure 8.12a](#) is reached, most people simply scan it linearly. A strength of computer displays is that they can at least offer a choice of access methods.

The central navigation button of [Figure 8.12a](#) invokes the Pinyin browser in [Figure 8.12b](#), which orders characters alphabetically by their Pinyin equivalent. The Pinyin codes for the titles are shown to the right of the figure, but again would not form part of the final display. Obviously, this arrangement is much easier for Westerners to comprehend.

8.5 Notes and Sources

The design and construction of the “Niupepa” (the Māori word for “newspaper”) collection of historical New Zealand newspapers is described by Apperley et al. (2002); Apperley et al. (2001) give a synopsis. The scanning and OCR operations involved in creating this collection are discussed at the end of Section 4.2. The project was undertaken in conjunction with the Alexander Turnbull Library, a branch of the New Zealand National Library, whose staff gathered the material together and created the microfiche that was the source for the digital library collection. This work is being promoted as a valuable social and educational resource and is partially funded by the New Zealand Ministry of Education.

The collection of Tang Dynasty rubbings in [Figure 8.3a](#) was produced in conjunction with the Chinese Department of Peking University. The Chinese literature in [Figure 8.3b](#) and the Arabic collection in [Figure 8.4](#) are small demonstrations built out of material gathered from the Web. The Chinese title browsers in [Figure 8.12](#) are mock-ups and have not yet been implemented, although the Greenstone structure makes it easy to add new browsers like this by adding a new “classifier” module (see Part II).

The official description of Unicode is given in International Standard ISO/IEC 10646-1, Information Technology—Universal Multiple-Octet Coded Character Set (UCS)—Part 1: Architecture and Basic Multilingual Plane. Part 1 covers the values U+000000–U+10FFFF; the other values are covered by Part 2. We mentioned that the ISO standard and the Unicode Consortium differ inconsequentially over the size of the character space. Another difference is that Unicode augments its definition with functional descriptions to help programmers develop compatible software. For example, it defines an algorithm for displaying bidirectional text (Unicode Consortium, 2006). The consortium uses the Web site www.unicode.org to expedite the release of versions, revisions, and amendments to the standard.

The fact that certain characters have more than one Unicode representation, which is a consequence of demanding round-trip compatibility, has been exploited to deceive computer users about what remote system they are communicating with on the Internet in what is called the “international domain name homograph attack” (see www.opera.com/support/search/view/788).

Different scholars transcribe the Fante word for *welcome* in different ways. We have chosen the version with an umlaut above the second *a* to illustrate interesting text representation issues. Other common renderings are *akwāba* and *akwaaba*.

The material on segmenting Chinese text is from Teahan et al. (2000), while [Table 8.4](#), giving English segmentation results, is due to Teahan (1997). Information about the word segmentation problem in Thai is from www.thai-language.com.

Visions: Future, past, and present

Whither digital libraries? Where are we headed, where have we come from, and what should we be doing now? This chapter looks at the past, present, and future—but not in that order.

This book begins with a whirlwind tour of the history of 25 centuries of physical libraries. A whirlwind tour of prognostications for the first 25 centuries of digital libraries, starting today, would be an excellent way to end. But predictions about technology are futile. We cannot foresee even the next 25 years: in fact, given the rate of change in the field, predicting what will happen in 25 months seems sufficiently ambitious! Nevertheless, we will present some visions of the digital library—images of today as well as visions of tomorrow. Digital libraries have certain obvious practical advantages over physical ones and seem to offer the promise of far greater universality. While all this is true, we believe that their real power lies elsewhere. In our vision, the digital library is an all-encompassing information medium that will envelop us as we work—like air. Just as oxygen unobtrusively powers our body and brain, so digital libraries will unobtrusively empower our mind.

Having glimpsed the vision, we turn to a troubling but vital issue that threatens to jeopardize the whole enterprise: preserving the past. We touched on this in Chapter 6 when discussing schemes for representing metadata. Here we discuss the issue in more detail. The mission of a library is fourfold: to collect, organize, and disseminate information, and to pass it down to succeeding generations as a record of culture. In short, the librarian's duties boil down to access and preservation: providing access to the world's literature for today's readers (the first three planks of the mission) and preserving it for future generations (the fourth). Libraries are containers in which things are put in and retrieved from—at different points in the space-time continuum.

This book talks a lot about access, but little has been said so far about the question of preservation, which is a devastatingly serious problem. As is clear from the history of libraries in Chapter 1, a great deal has been destroyed forever, but some ancient material has been preserved. It is sobering to realize that if the ancients had possessed our technology, we would probably have an even smaller record of the past—for who today can read their computer disks of even 20 years ago? And, although the problem of preservation is seriously exacerbated by technology, it will not be solved by technology alone. Preservation is a social issue as well as a technical one. And it's an economic issue, too: preservation is potentially expensive, and the Alexandrian principle introduced in Section 1.2 with reference to collection building applies here as well. Libraries cannot afford to preserve everything, so part of the problem is deciding what to preserve.

This chapter also highlights some trends in contemporary digital library research. We present specific examples in key areas: a reactive digital library environment; an institutional-sized digital library that you can carry in your pocket; and an interactive digital library that is augmented by an external

knowledge source to provide an experience that seamlessly blends searching and browsing. The examples illustrate capabilities that go well beyond the classic search and browse tasks initiated by a user from a desktop PC that have been the hallmark of digital libraries for the last decade.

We round out Part I of this book by asking whether digital libraries can be used by people from oral cultures. Why not? And does this have implications for developing countries? You bet! And what about our own society, where text and sustained, coherent argument are increasingly being hijacked by imagery and sound bites?

Library traditions have long been influenced by the belief that libraries should serve democracy. As part of their mission to serve as resource centers for citizens, public libraries maintain collections of records, policy statements, government documents, and so on. The U.S. Library of Congress is the oldest cultural institution in the nation's capital, and serves as a symbol of American democracy and faith in the power of learning. Indeed, the library gained support during World War II by publicly defending American culture against the threat of totalitarianism in Europe.

We believe that future digital libraries will find a new role to play in helping to reduce the social inequity that haunts today's world, both within our own countries and between nations.

9.1 Libraries of the Future

Digital libraries have the potential to be far more flexible than conventional ones. Of course, they are portable: they will be with you whenever you want them to be: in your home, on a plane, at the beach, in a Ugandan village, or on the street (when you want to play that new song for your friends). They will be extensive, giving access to your personal book collection, your town's public library, and your university library. Furthermore, they will ultimately be seamlessly integrated with national and international sources of information—interlibrary loan at your fingertips. Like H. G. Wells's vision of a permanent world encyclopedia (see Chapter 1), they will give access to the world's recorded knowledge. But they will certainly not be static. In 1931, shortly before Wells penned his vision, Ranganathan wrote his "five laws of library science" (see Chapter 2), one of which is that a library is a growing organism. The ultimate digital library will continually be revised and extended by original thinkers around the world.

But wait, there's more. The flexibility of digital libraries will extend well beyond matters of physical convenience. Future digital libraries will surround you with information in ways that we can yet only dimly perceive. When Karl Marx wrote *Das Kapital*, he worked in the reading room of the British Museum Library and wrote using pen and paper. In contrast, future revolutionaries not only will use a laptop but also will work "inside" their digital libraries in a stronger and more visceral sense.

Today's visions

Figure 9.1 shows the reading room of the New York Public Library (the outside of which is shown in Figure 1.3). The reading room is a magnificent place: spacious and airy, with bright yet soft lighting, comfortable furniture, and warm tones of wood, leather, and books, as well as the hushed sounds of people working. You can become absorbed in what you are reading, or your gaze can wander



Figure 9.1: New York Public Library reading room

around the immense spaces above you, the massive windows, the huge chandeliers, the painted ceiling. But the space is impersonal, as any large physical library has to be. One size fits all. No matter how divergent your intellectual pursuits, you work in the same place as your neighbor. The most you might expect by way of personalization is an assigned carrel where you can leave your books and materials undisturbed until tomorrow. Is this the vision we seek for digital libraries?

Figure 9.2 shows an example of a digital library, one that forms part of the British Library. It is just as impersonal as Figure 9.1, if not more so. You stand at the threshold and are confronted with row upon row of identical workstations. This space lacks the architectural beauty of the reading room in Figure 9.1—it looks utilitarian rather than spacious. We all know from bitter experience that the technology it contains, no matter how cutting-edge today, will appear old and out of date in just a few months. But perhaps the most striking thing about the vision is the staged nature of this publicity photograph: one man in an empty room, clearly posed as though at work. Every other workstation in the room shows the same screen—not only that, but beside each one, on a typing stand, is the same piece of paper. This Orwellian setting does not seem like a nice place to work. Is it the vision we seek?

A more engaging picture of a present-day digital library is Figure 9.3, which shows the computer room at the Kataayi cooperative in Uganda with which this book opened. From a Western perspective, the library is decidedly low-tech. The computers are ancient, the furniture is shabby, the walls



Figure 9.2: Digital library in the British Library. “The Virtue of Virtual Libraries,” by Matt Jones. *The Independent*, 16 July 2001. London, England



Figure 9.3: A peek inside the digital library at the Kataayi cooperative in Uganda

are bare, the setting is plain and utilitarian. But it looks like it works. We see real people interacting with information in a real environment, rather than a publicity machine’s conception of some kind of spine-chilling ideal. This picture serves as a salutary reminder that libraries are about connecting people with the information they need. Kataayi’s library may not be flashy, but it works.

Tomorrow’s visions

So much for the present: what about the future? By way of—literally—comic relief, [Figure 9.4](#) shows a sci-fi artist’s image of a digital library taken from a Marvel comic, the living computers of Xandar (dated 1979). On this planet they have stored the still-functioning brains of the population

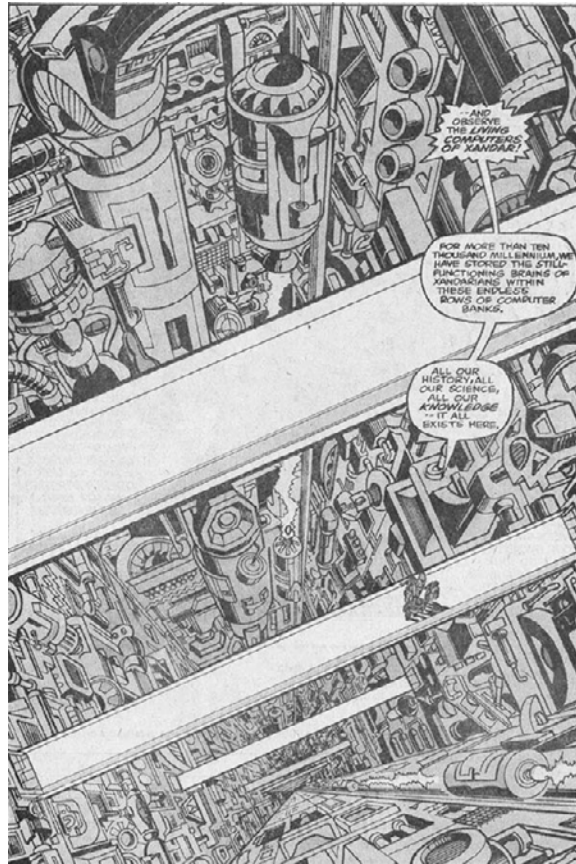


Figure 9.4: Xandar's digital library. Vol. 1, #25, 1979. FANTASTIC FOUR™ © 2002 Marvel Characters, Inc. Used with permission

for more than 10,000 millennia, giving—as our guide explains—a complete record of all their history, all their science, all their *knowledge*. Is this the long-term prognosis that we sought at the outset? (Note that our 25 centuries of libraries represent a mere eye-blink in Xandarian history.)

Here is a vision that emphasizes preservation over access. Xandarians can put the brains in, but can they get the knowledge out? It certainly appears from the illustration that their computer scientists have some work to do on the user interface. But the problem runs deeper. In storing living brains, what is lacking (or may be lacking—it's hard to tell from the picture) is *librarianship*—the selection, organization, and maintenance in the definition of digital libraries in Chapter 1, the wisdom that librarians put into the library by making value judgments about what information is to be included and how it should be organized. Figure 9.4 is reminiscent of a historical archive of the Web. There is no organization, no quality control: it's a mere repository. This is not a library.

Our vision of the digital library of the future is that it will be a personalized, comfortable space in which to work. Think of it as more like a kitchen than a library—a kitchen for knowledge

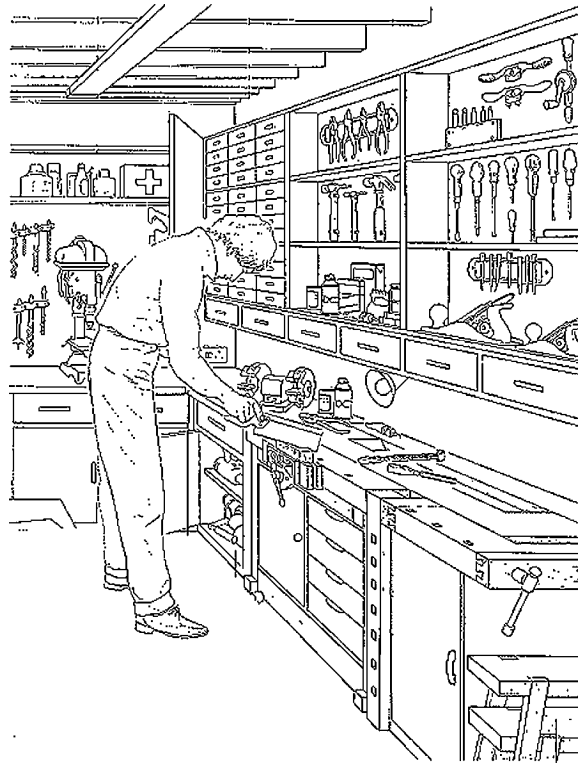


Figure 9.5: Carpenter's workshop. *Working in Wood*, by E. Scott. Putnam, 1980

preparation. If you like cooking, you will have arranged your kitchen to suit what you do: utensils ready to hand, pots by the stove, and spices for the kind of dishes you like to cook, placed just where you need them. Left-handed?—change your kitchen around. Short?—adjust the work surfaces. Like wine?—keep a glass handy. Stir-fries?—here's the wok.

Figure 9.5 shows another workspace that illustrates the vision we want to convey. This carpenter's workshop, although perhaps a little stiff and idealized, is a comfortable, productive workplace. Hand tools are arranged in convenient clusters, bench tools like the vise and drill press are conveniently located, nails and screws are boxed and labeled, and a well-positioned lamp illuminates the current job. This man knows where everything is, and the physical arrangement almost exudes productivity. Imagine how much better you could work here than in the messy corner, littered with assorted junk and half-finished projects, that most of us have in our garage. Imagine how this carpenter would feel if he had to carry all his tools into the New York Public Library reading room in Figure 9.1; unpack them, sort them, and arrange them before he began work; and pack them up again before lunch to leave the space clear for someone else.

Future digital libraries will be like the carpenter's workshop. Inside the computer your library will be arranged the way you like it and personalized for the kind of things you do. Externally it may look like Figure 9.1 (just bring your laptop and plug it in), Figure 9.2 (hopefully a little more welcoming

and ergonomic, with concealed computers, flat-panel screens, and wrist rests), or even [Figure 9.3](#) (although with more powerful equipment). Inside, not only will it give access to the world's recorded knowledge as Wells's vision did, but also it will feel like [Figure 9.5](#), arranged just for you and the kind of things you do.

You will need to invest in this personalization, just as the carpenter invested time and money establishing his workplace. And there are pitfalls aplenty: with the potential for flexibility comes the potential for confusion. A physical environment makes the possibilities that it offers for interaction—its “affordances”—openly manifest. There is no user manual for [Figure 9.5](#): you can step into this environment and see immediately how it works. It will be harder to make the intellectual environment your digital library provides so accessible and transparent.

But we're dreaming of the future: these problems will be solved. The computers so prominent in [Figure 9.2](#) will disappear. There will still be an interface—perhaps a screen, although it may disappear into the wall or into your spectacles; perhaps a keyboard, although it may be replaced by finger sensors or a microphone; and perhaps a mouse, although it may be a wand or a wave of the hand. But you will conceptualize this interface as a *library* rather than a *computer*: the computer will become invisible, just as the countless electric motors in your house have become invisible, disappearing into hair dryers, fans, electric razors, kitchen appliances, and DVD players. Tomorrow's digital library will feel less like a computer, and more like a kitchen or workshop.

As well as being personalized, your digital library will be dynamic, and not just in the sense that the information it contains will be totally up to date. The library will work alongside you, tracking your activity, unobtrusively rearranging itself to put what you might need in the context of what you are doing, just there where you can see and read it. When you quit for the day, it will continue to work for you, locating pertinent information, classifying and categorizing it, working through the implications, and researching on your behalf, so that the following day, when you recommence, you will start well ahead of where you left off.

Working inside the digital library

In Chapter 1 we characterize digital libraries as libraries without walls, although they do need boundaries (we argue that the very notion of a collection implies a boundary). Paradoxically, perhaps, in the future we will work inside the digital library in a new sense of “in”-ness that we can barely understand today. The library will be an environment that surrounds you in an intellectual, not a physical, sense. But virtual reality means that intellectual experiences can easily be translated into physical ones. More or less immersive (you can choose), the library will be an environment that reacts and responds to what you are doing, making the right kind of information available to you as and when you need it, and in an appropriate form. It will surround your mind.

[Figure 9.6](#) shows a Personal Information Retrieval Assistant developed at the University of Illinois, which hints at what is possible. It's a Web-based system for scholarly reading and writing that envelops users in a search-based environment. Users can control the sources in which related articles are sought and manage the citations to these articles once they have been located. [Figure 9.6a](#) shows the documents currently being worked on: Chapters 1 to 6 of this book, in fact. It also displays the

currently active digital library collections: CiteSeer, Wikipedia, Google Scholar, and DLHarvest Arizona. This selection can easily be modified, with sources switched in or out as desired.

The user clicks on Chapter 6, bringing up the main authoring page in [Figure 9.6b](#). On the left is the main writing area: in this case a section on MARC metadata. Beside it is a list of related keywords and phrases, including *library catalogs*, *card catalogs*, *library of congress online catalog*, and *anglo american cataloging rules*—which the system has identified automatically. These seed an ambient search, whose results are shown below, drip fed into the display area so as not to overwhelm the user. Two citations are listed, both from Wikipedia: *Library Catalog* and *AACR2*. Hovering over them brings up summary tool tips; clicking shows the source in a separate tab. An additional five citations have been found, as shown by the *New (5)* button above the citation display area. They will be gradually added in over time, but the user can display them immediately by clicking the button once for each item. Furthermore, the system continues to analyze the text and update the list of key phrases, which in turn generates more search terms.

Interesting citations can be saved by clicking the disk icon, and a citation folder is automatically created for each document being written (although users can create others too). [Figure 9.6c](#) shows the current citations for Chapter 6. All the usual browse and edit operations are available, and the list can be exported to formats like BibTeX, EndNote, and MODS (see Section 6.2 for more details about these formats).

Work created in this way is not only born digital, it is born in a library, fully contextualized and linked to the existing literature the moment it comes into the world. For instance, without any explicit searching, the system has suggested a citation to the *Resource Description and Access* content standard, a proposed successor to the Anglo-American Cataloging Rules—something that is not mentioned in the text. And exactly the same system supports reading. You can load an article into the document window and read it “in” the library. The article could be taken from the library or downloaded from elsewhere. The entire contents of the library are available with no effort, in context, at your fingertips, on the fly, as you read and write. This is a library that works along with you.

Another example of working within the digital library is MONK (Metadata Offer New Knowledge). MONK provides an environment within which humanities scholars can analyze texts and visualize the results. Software components for text-mining and Web-based data presentation are integrated into the digital library and configured to run on the fly, as needed. This makes it possible to locate works of literature that are similar, helping to establish, say, texts that influenced Jane Austen and vice versa. Alternatively, its visualization capabilities could be used to study the patterns of vocabulary usage and spelling in the works of Shakespeare.

9.2 Preserving the Past

Libraries link the past and the future, and preservation has always been a key function. Whatever form the cultural record is in, libraries ensure that it is preserved and made available for later use. Chapter 1 describes the tragic loss of the historic Alexandrian Library, willfully laid to waste during the decline of the Roman Empire. Acts of God—fires, floods, earthquakes—and acts of man—wars,



Figure 9.6: The Personal Information Retrieval Assistant: (a) my documents; (b) authorship;

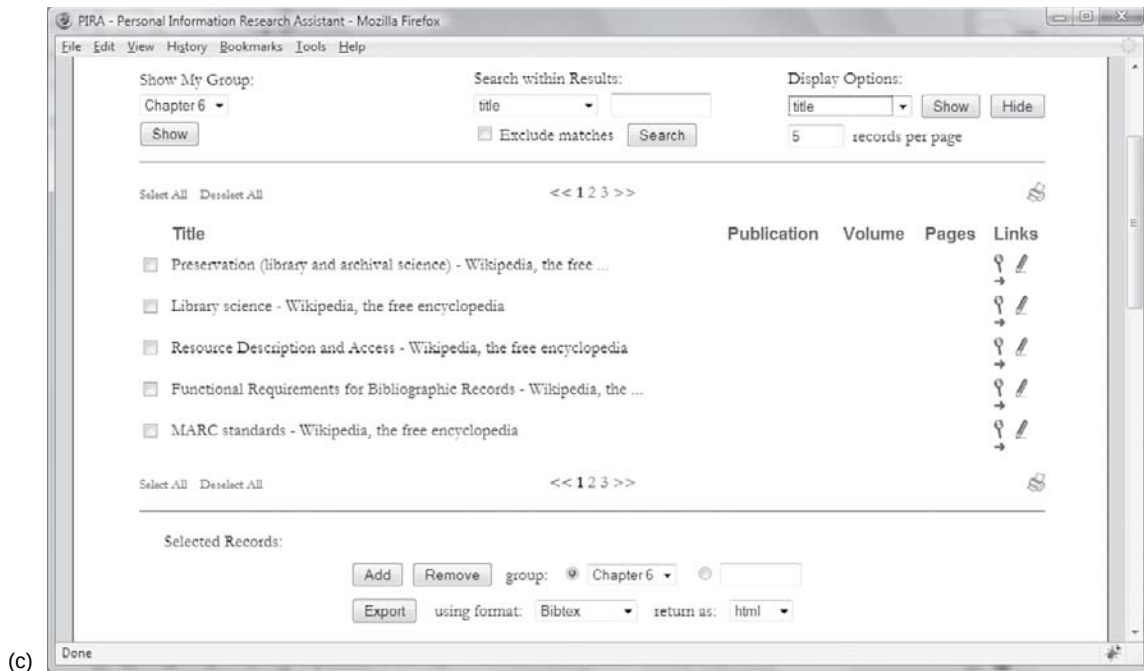


Figure 9.6, cont'd: (c) managing citations

revolutions—have damaged the holdings of many libraries, destroying forever much of the recorded history of human civilization. We gaze in awe at the stone steles of Xi'an (Figure 1.4) and the *Book of Kells* (Figure 1.9) not just because of their intrinsic beauty, but also because they have survived one or two millennia and have witnessed unimaginable strife and conflict.

The problem of preservation

The trouble with technological progress is that it seems to come at the expense of preservation. In Wolfenbüttel, Germany, Duke August's collection of 31,000 volumes of medieval literature with pale parchment bindings (Figure 9.7) is still there for any visitor to see and, barring catastrophe, our descendants in years to come will be able to share the experience. Xi'an's stone steles, older technology, have lasted longer. It is a tragic fact that your grandchildren and ours, while they can still visit Wolfenbüttel and Xi'an, will probably be unable to see our own parents' and grandparents' literature—despite the fact that it is far more recent.

Why? The reason is paper technology. Until the middle of the 19th century, nearly all paper used for written or printed material was made from cotton or linen rags. Paper made this way lasts for several hundred years without decomposing. Since then, however, ordinary paper has been made from wood pulp treated with acidic chemicals. The residual acid slowly decomposes the paper, causing it to become brittle and crumbly. The rate of decomposition depends on the paper's original quality and



Figure 9.7: Medieval literature in the library at Wolfenbüttel. Herzog August Bibliothek Wolfenbüttel: Globenkabinett

the conditions under which it has been stored. Light, heat, and humidity all accelerate decomposition. After a period of only a few decades, books made with acid-based paper decompose to the point where they can crumble into pieces, even when handled carefully. If this technological “advance” had been made in medieval times, the Duke’s collection would have disintegrated centuries ago.

The problem continues. Library organizations advise publishers to use acid-free paper when printing new books that are likely to have enduring value. However, fewer than 20 percent of hardcover books—and even fewer paperbacks—are printed this way. Deacidification processes have been developed that help prolong the life of books printed on acid paper, but they are expensive and time-consuming.

Nor is paper the only—or even the principal—problem. Other parts of our heritage have been irretrievably lost. Until 1951, the only type of film that was available for movie production contained nitrate, which caused it to decay quickly, even in controlled environments. Today half of the 21,000 feature-length films made in the United States before 1951 no longer exist. Those that have not been lost or destroyed have decomposed beyond repair. Similarly, analog audio recordings on wax cylinders or magnetic tapes need to be preserved by transfer onto digital formats, such as CD-ROM. While CD-ROMs are not necessarily long-lasting—certainly not compared with the acid-free paper used in the old days—at least a process of regular copying can be established to preserve digital material without loss.

A sorry tale

Despite its very short history, computer technology has an unbelievably bad record when it comes to preservation. There are many examples. A 1990 U.S. government report cited several cases of

significant digital records that had already been lost or were in serious danger of being lost. For example, the 1960 U.S. census narrowly escaped oblivion, because it was originally stored on tape that became obsolete faster than expected.

The Domesday Project

The Domesday Project is another example that exemplifies the problem of preservation, notable for its irony. The year 1986 was the 900th anniversary of the Domesday Book, a document commissioned by William the Conqueror to provide a record of the land he had conquered 20 years before. The original book, handwritten on parchment by (probably) a single monk, still exists and is kept at the U.K.'s Public Record Office (now part of the National Archives) in Kew, near London. The book sets out a record of all the lands in the kingdom and who held them. Of course, few of William's subjects saw the results of this labor—most couldn't read.

The Domesday Project was a national information-gathering exercise conceived and coordinated by the BBC in London to commemorate the anniversary. Schools around the United Kingdom were asked to survey their areas to produce a database of text and pictures that recorded how Britain looked in 1986. The materials provided by the schools were combined with central statistical, written, and visual information. The idea was to capitalize on the wide availability of microcomputers in British schools and libraries to capture and present the information uniformly.

Today, museums with publicly available interactive multimedia systems are commonplace: the Domesday Project was one of the first. Its scale was unprecedented: over a million people took part in one way or another. The information was recorded on two double-sided 12-inch laserdiscs, a quasi-analog format that was the forerunner of today's CDs and DVDs. One disk contained contributions from schools, and the other disk contained census and related data from government agencies, along with professional photographs and video footage. The laserdiscs contained analog photos and video footage, mixed with encoded digital data.

Demise

Most of the people who contributed to the database never saw the result of their labors—even though they could read. At a price of £4,000 (U.S. \$6,000), the disks cost roughly four times more than originally planned and couldn't be afforded by the organizations to which the project was targeted, so the project never saw the uptake its creators had planned. Apparently it would take over seven years to look at everything on the disks—but you would have had to work fast, because long before that time had elapsed, the system had vanished, almost without a trace. In 1996, on the project's tenth anniversary, a plea went out for information on any installations that were still available for public or research access. Ironically, at that point it would have been easier to go and see the original Domesday Book, then 910 years old. (Although only a few scholars are allowed to see the actual original, a handsome facsimile is on display in the U.K.'s National Archives.)

Resurrection

Following this debacle, the project underwent resurrection—albeit in a rather uncoordinated way, involving three groups that initially worked independently. In 1999, a joint Anglo-American university project began with the aim of developing and evaluating technical strategies for the long-term

preservation of digital material suffering from digital obsolescence, including resurrecting the data from the Domesday Project as a case study. The project's participants developed an emulator to access the information stored on the disks, and they demonstrated it three years later.

Meanwhile, a programmer named Adrian Pearce set about reverse-engineering the format on his own—without any documentation! Pearce was motivated by his wife, who in 1986 had led one of the original data-gathering teams and wondered if it was possible to make the information available to her current students. Both worked at the University of Brighton and, amazingly, not only did the university still have the equipment, but it was found to be in working order. Again independently, the U.K. National Archives became interested in resurrecting the work, basing their work on the Domesday terminal that had been bequeathed to them on completion of the 1986 project. The BBC helped them locate copies of the original 1-inch master tapes that contained the photos and videos, and these were digitized at a higher resolution than what was available on the laserdiscs.

The three teams met for the first time in 2002, at the above-mentioned demonstration of the emulator. In contrast to the Anglo-American approach, Pearce had developed a fresh application based on the extracted data, inspired by the original Domesday Project but including some innovation. Subsequent to this meeting, the higher quality digitized photos and videos produced by the National Archives were incorporated into Pearce's application, and today a public version of the application is available to visitors of the National Archives at Kew.

Following this success Pearce brought out a Web-based version in 2004. [Figure 9.8](#) shows a snapshot that displays information about Hastings, the location of the decisive battle in 1066 between William the Conqueror and King Harold. The screen is divided into quadrants displaying a map, text, photos, and search results. The map is clickable and supports zooming and panning. The text area displays section headings for Hastings, among them Introduction, Shops and Prices, Tourism, The Battle of Hastings (of course), Population, and Housing—all hyperlinked to further information. Below is a photo of the location; more are available through the "More Photos" button. Clicking a photo produces a higher-resolution version, with a caption. Photographic quality varies considerably, and users can lighten or darken the images. The final quadrant of the display allows searching.

Impressive as these technical feats of achievement have been, there is a social dimension that presents even bigger challenges than the technical ones. For instance, it is quite feasible for the Anglo-American team's emulator to be shared on the Internet, but this has not been done because of copyright concerns. Does the emulation constitute a copy or a new version? Apparently there is no definitive legal answer to this question, and the matter is still unresolved years after the emulation project was finished. The National Archives, working in concert with the BBC, have fared better in the legal realm, but with the untimely death of Pearce, the project's future is once again uncertain.

The digital dark ages

"Let us be absolutely clear from the outset," warns an article called "A Digital Dark Ages?" "no one understands how to archive digital documents." In stark, uncompromising language, the article develops several points.

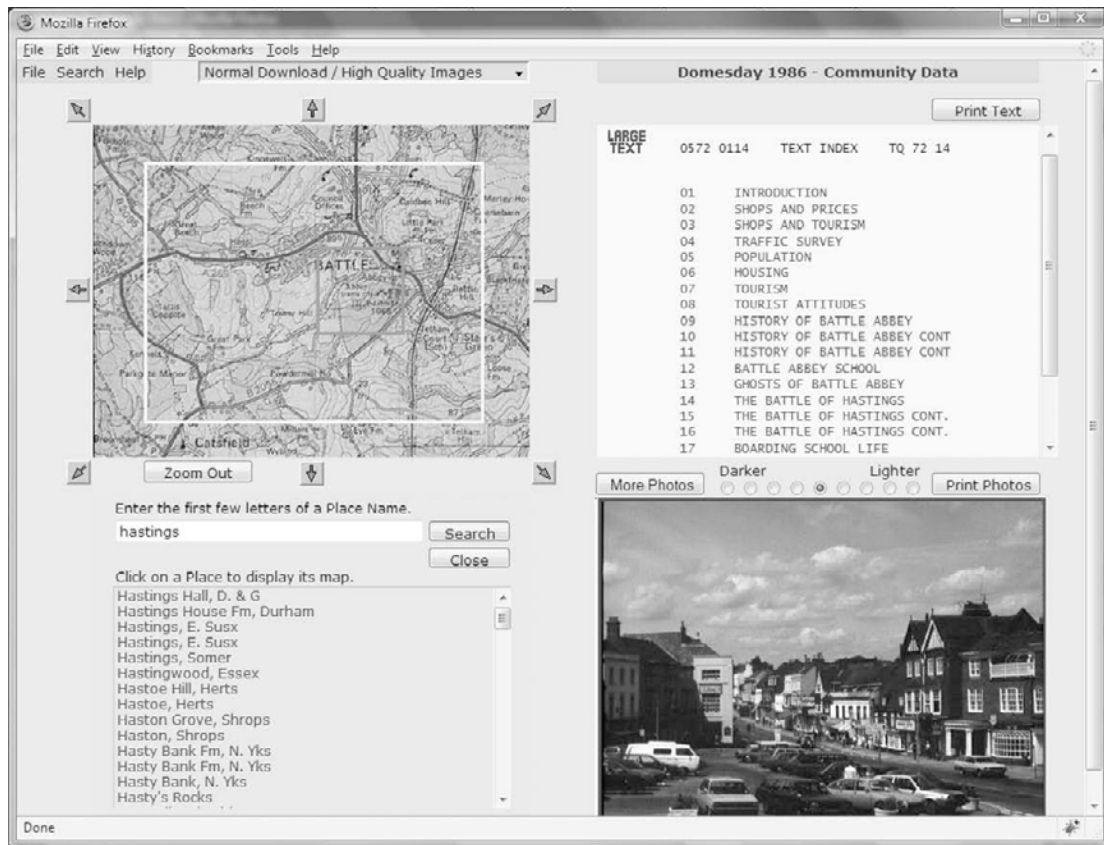


Figure 9.8: Entry for Hastings in the Web-based Domesday Project

First, with regard to preservation technology:

- Enormous amounts of digital information are already lost forever.
- Information technologies become obsolete very quickly.
- Document and media formats continue to proliferate.
- Technology standards will not solve fundamental issues in the preservation of digital information.

Second, with regard to the availability of material:

- Libraries will shortly see a demographic bulge of electronic material as the baby boom generation of authors and academics contribute material gathered during their careers.
- Much material will never make it into library collections for preservation because of increasingly restrictive intellectual property and licensing regimes.
- Archiving and preservation functions in a digital environment will increasingly become privatized as information continues to be commodified.

Third, with regard to the traditional library functions of archiving and preservation:

- Financial resources available to libraries and archives continue to decrease.
- Libraries and archives will be required to continue their existing archival and preservation practices as the current paper publishing boom continues.

This paints a gloomy picture. And it is not just one person's view. In 1996 the U.S. Commission on Preservation and Access issued the final report of the Task Force on the Archiving of Digital Information. An impressive group of 21 experts had spent a year studying the problem. The conclusion was alarming—there is, at present, no way to guarantee the preservation of digital information. The first line of defense against loss of valuable digital information rests with the creators, providers, and owners of that information. Unless something is done, it's every one for him- or herself.

A practical problem with preservation is that issues arise at different time scales. Many organizations are faced with a crisis: an urgent short-term need to save documents that are in imminent danger of becoming lost, documents that are already difficult to access. Fortunately, the vast bulk of information is not in immediate danger, but something must be done now to prevent it from being vulnerable to imminent loss in the medium-term future. In the long term, strategies must be developed that do not require continual emergency operations to be mounted on an ad hoc basis—strategies that are robust to unexpected technical and, particularly, social changes. Once lost, material is gone forever.

Preservation strategies

Digital documents are vulnerable to loss because the media on which they are stored decays and becomes obsolete. They become inaccessible when the software needed to interpret them, or the hardware on which the software runs, becomes obsolete. A luminary in the field declared ironically that “digital information lasts forever—or five years, whichever comes first.”

The situation is paradoxical. Digital formats have many advantages over analog ones, advantages that seem to *promote* preservation. Their prime asset is the lossless nature of copying: whereas photocopies of photocopies degrade, digital copies of digital copies are identical to the original—and automatic error checking can be incorporated to eliminate the effect of analog interventions, such as transmission errors. However, along with this cardinal advantage are others whose benefits quickly turn into drawbacks, such as

- ease of creation and copying
- independence of physical media
- constant improvement in hardware and software.

The problem is that these apparent advantages actually make digital preservation even harder, as a little thought shows. Ease of creation causes information glut. Ease of copying means that it is not clear which is the original—so every “copy” seems dispensable. Independence of media means that it seems hardly worth expending effort on saving the physical artifact, even if that were possible.

Constant improvement in hardware and software promotes obsolescence—cynics say this is what drives the computer industry.

“May all your problems be technical ones” is a blessing that nerds, among others, bestow upon one another. Computer people recognize that the technical stuff is the easy bit. It’s the human part that causes problems. The human interface is by far the largest and most complex part of virtually all of today’s software systems. Administrative problems consume time and beget frustration. Political processes like standardization require negotiation, compromise, and bargaining. It is the technical problems that have solutions that yield to honest intellectual work.

As we said earlier, preservation is not principally a technical problem, unfortunately. One of the approaches promoted throughout this book is to adopt open standards, for proprietary systems that keep information in secret formats are the bane of long-term preservation. Apart from this, there are five basically different preservation strategies:

- printing to paper or microfilm
- preserving hardware and software in museums and archives
- duplicating content far and wide
- emulating hardware and software
- migrating systems to newer versions.

The first two, printing material on paper (or microfilm) and preserving the technology in museums and archives, are not usually taken seriously as long-term preservation strategies, although we have seen where the latter paid off (by accident rather than design) with the Domesday Project.

The third is encapsulated in the slogan *lots of copies keep stuff safe*. LOCKSS is the name of an open source software system that allows libraries to collect, to preserve, and to disseminate material to its own readers: either open access material or published material to which it has subscribed. Library computers collect their copies using a specialized Web crawler that verifies that the publisher has granted appropriate permission. Libraries that have collected the same material cooperate to ensure its preservation using a networked system that automatically checks that all copies are the same. A participant whose copy is damaged will find that it differs from the majority’s copy and repairs it from the publisher or from other participating libraries.

The fourth and fifth preservation strategies highlight the distinction between preserving the physical stream of bits that constitutes each document and preserving the logical means by which these bits are interpreted as a document. The two mechanisms are independent. Although retaining the integrity of the physical bitstream is what first springs to mind when thinking about preservation, preserving the means by which it is interpreted is more taxing.

Emulation involves retaining the documents in exactly the same form as they are and emulating the functionality of the original, obsolete system on future, unknown systems, so that a digital document’s original software can be run in the future despite being obsolete. This is the route taken in the first Domesday Project resurrection, with notable success—except that the successful engineers were

not the principal copyright holders. To preserve the physical bitstream involves regular copying to new media, the application of error detection to determine whether degradation is occurring, and error-correcting codes to ensure that new generations are faithful copies of the original. To preserve the logical interpretation requires emulating old interpreters on new hardware—either by incorporating backward-compatibility into software, or by compiling special “historic” versions of the software that emulate its functions. For example, current incarnations of Microsoft Word can read (most) old Word documents, even though the formats are quite different: this is backward-compatibility. Although neither Microsoft nor Word may exist in 50 years, it may be possible to emulate the crucial parts of Word’s functionality on then-current hardware and so read and display old documents.

This example of Word highlights one of the problems. As discussed in Chapter 4, the binary file format for Word has until recently been proprietary. Without inside knowledge, you cannot write software even today that reliably reads old Word files. An important feature of any format used for preserving documents is that it is *open*: the details are made publicly available. And as well as being open in principle, it must be open in practice: documented well enough for others to understand and build their own interpreters. PostScript and PDF are good examples, and Open Office XML and the Open Document Format are two initiatives that seek to address this imbalance in the realm of editable office documents.

Migration involves translating the document from the old format, designed for now-near-obsolete software, to one that is accepted by new software. This involves not just copying the physical bitstream to new media, but translating it at the same time into a new logical format. The difficulty of this operation depends on the details of the software upgrade, but it will normally be very easy. For example, you might go through all your old Microsoft Word files, reading them into Word and writing them out in the latest version of the format. Word provides this functionality of course—no one would upgrade to new versions if they were unable to read at least the last generation file format—and, for the same reason, virtually all serious software will always provide this facility.

Emulation or migration? There are arguments on both sides. Migration may be cheaper, because no special emulation software needs to be written. Conversion software is almost always available if it is invoked in a timely manner. But proponents of emulation point to the fact that conversion is a kind of translation, and translation often loses features of the data. If each translator in the chain were reversible, the original document could be reconstituted. But this is not necessarily the case—at best, subtleties like format, font, footnotes, cross-references, citations, headings, and color might be sacrificed; at worst, entire segments, such as graphics or sound, might be omitted, or the whole document could be meaningless garbage. Would a modern version of the beautiful tale of *Beowulf* have the same literary impact if it had been translated through a series of intermediate languages, rather than from the earliest surviving text in ancient Anglo-Saxon?

These copying operations are actually inexpensive, particularly for a digital library. It is not as though the documents have to be copied by hand, as the Alexandrian scribes did with the Athenian manuscripts, or individually deacidified, as conservationists are doing with yesterday’s books. Once the procedure has been established, you can start a batch job that runs through all the documents and

converts them. For a large collection, the expense, amortized on a per-document basis, will be negligible. There will be ancillary costs, of course. Newly converted documents must be sampled and checked manually against the originals to ensure that the procedure is operating properly. Safeguards must be put in place against unexpected problems. You don't want to wind up with the digital equivalent of what happened to Richard Stallman's manual for EMACS, where, as mentioned in the Notes and Sources section at the end of Chapter 3, only every other page was copied. You will need to satisfy yourself that the new collection is complete and correct before finally destroying the old versions. You may need to change the digital library system to use the new format. But, again measured per document, the cost is minimal.

The danger is that you might miss a generation. When you come across a Word file from long ago, current versions may fail to read it. When you encounter a dusty old magnetic tape, the drive might be unable to read it for physical reasons, or you might not be able to find a suitable drive. All it takes is one broken link to lose the documents forever. Through fire and flood, pestilence and famine, war and revolution, the procedure must go on forever. Forever is a long time. The problem is administrative, institutional, and political, not technical.

And it is all very well to say that copying is inexpensive per document, but it is not so overall, and copying is a continual operational expense rather than a one-off cost. When it comes to copying from one media to another, and converting from one format to another, who will pay?

Preservation in practice

National libraries, archives, and others have been grappling with the issue of digital preservation in earnest since the beginning of this century. They must wrestle with the tension that arises from the dual issues of access and preservation—including the contrast between preservation in perpetuity and the fixed-term political appointments that hold the purse strings.

Public institutions want to make their content more accessible. We are witnessing a great upsurge in nationally funded digitization projects—such as Papers Past (Figure 3.4). Libraries, museums, and archives proudly declare that they are creating a national identity and putting it out on the Web for people to experience: a showcase of cultural memory. Such initiatives please their political masters. And cultural memory is indeed an integral and vital part of healthy modern societies. It benefits people by contributing to their enjoyment and inspiration, promoting their sense of identity through shared values, and enriching their lifelong learning potential.

But what about sustainability? The work undertaken should not be the technological equivalent of an evolutionary dead end, but the rate of change in Web technology makes this difficult to avoid. The New Zealand National Library's Papers Past in Figure 3.4 is actually the third significant incarnation of this resource; in this case the latest incarnation stemmed not from obsolescence of equipment or file formats, but due to a change to new software that provided enriched ways to deliver the content. The whole endeavor is an expensive and time-consuming process. Just in the small country of New Zealand, the National Library has identified three million newspaper pages that are not held anywhere else in the world and are consequently at risk. They have embarked upon the process of microfilming and digitization, which at current funding levels will take two decades. After that, there

is more work to do: there are a further two million archived newspaper pages that are neither micro-filmed nor digitized, and 24 million undigitized pages already on microfilm.

At the same time, libraries need to cope with the rising tide of content born digital. In some countries, legal deposit legislation imposes a frantic urgency on national libraries to put in place infrastructure for preserving digital documents. But, as pointed out in “A Digital Dark Ages?” the long-predicted decline in print publishing has failed to materialize, and the existing infrastructure must be maintained—and expanded—too. A 2004 survey revealed that major libraries were adding kilometers of shelving space—in one case, 10 km *per year*—and that they still preferred print for long-term preservation, even when digital material is also supplied.

The British Library has set itself the target of placing all its digital collections in a secure repository by 2016. This will use an open-ended range of preservation strategies, including normalization, migration on request, emulation, and “desiccation”—the last referring to documents derived from the originals that are inferior quality but capture their main scholarly value in a less perishable form. The implementation plan includes

- storing multiple copies of each digital object in different physical locations,
- utilizing backups and integrity checking to ensure that nothing is lost due to media decay or catastrophe,
- characterizing and validating each object in order to determine requirements for its preservation,
- recording metadata about each object and allocating unique, persistent identifiers to ensure that none will be lost.

The first two points are technical; the last two require human mediation—and underline once more why preservation is principally a social issue. The third recognizes that different preservation solutions are appropriate in different situations. The fourth involves the kind of preservation metadata discussed in Chapter 6.

The national libraries of Germany and the Netherlands have been working on a software architecture that leverages a decade of research on digital preservation at IBM. As a foundation, it requires the implementation of a carefully prescribed Universal Virtual Computer. A workflow and data representation scheme is built upon this that includes a traceable path of provenance for any digital object and authentication for both bitstreams and metadata.

The Internet Archive

We met the Internet Archive in Chapter 5, where we described how it is a vast treasure trove of resources spanning many forms of digital media, including an Audio Books and Poetry collection (Figure 5.7). Prior to its founding in 1996 there had been no attempts to capture a comprehensive record of the text and images in the documents that appear on the Web. The founder of the Internet Archive, Brewster Kahle, noted that the history of print and film is littered with stories of loss and partial reconstruction, and he sought to avoid this scenario for the Web, which even then, only a couple of years after its inception, was evolving into a storehouse of scientific, cultural, and historical information.

The Internet Archive's mission statement is:

Most societies place importance on preserving artifacts of their culture and heritage. Without such artifacts, civilization has no memory and no mechanism to learn from its successes and failures. Our culture now produces more and more artifacts in digital form. The Archive's mission is to help preserve those artifacts and create an Internet library for researchers, historians, and scholars.

The archive has produced the Wayback Machine, a free Internet service that allows users to see archived versions of Web pages over time. In fact, in the Notes and Sources section of this chapter, we use it to ascertain the status of the Web site for Adrian Pearce's resurrection of the Domesday Project mentioned earlier. The Internet Archive also provides a subscription service for users who want to ensure that their material is archived immediately, perhaps so that they can cite an archived version. Such material is periodically amalgamated with the general Wayback Machine.

One fascinating aspect of this project is the controversies it has sparked, which highlight the difficulties of archiving material—particularly material that is not your own. People have had mixed success in citing the Wayback Machine archives as a source of admissible evidence in court. The United States and European patent offices accept its date stamps in order to determine whether a Web page is available as prior art for patent applications. The Internet Archive strives to exclude material belonging to those who do not want their material to be archived, and it deletes material upon request of the creator or site owner. (Of course, this does not necessarily prevent the violation of copyright laws.) There have been problems when the Archive responded to requests by people other than the site owner, resulting in the removal of material that owners did *not* want removed.

The entire Internet Archive is mirrored at the Bibliotheca Alexandrina in Egypt, a library and cultural center located on the shores of the Mediterranean, which commemorates the ancient Library of Alexandria.

9.3 Trends in Digital Libraries

There are many noteworthy themes in digital library research. The Personal Information Retrieval Assistant described in [Section 9.1](#) exemplifies two: it embeds authorship in a digital library environment, and it includes integration with external services through a “mashup” that searches other resources. Other themes include mobile or portable digital libraries and knowledge-based information retrieval. To illustrate, we describe below how the iPod personal media player has been repurposed as a large self-contained digital library that can literally fit in your pocket, and then show how traditional searching can be augmented with semantic information gleaned from Wikipedia to provide knowledge-based information retrieval.

Mobility: Portable collections

Mobile digital libraries usually imply network connections from handheld devices to digital library servers. Instead, imagine being able to carry a digital library around with you. Through searching and

browsing, you can have full fingertip access to millions of items—text, image, audio, or video—wherever you are. Wherever you are on earth (or in space): there’s no need to be at a wireless hotspot or to register with an Internet service provider. No waiting for rich-media content to be transferred to your device, not to mention the accompanying rapid depletion of battery power that goes with all that communication. Everything travels with you: no restrictions on airplanes, no worrying about connectivity in other countries. Your own personal copy of a digital library, right there in your pocket. Take it wherever you go—try doing that with the bricks-and-mortar counterpart!

A low-cost personal multimedia player like Apple’s iPod can store an astonishing volume of information, and can be programmed to incorporate your own software—such as a digital library application. In terms of text storage, consider a medium-sized university library with a million books. At 80,000 words per book and 6 letters per word (including the inter-word space), each book comes to half a million bytes, maybe 150,000 bytes when compressed. The whole library amounts to about 150 GB, less than the capacity of today’s high-end portable devices. This is text only: it does not allow for illustrations.

What about illustrated books? It’s hard to generalize about their size, but our experience with collections of practical information—like the Humanity Development Library illustrated in Figure 3.1, where 2,000 fully illustrated, and fully indexed, books fit comfortably on a CD-ROM—indicates that a 160 GB iPod would hold half a million such books. Alternatively, it could accommodate 250,000 pages from the Papers Past collection (Figure 3.4), including the full-text index, highlighting search terms in the images, and metadata to allow switching between page and article views. Many more articles could be stored if word highlighting, which consumes half the space occupied by supplementary files, were omitted.

And multimedia? Over 2,000 hours of audio or 200 hours of video fit comfortably onto the device. This is particularly attractive if users are illiterate or semi-literate (see Section 9.4 below). And cost need not be a big concern, for there is a buoyant and growing market in second-hand personal media players.

Figure 9.9 shows snapshots taken when browsing an iPod-touch digital library prototype we have developed using Greenstone. The application is launched by pressing the Greenstone (g^2) icon on Apple’s Springboard (Figure 9.9a). From the resulting display of collections the user chooses *The Complete Works of Shakespeare*. Traversing the subject hierarchy for *Histories*—touching an item scrolls the screen left to show the next level—the user reaches Figure 9.9b three screens later. Touching *The Tragedy of Antony and Cleopatra* brings up an illustrated version of the play (Figure 9.9c), and the standard features of the iPod can be used to pan and zoom around the document—panning by dragging a single finger across the screen, zooming in or out by pinching two fingers together or opening them out. As usual on the iPod, the view automatically switches from portrait to landscape when the device is tipped on its side. Along the bottom of the screen in Figure 9.9d are tabs for Browse, Search, and Preferences. The user selects Search and enters the term *murder* in Figure 9.9e using a virtual touch-keyboard. From the list of results in Figure 9.9f, the user selects *Romeo and Juliet* and views the result in Figure 9.9g. The Preferences tab gives the usual control over searching, for example, stemming and case-sensitive matching, and also lets users change the language of the interface.

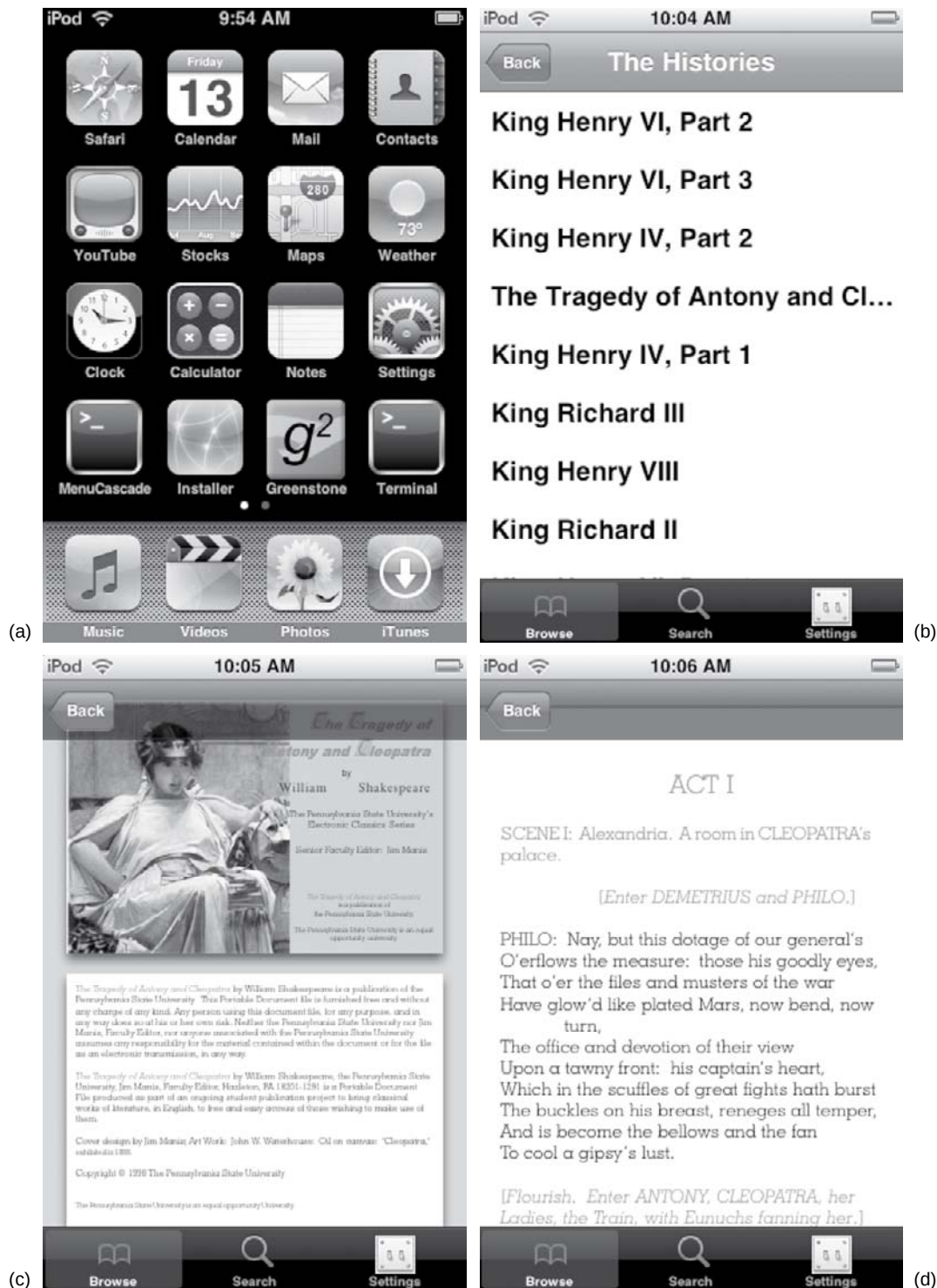


Figure 9.9: Portable digital libraries: (a) Springboard with Greenstone as one of the applications; (b) browsing Shakespeare's *Histories*; (c) *The Tragedy of Antony and Cleopatra*; (d) zooming in to the beginning of Act 1;

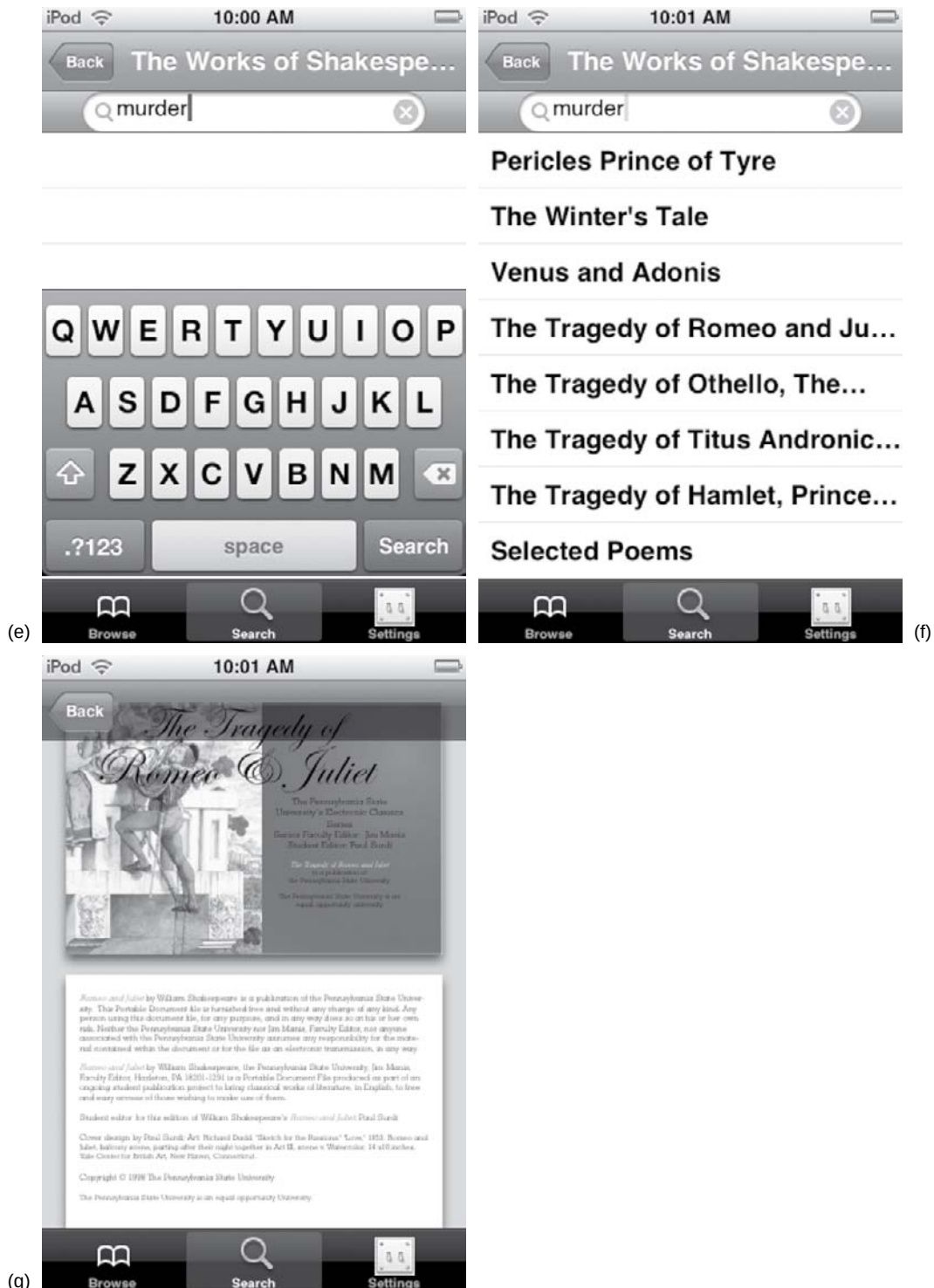


Figure 9.9, cont'd: (e) searching for *murder*; (f) search results; (g) *Romeo and Juliet*

Carrying a digital library around is nice, but the miniature screen is a clear disadvantage. One solution is for the portable device to serve collections to a workstation or over a local area network. In either case, Greenstone can accomplish this using a built-in Web server, the same way it serves its collections in the normal course of events. Fortunately, a Web server is pre-installed on the iPod-touch—at least on a jail-broken device. Alternatively, it could be used in disk mode, with pre-compiled Web-server binaries for popular host machine architectures. Whenever the iPod is plugged into a workstation, the appropriate server starts up. The result is that as soon as the standalone digital library device is connected to a host machine, it begins to serve the same content over the local area network—or beyond. Portable, self-sufficient digital libraries have many interesting potential applications. Greenstone has also been ported to the Android software platform for mobile devices.

Knowledge-based information retrieval

It is tempting to imagine that full-text searching solves the entire problem of finding information in a digital library—or on the Web. But this is far from the truth. How to find new information presents a very old philosophical conundrum. Around 400 B.C. the Greek sage Plato spoke of how his teacher Socrates examined moral concepts like Good and Justice, important everyday ideas that are used loosely without any real definition. Socrates probed students with leading questions to help them determine their underlying beliefs and to map out the extent of their knowledge—and ignorance. In a discussion about Virtue, Socrates' student Meno stumbled upon a paradox:

And how will you enquire, Socrates, into that which you do not know? What will you put forth as the subject of your enquiry? And if you find out what you want, how will you ever know that this is the thing that you did not know?

In other words, what is this thing called search? How can you tell when you have arrived at the truth when you don't know what the truth is? Digital library users, this is a question for our times.

How can you describe the unknown? That is precisely what must be done to formulate a cogent query. To make matters worse, search engines, unlike people, are incapable of reasoning about descriptions. Search engines treat queries as nothing more than excerpts—a few words—from a relevant document. To search effectively, one must predict not only the information that relevant documents contain, but also the terms by which it is expressed. In short, you have to know a great deal of what is being sought in order to find it.

What knowledge seekers need—at least those who are not clairvoyant—is a bridge between what they know and what they wish to know, between their vague initial query and the concrete topics and terminology available in the library. One possibility is a thesaurus: a map of semantic relations between words and phrases. People who cannot identify effective terms for their query could benefit from a thesaurus that covers the terminology of both documents and potential queries and that understands the relations between them. Those who cannot formulate a query at all could use a well-organized thesaurus that exposes the topics available in a way that they can be explored.

Thesauri have the potential to greatly advance the art of information retrieval. In practice, however, they are not widely used. The trouble is that generic thesauri have shortcomings in any particular technical domain. Domain-specific thesauri exist, but they are expensive to produce, and their use

often requires specialist technical knowledge. Thus they are available only for a small proportion of document sets and appeal mainly to expert users.

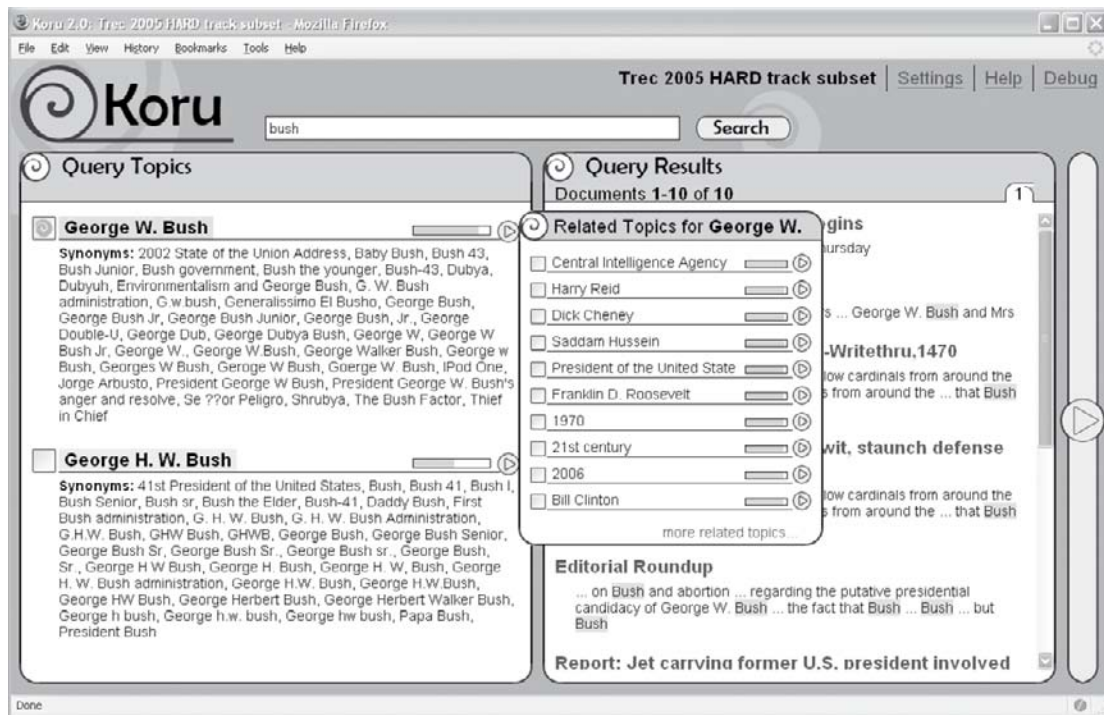
Figure 9.10 depicts a prototype digital library that strives to overcome these limitations by using the information in Wikipedia as a vast general-purpose thesaurus. (We introduce the idea of using Wikipedia as the basis for a thesaurus in Section 6.6.) Wikipedia is a constantly evolving encyclopedia containing manually defined structures that can be tailored to expose the topics, terminology, and relationships in individual document collections. The system illustrated exhibits an understanding of the topics involved in both queries and documents. It allows documents to be matched accurately, and queries to evolve interactively or automatically.

The word *bush* has been entered as a query in Figure 9.10a. Below the search box are two panels: the left one for query topics, the right one for query results. There is a third panel for viewing a document, out of sight but accessible by clicking the vertical bar on the right. This collection is a body of 1,200 newswire stories concerning a wide range of topics. When the search button is pressed, two items appear as query topics, and a list of documents as query results (the latter are partially obscured by a pop-up window). Using the taxonomy and link structures derived from Wikipedia, two interpretations of the word *bush* have been identified, namely former presidents *George H. W. Bush* and *George W. Bush*. Many synonyms for both appear below the query topics. There are many other interpretations of the query term, such as *small tree*; the system filters them out automatically because they are irrelevant to the document collection.

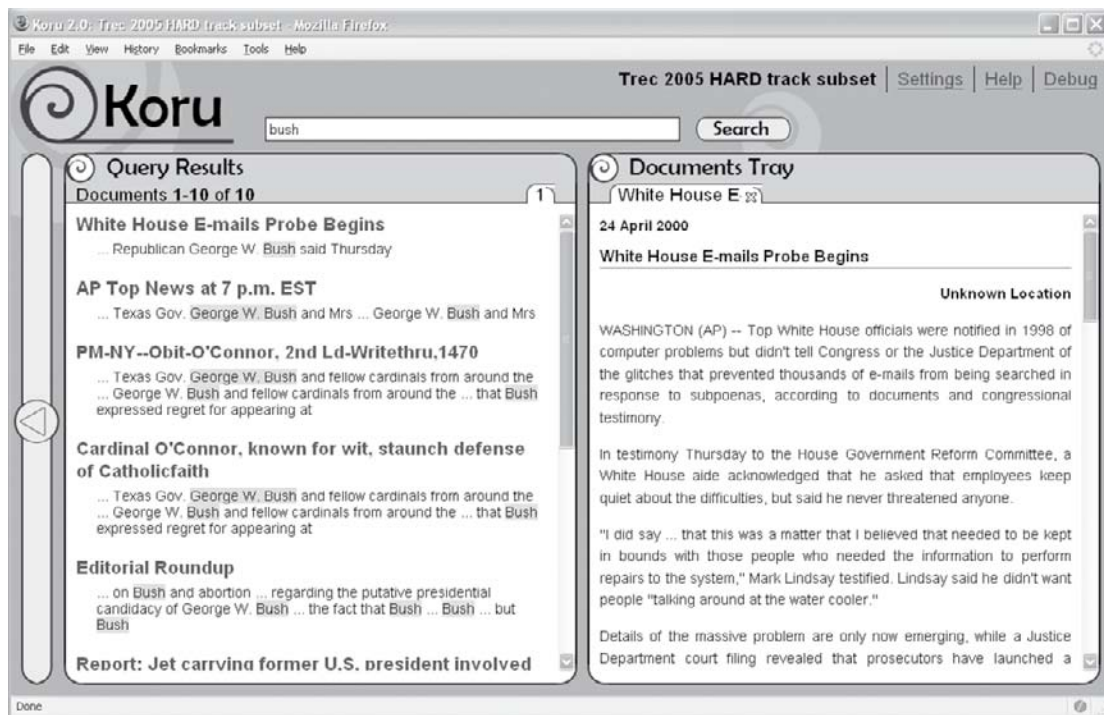
The topic *George W. Bush* is preferred, because it occurs most frequently in the document collection, and the results contain documents that match this topic only. Occurrences of *Bush* and *George W. Bush* are highlighted. If they wish, users can click the selector box beside the alternative query topic instead of the one beside the first query topic—or in addition to it, since multiple topics can be chosen—and the search results would change accordingly. Users can also add new topics to the two that have been identified. Instead, our user explores terms related to *George W. Bush* by clicking the triangular green button next to the query term. A list of related terms pops up, any of which can be selected by clicking its checkbox—whereupon that term is added to the query topics panel and the results panel updates immediately. Here, *Central Intelligence Agency*, *Saddam Hussein*, *President of the United States*, and *Bill Clinton* are among the related terms. Any of these can be expanded in the same way by pressing their triangular green button, and the user can continue to explore the hierarchy of related terms. Any useful terms that are encountered can be added to the query topics panel with a single click.

Instead of refining the query, the user turns her attention to the results panel. Clicking on the first item, *White House E-mails Probe Begins*, scrolls the screen leftward to reveal the third panel, the Documents tray shown in Figure 9.10b. The selected document is displayed here, and the query results are now on the left. The user can click other items in the result set to view them. The vertical bar on the left scrolls the screen rightward to bring the query topics panel back into view, returning the display to Figure 9.10a.

This digital library interface seamlessly blends searching and browsing. A query term or two are needed at the start, but users switch naturally between browsing and searching modes without being



(a)



(b)

Figure 9.10: A new search interface: (a) searching for *bush*; (b) first matching document for *George W. Bush*

consciously aware of any difference. Perhaps this confluence illustrates what will eventually become the future of computer-based information access.

9.4 Digital Libraries for Oral Cultures

Libraries are about literature. Our dictionary defines literature as “the writings of a society, in prose or verse,” and goes on to add, “Broadly speaking, literature includes all types of fiction and nonfiction writing intended for publication.” Thus, literature seems to be firmly dependent upon writing. But the earliest cultural tradition in New Zealand, for example, is that of the Māori, whose literature consisted of history, tales, poems, and myths handed down by oral tradition. Only after missionaries arrived from Europe were these works written down and transformed into “literature” by the dictionary definition. We have much to learn from early cultures. One trend in modern writing, for example—particularly Australian and North American—is to draw on the oral storytelling traditions of aboriginal cultures.

The ultimate aim of multimedia digital libraries is to give nontextual material first-class status in a digital library—if you like, first-class status in “the literature.” This has important cultural ramifications. It should be possible to create digital library collections intended for use by people in oral cultures, who may be illiterate or semi-literate, or by people who, although they can read and write their own language, cannot speak or read the language of the digital library.

Imagine having access to collections that spring out of the rich cultures of China or Arabia, created by people who grew up in these cultures, without having to learn a new language. More practically—since you, dear reader, are culturally privileged and can probably access this kind of information in translation—imagine giving someone in the highlands of Peru, fluent and literate in her native language of Quechua, first-hand access to the information in humanitarian collections like the Humanity Development Library (currently available only in English and French) or the Biblioteca Virtual de Desastres (a collection of information dealing with disasters and emergencies, currently mainly in Spanish).

If you find it difficult to imagine how serious practical information could possibly be conveyed in purely iconic form, look at [Figure 9.11](#), which shows how to splint a broken forearm. It is taken from a 120-page manual on first aid that contains not a single written word. Literate as you undoubtedly are, in an emergency you might even prefer these pictures—which in the book are vividly colored, adding further clarity—to a textual account of how to deal with a medial radius fracture. Another striking example is the “user manual” for Mexico City’s underground transport system: it is a brilliant iconic design, with no words whatever, that gives clear instructions on how to use the metro. A third example, and a historical precedent, are the so-called Beggar’s Bibles (*Biblia pauperum*), which were designed in the Middle Ages for the religious instruction of illiterate believers. They used numerous illustrations to explain passages from the Bible and have been called prototypes of our present-day comics. Produced before Gutenberg, each printed page was carved in a woodblock. You can see them today at Wolfenbüttel.

These examples are static images. Imagine what you could do with motion, sound, video, interaction, 3D objects, simulations, and virtual reality.

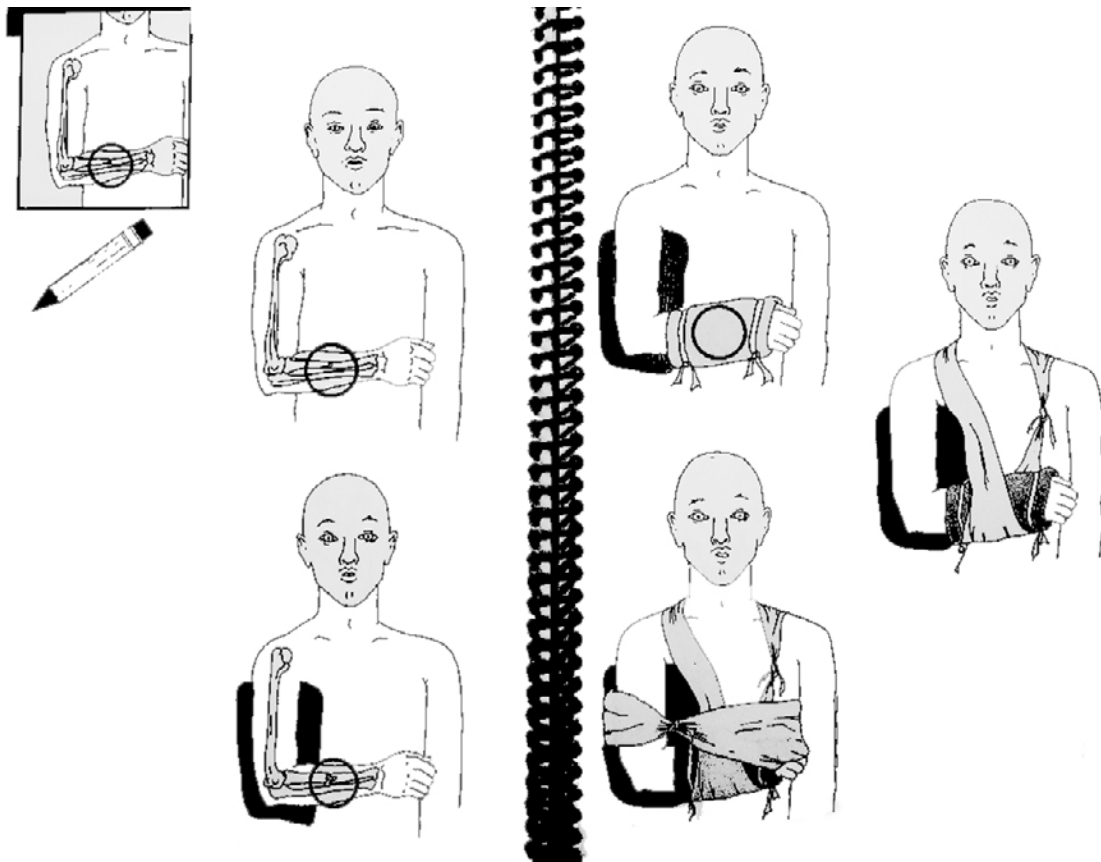


Figure 9.11: First aid in pictures: how to splint a broken arm. *First Aid in Pictures*, by M. B. DeLong, J. G. Brady, L. D. Bourgeois, and L. J. Niemiec. Barbara A. Bear, Illustrator. Edward L. Meador, Editor. Vade Mecum Press, 1988

The apparently paradoxical notion of libraries for the illiterate has its resolution in the notion of generalized documents. The challenge is to design digital libraries in a way that they are usable by people from oral cultures—not so different from designing searching and browsing mechanisms for non-textual material. In fact it is even possible to imagine such people creating their own digital library collections. The LINCOS project, for example, equips villages in developing countries in Central and South America with information technology, housed in ordinary transportation containers. The units form a kind of “school,” typically attended by children and elderly people—the economically active segment of the population does not enjoy the luxury of leisure time for continuing education. In the LINCOS units, people—including the semi-literate—enjoy learning to use computers for creative and artistic activities like drawing, capturing photographs, and making movies. Suitably equipped, the modern microcomputer transcends text.

With the advent of low-cost consumer technology for image and movie capture, a quiet revolution is taking place in our homes. Now anyone can keep a photograph album on a home computer, or shoot video and use sophisticated editing techniques to produce a professional-quality movie, or make a CD-ROM of one's own music (or that of others). Ever since the advent of broadcast television, our text-dominated society has gradually become attuned to the more visceral medium of moving images. We book lovers may deplore the decline of the printed word, laud the sustained argument carefully built up over pages, and praise the power of the written word to conjure up far more imaginative and vivid imagery than any TV can. We may lament the decrease in attention span, the reduction of arguments to sound bites. We may wish that our children spent more time reading books and less time playing video games. But we must live in the world too, and the world is changing.

Perhaps digital libraries for people from oral cultures will find a place in our own society, too, and help reduce the various "digital divides" that cleave our world—the "social divide" between the information-rich and the information-poor in our own nations; the "democratic divide" between those who do and those who do not use the panoply of digital resources to engage, mobilize, and participate in public life; and the "global divide" that reflects the huge disparity in access to information between people in industrialized and developing societies.

Is this too much to hope?

9.5 Notes and Sources

We thank Rob Akscyn for the vision of digital libraries as kitchens for knowledge preparation; it has greatly influenced our thinking. Back in 1999, Steve Jones conceived and constructed a system called Phrasier that illustrated these ideas (Jones, 1999; Jones and Paynter, 1999; Jones and Staveley, 1999). The Personal Information Retrieval Assistant (PIRA) illustrated in [Figure 9.6](#) pursues and updates these ideas in a Web context (Twidale et al., 2008). You can try PIRA yourself at writencite.com. More details about MONK can be found through the Web site www.monkproject.org.

The government report that cited cases of digital records that had been, or were about to be, lost—including the 1960 census results—was published by the U.S. Congress (1990). More studies of preservation and access can be found at the Web site of the U.S. Council on Library and Information Resources, www.clir.org.

A report on the Domesday Project, written by Andy Finney, appears at www.atsf.co.uk/dottext/domesday.html. Finney has kept the report up to date for over a decade. Of the many aspects covered, the detail it includes about the video digitization is of note. For more information about the role Adrian Pearce played, see Darlington et al. (2003). At the time of writing the home page for the commercial project Pearce founded, domesday1986.com, was inactive, and, according to the Internet Archive's Wayback Machine, it has been inactive since early 2008.

The changing world of preservation is a matter of serious concern for democratic societies. As George Orwell said in *Nineteen Eighty-Four* (published in 1949), "Who controls the past controls

the future. Who controls the present controls the past.” It was Jeff Rothenberg (1997) who quipped that “digital information lasts forever—or five years, whichever comes first.” He has published an interesting *Scientific American* article about preservation called “Ensuring the longevity of digital documents” (Rothenberg, 1995), which promotes emulation as the only workable solution, and a more extended report that develops the same theme (Rothenberg, 1999).

The ominous article on the digital dark ages is by Kuny (1998). Lynch (1999) has published an interesting analysis of the issues that arise in managing digital objects over time, in particular the need to track authenticity as part of an object’s provenance, to maintain its integrity, and to ensure the integrity of links to that object from other objects (or from metadata), as well as to reformat the object without affecting its integrity. Rusbridge (2006) has an interesting discussion of some fallacies of digital preservation. For an introduction to the Universal Virtual Computer, see van der Hoeven et al. (2005); Gladney (2007) gives a broader picture of this approach to digital preservation.

The poem *Beowulf* survives in the British Museum manuscript Cotton Vitellius A.xv, probably written about 1000 A.D.; most scholars accept that it took its present form somewhere between 680 and 800 A.D. It opens with these words:

Hƿæt ƿē Gār-Dena in ġeārdagum
 þeodcýninga þrym ġefrūnon
 hū ðā æþelingas ellen fremedon.

An eloquent translation by Crossley-Holland (1968) begins:

Listen! The fame of Danish kings
 in days gone by, the daring feats
 worked by those heroes are well known to us.

Just thirty years later, Seamus Heaney (1999) gave a fresh rendering:

So. The Spear-Danes in days gone by
 and the kings who ruled them had courage and greatness.
 We have heard of those princes’ heroic campaigns.

It is hard to imagine that such stirring words would ever have resulted from a sequence of paraphrasings of earlier interpretations.

The LOCKSS system was produced by researchers at Stanford University (Rosenthal and Reich, 2000; Rosenthal et al., 2005). Brewster Kahle described his vision of the Internet Archive in *Scientific American* (Kahle, 1997); the Web site is simply www.archive.org. The Bibliotheca Alexandrina is at www.bibalex.org.

The iPod-based digital library is due to Bainbridge et al. (2008). The knowledge-based information retrieval system described in [Section 9.3](#) is Koru (Milne et al., 2007), which can be accessed at www.nzdl.org/koru.

We gratefully acknowledge Maria Trujillo for helping us focus on digital libraries for oral cultures. The wonderful book from which [Figure 9.11](#) is reproduced is *First Aid in Pictures* (DeLong et al., 1987). The mission of LINCOS, which stands for “little intelligent communities,” is to help disadvantaged communities from developing countries to use a series of services and applications—telemedicine, the Internet, electronic communication, videoconferencing, electronic trade, and educational applications. You can find out more about this project at www.lincos.net. Finally, digital divides and their ramifications are thoroughly discussed in a thought-provoking book by Norris (2001).

Building collections

We are hoping that you are reading this book because you actually want to *build a digital library*! If so, now is the time to get started.

Greenstone is introduced briefly in Chapter 1: Section 1.7 reviews its features to help convey the requirements for digital library software. Virtually all the user interfaces in Chapter 3 show actual Greenstone collections. In Chapters 4, 5, and 6 we encounter various document, multimedia, and metadata formats, almost all of which the software supports.

In Part II of this book we focus specifically on Greenstone: how to use it and how it works. We assume that you have installed it on your computer, whether it runs Windows, Mac OS X, or Linux. The installation procedure—particularly for Windows—is very simple and should only take a few minutes. The best thing to do is simply to accept all the defaults; just keep clicking **Next**. After installing Greenstone, you enter it in Windows from the Start menu; on Mac or Linux you navigate to the folder where you installed Greenstone and double-click *gs2-server.sh* (or else change directory to that location and run *./gs2-server.sh*). (Greenstone works on all versions of Unix, but needs to be compiled from source for versions other than Linux.) Click on the **Enter Library** button in the window that appears. This takes you to the home page of your Greenstone installation, which gives access to a tiny demonstration collection that we call the Demo collection. (Other collections may also appear on the home page, depending on how you installed it.)

There is an important distinction between the processes involved in *building* collections and those involved in *delivering* the information they contain to users. The latter—accessing collections from the Greenstone home page and searching or browsing them—is done through what we call the Reader’s interface. To serve users effectively, information structures have to be carefully prepared in advance, and this is done through the Greenstone Librarian Interface, or GLI, which is an interactive platform-independent Java application. Usually it runs on the same computer that operates the Greenstone digital library server. However, you can set things up so that the Librarian interface lets you access collections built on a remote computer, as described in the next chapter.

This chapter demonstrates the use of Greenstone. All sections except the next one are concerned with the Librarian interface. Some sections include extensive exercises with detailed step-by-step instructions. Doing the exercises is really the best way to learn how to use the software, and we strongly recommend that you follow along with your computer while reading. Instructions are written in bold-face to draw your attention to them and to make them easier to follow. To do the exercises you will need to have some sample documents of different types; you can download an appropriate collection of files from the *How to Build a Digital Library* Web site at www.greenstone.org/howto. You do not

have to finish each section before going on to the next; feel free to skip over parts that don't concern you.

This chapter is not a comprehensive user manual. The many options that Greenstone provides are documented online in the interface. Working through this chapter will teach you about the concepts involved and will give you enough experience that you can continue to work independently, learning as you go. The main task here is to learn to *build* collections, but to do so effectively, you need to appreciate how readers will use the collections, so we begin by taking a look at what readers see when they use Greenstone collections. As an example we use a small subset of the Humanity Development Library with which this book opens. Then we give a preview of the Librarian interface, explaining what it does and walking through the process of using it to build collections, illustrated with a larger subset of the Humanity Development Library. This is a fully illustrated walk-through, so you do not need to follow along with it on your computer. At each stage we point out the options that are available but do not actually explore any of them.

[Section 10.3](#) is where your real work begins. It takes just a few minutes to build a collection with Greenstone, and we describe how to build many of them. We start with a tiny collection of HTML files and progress to one containing Word and PDF documents. Next we look at some more advanced facilities that can enhance the processing of all three of these file types. Then we examine a larger collection and show how to add additional searching and browsing structures to make it more useful for readers.

Greenstone is extremely flexible in the facilities it provides for you to configure the Web pages it produces. Central to accomplishing this are *format statements*. Writing these statements, though, is perhaps the most difficult topic that people encounter when using the Librarian interface. Extreme flexibility comes at a price! [Section 10.4](#) introduces this topic. On first reading you may prefer to skim or skip this section and return to it when you come across particular format statements later on. However, to become proficient, you will eventually need to learn something about format statements.

In [Section 10.5](#) we describe how to deal with metadata. By this point you will have used metadata in the HTML, Word, and PDF collections, but there are many more things to know. Not only can you enter new metadata, but also you can create new metadata *sets* and edit old ones, adding new fields—perhaps ones that are particular to your organization or library. We explain how metadata is stored internally in XML files, which can be used to bring existing metadata along with documents into collections. Many collections contain bibliographic records, not full-text documents, that are supplied in standard formats, such as the Library of Congress MARC format (see Chapter 6). In other cases, full-text documents will already be associated with some or all of the bibliographic records and must be imported into the collection along with the metadata. And frequently new full-text documents corresponding to existing metadata records come along and must be included, too. All of these things are easy to do.

Finally we turn to collections of multimedia documents, unifying the different formats by mapping them to document surrogates in Greenstone—so that they can be handled in a similar way to purely textual documents. However, there is still much to be learned about different ways of organizing information. The final two exercises in [Section 10.6](#) are quite elaborate—and difficult. They produce polished collections. One collection is of pages where full text is available alongside the page

images. The other includes many different media and document types with highly customized pages, sporting special-purpose buttons for accessing them.

Like many open source projects, Greenstone is an evolving system. Throughout Part II of this book we describe the “production” version, sometimes called Greenstone 2. In fact, we assume Greenstone 2.83, the current release at the time of writing—but the features and functionality have been stable for several years, and we expect these instructions to continue to apply to future versions of the software. However, work is underway on a completely new design, Greenstone3. From the ordinary user’s point of view there is little difference between the two; the main advantage of the new version is that it makes life easier for programmers who wish to enhance the system or to add completely new facilities. Most of the information we give here applies to both versions—with one important exception. The scheme of format statements, described in [Section 10.4](#) and used occasionally throughout this chapter, is different in Greenstone3 (which uses XSLT, the XSL Transform language). The new scheme is documented on the Greenstone Web site, but is undergoing further development—which is why we use the tried-and-tested version of Greenstone here. More information about the Greenstone3 project is given at the end of Chapter 12.

10.1 The Reader’s Interface

Before building any collections, we briefly explore the Reader’s interface. For this we use the Demo collection, a small subset of the Humanity Development Library that is installed whenever Greenstone is. Other collections offer similar facilities. If you prefer, you can visit www.nzdl.org and do this exercise using the Humanity Development Library, which is very similar—only much bigger.

Remember, instructions are written in boldface to draw your attention to them and to make them easier to follow. The boldface instructions include menu selections and buttons to click.

The Greenstone digital library

If you are using Windows, launch the Greenstone Local Library Server by clicking

Start→All Programs→Greenstone-2.83→Greenstone Server

Click **Enter Library** in the dialog box and your Web browser will display the Greenstone home page. Simultaneously, the Greenstone server program minimizes itself: it can be reopened at any stage by clicking on the “world globe” icon that appears in the Windows task bar. The Local Library Server, despite its name, includes a light weight Web server and can serve content to other users over a network if this is switched on using the File→Settings menu.

It is sometimes necessary to turn off the proxy settings of the browser. The Local Library Server normally detects this and pops up a window alerting you to the problem. If you continue to have problems, there are a number of different options for the address resolution method on the File→Settings menu of the dialog box.

Things are much the same on Mac or Linux, except that the Apache Web server is used instead of the Local Library Server. We refer to this as the Web Library version. A program called *gs2-server.sh* located in the top-level folder of your Greenstone installation launches Greenstone. Locate this

application using a file browser and double-click it, or change directory (`cd`) to the location of your Greenstone installation and type `.gs2-server.sh`. This application starts, stops, and reconfigures the Apache Web server that is installed with the Mac and Linux versions of Greenstone.

Again, a possible pitfall is the Web proxy settings. If you normally use a Web proxy in your browser, you may need to ensure that no proxy is set for localhost. By default, Greenstone's Web server uses port 8080. If this is already being used by another program on your computer, Greenstone will automatically select another port—it can also be altered manually using the File→Settings menu of the dialog box.

Digital library installations usually contain many separate collections. Readers access them by clicking them in the Greenstone home page. It is possible to set up private collections that don't appear on the home page but are still accessible if you know the URL. To get back to the home page, click the **Home** button that appears in every collection, near the top right-hand corner of the window.

To exit from Greenstone, close the Greenstone server by selecting the world globe icon in the task bar (to bring the window to the front) and then clicking the exit hotspot—usually marked as a red cross. The server exits, but your Web browser continues to run.

Exploring the Demo collection

Click on the *Demo* collection on the Greenstone home page.

You are looking at the *About* page in [Figure 10.1](#). Each collection has a page like this to give information about it. To return there at any time from within the collection, click on the collection icon that appears at the top left of all searching and browsing pages.

The easiest way to learn how to use the interface is to try it out. Click liberally: most images that appear on the screen are clickable. If you hold the mouse stationary over an image, most browsers will pop up a “mouse-over” message that tells you what will happen if you click. Experiment! Choose common words such as *the* and *and* to search for—that should evoke some response, and nothing will break. (Note: all words are indexed; there are no stop words. See Section 4.1 for more details.)

All icons in [Figure 10.1](#) are clickable. Several appear at the top of almost every page; [Table 10.1](#) defines what they mean. The *search ... titles ... subjects ... organizations ... how to* navigation bar at the top of the figure gives access to the searching and browsing facilities. These buttons provide five different ways to find information in the Demo collection. The leftmost one is for searching, and the others—four, in this collection—evoke different browsing facilities. The buttons in this navigation bar typically differ from one collection to another.

Browsing

Click *titles* (in the navigation bar).

Access the first book in the list of titles by clicking the book icon next to the title *Butterfly Farming in Papua New Guinea*. You can tell that you have arrived at an individual book because there is a

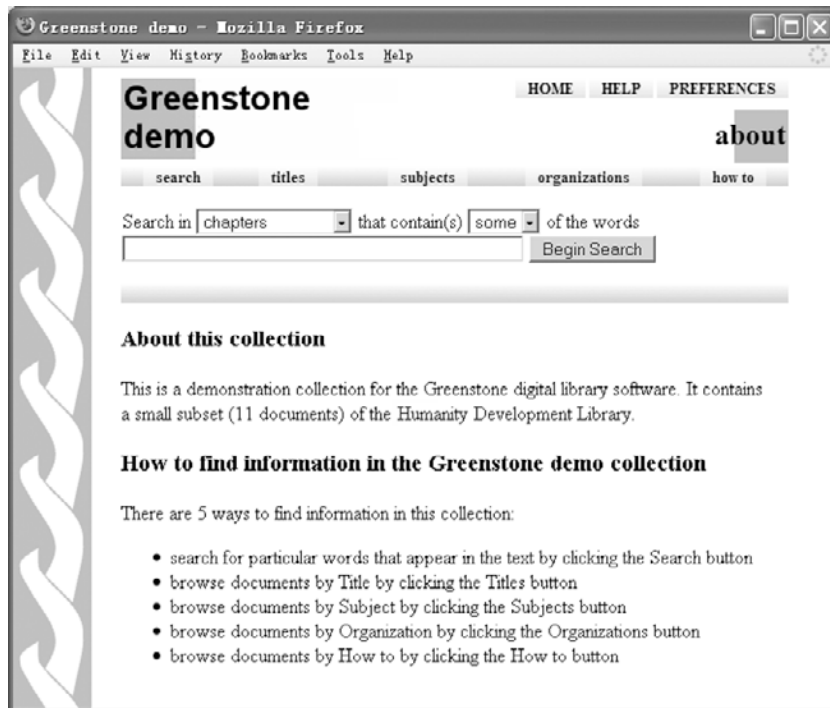


Figure 10.1: The Demo collection's *About* page

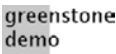
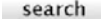
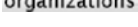
picture of its front cover. If necessary, use the scroll bar to view the full length of the page; the button at the end takes you to the next section. Explore the table of contents near the top by clicking some of the page icons. This book only has one folder—Appendixes—containing subsections; other books have many more folders, and folders within folders, that can be expanded and contracted by clicking them.

Sometimes—especially for printing—it's good to see the whole book in a single long Web page. To do this, click **Expand Text** just below the cover picture. For long documents, which take some time to load, Greenstone seeks confirmation for this action: clicking **Continue** loads the full document. You can also expand the table of contents by clicking **Expand Contents**.

You can close this document by clicking the book icon next to the book's title at the top of the table of contents (this signifies closing the book). Or you could just click **titles** in the navigation bar.

Browse by subject by pressing the **subjects** button. Subjects are represented by bookshelves that can be further expanded by clicking on them. Of course, this collection is very small and has hardly any books on the shelves! As we have seen, the **titles** button brings up the documents in alphabetical order. The **organization** button brings up a list of the organizations that commissioned the books, and **how to** brings up a list of "how to" hints. Again, this was structure designed for several thousand books; the Demo collection has a paltry 11.

Table 10.1: Icons and their meaning

Top of each page		Takes you to the About page
		Takes you to the Greenstone home page
		Provides help text
		Controls user interface and searching options
Search/browse bar		Search for particular words
		Access publications by title
		Access publications by subject
		Access publications by organization
Miscellaneous		Access publications by “How to” listing
		Click to read the book
		Click for books on this subject
		View this document
		Open folder and view contents
		Close the book
		Close the folder
		Go on to the next section ...
		... or back to the previous section
		Open this page in a new window
		Expand table of contents
		Display all text
		Highlight search terms

Browse around and peruse some other documents.

In some collections the documents do not have a hierarchical chapter/section structure. Then no table of contents is displayed when you get to an individual document—just the document text. In other collections, such as one with digitized newspapers, each document is split into separate pages, and you can read sequentially or jump around from one page to another.

Searching

Access the search page by clicking *search* in the navigation bar.

The Search page is just like the *About* page in Figure 10.1 except that it doesn't show the *About this collection* text. Enter the term *gender* in the query box, then click **Begin Search**. The Web browser shows the results. There are rather a lot of them because this search is at *chapter* level, and each

chapter (and section, and subsection) containing the search term is shown individually. The first drop-down list above the query specifies what part of the document you are searching in, in this case the page text (alternatives are titles, subjects, etc.). The second drop-down list allows you to search instead at *book* level, in which case just three books would be returned.

There are many small details that relate to the search process. Search terms contain only alphabetic characters and digits and are separated by white space. If other characters, such as punctuation, appear, they serve to separate terms just as though they were spaces—and then they are ignored. You can specify a phrase search by enclosing your terms in quotation marks.

Section 3.4 explains two different types of query, ranked and Boolean. Initially, the interface gives a simplified set of choices:

- queries for all the words
- queries for some of the words.

The first is interpreted as a Boolean AND query, the second as a ranked OR query. Users are encouraged to employ as many search terms as they like—a whole sentence, even a whole paragraph. The interface does not use the words “Boolean” and “ranked” to keep the language as simple as possible for users. More control over searching is available through the Preferences page, discussed in the next section; there is also a detailed Help page.

In this collection you can search within five different indexes: text, titles, subjects, organization, and “how to,” at both book and chapter levels. Other collections have different indexes—for example, author and title indexes, or chapter and paragraph indexes. In most collections the full matching document is returned regardless of which index you search, although this need not be the case.

Preferences

Access the Preferences page by clicking *preferences* at the top right.

The Preferences page allows you to control many features of the interface (though many readers do not notice it). Each collection has a default presentation language, but you can choose a different one if you like. You can also alter the encoding scheme that Greenstone uses to communicate with the browser—the software chooses sensible defaults, but with some browsers better visual results are obtained by switching to a different encoding scheme. (Some languages, such as Arabic, Russian, and Chinese have special encodings that are selected automatically; changing them may cause problems.) All collections allow you to choose a textual interface format rather than the standard graphical one, which is useful for visually impaired users who use large screen fonts or speech synthesizers.

Try the interface in another language, for example Chinese.

The Preferences page also gives more control over the search operation. This includes case-folding and stemming, an advanced query mode that allows users to specify Boolean operators, a fielded query form, and a mode that displays search history.

Experiment with different query modes and styles.

Depending on the collection, there may be other options that control the presentation. In some collections of Web pages, you can suppress the navigation bar at the top of each document page, so that clicking on a search result lands you at the matching document's page without any Greenstone header. (To do another search you would have to use the browser's **Back** button.) In some Web collections you can control whether the links on the Search Results page take you straight to the actual URL in question, rather than to the digital library's own copy of the page. Other collections comprise several subcollections that can be searched independently or together as one unit. If so, you can select from the Preferences page which subcollections to include in your searches.

10.2 The Librarian Interface

The Librarian interface is designed for creating and maintaining digital library collections. It helps librarians and others who compile electronic anthologies to construct and organize collections of information. Only a few minutes are needed to set up a collection based on a standard design, assuming that documents and metadata are already available in electronic form.

Actually building the full-text indexes and browsing structures that support a collection may take longer. Some collections contain gigabytes of text—millions of documents. There may be even larger volumes of associated information—typically audio, image, and video, with textual metadata. Once initiated, the mechanical process of collection building may take a few moments for a small collection, or several hours for a multi-gigabyte one that involves many full-text indexes. The Librarian interface monitors all this and provides visual feedback about progress.

Naturally, customized collections that have their own idiosyncratic requirements—as most substantial collections do—take longer to set up, and the design and debugging process can take days, or weeks if iterative usability testing is involved. The Librarian interface is designed to support precisely this sort of activity.

Users and functions

The Librarian interface is targeted at three different levels of user. *Library Assistants* gain access to basic features: adding documents and metadata to existing collections, creating new collections whose structure mirrors existing ones, and rebuilding collections to reflect changes. *Librarians*, the regular or default users of the Librarian interface, perform all the above tasks and can also design new collections—adding new document types, new full-text indexes, and new metadata browsing features. *Expert users* are those who are experienced with Greenstone. They have more control over building collections, including the ability to schedule regular automatic rebuilding. This level also presupposes some familiarity with running Perl scripts and examining their output.

The Librarian interface supports six basic activities, which can be interleaved but are normally undertaken in this order.

- If required, *download* any documents from external sources that need to be included in the collection. This step is only relevant when the collection will contain material from elsewhere. Use protocols like HTTP for Web pages, and OAI-PMH or Z39-50 for metadata.

- *Gather* documents together—whether to populate a new collection or to update an existing one. Metadata files may also be brought in. Users browse the computer’s files to find documents or folders to include, and drag and drop them into place. Any documents imported from other Greenstone collections come with existing metadata attached.
- *Enrich* the documents by adding metadata to them. Documents can be grouped into folders (existing folder structures are retained when dragging files in), and any metadata assigned to folders is inherited by all documents nested within them.
- *Design* the collection by determining its access facilities, such as full-text search indexes and browsing structures. The design facility is not available at the Assistant Librarian user level.
- *Create* the collection. This work is done by the computer; users are presented with a progress bar. This is the point where expert users might examine the output of Perl scripts, which are presented in a scrolling window, to determine if anything is going wrong.
- *Format* the collection appropriately, so that it looks good on the reader’s screen.

Whenever a collection is created, it is automatically installed in the digital library on your computer, and a Web page is opened to the collection’s *About this collection* page, which acts as its home page in Web parlance. The collection can easily be transferred to another computer and placed in an existing Greenstone installation there.

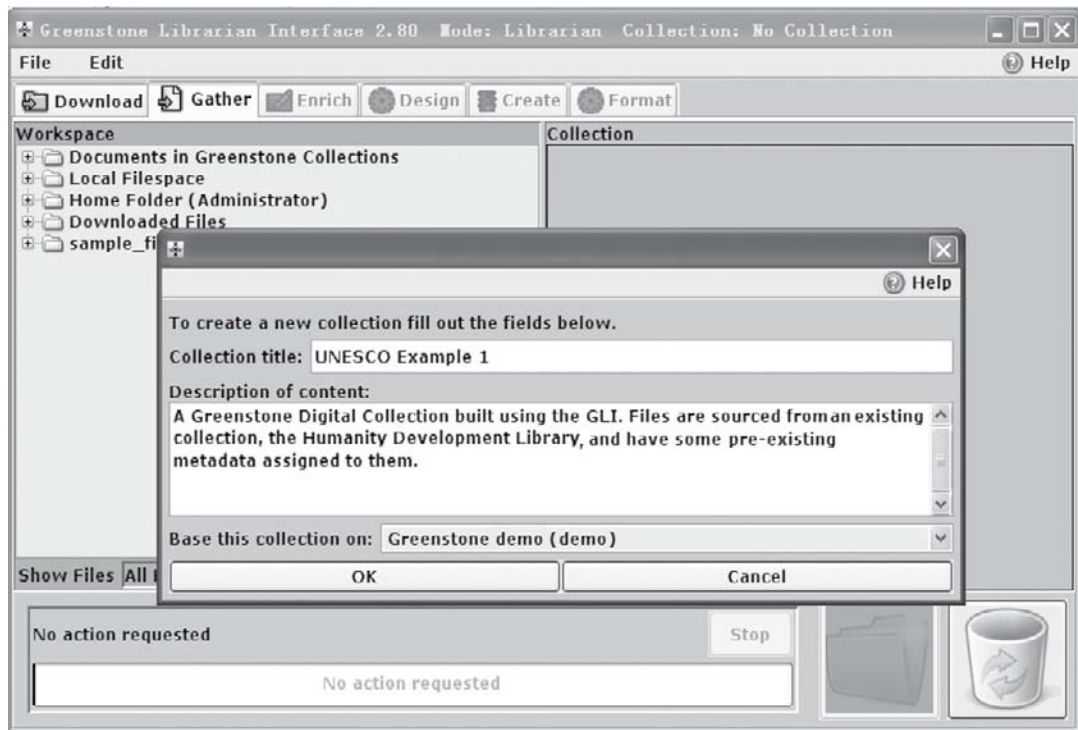
A walk-through

To convey the operation of the Librarian interface, we walk through a simple example. [Figures 10.2a–l](#) show screen snapshots at various points during the interaction, so you do *not* need to follow along on your computer. This example uses documents in the Development Library Subset (DLS) collection, a larger subset of the Humanity Development Library, which is distributed on Greenstone CD-ROMs. For expository purposes, the walk-through takes the form of a single pass through the steps listed above. A more realistic pattern of use, however, is for users to switch back and forth through the various stages as the task proceeds.

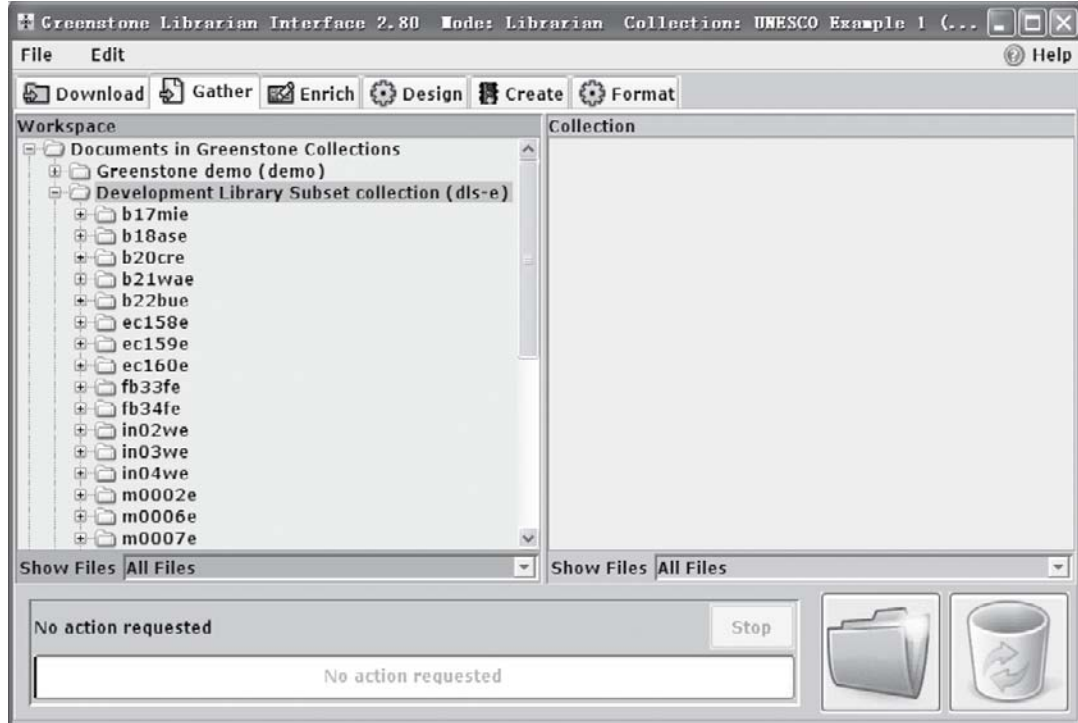
Getting started

The Librarian interface is launched under Windows by selecting Greenstone from the Start menu and choosing Librarian Interface. (On Mac or Linux, having used a `cd` command in a terminal window to move to where Greenstone was installed, invoke the Librarian interface by running `gli/gli.sh`. Alternatively, you can navigate to the `gli` folder in a file browser and then double-click `gli.sh`.)

The first step is to open an existing collection or to start a new one. In [Figure 10.2a](#), the user is starting a new collection. She has selected New from the File menu and begun to fill out general information about the collection—its title and a brief description of the content—in the pop-up window. The collection title is a short phrase used to identify it: existing collections have names like *Food and Nutrition Library*, *World Environmental Library*, and so on. From it, the system assigns a unique mnemonic identifier, the collection name, for internal use. The brief description that the user enters below this is intended to be some statement describing the principles that govern what is included in the collection. It appears under the heading *About this collection* on the home page, and while the



(a)



(b)

Figure 10.2: Walking through the Librarian interface: (a) starting a new collection; (b) exploring the local file space;

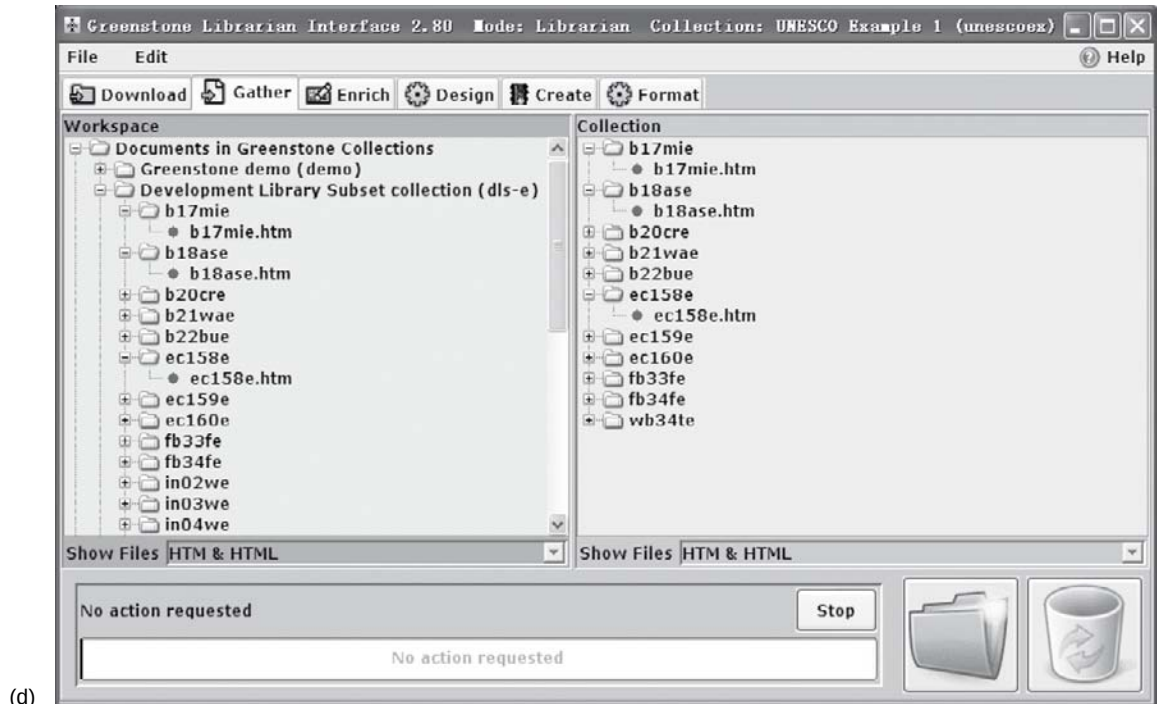
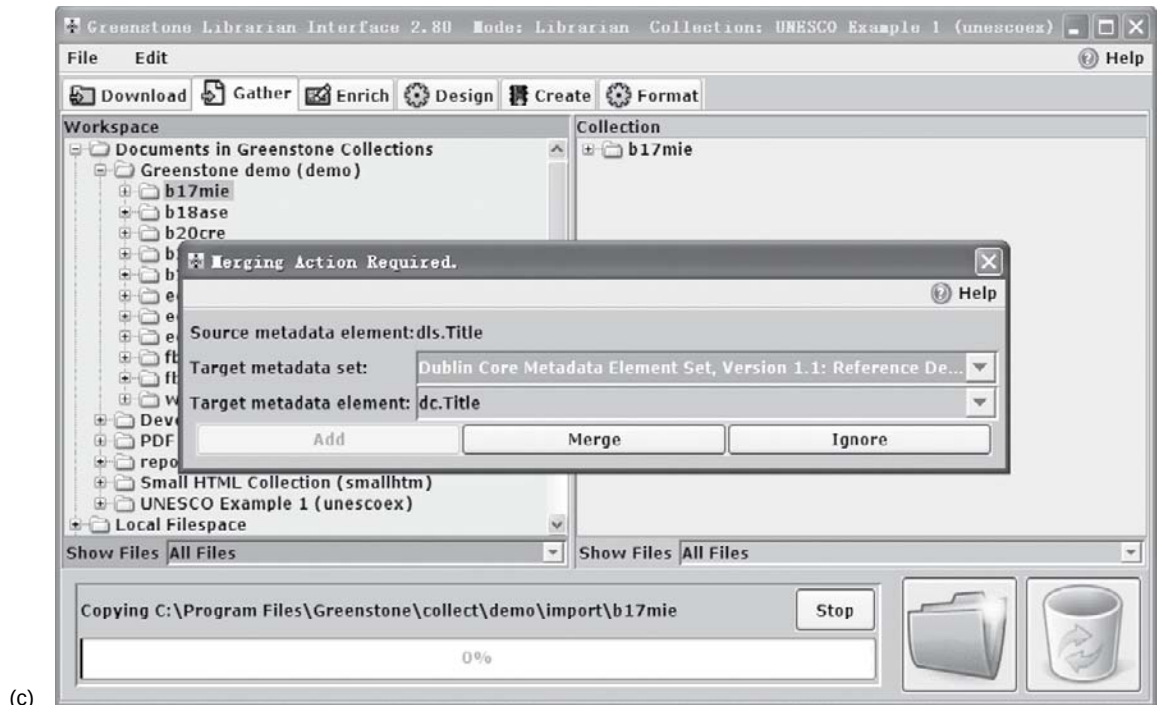
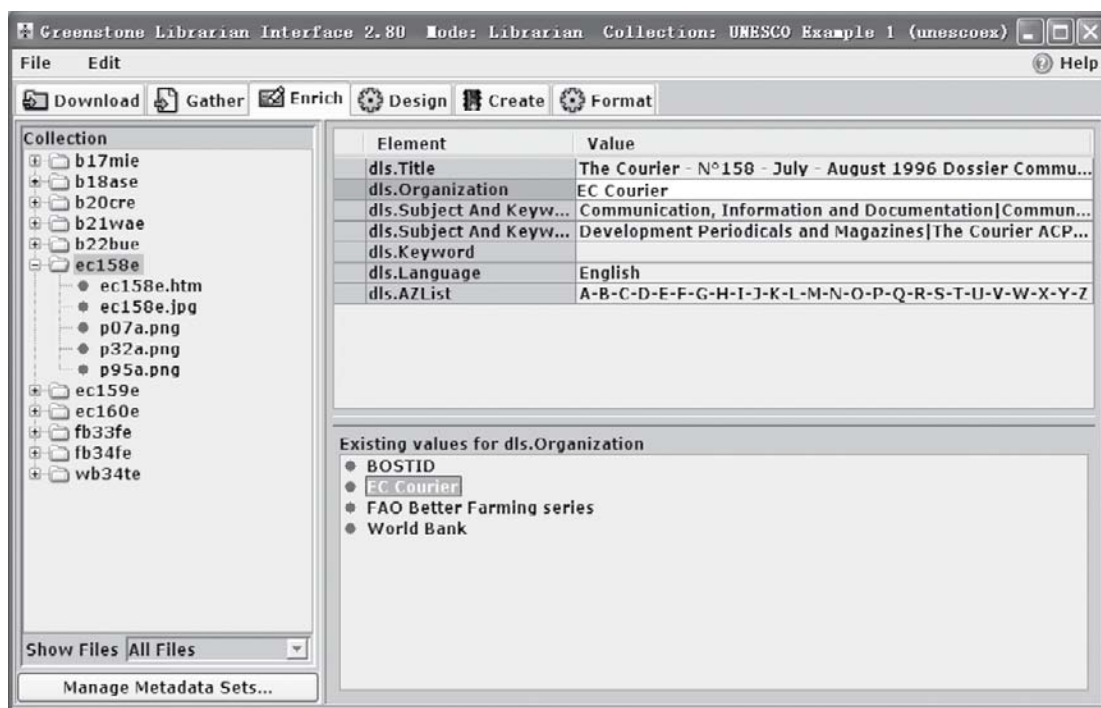
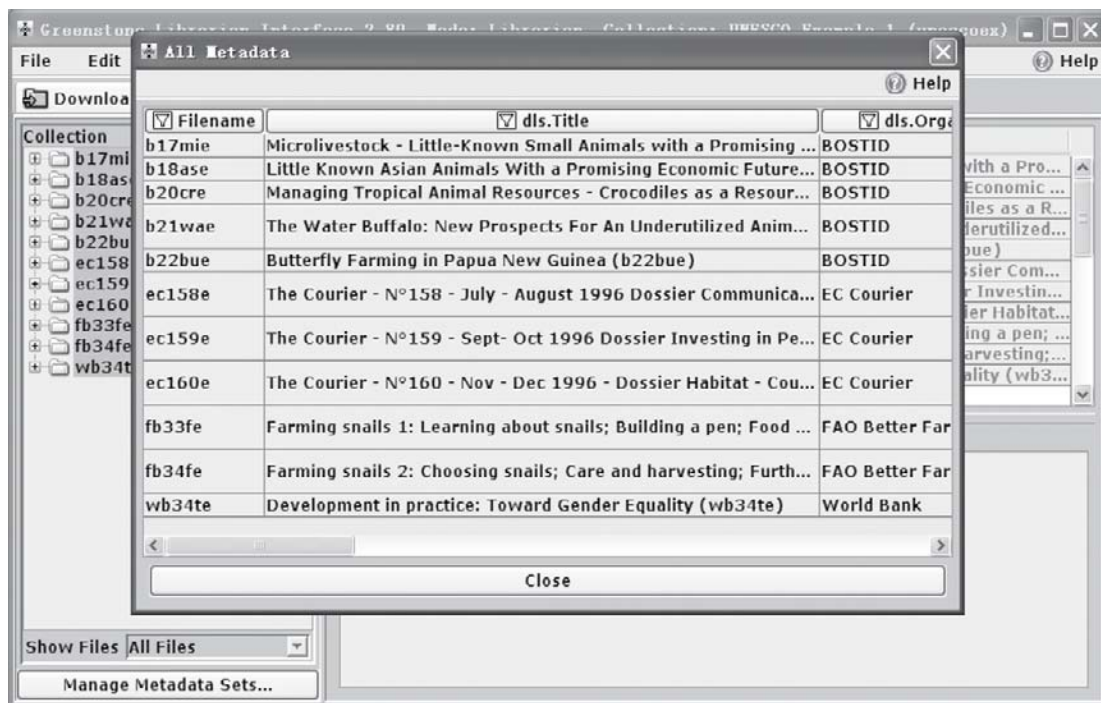


Figure 10.2, cont'd: (c) importing existing metadata; (d) filtering the file trees;



(e)



(f)

Figure 10.2, cont'd: (e) assigning metadata using the Enrich view; (f) viewing all metadata for selected files;

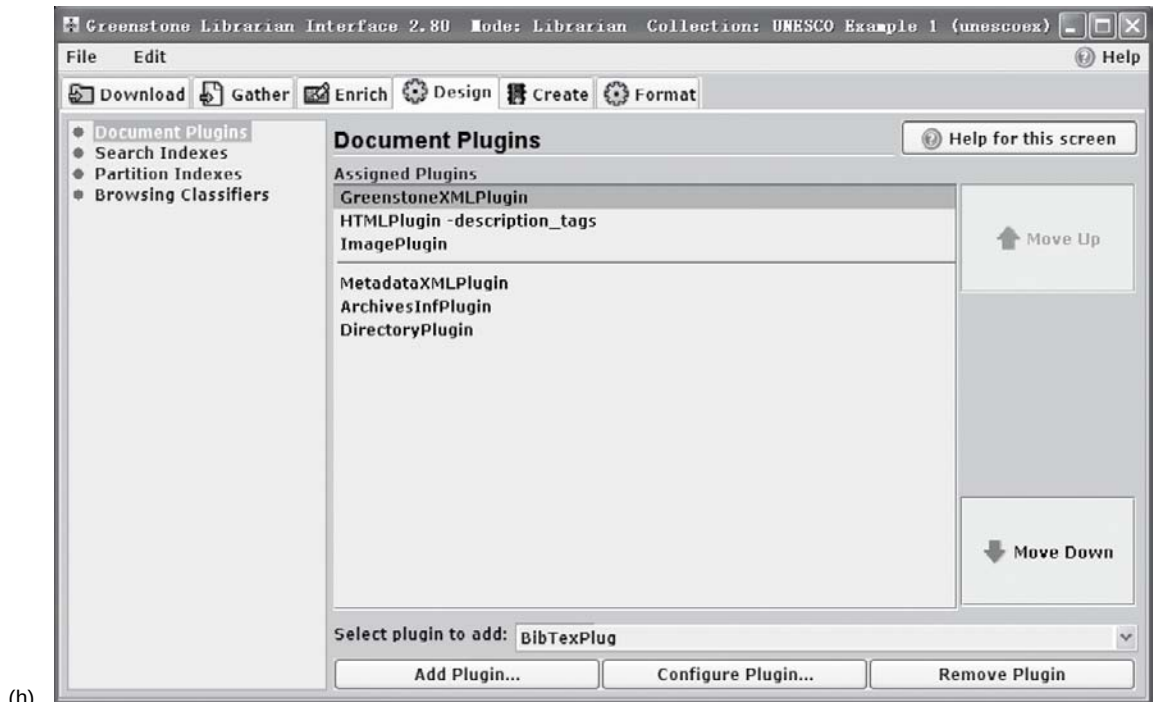
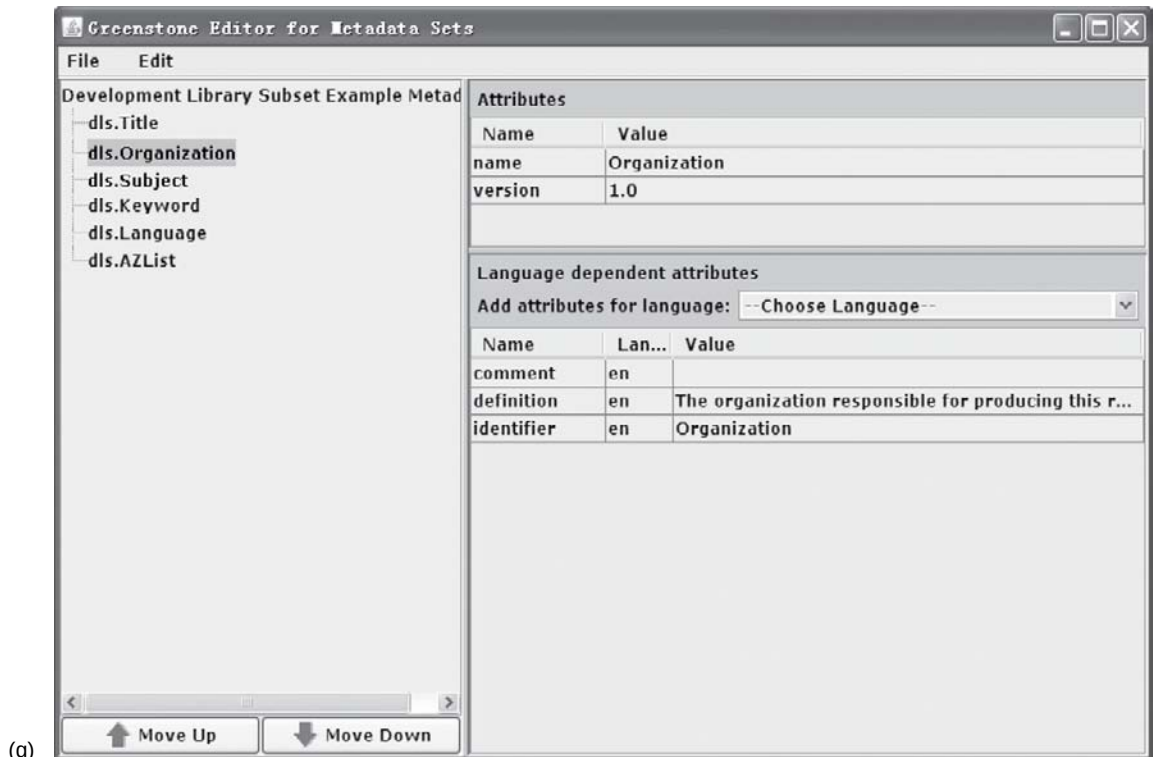


Figure 10.2, cont'd: (g) editing the metadata set; (h) specifying which plug-ins to use;

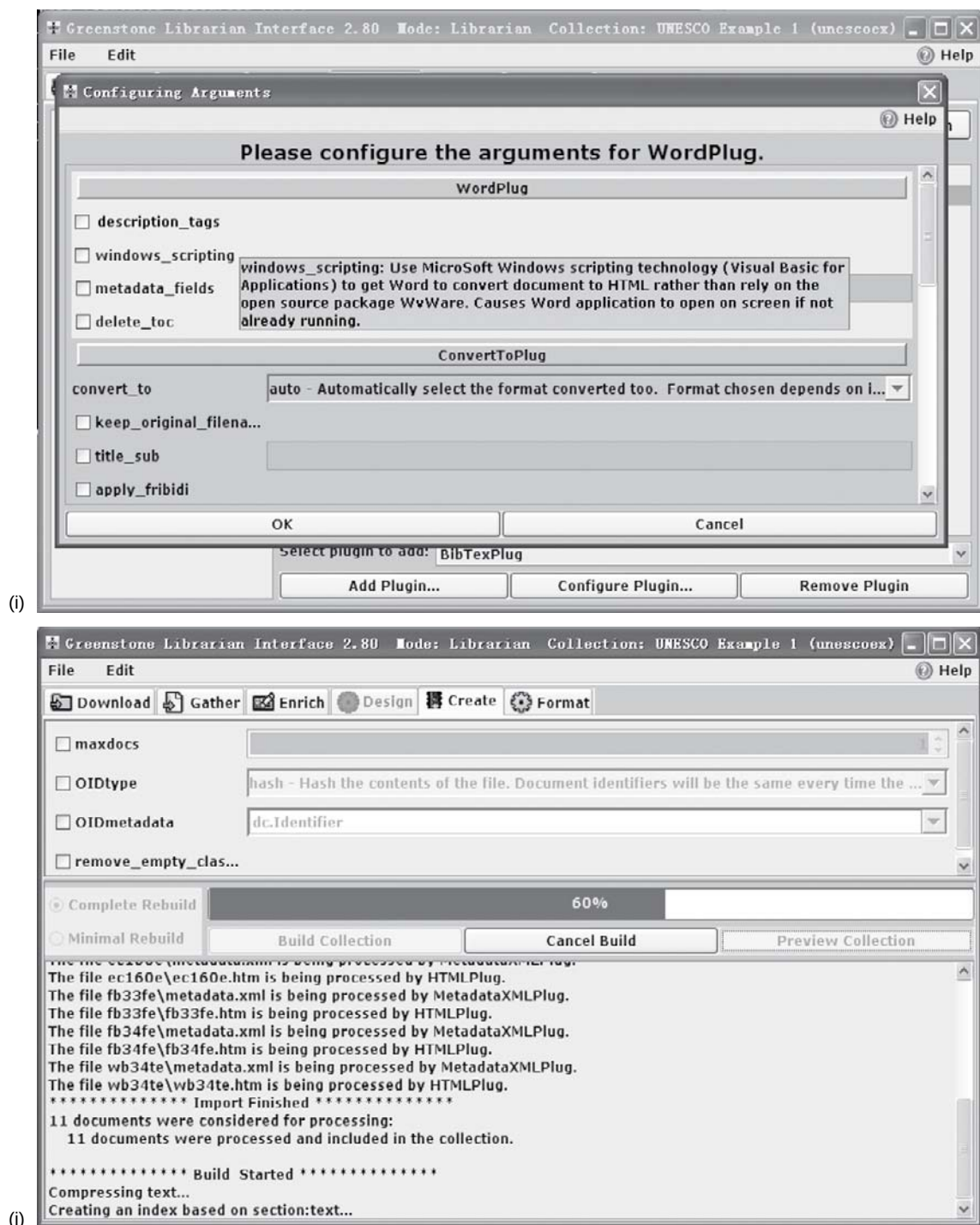


Figure 10.2, cont'd: (i) configuring arguments to a plug-in; (j) in the process of building the new collection;

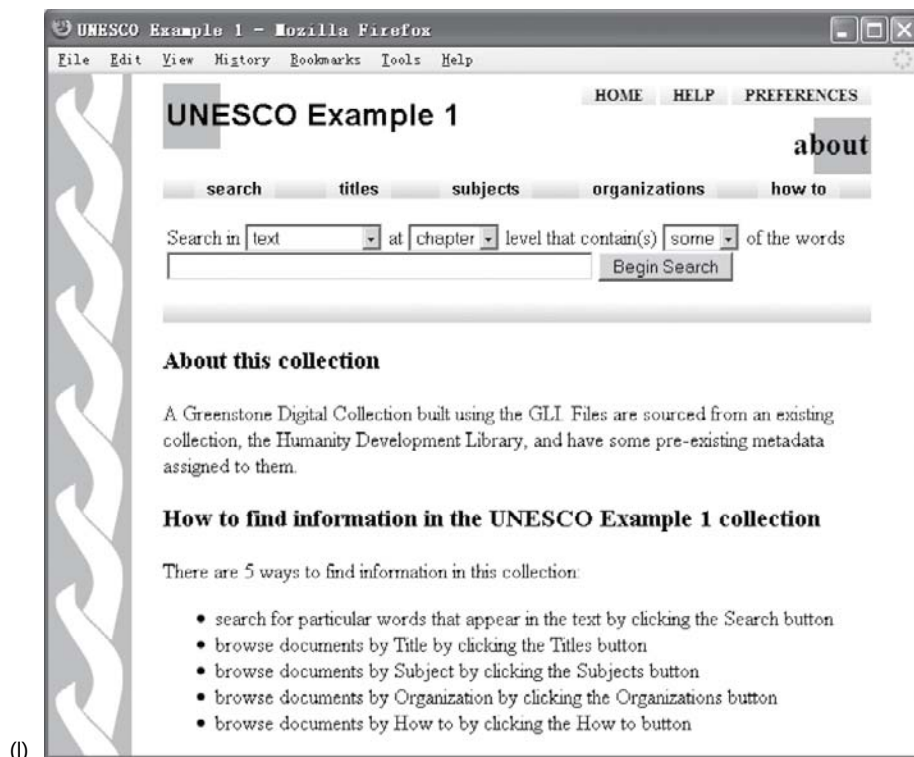
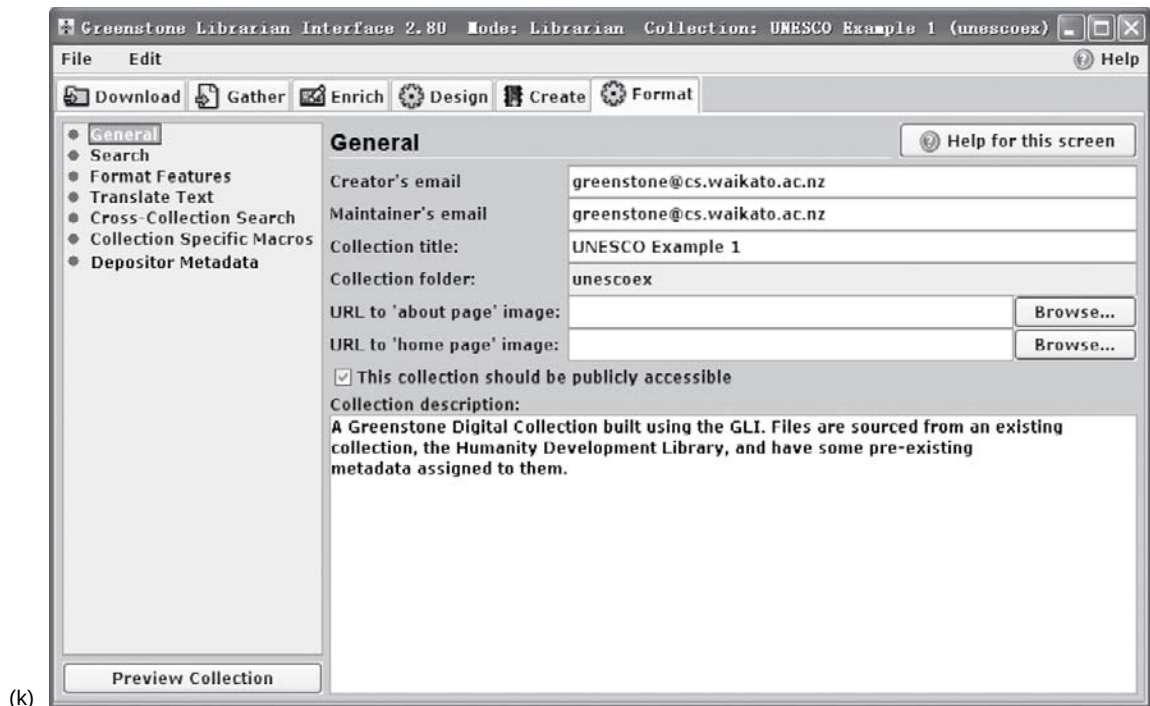


Figure 10.2, cont'd: (k) formatting the collection's appearance; (l) previewing the newly built collection

early examples we will encounter use only plain text, the field does in fact support the full range of HTML syntax.

At this point, the user decides whether to base the collection on the same structure as an existing one or to build an entirely new kind of collection. In [Figure 10.2a](#) the user has chosen to base it on the Demo collection. This implies that the DLS metadata set used in the Demo collection will also be used for the new collection. (In fact, this simple metadata set underlies several Greenstone collections that share a common structure and organization but with different content.) It contains these items:

- Title
- Subject
- Language
- Organization
- Keyword.

If, instead, the user had chosen to create a New Collection at this point, it would have been based on a default standard structure and equipped with the unqualified Dublin Core metadata set (see Section 6.2 for more details). Metadata sets can be added to a collection at any time using the **Manage Metadata Sets** button on the Enrich panel. Several different sets can be associated with the same collection; the system keeps them distinct (so that, for example, documents can have both a Dublin Core Title and a DLS Title). The different sets are clearly distinguished in the interface. Behind the scenes, metadata is represented in XML.

Assembling the source material

After the **OK** button is clicked on the New Collection pop-up, the remaining parts of the interface, which were grayed out before, become active. Initially the Gather panel, selected by the eponymous tab near the top of [Figure 10.2a](#), is displayed. It lets users explore the local file space and existing collections and gather together selected documents for the new collection. The panel is divided into two sections; existing structures on the left and the documents in the collection on the right.

Operations available at this stage include:

- Navigating the existing file structure hierarchy, and the one being created.
- Dragging and dropping files into the new collection.
- Multiple selection of files.
- Dragging and dropping entire sub-hierarchies.
- Deleting documents from the collection.
- Creating new sub-hierarchies within the collection.
- Filtering the files that are visible in both the local file system and the collection, based on predetermined groups or on file matching expressions.
- Invoking an appropriate program to display a selected file's contents by double-clicking it.
- "Exploding" a file containing multiple metadata records into individual records.

In [Figure 10.2b](#) the user is using the interactive file tree display to explore the local file system. At this stage the collection on the right is empty; the user populates it by dragging in files of interest from the left-hand panel. The files are copied rather than moved, leaving the original file system undisturbed. Greenstone follows the usual conventions for multiple selection, dragging and dropping, structuring the new collection by creating folders, and deleting files from it by moving them to a trashcan.

Existing collections are represented by a subdirectory on the left called Documents in Greenstone Collections, which can be opened and explored like any other directory. The documents therein differ from ordinary files because they may already have metadata attached, and this metadata is preserved when they are copied into another collection. Conflicts may arise because the metadata may have been assigned using a set that is absent from the target collection, in which case the user must resolve the conflicts. This will not happen in the present situation because the new documents also use DLS metadata, but [Figure 10.2c](#) illustrates a user dragging some documents from an existing collection into one with a different metadata set (Dublin Core). A window pops up explaining that the `dlc.Title` element cannot be automatically imported; the user can choose a metadata element and press Merge to effectively merge the old and new metadata elements. The system remembers all choices that are made and applies them in subsequent file copies.

When large file sets are selected and dragged into the new collection, copying may take some time—particularly if metadata conversion is involved. To indicate progress, the interface shows which file is being copied and what percentage of files have been processed.

Special facilities are also provided for working with large file sets. For example, the user can choose to filter the file tree to show only certain files, using a drop-down menu of file types displayed underneath the tree. In [Figure 10.2d](#), only the HTM and HTML files are being shown (and only these files will be copied by drag and drop).

Enriching the documents

The next phase in collection building, which is optional, is to enrich the documents by manually adding metadata. The Enrich tab brings up a new panel ([Figure 10.2e](#)) that shows the documents in the collection on the left and a metadata panel on the right. In this panel, metadata values can be assigned to documents. Existing values can be re-used or new ones typed in. If the values have a hierarchical structure, the hierarchy can be extended with new branches.

When more than one document has been selected, metadata values can be assigned to them all in a single operation. They can also be assigned to folders. Documents in these folders for which this metadata is unspecified inherit the metadata values. However, they can subsequently be overridden by supplying different ones for the document itself.

Operations at this stage include:

- Assigning new and existing metadata values to documents.
- Assigning metadata to an individual document.
- Assigning metadata to a folder (inherited by all documents in the folder, including nested folders).

- Assigning hierarchical metadata and updating its structure.
- Editing or updating assigned metadata.
- Reviewing the metadata assigned to a selection of files and directories.
- Reviewing the metadata sets, and their elements, used in the collection.

In [Figure 10.2e](#) the user has selected the folder *ec158e* and is examining its dls.Organization metadata; the value is *The Courier*. The existing values for this metadata element can be seen in the box at the lower right.

Users can view all the metadata that has been assigned to documents in the collection by selecting a set of documents and choosing Assigned Metadata from the right-click menu. This pops up the window shown in [Figure 10.2f](#), showing the metadata in spreadsheet form. If the user has specified a file filter, as mentioned above, only the selected documents are shown in the metadata display.

The panel in [Figure 10.2g](#) allows the user to edit metadata sets; you arrive at it by clicking Manage Metadata Sets at the lower left of [Figures 10.2e and f](#). Here, the user is looking at the Organization element of the DLS set. New elements can be added (by right-clicking the set name on the left-hand side), the ordering of items can be changed, translations for existing elements can be included, and new sub-elements can be added (by right-clicking an element).

Designing the collection

The Design panel ([Figures 10.2h and i](#)) is where users determine the collection's structure and organization. This involves a series of separate interaction screens, each dealing with one aspect of the collection design.

Operations include:

- Reviewing and modifying the plug-ins used to process documents.
- Configuring individual plug-ins.
- Defining what full-text indexes are to be built.
- Creating subcollections and having indexes built for them.
- Adding browsing classifiers to the collection.
- Configuring individual classifiers.

On the left of [Figure 10.2h](#) are listed the various facets that the user can configure: Document Plugins, Search Indexes, Partition Indexes, and Browsing Classifiers. Appearance and functionality vary depending on which item is selected. The figure shows the Document Plugins screen, which allows users to add, remove, or configure plug-ins and to change the order in which they are applied to documents.

Plug-ins and classifiers have many different options. The dialog box in [Figure 10.2i](#) shows the user specifying arguments to one of the plug-ins. The grayed-out fields become active when the user adds the option by clicking the tick-box beside it.

Because Greenstone is a continually growing open source system, the number of options tends to increase as developers add new facilities. To help cope with this, there is a “plug-in information” utility program that lists the options available for each plug-in along with an explanation of what each does, and the Librarian interface automatically invokes this to determine what options to show. The explanation appears as a tool-tip when the user hovers the mouse cursor over the option, as seen in [Figure 10.2i](#) for the *windows_scripting* option; the explanation uses the language the Librarian interface is set to. This allows the interactive user interface to automatically keep pace with developments in the software.

Building the collection

The Create panel ([Figure 10.2j](#)) is used to construct a collection based on the documents and assigned metadata. The brunt of the work is borne by the underlying command-line scripts that the Librarian interface initiates. The user observes the building process through a window that shows the text output generated by the importing and index-building scripts, along with a progress bar.

At the top of [Figure 10.2j](#) are some options that can be applied during the creation process, which are really for more advanced use of Greenstone. On this occasion, no changes to the options have been made, and the user has proceeded to clicking Build Collection.

Formatting the pages

Users can access the Format panel ([Figure 10.2k](#)) before or after they build the collection, to determine the appearance of each page in it. Changes made here take effect immediately. Again, the process involves a series of separate interaction screens, each screen dealing with one aspect of the formatting.

Operations include:

- Reviewing and editing collection-level metadata, such as title, author, and public availability of the collection.
- Assigning names to the various search indexes in the collection.
- Assigning formatting strings to controls within the collection.
- Supplying text in other languages for users who change the interface language.
- Determining which other collections are to be searched when this one is.
- Supplying collection-specific macros (Section 12.1) to control the appearance of each page.
- Customizing metadata sets for the Depositor, a Greenstone facility for institutional repositories (Section 11.6).

On the left of [Figure 10.2k](#) are listed the various facets that the user can configure: General, Search, Format Features, Translate Text, Cross-Collection Search, Collection Specific Macros, and Depositor Metadata. Appearance and functionality vary among these facets. In [Figure 10.2k](#), the user has clicked the General tab and is reviewing the information that was entered when the collection was created.

Previewing

The **Preview Collection** button (grayed out in [Figures 10.2j](#) and active in [Figure 10.2k](#)) is used to view the collection that has been built. It launches a Web browser showing the collection's *About* page ([Figure 10.2l](#)). In practice, previewing often shows up deficiencies in the collection design, or in the individual metadata values, and the user frequently returns to earlier stages to correct them. The newly created collection is automatically installed as one of the collections on the home page of your Greenstone installation.

Help

Online help is always available and is invoked using the Help item at the right of the main menu bar at the top of each figure. This opens up a hierarchically structured file of help text, and account is taken of the user's current context to highlight the section that is appropriate to the present stage of the interaction. Furthermore, as noted above, whenever the mouse cursor is held still over any interactive object, a small window pops up to give a textual "tool tip," as illustrated in [Figure 10.2i](#).

10.3 Working with Documents

This, as we warned at the beginning, is where the work begins. We describe the process of building several small collections, and we *strongly* encourage you to work through the process on your own computer. Again, actions are written in boldface. Building a collection takes just a few minutes, but equipping it with metadata and appropriate searching and browsing facilities takes longer. Nevertheless, you should be able to work through this section in half a day or so. We use HTML, Word, and PDF files, because they are the most common types used in practice.

HTML documents

We start with a tiny collection of HTML Web pages. We recommend that you download the files from the book Web site (www.greenstone.org/howto) that are expressly designed for use with these exercises. The *simple_html* folder contains sample files for this collection. Alternatively, you could use your own HTML files.

First, start the Librarian interface. In Windows,

Start→All Programs→Greenstone-2.83→Librarian Interface

After a short pause, a black command window appears, and then, after a slightly longer pause, the main Librarian interface appears. You do not need to use the command window any more: minimize it if you like. On Mac or Linux, open a command window, change to the folder in your Greenstone installation called *gli*, and execute the program *.gli.sh*, which starts up the Librarian interface program under Java—only without the annoying black window that pops up under Windows.

Next, start a new collection within the Librarian interface:

File→New

A window pops up. Fill it out with appropriate values—for example, the title *Small HTML Collection* and the description *A small collection of HTML pages*. Whatever you write here will appear on the collection’s *About* page—you looked at this page for the Demo collection at the beginning of [Section 10.1](#). Leave the setting for *Base this collection on* at its default (New Collection) and click **OK**.

You are now on the **Gather** panel. Currently the collection, which is represented by the large green area to the right of the interface, is empty. Notice, by the way, that you can re-size the panels by mousing over the gray bar that separates them—the shape of the pointer changes—and dragging. This kind of functionality is supported throughout the Librarian interface. The next step is to gather together files for the collection. A suitable set has been placed in the *simple_html* folder. Using the left-hand (blue) side of the **Gather** panel, interactively navigate to this folder (or some other folder containing a few HTML files). Drag the entire folder from the left-hand side and drop it into the right-hand (green) area.

At the bottom of the window is a status area that shows the progress of actions involving files (copying, moving, and deleting). The **Stop** button stops any action currently in progress. In this case the progress bar at the bottom shows some brief activity as the files are quickly copied across into the collection. You can inspect what has been copied by double-clicking on the folder in the right-hand side.

Since this is the first collection, we won’t complicate matters by manually assigning metadata or altering the collection’s design; instead we rely on default behavior. So pass directly to the **Create** panel by clicking its tab and start building the collection by clicking the **Build Collection** button.

A window pops up to alert you when the collection has been built. Click **OK**; then click the **Preview Collection** button to examine the end result. This loads the relevant page into your Web browser (starting it up if necessary).

When you look at the collection through the Reader’s interface, you will see that the navigation bar has three buttons—*search*, *titles*, and *filenames*. This is the default structure for collections produced by the Librarian interface. The *filenames* browser is not particularly interesting, but is included to give a way of accessing all documents in the collection. If a document does not, for some reason, have a title, it will not appear in the *titles* list; but all documents have filenames.

The *titles* browser shows the three documents in the collection. Where did the titles come from? Greenstone extracted them from the HTML documents.

Back in the Librarian interface, click the **Enrich** tab to view the metadata associated with the documents. To the right is a table that shows metadata for any selected files or folders in the collection. Columns can be re-sized by dragging the separating line in the column header, which is gray. Navigate to one of the documents by double-clicking the *simple_html* folder to expand its contents and single-clicking *aragon.html* to display all its metadata in the right-hand side of the panel. The initial fields, starting *dc.*, are empty. These are Dublin Core metadata fields for manually entered data, and presently there is no manually assigned metadata. However, the act of building the collection has extracted metadata from the documents.

Use the scroll bar on the right to view the bottom part of the list. There you will see fields starting *ex* that express the extracted metadata. For example, *ex.Title* is based on the text within the HTML `<title>` tags (see Section 4.3), and *ex.Language* is the document's language (represented using the ISO standard 2-letter mnemonic), which Greenstone determines by analyzing the text.

Now return to the Web browser and look at the documents in the collection. Like most HTML documents, they contain hyperlinks. Hyperlinks in a Greenstone collection work like this: if the link is to a document that is also in the collection, clicking it takes you to that document in the collection. If the link is to a document that is not in the collection, clicking it takes you to that document on the Web.

Because the collection's so tiny, most of its links are external. Try clicking almost any link: it will take you out onto the Web. However, there is an internal link. Open *boleyn.html* and look for the link to Katharine of Aragon in the fifth paragraph of the Biography section. This link is to a document inside the collection, namely *aragon.html*. View this document by clicking the link and check that it takes you to the document within Greenstone.

Word and PDF files

Before continuing, we advise you to set up a shortcut within the Librarian interface to make it easy to access the sample files. In the **Gather** panel, navigate to the folder in your local file space that contains the files you want to use—in our case, the *sample_files* folder that contains all the files you will need for these exercises. Select it, right-click it, and choose **Create Shortcut** from the menu. You can enter a name for the shortcut in the Name field, or just accept the default *sample_files*; then click **OK**. Close all the folders in the file tree in the left-hand pane, and you will see the shortcut to your source files. This will make life much easier for you in future.

Before going on to make a new collection of Word and PDF files, you could close the one you have just been working on by clicking **File→Close**. In fact, this is unnecessary because Greenstone always saves what you have done: the Close menu item is provided only because otherwise some users become nervous. Feel free to close your collection if that makes you more comfortable!

Start a new collection (**File→New**) called *reports*, again based on the default (New Collection). In the **Gather** panel, open the *Word_and_PDF* folder in *sample_files*, then the *Documents* folder, and copy the files into the collection. There are nine: four Word files (*.doc*), two PDF files (*.pdf*), one RTF file (*.rtf*), and two PostScript files (*.ps*). You can copy them one after another, or select multiple files by clicking on the first, shift-clicking on the last, and dragging them all across together. (This is the standard technique of multiple selection.) Switch to the **Create** panel and click **Build Collection**. When building has finished, **Preview** the collection. It contains nine documents.

We will now do some work in the Enrich, Design, and Format panels to illustrate their purpose.

Enriching with metadata

In the Librarian interface, click the **Enrich** tab to view the automatically extracted metadata. This collection contains no manually assigned metadata, and you will need to scroll down to see the

extracted metadata, which begins with *ex*. Titles have been automatically extracted from the documents, and you will find that some are incorrect. This is often true for extracted data: it's cheap and cheerful, and sometimes wrong.

You can add metadata manually to documents. Select one and enter the correct title into the *dc.Title* box, which represents the Dublin Core Title field. Note that documents can be opened from the Librarian interface by double-clicking them, which makes it easy to check the titles (for this to work—particularly on Linux—you may need to set it up as explained at the end of [Section 10.7](#)). You can then transfer the title using copy-and-paste. However, when you open documents by double-clicking, be sure to close them (in Microsoft Word, or Acrobat, or whatever) when you have finished with them. In Windows, external programs that remain open to view documents can cause errors in collection building.

Add Dublin Core Creator metadata as well. Some documents have several authors, and several values can be added for the same field. When you press *Enter* in a metadata value field, the interface generates a new empty field of the same type. Instead of putting all the authors into the same field, it's better to put each one into a separate field. Continue to add Title and Creator metadata for a few of the other documents. As you add more values, they appear in the Existing Values box below the metadata table. If you are adding the same metadata value to more than one document, you can select it from this list. This is often the case for authors (and keywords), and in this collection some documents even share the same title.

Note that although you can add and alter Dublin Core metadata, you are not able to edit extracted metadata. This is because the next time the collection is built, the metadata will be extracted again, which would overwrite any changes you made.

If you build and preview the collection at this point, you will see that the *titles* list now shows the newly entered Dublin Core titles in preference to the extracted ones. However, the Creator metadata is not displayed. You need to alter the collection design to use this metadata.

Designing the collection

Go to the **Design** panel and click **Document Plugins** in the list on the left. Greenstone processes documents using software modules called *plug-ins* when bringing them into the collection. Here you can add, configure, or remove plug-ins to be used in the collection. There is no need to remove any plug-ins. However, in this collection there are only Word, PDF, RTF, and PostScript documents, and it would do no harm to remove the ZIP, Text, HTML, Email, Image, ISIS, and Nul plug-ins. To delete a plug-in, select it and click **Remove Plugin**. GreenstoneXMLPlugin is required for any type of source collection and should not be removed.

The next step in the design panel is **Search Indexes**. These specify what parts of the collection are searchable (e.g., searching by title and author), and can be based on any metadata field, or combinations thereof. Delete the **ex.Source** index, which refers to filenames and is not particularly useful, by selecting it and clicking **Remove Index**. Add a new index based on Creator metadata by clicking **New Index**, selecting **dc.Creator** in the list of metadata, and clicking **Add Index**.

The next step in the design panel is **Partition Indexes**, which is used to split the full-text indexes into different parts that can be searched separately. We will not do this now. Instead click **Browsing Classifiers**, which adds *classifiers*. These are modules that equip the collection with browsing functions. Observe that Greenstone has provided two classifiers. They correspond to the *titles* and *file-names* buttons on the collection's navigation bar. Both are alphabetically ordered lists: the former is based on Dublin Core and extracted Title; and the latter is based on extracted Source (i.e., filename) metadata. Remove the **ex.Source** classifier by selecting it and clicking **Remove Classifier**.

Now add a new classifier: we'll use a List classifier for Creator metadata. Ensure that **List** is selected in the drop-down list beside **Select classifier** (it's selected by default) and click **Add Classifier**. A window pops up in which you configure the arguments for this classifier. Select **dc.Creator** from the **metadata** drop-down list, check the box labeled **bookshelf_type**, and select from its drop-down list the item that starts "always – Create a bookshelf icon even ...". Finally, click **OK**. This classifier lists all items in alphabetic order of the specified metadata element, grouping values that appear multiple times together into a new node, shown as a bookshelf icon. Switching on **bookshelf_type** and setting its drop-down list entry to "always" ensures that a bookshelf is created even for a value that only appears once. (Recall the **organization** button in the Demo collection, which brought up a list of organizations represented by bookshelves.)

Switch to the **Create** panel, and build and preview the collection. At this point you may become annoyed with the window that constantly pops up to alert you to the fact that the collection is ready for previewing. To suppress this, tick the check-box on it to prevent it from being shown again.

Check that all the facilities work properly. There are three full-text indexes, called **text**, **dc.Title**, **ex.Title**, and **dc.Creator**, which can be selected for searching. On the navigation bar are *search*, *titles*, and *creators* buttons. The *creators* list shows one bookshelf for each author that has been assigned as Creator metadata. Clicking that bookshelf takes the reader to all documents that that person authored.

Changing the format

The display text for the indexes in the drop-down list on the search page contains the metadata field name for the index, which looks ugly. To change this display text, switch to the **Format** panel by clicking its tab. This panel is split into several sections, each controlling some aspect of presentation; we will only touch on it here. Select **Search** in the left-hand list. Here you can modify the text displayed for the drop-down lists in the search form. Set the **Display text** for the **dc.Creator** index to *authors*. (Note: for multilingual interfaces it is usual to specify a macro here, rather than verbatim text.) When changes are made in the **Format** panel it is not necessary to rebuild the collection using the **Create** panel. Instead, just click the **Format** panel's **Preview Collection** button. The new text is displayed in the search menu.

Enhanced Word document handling

Greenstone processes Word documents by converting them to HTML format using a third-party, open source program (called *wvWare*). Parsing Microsoft Word files is a tough job given that, for the main part, the company does not publish internal details about the format. This means that the con-

verted version is often a poor rendering of the original document. One solution is to edit it by hand. If you right-click on the Word document in the **Gather** or **Enrich** panels, you can replace it with the HTML file. This is an irreversible process: the original document is removed from the collection. You can then edit the converted (HTML) version. To facilitate this, you could use the File Associations item on the File menu (described more fully at the end of [Section 10.7](#)) to associate an HTML editor with HTML files, whereupon it will be used to open such files when they are double-clicked.

An alternative, if you are using Windows with Microsoft Office installed, is to co-opt the Word application to automatically perform the conversion instead of *wvWare*, using Windows native scripting. If any properties were defined in the original Word version of the document, Greenstone can capture them as metadata. If the original document was structured using Word styles, they can be captured too. It is also the case that Word can often do a better job of converting documents to HTML form (especially if the result is then viewed in Internet Explorer) and works on a broader range of files.

Extracting document structure

Use the Reader's interface to check the HTML versions of the Word documents in the collection you have just built. Note that they have no structure—they have been converted to flat text. Returning to the Librarian interface, switch to the **Design** panel and select the **Document Plugins** section on the left-hand side. Configure **WordPlugin** by double-clicking it. Plug-ins (like classifiers) have many options, and they vary from one plug-in to another. Explanatory text appears when the mouse cursor hovers over each option. Notice that the options are arranged in groups corresponding to the different kinds of operation that the plug-in performs. If you are running the software on a Windows computer, a *windows_scripting* option will appear; switch it on. Return to the **Search Indexes** section and check the **section** box to build the full-text indexes on section level as well as document level.

Build the collection. You need to have Microsoft Word installed for this part of the exercise, and you will see it starting up for each Word document—the document is saved as HTML from Word itself. When you preview the collection you will notice that two of the documents (*word03* and *word06*) now have a book icon rather than a page icon in the *titles* list. These documents appear with hierarchical structure.

The Word plug-in splits the document according to Word's *Heading 1*, *Heading 2*, and *Heading 3* styles, and if you open these documents in Word, you will see that the sections use these styles. (Use Word's *Format/Style* menu to see which styles a document uses.) If documents do not use styles (e.g., *word01*), no structure can be extracted. If there are user-defined styles, the plug-in can be configured to recognize them.

Detecting user-defined styles

If you examine *word05* in Word (by double-clicking it in the **Gather** or **Enrich** panel), you will find that it uses these styles:

- ManualTitle
- ChapterTitle

- SectionHeading
- SubsectionHeading
- AppendixTitle.

Back in the **Design** panel's **Document Plugins** section, double-click **WordPlugin** to configure it. Four types of header can be set, defining which styles are to be treated as the title and level 1, 2, and 3 headers. Note that these options cannot be seen unless you are running Windows. Ensure that *windows_scripting* is checked, and set the options as follows. (Spaces in Word styles are removed when they are converted to HTML styles, and these options must match the HTML styles).

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

The parentheses in the first specification do not form part of the text, and the vertical bar is interpreted as OR. The result is that both ChapterTitle and AppendixTitle are treated as level 1 headers. This is because the specifications above are interpreted as regular expressions. If you have documents with different formatting styles, you can use (...) to specify the alternatives.

Once these headers are set, click **OK**, build the collection (be sure to close any documents that are still open in Word first), and preview it. Look in particular at *word05*: it is now hierarchically structured.

But there is still a small problem. In the HTML versions of *word05* and *word06* there are two tables of contents. One is generated by Greenstone based on the document's styles; the other was already defined in the Word document. The plug-in can be configured to remove predefined tables of contents (and other items)—provided they are defined with Word styles.

To remove the tables of contents and figures from *word06* and the table of contents from *word05*, switch on the *delete_toc* option in the **Word** plug-in. Set the *toc_header* option to

```
(MsoToc1|MsoToc2|MsoToc3|MsoTof|TOA)
```

which lists the style names used by the table of contents and list of figures in this document. Click **OK**, build and preview the collection, and check that these two documents now have only one table of contents.

Extracting document properties

Microsoft Word allows authors to define properties for their documents, and Greenstone can extract these properties as metadata. Only the title is extracted by default, but other properties can be extracted, too. For example, if you open *word05* and *word06* in Word and check their properties (File→Properties), you will see that they have titles, authors, subjects, and keywords. However, if you examine the metadata that has been extracted from these documents in the **Enrich** panel, you will find only titles.

To extract the other properties, in the **Design** panel, under **Document Plugins**, configure the **Word** plug-in once again. Check that *windows_scripting* is still set and switch on the *metadata_fields* option. Set its value to

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

This instructs the **Word** plug-in to seek Title, Author, Subject, and Keywords metadata. The angle brackets after Author rename this metadata: from now on it will be Creator metadata. (Later on, in [Section 10.4](#), “Formatting Exercise 2,” we return to this collection and want to refer to it this way.) Ensure that you have closed all the documents that were opened, then rebuild the collection.

Now *ex.Creator*, *ex.Subject*, and *ex.Keyword* metadata for the two documents appear in the **Enrich** panel. This metadata can be used for searching or browsing. In fact, you can do this easily because if you followed through the instructions in the previous section, you already have an author classifier. But it only uses Dublin Core Creator metadata, and you need to make it use extracted Author metadata too. To do so, configure it in the **Design** panel. Under **Browsing Classifiers** double-click the **List** classifier, and in its configuration panel change the metadata it is based on to **dc.Creator;ex.Creator**. You cannot do this by selecting; you will have to type it. By the way, the order is significant: it specifies that Dublin Core metadata is preferred, and extracted metadata is used only if the Dublin Core field is empty.

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

Enhanced PDF document handling

As we saw in the last section, the default method for processing Word documents can sometimes be improved upon, and the same is true for PDF. For PDF files, the default conversion is also done by a third-party program (in this case, *pdftohtml*). Converting to HTML lets users view these documents even if they don’t have PDF-reading software installed; however, the formatting can leave much to be desired. Also, it is sometimes more convenient to display the PDF files as a sequence of page images—for example, the original files might be very large and best viewed page by page. The PDF plug-in has options for this kind of display. Some are based on the default conversion program, but others use different open source programs—called ImageMagick and Ghostscript. The former copes with general image conversion; the latter converts Postscript and PDF files to other formats.

Trouble-shooting PDF collections

In the Librarian interface, start a new collection called *PDF*, based on New Collection as usual. In the **Gather** panel, from **sample_files** → **Word_and_PDF** → **Documents**, drag just the two documents, *pdf01* and *pdf03*, into the new collection, and also drag in the two documents in **sample_files** → **Word_and_PDF** → **difficult_pdf**. Go to the **Create** panel and build the collection. You will notice messages from the build process stating that one document could not be processed, and saying why:

```
The file pdf05-notext ... could not be processed by any plug-in
3 documents were ... included in the collection; 1 was rejected
```

When you preview the collection, you will see that *pdf05-notext* does not appear at all and *pdf06-weirdchars* looks very strange. The other files are presented as long documents with no sectioning. However, in order to work out why *pdf05-notext* could not be processed, you need to switch the Librarian interface to Expert mode.

Switching modes in the Librarian interface

We note in [Section 10.2](#) that the Librarian interface has three different modes, each targeted at different levels of user. Click **File→Preferences→Mode** and see what functionality they give access to. *Librarian* is the default mode. However, we will need to examine a detailed list of what happened during processing in order to determine what has gone wrong, and for this you must switch to **Expert** mode.

Having done that, build the collection again. The **Create** panel looks different because Expert mode gives more options: the **Build Collection** button appears near the bottom of the window. Now, amid all the output, a message appears saying that the file could not be processed, and why:

```
Error: PDF contains no extractable text
Could not convert pdf05-notext to HTML format
```

This particular file is a sequence of images, without any explicit text, and Greenstone cannot currently extract text from images.

To avoid confusion, we recommend that you switch back to Librarian mode for subsequent exercises.

Splitting PDF documents into sections

Greenstone can split PDF files into individual pages. In the **Document Plugins** section of the **Design** panel, configure the PDF plug-in by switching on the *use_sections* option. Under **Search Indexes**, check the *section* check-box to build the indexes on the section level as well as the document level, allowing readers to search by page as well as by document. Build and preview the collection. The documents are now split into a series of pages, with a *go to page* page-turning widget. However, the format is ugly, and *pdf05-notext* is still not processed.

Converting PDF documents to page images

These problems can be solved by converting PDF documents to a series of images, one per page. Return to **Document Plugins** and configure the PDF plug-in again, setting the *convert_to* option to one of the image types, e.g., JPEG format. (The *use_sections* option is not used with image conversion, so you can switch it off.) Build and preview the collection. All the documents—including the one that could not be processed before—have been processed and split into pages, but—disappointingly—instead of the text, a message saying *This document has no text* appears. When PDF files are converted to images, no text is extracted.

In order to see the document contents, you need to modify the format statement that displays the document text. Format statements are perhaps the most difficult part of Greenstone, and we discuss them later, in [Section 10.4](#). But only a simple change is needed to see these PDF documents, and the result is so satisfying that we describe how to do it here. Go to the **Format** panel and select the **Format**

Features section. There is a drop-down menu in the middle that says *All Features*. Instead, select **DocumentText** from the drop-down list. Under **HTML Format String**, replace

```
[Text]
```

with

```
[srcicon]
```

Be careful that you get this right! Now, when you preview the collection, page images are displayed instead of the extracted text. Both of the troublesome files appear, and both look respectable. (Note that the format statement applies to all documents in the collection, so if there were other kinds of documents, they would not be displayed correctly. For this, a conditional format statement is needed, which is explained in [Section 10.4](#).)

Working with mixed PDF collections

One disadvantage of using image conversion is that, since no text is extracted, none of the documents can be searched. This cannot be avoided for those whose source files contain page images, but it may be worthwhile to treat ordinary textual PDF files in the normal way so that they can be searched. To do this, the image format should be used only where the HTML conversion does not work properly. This can be achieved by including two copies of the PDF plug-in, configuring them differently, and directing them to process different sets of files. The simplest way is to place the problem files in a separate folder.

You can make folders and reorganize files in the **Gather** panel; you can also delete parts of the collection by dragging files and folders into the trash (or by right-clicking and selecting **Delete**). To create a new folder, right-click anywhere in the area that represents the collection and choose **New Folder**, or else select an existing folder in the file tree and click the **New Folder** button. You will be prompted for the folder's name, and the new folder will appear within the selected one—or at the top level if none is selected.

Go to the **Gather** panel and make a new folder; call it *notext*. Drag the two troublesome files (*pdf05-notext* and *pdf06-weirdchars*) into it. We will configure Greenstone to treat files in this folder differently from other PDF files.

Go to the **Design** panel, **Document Plugins** section. Add a second PDF plug-in by selecting **PDF-Plugin** from the drop-down list and clicking **Add Plugin**. It will appear at the end of the plug-in list. For each document the plug-ins are checked in order, and the first relevant one is applied to the document. This new plug-in will process PDF files in the normal way, converting them to HTML (which is the default). Switch on the **use_sections** option and click **OK**.

Now configure the *first* PDF plug-in to process only files in the *notext* folder that you created above by setting its *process_exp* option to *notext.*\pdf*. This is a regular expression that specifies any file-name that includes the characters *notext*, followed by any characters (that is the meaning of “.”), followed by a literal period character (“\.”), followed by *pdf*. Such is the mysterious language of regular expressions. Then set this plug-in's *convert_to* option to JPEG format. Note that it is essential that this plug-in precede the one that we just added. It will process only files that match the regular

expression, that is, only files that include *notext* in their filename; in this case, only files that appear in the *notext* folder. The second PDF plug-in will process any PDF documents that are not processed by the first one. All plug-ins have the *process_exp* option, and this can be used to customize which documents are processed by which plug-in.

In order to be able to see the documents properly you will have to use a more complex format statement. Check the instructions above for the **DocumentText** format statement, and change it to make it read

```
{If}{[NoText],[srcicon],[Text]}
```

Once you have read [Section 10.4](#), the meaning of this rather arcane instruction will become clearer. Build and preview the collection, and check how its documents look. When searching you will be able to find the documents that were converted to HTML (e.g., search for *bibliography*), but not the ones that were converted to images (e.g., *METS*).

Highlighting search terms

Finally, it would be nice, after searching for a document, to display it with the query terms highlighted if the PDF icon is clicked. This is possible using Acrobat Reader version 7.0 or higher and currently works only on a Microsoft Windows platform. It involves a new format statement, and a fairly intricate one at that, so we will not explain how to do it here—however, in [Section 10.7](#) we explain how to find out this information from the Greenstone mailing list archive.

Enhanced HTML document handling

We learned above how to instruct Greenstone to pick up additional metadata that can be specified in Microsoft Word documents and to detect any hierarchical division into sections and subsections. The same can be done with HTML. HTML files that have been downloaded from the Web also have another very common requirement: to ensure that when a document is clicked, it is retrieved from the original Web site for presentation to the user, rather than being displayed by Greenstone from its local copy. This is very easy to do, but since it involves a format statement, we leave it to [Section 10.4](#).

Extracting metadata

Return to the tiny collection of HTML documents that you built from the *simple_html* folder in the first Librarian exercise above. There you examined the metadata that was extracted from some of the documents and found *ex.Title* metadata, based on the text within the HTML `<title>` tags, and *ex.Language*, the document's language, which Greenstone determines by analyzing the document's text.

Many Web documents contain metadata in `<meta>` tags in the `<head>` of the page. In the **Enrich** panel, open up the *boleyn.html* file by double-clicking it in the tree on the left-hand side. In your Web browser, view the HTML source of the page (from the View menu). If you check carefully, you will see that this page has *page_topic*, *content*, and *author* metadata (and more besides).

To configure the HTML plug-in to look for this metadata, switch to the **Design** panel and select the **Document Plugins** section. Double-click **HTMLPlugin** to configure it. Switch on the **metadata_fields** option and set the value to

```
Title, Author, Page_topic, Content
```

Click **OK**. (Note that capitalization is *not* taken into account when matching against metadata field names in the HTML file.) Now rebuild the collection, return to the **Enrich** panel and examine the extracted metadata for the *boleyn.html* file. The new metadata will now be visible.

As we mentioned earlier with Word documents, you can rename this metadata at the same time. For example, if you set the *metadata_fields* to

```
Title, Author <Creator>, Page_topic <Description>, Content
```

you will find that the author and page topic metadata have been saved as *ex.Creator* and *ex.Description*, respectively. We will not bother using this metadata now, although it would be very easy to do so.

Extracting document structure

Greenstone can also detect the HTML heading structure (`<h1>`, `<h2>`, ...). To do this, double-click the HTML plug-in and enable the *sectionalize_using_h_tags* option. This automatically creates a sectioned document using the standard HTML heading tags, where the contents of the `<h1>`, `<h2>`, ... tags will be used as section titles.

Metadata for hierarchical documents

For Word documents, Greenstone can identify user-defined styles as well as the standard ones. In contrast, HTML documents invariably use the standard header tags. However, there is a facility for allowing user-defined sections instead. The main reason for this is to provide a way of associating metadata with individual sections.

Documents in Greenstone are structured hierarchically, and metadata can be defined at any level of the hierarchy. For example, section titles are defined as standard *Title* metadata that is associated with the section itself instead of the entire document. It is possible to define any kind of metadata at any level of a hierarchical document. In an edited book, chapters might be written by different authors and have different keywords, and these items can be defined as metadata elements associated with that level of the book.

To accomplish this, users define sections in HTML documents and tag them with metadata by subverting HTML comments to specify additional information. For example, [Figure 10.3](#) demonstrates a document with two chapters, the second of which contains two subsections. When Greenstone encounters a `<Section>` tag in one of these comments, it starts a new subsection of the document. This is closed when a `</Section>` tag is encountered. Metadata can be added for each section—in this case, Title metadata has been specified for each section, and Creator metadata for the two top-level sections (chapters). Sections can be nested to create arbitrary tree-structured documents.

```

<!--
  <Section>
    <Description>
      <Metadata name="Title">Our book</Metadata>
    </Description>
  </Section>
  <Section>
    <Description>
      <Metadata name="Title">Chapter 1</Metadata>
      <Metadata name="Creator">David Bainbridge</Metadata>
    </Description>
  -->
  (text of chapter 1 goes here)
  <!--
    </Section>
    <Section>
      <Description>
        <Metadata name="Title">Chapter 2</Metadata>
        <Metadata name="Creator">Ian H. Witten</Metadata>
      </Description>
      <Section>
        <Description>
          <Metadata name="Title">Subsection 1</Metadata>
        </Description>
      -->
      (text of sub-section 1 goes here)
      <!--
        </Section>
        <Section>
          <Description>
            <Metadata name="Title">Subsection 2</Metadata>
          </Description>
        -->
        (text of sub-section 2 goes here)
        <!--
          </Section>
        </Section>
      </Section>
    -->
  -->

```

Figure 10.3: Section tagging for a document with two chapters

In order for this embedded information to be recognized by Greenstone, the *description_tags* option must be set in the HTML plug-in. In fact, this facility works in Word documents as well, because the key operation of WordPlugin is to convert the source document into HTML, which is then processed by the HTML plug-in. Consequently, *description_tags* is also available through the Word plug-in.

Scaling up

The collections we have dealt with so far have involved a tiny handful of documents, and some of the organizational features that Greenstone provides can only be meaningfully illustrated with larger collections. We will use a collection of around 150 Web pages to show how to add hierarchically

structured metadata and use it in a classifier, how to partition the full-text index based on metadata values, and how to control the building process.

Start a new collection called *tudor*, again based on the default (New Collection). In the **Gather** panel, open the *tudor* folder in *sample_files* and drag *englishhistory.net* into the collection. Switch to the **Create** panel, build the collection, and preview it. The titles list is now segmented into alphabetic ranges—four of them. Greenstone chooses these ranges to put about 20 to 30 documents on each page. If there are fewer documents in the collection, it simply suppresses the range mechanism, which is why you have not encountered it before.

Examining different file types

Back in the Librarian interface, it is often convenient with larger collections to look at different views of the files they contain. Switch to the **Gather** panel, and on the right-hand side open **englishhistory.net** → **tudor**. Change the **Show Files** menu for the right-hand side from **All Files** to **HTM & HTML**. Alternatively, you can show only images, or some other file types. Remember, if the Gather or Enrich panels do not seem to be showing all your files, this could be the problem. You can even specify a custom filter by typing in a pattern to match filenames against. Use standard file system abbreviations like **.doc* (* matches any characters).

Adding metadata

Next we manually add metadata to express a subject hierarchy; then build a classifier that exploits it by allowing readers to browse documents about Monarchs, Relatives, and Citizens separately. In the **Enrich** panel, select the *citizens* folder (a sub-folder of *englishhistory.net*→*tudor*). Set its Subject and Keywords metadata to *Tudor period|Citizens*. Greenstone uses the vertical bar (|) as a hierarchy marker. Selecting a folder and adding metadata has the effect of setting this metadata value for all files contained in this folder, its sub-folders, and so on. A pop-up alerts you to this fact; click **OK** to close it.

Repeat this for the *monarchs* and *relative* folders, setting their Subject and Keywords values to *Tudor period|Monarchs* and *Tudor period|Relatives*, respectively. Note that the hierarchy appears in space reserved for existing values of this metadata element, below the metadata list. If you don't want to see the pop-up each time you add folder-level metadata, tick its check-box to prevent it from being shown again. (We did this earlier, for the Preview message. File→Preferences has an item called Warnings that allows you to switch these messages back on if you like.)

Finally, select all remaining files—the ones that are not in the *citizens*, *monarchs*, or *relative* folders—by selecting the first and shift-clicking the last. Set their Subject and Keywords metadata to *Tudor period|Others*: this is done in a single operation. (There is a short delay before it completes. In fact, it will be quicker if you are showing only the HTML files, because then metadata is assigned to just these files—which is all you need.) When multiple files are selected in the left-hand panel, all metadata values for all files are shown on the right-hand side. Items that are common to all files are displayed in black—e.g., Subject and Keywords—while others that pertain to only one or some of the files are displayed in gray—e.g., any extracted metadata. Metadata inherited from a parent folder is indicated by a folder icon to the left of the metadata name. Select one of the files in the *relative* folder to see this.

Adding a hierarchy classifier

Switch to the **Design** panel and select **Browsing Classifiers** from the left-hand list. Find the **Hierarchy** classifier in the drop-down menu near the bottom and click **Add Classifier**. Configure this classifier to use Dublin Core Subject and Keywords by selecting that from the list; then click **OK**. For tidiness's sake, remove the classifier for Source metadata from the list of currently assigned classifiers, because this adds little to the collection. Now switch to the **Create** panel, build the collection, and preview it. Choose the new *subjects* link that appears in the navigation bar, and click the bookshelves to navigate around the four-entry hierarchy that you have created.

Partitioning the index

Next we partition the full-text index into four separate pieces. We first define four subcollections obtained by filtering the documents according to a criterion based on their Subject and Keywords metadata. Then we assign an index to each subcollection, which will enable users to restrict their searches to a subset of the documents.

Switch to the **Design** panel and click **Partition Indexes**. Ensure that the **Define Filters** tab near the top is selected (the default). Define a subcollection filter with the name *monarchs* that matches against *dc.Subject and Keywords*, and type *Monarchs* as the regular expression against which to match it. Click **Add Filter**. This filter includes any file whose Subject and Keywords metadata contains the word *Monarchs*. Define another filter, *relatives*, which matches Subject and Keywords against the word *Relatives*. Define a third and a fourth, *citizens* and *others*, which match against the words *Citizens* and *Others*, respectively.

For these filters we simply entered the text *Monarchs*, *Relatives*, etc. to match against, and any document that contains this text as part of its Subject and Keywords metadata will be included by the filter. However, more sophisticated matching can be achieved by entering a general regular expression. Also, you can either include documents that match the specification, or exclude them. For example, had the *Others* metadata values not been defined explicitly, a filter that *excluded* (*Monarchs OR Relatives OR Citizens*), formulated as a regular expression, would have achieved the same effect. Finally, you can specify any of the standard Perl regular expression flags to use when matching (e.g., “i” for case-insensitive matching).

After defining the subcollection filters, the next job is to partition the index into corresponding parts. Click the **Assign Partitions** tab. Select the *citizens* subcollection and click **Add Partition**. Next select *monarchs*, and click **Add Partition**. Repeat for the other two subcollections, so that you end up with four partitions, one based on each subcollection filter. The order in which they appear in the list here is the order in which they will appear in the drop-down menu on the search page. You can change this using the **Move Up** and **Move Down** buttons.

Viewing and refining the collection

Build and preview the collection. The search page includes a drop-down menu that allows readers to select one of the partitions for searching. For example, try seeking *mary* in the *relatives* partition and then seek the same word in the *monarchs* partition. To allow users to search the collection as a whole as well as each subcollection individually, return to the **Partition Indexes** section of the **Design**

panel and select the **Assign Partitions** tab. Select all four subcollections by checking their boxes (or use the **Select All** button) and click **Add Partition**. Ensure that the combined index appears first in the list on the reader's Web page by using the **Move Up** button to bring it to the top of the list in the Design panel.

Then build and preview the collection. Search for a common term (like *the*) in all five index partitions, and check that the numbers of words (not documents) add up.

The text in the drop-down box on the search page is based on the filters on which each partition was built. To change it, go to the **Search** section of the **Format** panel. The single filter partitions have sensible default text, but the combined partition does not. Set the display text for the combined partition to *all* and preview the collection (it is not necessary to rebuild it).

Developing a new collection usually involves numerous cycles of building, previewing, adjusting enrich and design features, and so on. While prototyping, it is best to temporarily reduce the number of documents in the collection. The building process has a parameter that accomplishes this. Switch to the **Create** panel and view the options that are displayed in the top portion of the screen. Select **maxdocs** and set its numeric counter to 3. Now build.

Preview the newly rebuilt collection's Titles page. Previously this listed more than a dozen pages for each of four alphabetic ranges, but now there are just three—the first three files encountered by the building process. Remember to deselect **maxdocs** when you want to rebuild the entire collection.

10.4 Formatting

One of the trickiest parts of Greenstone is the way in which users control how text and images appear on the screen. The Web pages that readers see are not pre-stored but are generated on the fly as they are needed. Format commands are used to change the appearance of the pages. The mechanism is extremely flexible: unfortunately, it is tricky to use. The formatting language is powerful but mysterious. And, in truth, it is not very interesting. The sections of this chapter that follow tell you how to go beyond textual documents and to work with files containing metadata, images, audio, and other multimedia—which *is* interesting, and not very difficult at all. But in order to see the results of your work, you need to know something about the format mechanism.

The Format panel

The Format panel of the Librarian interface contains several parts, selected using the facets on the left. When you make changes to any part of the format, there is no need to rebuild the collection; simply click the **Preview** button to view the changes. We will spend most of our time with Format Features, but first we review the other parts.

The *General* part of the format panel allows you to edit the information you supplied when creating the collection: its title, and—importantly—the description that appears on the collection's *About* page. You can also specify an image that will appear (by default) on the top left of each page in the

collection, and another image that will be used to identify this collection on the Greenstone home page; and you can control whether the collection will be shown on that page.

The *Search* part is where you specify names for the search indexes, which will appear as display text in the drop-down menu on the collection's search page. (See the discussion of working with the collection of Word and PDF files in [Section 10.3](#).) For each index you create, just type in the name you want readers to see on the search page.

The *Translate Text* part gives you an opportunity to specify text fragments for parts of the collection's interface in other languages. Choose an entry from the list of features and its associated language-specific strings appear below. Select the target language from the drop-down list and type the translated text into the text area. You can add new translations, remove existing ones, or replace translations by an edited version.

Cross-Collection Search makes it possible to search several different collections as though they were one. Just specify which other collections should be searched along with the current one from the checklist provided (on this list, the current collection is ticked and cannot be deselected). To add other collections to be searched in parallel, simply click them. If only one collection is selected, there is no cross-collection searching. If the individual collections do not have the same indexes as each other (including subcollection partitions and language partitions), cross-collection searching will not work properly. The user will be able to search using only the indexes that are common to all collections.

The appearance of every page of the collection is governed by *macros*. The *Collection Specific Macros* view is where macros can be defined that are specific to a particular collection.

The *Depositor Metadata* view allows you to specify what metadata items should be requested when users add documents to the collection through the Web interface described in [Section 11.6](#).

Format Features

Go to the **Format** panel and click the **Format Features** link. This link affects things like which buttons appear when a document is shown, and what text and metadata are displayed by classifiers. You can apply a format command to anything in the **Choose Feature** drop-down list in the center of the panel, which you should investigate now. It includes a predefined list of features. Depending on which you select, there are two types of control. Some features are simply enabled or disabled, which is indicated by a check-box. Others require a textual *format string* to be typed.

Above the drop-down list is a panel that shows all the format strings that have been assigned to the collection. You cannot edit this directly.

For example, select the **DocumentArrowsBottom** feature from the drop-down menu. The only active control is **Add Format**. Enable this and you will notice that a corresponding entry is added to the format strings in the upper panel. You can determine whether this control is *true* or *false* using the Enabled check-box. The effect of this feature is to display next and previous arrows at the bottom of the page that shows the document. To see the arrows, go to the Demo collection, look at a document, and scroll down to the bottom of the page.

Table 10.2 lists the format options. We have already encountered DocumentText in the section on enhanced PDF document handling. Note that you can remove existing format strings or set them to the default value by clicking them on the panel above the drop-down menu showing all format strings assigned to the collection.

If you investigate the format commands in the drop-down list, you will find that the following are missing from Table 10.2:

- Document
- Search

Table 10.2: Collection-specific format options

Item	Description
DocumentImages	Display a cover image at the top left of the document page (default <i>false</i>)
DocumentTitles	Use DocumentHeading to display the title (default <i>true</i> ; only applies when DocumentImages is <i>false</i>)
DocumentHeading	Defines the document header shown at the top left of the document page (default: title metadata)
DocumentContents	Display table of contents (if document is hierarchical), or next/previous section arrows and page selector (default <i>true</i>)
DocumentButtons	Determines whether to show <i>Detach</i> and <i>Highlight</i> buttons with a document (default both)
DocumentText	Governs the format of the text that is displayed on a document page (default: center the text in a fixed-width table)
DocumentArrowsTop	Display next/previous section arrows at top of document (default <i>false</i>)
DocumentArrowsBottom	Display next/previous section arrows at bottom of document pages (default <i>true</i>)
DocumentSearchResultLinks	Display link to next and previous search results at the top of document pages (default <i>false</i>)
DocumentUseHTML	Display each document in a separate frame; options applicable to collections of HTML documents will be added to the Preferences page (default <i>false</i>)
SearchTypes	Whether the interface is for plain search, or form search, or both (default both)
NavigationBar	Provide a drop-down list in place of the usual horizontal navigation bar that contains search and classifier options
AllowExtendedOptions	Control the entire content of the document page by DocumentHeading and DocumentText format statements, suppressing detach, highlight, and expand buttons and automatically generated table of contents. The following metadata is set: [DocumentButtonDetach], [DocumentButtonHighlight], [DocumentButtonExpandText], [DocumentButtonExpandContents], [DocTOC] and can be embedded in DocumentHeading and DocumentText format statements

- CL1
- CL2
- (others starting with CL ...)

The first (which is unfortunately named) formats the table of contents of a document; it is rarely used and we will ignore it here. The second formats the search results page. The remainder format the pages corresponding to each classifier that has been included in the collection. The classifier pages are designated CL1, CL2, etc., up to the total number of classifiers.

The most important format items are those for the search results and various classifier pages, along with the document text itself (*DocumentText* in [Table 10.2](#)). They have an associated drop-down list called Affected Component for selecting which part of the feature the string applies to—either VList (for vertical list) or HList (horizontal list)—or, in some cases, DateList. The default value of VList is the only one we are concerned with here (the other options can be dismissed as vestigial). Below that is a text area for entering an HTML format string, which is grayed out in most cases.

Default format statements

A powerful and—let’s face it—difficult aspect of the format mechanism is the reason why some format statements are grayed out. The grayed-out format statements correspond to the search results page and the classifier pages. By default, the same format is used for each of these pages. You might have noticed when building your early collections that the *titles* and *filenames* browsers contained the same information. In fact, if you were to go back and look at your first collection ([Section 10.3](#)), the three documents are the same in both browsers but are listed in a different order—Anne Boleyn, Charles Brandon, and Katherine of Aragon in *titles* and Katherine, Anne, and Brandon in *filenames*. In fact, they are ordered by document title in the first case and filename (*aragon, boleyne, brandon*) in the second. It might be worthwhile to bring this to the reader’s attention by formatting the pages generated by the two classifiers differently, but in the default collection structure, the search results page and all classifier pages are generated by the same format string.

This default format string is specified on the **Format Features** pane under the **All Features** menu item. If you were to edit the format string under this menu item, you would find that the very same changes appear in the gray text under the CL1, CL2, and Search menu items too. Check this by making a random change to that string—you can always reverse it using the **Undo** button—and taking a look.

It’s easy to assign different formats to the search results page and various classifier pages. Select **Search** from the drop-down menu, ensure that **VList** is selected as the **Affected Component** (which it is by default), and click the **Add Format** button. This adds a new format string specifically for the search results page, a copy of the current default one. The panel changes from gray to white and the text becomes editable. Now you can make it different from the others. To remove it again, click **Remove Format**. You can also reset it to the default format string, and (as noted above) you can undo individual changes.

Format strings

What about the format strings themselves? Basically, they are fragments of HTML, with some additional facilities. A format string is interpreted in the context of a particular document, and one important facility is the ability to interpolate the value of any metadata item for that document into the output. This is done by enclosing the metadata name in square brackets. For example, this string will print out the title (italicized) and author of the document, assuming that the relevant Dublin Core items are defined:

```
<i>[dc.Title]</i>, by [dc.Creator].
```

Or, to put it a different way,

```
[dc.Creator] wrote "[dc.Title]."
```

These will result in something like

```
Hamlet, by William Shakespeare.  
William Shakespeare wrote "Hamlet."
```

respectively. Any text that is not in square brackets is interpreted as regular HTML. Any text that is not in square or angle brackets is just put out to the screen. This is how you display metadata values. The text of the entire document (which, in the case of *Hamlet*, could amount to some hundred pages) is represented by the word *text* in square brackets. Thus

```
Here's the whole thing:<br>[text]<br>THE END
```

would result in

```
Here's the whole thing:  
ACT I  
SCENE I. Elsinore. A platform before the castle.  
(about 130 pages of text omitted ...)  
A dead march. Exeunt, bearing off the dead bodies; after which a  
peal of ordnance is shot off  
THE END
```

Conditionals

This fact that metadata values can be interpolated into the output text introduces a need for conditional statements. Suppose the word *anonymous* is to be generated for documents whose Creator metadata is undefined, like

```
<i>[dc.Title]</i> (anonymous).
```

In order to accommodate such requirements, the format language allows *if* statements. These are conditional on such things as the existence or non-existence of certain metadata values (Creator, in this example), or on other variables. [Table 10.3](#) summarizes the structure of conditional statements, for reference. Of course, the word “if” must somehow be protected—otherwise it would be interpreted as part of the HTML string and printed out on the page—and Greenstone uses curly brackets {If} for

Table 10.3: Conditional expressions

Item	Description		
{If}{[metadata], action-if-non-null, action-if null}	If the value of this metadata element is defined, output the first clause. otherwise output the second clause. Either clause may be empty.		
{If}{[metadata] op value, action-if-true, action-if-not-true}	Conditional expressions can be string comparisons or numeric comparisons. Valid operators are:		
	String	Numeric	Meaning
	eq	==	equals
	ne	!=	not equals
	gt	>	greater than
	ge	>=	greater than or equal to
	lt	<	less than
	le	<=	less than or equal to
	sw		starts with
	ew		ends with
{Or}{[metadata], [metadata2], [metadata3]}...	Evaluate metadata items in turn and output the first one that exists. The last can be plain text, which always exists.		
nested If/Or	{If} can be arbitrarily nested; {Or} can have another conditional as its final option		

this purpose. To return to the anonymous author example, the following statement would have the desired effect of using the author’s name when it was known, and “anonymous” at other times:

```
<i>[dc.Title]</i> ({If}{[dc.Creator],[dc.Creator],anonymous})
```

“If” evaluates the first part ([dc.Creator]), and if it has a value, the second part is displayed (the value of [dc.Creator]). If the first part does not have a value, the third part is displayed (the string “anonymous”).

The same functionality could be achieved with an “or” statement:

```
<i>[dc.Title]</i> ({Or}{[dc.Creator],anonymous})
```

This evaluates the first part ([dc.Creator]), and if it has a value, displays it. (Note the difference from the “if” statement, which does not *display* the value of the first part.) If the first part does not have a value, the second part (the string “anonymous”) is evaluated and, if it has a value, displayed. In this case it does have a value—any literal string has a value. This overall effect is the same as that of the above “if” statement: to print the value of [dc.Creator] if there is one, and the string “anonymous” if not. Note that an “or” statement can have any number of parts, not just two, as in this example: it evaluates the first item and uses its value if one exists, and failing that moves on to try the next item, and so on. The “if” statement has only three parts: the condition, what to do if it is true, and what to do if it is false. In fact, the third part can be omitted, in which case nothing is done if the condition is false.

Referring to document nodes and multiple-valued metadata

The basic way of specifying metadata is to place it in square brackets, which displays the value of that particular metadata element for the current document or section. However, Greenstone documents are hierarchical, which introduces a need to refer to different parts of the hierarchy. For example, when a search is made in the Demo collection, the search results page contains the name of the relevant section for each hit, preceded by any enclosing sections, preceded by the book title. In general, it may be necessary to refer to the parent of a node, or all its ancestors. For this reason metadata names may be preceded by *parent* or *child*.

Another complication is caused by the possibility of multiple values for the same metadata element. These are treated as *siblings*. For example, [dc.Creator] specifies just the first author. If there are several authors they can all be referred to by [sibling:dc.Creator]. Space is used as the default separator, but different choices can be specified (for example, Table 10.4 contains an example in which colon is used as a separator). Also, it may be necessary to refer to a particular sibling—for example, the second or the last. Table 10.4 summarizes the ways in which one can refer to different hierarchically related components. The examples all use Dublin Core Title or Subject metadata, but any metadata could be used. Metadata names can also be prefixed by *cgisafe:*, which formats the value in a way that makes it safe to include in a URL.

Table 10.4: Extended metadata names

[parent:dc.Title]	Title of the immediate parent section
[parent(Top):dc.Title]	Title of the topmost parent section
[parent(All: '):dc.Title]	Titles of all parents, separated by colon (or whatever appears inside the quotes)
[child(All: '):dc.Subject]	Subjects of all children of the current node, separated by colon
[child(2):dc.Subject]	Subject of the second child of the current node (numbering starts from 1)
[child(last):dc.Subject]	Subject of the last child of the current node (<i>first</i> is also a valid option)
[sibling:dc.Subject]	All Subjects of the current section, separated by space
[sibling(All: '):dc.Subject]	All Subjects of the current section, separated by colon
[sibling(2):dc.Subject]	The second Subject metadata value for the current node (numbering starts from 1)
[sibling(last):dc.Subject]	The last Subject metadata value for the current node (<i>first</i> is also a valid option)
[parent:sibling:dc.Subject]	Sibling can be combined with parent to give all (or specific) values for the parent node(s); all parent and sibling qualifiers can be used
[child:sibling:dc.Subject]	Sibling can be combined with child to give all (or specific) values for the child node(s); all parent and child qualifiers can be used
[cgisafe:parent(Top):dc.Title]	Title of the topmost parent section, made safe for URLs
[cgisafe:sibling:dc.Subject]	All Subjects of the current section, made safe for URLs
[format:dc.Date]	A nicely formatted version of Date metadata (<i>format</i> can also be used with Language metadata)

Predefined items

Square brackets are used in format strings for other purposes as well. They are used to create hyperlinks to the document content—either the Greenstone version of the document or the source version if it has been converted from a different format, such as Word or PDF. They are used to generate an appropriate icon for other formats, e.g., the standard Word or PDF icons. They are used for document identifiers. Table 10.5 summarizes them. Some of these predefined items can best be understood in the context of a particular example.

Formatting exercise 1: Tudor collection

In order to complete the remaining practical exercises in this chapter, you need to gain some familiarity with format statements. If you are intending to read through the exercises rather than actually doing them, feel free to skip the remainder of this section. You will still be able to learn a lot about Greenstone’s capabilities without a detailed technical knowledge of format statements.

Open up your **tudor** collection, go to the **Format** panel and select **Format Features** from the left-hand list. Leave the editing controls at their default value, so that **Choose Feature** displays **All Fea-**

Table 10.5: Items in format strings.

Item	Description
[link]/[/link]	Link to the document (Greenstone version)
[href]	The href of the link to the document (Greenstone version), without the <a> tag
[srclink]/[/srclink]	Link to the original document if the original was converted to another form
[icon]	Appropriate icon for a classifier/document node, e.g., bookshelf, book, chapter, page
[srcicon]	Appropriate icon for the original source document, e.g., Word, PDF, PostScript
[num]	Position of the document in the search results (useful for debugging)
[numleafdocs]	Number of documents below the current classifier node
[Text]	Text of the current section
[RelatedDocuments]	Related document information (if available): a vertical list of Titles (or Subjects if Titles aren’t available) that link to the related documents—based on “relation” metadata, which is a space-separated list of collection, OID pairs
[highlight]/[/highlight]	Used to bold selected section in a hierarchical table of contents, and the selected node in a classifier (has no effect otherwise)
[Summary]	Displays summary metadata or creates a short summary on the fly
[DocOID]	Internal identifier of the current section of the document
[DocTopOID]	Top-level internal identifier of the current document
[DocRank]	Rank of the document—used in search results
[DocImage]	URL of the document’s cover image, if available
[collection]	Directory name of the collection this document is from, for use in cross-collection searching
[collection:metadata]	Collection metadata for the collection this document is from, for use in cross-collection searching, e.g., [collection:collectionname]
[metadata]	Value of this metadata element for the document

tures and VList is selected as the **Affected Component**. The text in the **HTML Format String** box is shown in Figure 10.4a. For a particular document whose Title metadata is *A discussion of question five from Tudor Quiz: Henry VIII* and whose Source metadata is *quizstuff.html*, this displays something that looks like Figure 10.4b. The same format appears in the search results list and the titles list, and also when you get down to individual documents in the hierarchical subject browser. This is

- (a)

```
<td valign=top>[link][icon][/link]</td>
<td valign=top>[ex.srclink]{Or}{[ex.thumbicon],[ex.srcicon]} [ex./srclink]</td>
<td valign=top>[highlight]
  {Or}{[dc.Title],[exp.Title],[ex.Title],Untitled}[/highlight]
{If}{[ex.Source],<br><i>([ex.Source])</i></td>
```
- (b)  A discussion of question five from Tudor Quiz: Henry VIII
(*quizstuff.html*)
- (c)  **Tudor period**
 Citizens
 Monarchs
 Others
 Relatives
- (d)

```
<td>[link][icon][/link]</td>
<td>[ex.Title]<br>
  <i>([ex.Source])</i>
</td>
```
- (e)  **Tudor period**
 ()
 Citizens
 ()
 Monarchs
 ()
 Others
 ()
 Relatives
 ()
- (f)

```
<td>[link][icon][/link]</td>
<td>[ex.Title]</td>
```
- (g)

```
<td>[link][icon][/link]</td>
<td>[ex.Title]<br>
  [dc.Subject]
</td>
```

Figure 10.4: Formatting the Tudor collection: (a) default format statement; (b) link to a document; (c) hierarchy browser; (d) simpler version of default format statement; (e) hierarchy browser with modified format statement; (f) format statement for hierarchy browser; (g) format statement that displays subject metadata;



(h)

```
<td>[link][icon]/>
<td>{If}{numleafdocs},<i>Bookshelf title:</i> [ex.Title],
<i>Title:</i> [ex.Title]}
</td>
```

(i)

Figure 10.4, cont'd: (h) screenshot showing new search results list; (i) advanced format statement for hierarchy classifier

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

Delete the contents of the **HTML Format String** box and replace it with the simpler version in [Figure 10.4d](#). **Preview** the result (you don't need to build the collection, because changes to format statements take effect immediately). Look at some search results and at the titles list. They are just the same as before! Under most circumstances this far simpler format statement is entirely equivalent to the more complex default.

The meaning of the format string is this. The first line creates a hyperlink to Greenstone's version of the document, using an appropriate icon as anchor text. This is placed in the first column of a table. In the second column is the value of the extracted Title metadata, followed by the filename (*ex. Source*) in parentheses. Ordinary HTML commands are used for formatting, such as italics. Any characters in plain text will simply be displayed as they are. Put some X's in after the title to see what happens. If you would prefer to use the title as a link anchor as well as the icon, replace *[ex. Title]* in the above format string by *[link][ex.Title]/[link]*. If you don't like the icon, just take it out.

However, there's a problem. Beside the bookshelves in the hierarchical subject browser, which used to look like [Figure 10.4c](#), a mysterious pair of parentheses has crept in beneath each subject, as in [Figure 10.4e](#). This is because what is printed for these bookshelves is governed by the same format statement. Nodes in the hierarchy—bookshelf nodes—have associated *ex.Title* metadata, which is the set to the value of the metadata that corresponds to that level of the hierarchy. However, they do not have Source metadata, so it comes out blank.

As noted above, the **Choose Feature** menu displays **All Features**. That implies that the same format applies to the search results, titles, and all nodes in the subject hierarchy—including internal nodes (that is, bookshelves). The menu can be used to restrict a format statement to a specific one of these lists. We will override this format statement for the hierarchical subject classifier. In the **Choose Feature** menu, scroll down to the item that says **CL2: Hierarchy-metadata dc.Subject and Keywords** and select it. This is the format statement that affects the second classifier (i.e., CL2), which is a hierarchy classifier based on Subject metadata. Click **Add Format** to add this format statement to the collection; the panel below changes from gray to white and the text becomes editable. Now you can make it different from the others. Edit the **HTML Format String** box below to the text shown in [Figure 10.4f](#).

Preview the subjects list in the collection. First, the offending parentheses have disappeared from the bookshelves. Second, when you reach a list of documents in the subject hierarchy, the filename does not appear beside the title, because *ex.Source* is not specified in the format statement and this statement applies to all nodes in the subject classifier. Note that the search results and titles lists have not changed: they still display the filename beneath the title.

We now change the search results format to display Subject metadata instead of the filename. In the **Choose Feature** menu (under Format Features on the Format panel), scroll down to the item **Search** and select it. Ensure that **VList** is selected as the **Affected Component** (which it is by default). Click **Add Format** to add this format statement to the collection. Change the format string box to the text in [Figure 10.4g](#). To insert the `[dc.Subject]`, position the cursor at the appropriate point and either type it in, or—better still—select it from the **Insert Variable** drop-down menu, which shows the things you can put in square brackets in the format statement.

Preview the collection. Documents in the search results list are displayed as shown in [Figure 10.4h](#). The vertical bar is there because this metadata is hierarchical. For most metadata, such as title and author, this isn't a problem, but if necessary the individual components of the hierarchy can be extracted using JavaScript. (Design pattern 2 in Section 12.2 describes the approach in general.)

Finally, let's return to the subject hierarchy and learn how to do different things to bookshelves and the documents themselves. In the **Choose Feature** menu, re-select the item **CL2: Hierarchy-metadata dc.Subject and Keywords**. Edit the format string below to the text in [Figure 10.4i](#). Again, you can insert the items in square brackets by selecting them from the drop-down box; indeed, it's best to do so to avoid typing errors. The *If* statement tests the value of the variable *numleafdocs*. This variable is only set for internal nodes of the hierarchy—that is, bookshelves—and gives the number of documents below that node. If it is set, the first branch is taken, if *numleafdocs* is not set the second branch is taken. Commas are used to separate the branches. The curly brackets serve to indicate that the *If* is special—otherwise the very word *If* itself would be output.

Preview the collection and examine the subject hierarchy to see the effect of your changes. Bookshelves should say *Bookshelf title* and then the title, while documents will display *Title* and the title. Note that the number of documents in the bookshelf is not displayed: *numleafdocs* is only being used to test the current item in the list.

Pointing to documents on the Web

As we discuss earlier under “Enhanced HTML document handling,” when an HTML document is clicked, it is often best to retrieve it from the original Web site for presentation to the user, rather than having Greenstone display its local copy. In the *tudor* collection, go to the **Gather** panel and inspect the files you dragged into it. The first folder is *englishhistory.net*, which opens up to reveal *tudor*, and so on. The files represent a sweep of the pages (and supporting images) that constitute a complete section of the *englishhistory.net* Web site. Greenstone downloads them in a way that preserves the structure of the original site, and the HTML plug-in reconstructs each page’s original URL from the folder hierarchy and stores it as metadata.

When pointing to the original documents, it is vital that the URLs begin with the Web domain name (*englishhistory.net* in this case). If you dragged over a sub-folder, for example *tudor*, the URL metadata will be set to something like

```
http://tudor/monarchs/...
```

rather than

```
http://englishhistory.net/tudor/monarchs/...
```

In the **Document Plugins** section of the **Design** panel, double-click **HTMLPlugin** to configure it. Locate the *file_is_url* option (about halfway down the first block of items) and switch it on. This option makes the HTML plug-in set additional metadata for each document to facilitate retrieval from the Web. To use it, go to the **Format** panel, select the **Format Features** section, check that **Choose Feature** is set to *All Features*, and then edit the format statement by replacing

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

with

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

Build and preview the collection. Note that the document icons have changed. The collection behaves exactly as before, except that when you click a document icon, your browser retrieves the original document from the Web.

If you are wondering how this works, the new metadata is called *[weblink]*, and it gives the URL in a manner suitable for linking to. For the file *boleyn.html*, it is set to

```
&lt; a href=&quot;  
http://englishhistory.net/tudor/monarchs/boleyn.html&quot;&gt;
```

which is simply a heavily disguised way of saying

```
<a href="http://englishhistory.net/tudor/monarchs/boleyn.html">
```

Behind the scenes, the *[link]* element used in format statements is dealt with in the same way, although its value is not exposed in the Enrich panel as with *[weblink]*. In addition, *[webicon]* metadata is set to an icon for the Web.

Formatting exercise 2: Word and PDF collection

To reinforce the lessons learned when formatting the Tudor collection, let us study and modify the format statements for the Word and PDF collection you built earlier.

Open up the **reports** collection (assuming that's what you called it), go to the **Format** panel, and select **Format Features** from the left-hand list. Leave the editing controls at their default value, so that **Choose Feature** displays **All Features** and **VList** is selected as the **Affected Component**. The default **HTML Format String** is as shown in Figure 10.4a. As noted earlier, this format statement is used for any vertical list, such as search results, classifiers, and document table of contents.

Examine it more closely. *{Or}{[ex.thumbicon],[ex.srcicon]}* displays extracted *thumbicon* metadata if it's present, and *srcicon* otherwise. If both are absent, nothing is displayed. This collection has no thumbnails, since it contains no images, so the choice is unnecessary and the whole clause above can be replaced by *[ex.srcicon]*. You will learn about "exploded" metadata below, but here there is none, so *exp.Title* can be removed from the fourth line. The resulting format statement is shown in Figure 10.5a. Edit the **HTML Format String** accordingly and preview the collection to make sure the display is unchanged.

When documents undergo a conversion process during importing (e.g., Word and PDF documents), the original file is stored in the collection along with the converted version. The format statement in Figure 10.5a links to both versions. *[ex.srclink][ex.srcicon][ex./srclink]* links to the original file; *[ex.link][icon][ex.link]* links to the converted one. Storing and displaying the original allows users to see the correct format, but it requires the user to have the relevant program installed. (It also increases the size of the collection.) The Greenstone version can be viewed in a browser, but it may not look as nice.

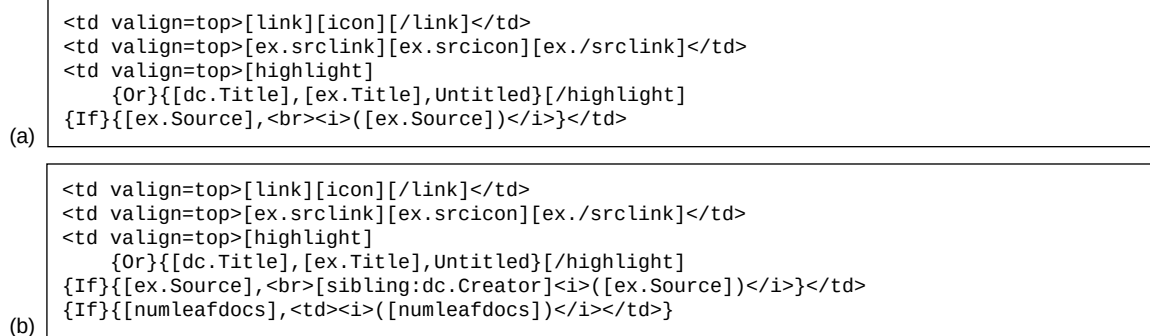


Figure 10.5: (a) Simplified format statement; (b) modified CL2 format statement to display all authors and the number of items by each one

Choose **Search** in the **Format Features** menu, ensure that **VList** is selected as the **Affected Component**, and click **Add Format**. Experiment with removing either of the two links from the format statement. To see the results of your changes, preview the collection and do a search. The changes will only apply to search results.

Next we customize the format for the *creators* browser that was built in [Section 10.3](#). Bookshelves have only two pieces of metadata to display: *ex.Title* and *numleafdocs*. Regardless of the metadata that has been used to build the classifier, the bookshelf label is always stored as *ex.Title*. That is why authors are printed for each bookshelf even though *[dc.Creator]* is not specified in the format statement. As noted earlier, the variable *numleafdocs* can be used in an *If* statement to make bookshelves and documents display differently.

To make each creator's bookshelf show how many entries it contains, select the **CL2** classifier (which is based on Dublin Core Creator metadata) from the **Choose Feature** drop-down list in the **Format** panel's **Format Features** section; again ensure that **VList** is selected as the **Affected Component**. Click **Add Format** to make it editable and append the text shown in the sixth (last) line of [Figure 10.5b](#) to the end of the statement (we will add the fifth line shortly).

When you preview the collection, the creator's bookshelves will display how many documents they contain. This additional format line has the effect of specifying in brackets how many items appear beneath a bookshelf. Only bookshelves display this information because only they define *numleafdocs*. By modifying the format statement for the CL2 classifier, you have applied the change only to the second classifier.

Next we modify the document entries in this classifier to display all authors. Ensure in **Format Features** that the **CL2** classifier is still selected. Finish editing the format statement so it looks like that of [Figure 10.5b](#)—this involves modifying the fifth line to include the text related to *sibling*. This modification is protected by an *if* statement that tests extracted source (filename) metadata, which is undefined for bookshelves and can also be used to differentiate them from documents.

The result displays the Greenstone link, the link to the original document, and then the title. For bookshelves, it also shows how many documents the bookshelf contains. For documents, it shows all authors (creators), and the source filename. *[sibling:dc.Creator]* displays all Creator metadata for the document, separated by a space, whereas *[dc.Creator]* displays only the first author. Check this by previewing the creators list.

You can change the separator between authors by replacing *[sibling:dc.Creator]* with *[sibling(All'
'):dc.Creator]*. This adds a new line (
) after each author. Moreover, if you completed the enhanced Word document handling exercise in [Section 10.3](#), the collection will have both Dublin Core and extracted Creator metadata. To display both, use

```
[sibling:dc.Creator] [sibling:ex.Creator]
```

To display just the Dublin Core creator if present, or otherwise the extracted creator, use

```
{Or}{[sibling:dc.Creator],[sibling:ex.Creator]}
```

As you can see, getting a suitable display for multiple-valued metadata requires some thought. Manually assigned metadata can be used to replace or enhance what can be extracted automatically, and the right thing to do depends on the circumstances and the meaning of the metadata.

```

package Style
_cssheader_{
  <link rel="stylesheet" href="_httpcstyle_/style-blue.css" type="text/css"
    title="My Style" charset="UTF-8">
}

```

Figure 10.6: Macro that specifies a non-standard style sheet, *style-blue.css*

Formatting exercise 3: Branding your collection

After installing Greenstone and having built a few collections, most people want to make them look like *their* collections, not Greenstone's, by changing their appearance—or the appearance of the entire installation—to match their institutional style or the style of their other Web pages. The layout and presentation of every page that Greenstone generates is governed by *macros*, and, as noted earlier, ones specific to a particular collection can be defined in the *Collection Specific Macros* view. Macros are very powerful; they are discussed further in Chapter 12. However, a simpler (but more modest in what can be achieved) way to change the appearance of your pages is to use cascading style sheets (see Section 4.4), which we look at next.

In the Librarian interface, open any collection (the *demo* collection is a good choice). Click **Collection Specific Macros** on the left-hand side of the **Format** panel and copy the contents of Figure 10.6 into the text box. If you preview the collection now, you will find that all the style information has been lost and the pages look very plain. This is because the macro in Figure 10.6 specifies a style sheet called *style-blue.css* that does not yet exist.

The next step is to move a file into the collection, outside the Librarian interface. In the Windows file browser, go to the place where the collections are kept, that is, the folder in your Greenstone installation called *collect*. This has a sub-folder for each collection. You can determine the internal name of your collection by looking at the title bar of the Librarian interface: it appears in brackets at the end. Look in the sub-folder given by the internal collection name. We will learn about the make-up of collections in the next chapter, but for now you need to find its *style* sub-folder. Copy the file *style-blue.css* (do not move it; *copy* it) from the *custom* folder of *sample_files* into the collection's *style* sub-folder. (This folder is what the mysterious *_httpcstyle_* macro in Figure 10.6 refers to.) Preview the collection again; it will look radically different, with a blue theme. (If not, you need to carefully check what you have done.)

To play around with the style of this collection, use a text editor, e.g., WordPad or Emacs, to open the *style-blue.css* file that you have just placed in the collection's *style* sub-folder. The extra lines in the style sheet that give the blue coloring are given at the beginning of the file. Remove these lines one by one, watching your page lose its bluish hue by viewing the result after each time. To see the changes, just reload the page in the Web browser—but note that stylesheet changes may require a forced reload. Maybe we should call such a configuration “bluestone”!

This operation made a local change to just one collection. You can change all the pages in one go by modifying the files located within the top-level *web* folder of your Greenstone installation. To achieve site-wide changes along the lines of the collection-specific ones described above, you need to locate the file *style.css*. In a file browser, navigate to your Greenstone installation and select the *web*

folder, followed by the *style* folder. This is where *style.css* is located. Open it in a text editor, having first saved a backup copy. The changes you make now will apply to all collections—except the one you have just been working on, which uses *style-blue.css* instead. After each change, preview any other collection to ensure that it has worked properly. (Reload the page in your Web browser to see the effect of the change.)

Here are some suggested modifications to *style.css*. Change some of the colors by adding the line

```
color: teal;
```

to the *body* style instructions at the beginning of the file, to make it read:

```
body {
    background: #ffffff;
    color: teal;
}
```

Find *a.collectiontitle* and change its color from black to blue (that will change it on your Greenstone home page); find *p.collectiontitle* and do the same thing. When you preview any collection, text in the page body is light green (teal), and the collection title has changed from black to blue—both on the Greenstone home page (*a.collectiontitle*) and in the collection itself (*p.collectiontitle*). Of course, you won't see the change in collections that use an image for the title.

Next alter the positions of the *Home*, *Help*, and *Preferences* buttons and the collection name or image:

- For *div.pageinfo*, set both **float** and **text-align** to **left**.
- For *div.collectimage*, set both **float** and **text-align** to **right**.
- For *p.bannertitle*, set **background-position** to **top left**.
- For *p.collectiontitle*, set **background-position** to **top right**.

Now when you look at any collection the three buttons are in the left upper corner and the collection title is on the right.

You probably want to change the images used on the Greenstone home page, and in the collections themselves, to reflect your own house style. The simplest way to do this is to determine the names of the image files and replace them with files of your choice. The images are kept in the *images* subfolder located within the top-level *web* folder of the Greenstone installation.

For example, the image at the top right of the home page that proclaims “Greenstone digital library software” is called *gsdlhead.gif*. To determine this, visit the home page in a browser, right-click the image, and select **Save picture as** (or equivalent, depending on the browser you are using). A dialog pops up displaying the image name: *gsdlhead.gif*. Click **Cancel** to close the dialog—you don't need to save the image, you just want to know its name. Do the same for the background image by right-clicking on the green swirly bar down the left. This time choose **Save background as** to find the name (*chalk.gif*), then click **Cancel**. (These instructions apply to Internet Explorer; other browsers are slightly different. For example, Firefox has a **View Background Image** option that you will need to invoke before saving.)

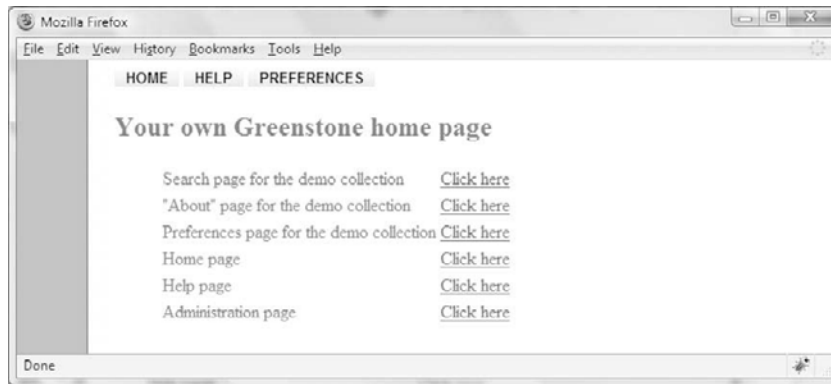


Figure 10.7: New Greenstone home page

Two new images can be found in the *custom* folder of *sample_files*: *newheading.gif* and *newbgimg.gif*. Copy these into the folder called *images* in your Greenstone installation, calling them *gsdlhead.gif* and *chalk.gif*, respectively. But don't forget to make backup copies of the original versions of these two images first! When you preview the result, look at your Greenstone home page and at any collection.

The CSS files and images allow you to make superficial changes to the appearance of your installation. You can make far more radical changes to the structure of all pages by altering the macro files that are used to generate them. We will learn about macros in Chapter 12, but here we illustrate by using an alternative to the home page that has been prepared in advance and included in your Greenstone installation.

Macros are kept in a top-level folder in your Greenstone installation called *macros*, and have file-names ending with *.dm*. For example, the Greenstone home page is generated by the file *home.dm*. Make a backup of this file, and then copy the file *yourhome.dm*—which is provided just for this exercise—into *home.dm* (alternatively, just rename the file). If you are using the Windows Local Library server, you will need to restart it to see the changes that result from altering the macro files: click the “world globe” icon on the task bar to make the Local Library server window appear and press **Restart Library**. Otherwise, just reload the page in the Web browser—as with changes to style sheets and images, changes to macro files may require a forced reload in the Web browser. Now you will find that your Greenstone home page is almost unrecognizable—it looks like the snapshot in Figure 10.7. If you look at the contents of *yourhome.dm* you can see how to link into different Greenstone facilities.

Finally, you should reverse this last change by renaming your backup copy of *home.dm*. You might also like to reverse the other changes you have made.

10.5 Dealing with Metadata

Along with full-text search, metadata is the key to giving readers access to the documents in a digital library. You have used both extracted and manually assigned metadata in the preceding exercises.

However, there is more to learn. We will review the facilities provided by the Enrich panel, including the ability to define new metadata sets and convert metadata from one set to another when importing documents with attached metadata. Then we examine how metadata is stored internally, in XML files.

It is often necessary to import metadata from existing databases. We use MARC records as an example, but Greenstone can ingest many other formats too, such as metadata records downloaded over the Open Archives Initiative Protocol for Metadata Harvesting described in Section 7.2; CDS/ISIS, which is widely used in developing countries; BibTeX, which pervades mathematics and computer science; and the commercial product Procite. It can also ingest metadata expressed in a spreadsheet, where the first row contains the names of the metadata fields and subsequent rows contain metadata records. (In this case, the file must be saved in the CSV, or “comma-separated value,” file format and processed by Greenstone’s CSV plug-in.)

In each case, the operations and facilities are exactly the same. If you can build collections from MARC records, you can do so from any of these other formats. Full-text documents are often associated with the metadata records too, and the full capability of a digital library is only realized when these are brought into the collection and associated with the correct metadata. We show how this is done with Greenstone.

The Enrich panel

We have used the **Enrich** panel many times in the preceding exercises above. On the right is a table that shows metadata for selected files or folders in the collection. Columns can be re-sized by dragging the separating line in the gray heading at the top. If several files or folders are selected, black text indicates values common to them all, while gray indicates that the values are different. Editing a gray value only affects documents with that value. Entering new metadata adds it to all selected items.

A folder icon beside a metadata entry indicates that the value is inherited from a parent (or ancestor) folder. Inherited metadata cannot be edited or removed, only appended to. Click the folder icon to go immediately to the place where the metadata was assigned.

Clicking on an element displays its existing values below the table. Clicking an existing value automatically places it in the value field. Conversely, typing in the field selects the existing value that begins with the characters typed, and pressing the tab key auto-completes the typing. Annoyingly, typing errors that have been corrected in the metadata itself persist in the existing values list but can be removed by re-opening the collection, which reconstructs the list from scratch.

Metadata values can be organized into a hierarchy, shown in the existing values using folder icons. When entering hierarchical values, the character “[|” is used to separate levels—for example, *Cards|Red|Diamonds|Seven* might be used in a hierarchy that represents a pack of playing cards. Hierarchies enable values to be grouped together, and intermediate levels—groups—can also be assigned as metadata to files.

Greenstone extracts metadata automatically from documents into a set whose elements are prefixed by *ex*. These values cannot be edited, and existing values are not shown. The extracted metadata set

is often used as the default, so if a metadata element name appears without an identifier, *ex* is assumed.

Metadata sets

As explained in Section 6.2, in qualified Dublin Core, fields can be refined or *qualified*. For example, the date can be qualified as date created, date valid, date available, date issued, or date modified. Greenstone allows metadata elements to have qualifiers, called sub-elements. Sub-elements are separated from the element name by a caret (^). To see this, go to any collection in the Librarian interface and click **Manage Metadata Sets** in the **Enrich** view, then **Add** in the pop-up that appears. This produces a further pop-up: select **Qualified Dublin Core** and then press **OK**. You may be warned that this new metadata set has the same namespace (dc) as an existing one, but proceed anyway. Finally, press **Close**. Now, back in the Librarian interface, select a document. Under date, for example, you will see metadata elements

- dc.Date^Created
- dc.Date^Valid
- dc.Date^Available
- dc.Date^Issued
- dc.Date^Modified

(among others). You can use these extended fields just like any other.

To create a new metadata set, click **Manage Metadata Sets** in the **Enrich** view followed by **Add** (as before), but this time click **New** to launch the Greenstone Editor for Metadata Sets (you can also run this from the Windows Start menu, or by running *gems.bat* or *gems.sh* in the *gli* folder of your Greenstone installation). You are prompted for the set name, name-space, and description. If you base the set on an existing one, it will inherit all its elements.

After this initial dialog, the main window shows the metadata set on the left-hand side and some attributes of the set itself on the right. If you have based the set on an existing one, one or more elements will be shown; clicking one displays its attributes on the right. Note the distinction between the attributes of the set as a whole and the attributes of elements in the set. To add a new element, right-click on the set and choose **Add Element**. To add a new sub-element (that is, a qualifier), right-click on the element and choose **Add Subelement**. Elements and sub-elements can be deleted using the right-click menu.

With the Metadata Set Editor you can edit existing metadata sets as well as create new ones. Back in the Librarian, having pressed **Manage Metadata Sets** on the **Enrich** panel, select a set from the list of assigned metadata sets and then click the **Edit** button to edit a set. When you have finished, save it and close the Metadata Set Editor. If a collection no longer needs a metadata set, select it and press **Remove**. Any metadata that has been assigned to its elements is not lost: you will be asked how to deal with it when you next open the collection.

Reviewing assigned metadata

Sometimes it is convenient to see the metadata assigned to many files together—for instance, to determine how many files are left to work on, or to get some idea of the spread of dates. Select some

files in the Enrich panel, then right-click and choose **Assigned Metadata**. The first column in the table that appears shows file names; the rows show all values assigned to those files.

If the table is too large you can reduce it by applying filters to the columns. As new filters are added, only those rows that match remain visible. To set, modify, or clear a filter, click the “funnel” icon at the top of a column. You are prompted for information about the filter. When set, the column header changes color.

The filter pop-up window has Simple and Advanced tabs. Simple filters show rows that contain a certain metadata value; you can select values from the drop-down list (* means any value). Advanced filters allow matching operations that use string comparisons, and case-sensitive or case-insensitive matching. You can specify a second matching condition: use this to specify a range of values (by selecting AND) or alternative values (OR). Finally, you can change the sort order (ascending or descending). When finished, click **Set Filter** to apply the new filter to the column, or **Remove Filter** to remove it. Note that the filter details are retained even when the filter is removed, and can be reinstated at a later point.

For example, to sort the metadata table, choose a column, select the default filter setting (a Simple filter on “*”), and choose ascending or descending.

Importing documents with assigned metadata

When creating a collection, you can drag in documents from existing collections (in the **Gather** panel). In this case, any metadata that has been assigned is imported automatically.

If different metadata sets are involved, the metadata must be mapped. You will be prompted for the necessary information and shown the name of the element that is being imported, just as it appears in the source file—this cannot be edited or changed. You choose which set to map this element to. You can either have it added to the set as a new element or map it to an existing element. The system automatically identifies the closest match, in terms of set and element, and presents it as the default. Click **Merge** to map this element to the selected set and element, or **Add** to augment the chosen set with this new element, or **Ignore** to ignore this element entirely. Once you have specified how to import a given element, the mapping information is retained for the lifetime of the collection.

How metadata is stored

Behind the scenes, the Librarian interface stores any metadata manually assigned to documents in automatically created XML files called *metadata.xml*. In fact, if you create your own metadata files in this format and simply place them in the folder along with your documents, the Librarian interface will automatically pick up the metadata when you drag the documents in.

Figure 10.8a shows a metadata file, while Figure 10.8b shows the XML document type definition (DTD) for these files (see Section 4.3, Basic XML). The example file contains two metadata structures. In each one, the *filename* element describes files to which the metadata applies, in the form of a regular expression. Thus `<FileName>nugget.*</FileName>` indicates that the first metadata record applies to every file whose name starts with “nugget”. For these files, *Title* is set to “Nugget Point

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE DirectoryMetadata SYSTEM
  "http://greenstone.org/dtd/DirectoryMetadata/1.0/DirectoryMetadata.dtd">
<DirectoryMetadata>
  <FileSet>
    <FileName>nugget.*</FileName>
    <Description>
      <Metadata name="Title">Nugget Point Lighthouse</Metadata>
      <Metadata name="Place" mode="accumulate">Nugget Point</Metadata>
    </Description>
  </FileSet>
  <FileSet>
    <FileName>nugget-point-1.jpg</FileName>
    <Description>
      <Metadata name="Title">Nugget Point Lighthouse, The Catlins </Metadata>
      <Metadata name="Subject">Lighthouse</Metadata>
    </Description>
  </FileSet>
</DirectoryMetadata>

```

(a)

```

<!DOCTYPE DirectoryMetadata [
  <!ELEMENT DirectoryMetadata (FileSet*)>
  <!ELEMENT FileSet (FileName+,Description)>
  <!ELEMENT FileName (#PCDATA)>
  <!ELEMENT Description (Metadata*)>
  <!ELEMENT Metadata (#PCDATA)>
  <ATTLIST Metadata name CDATA #REQUIRED>
  <ATTLIST Metadata mode (accumulate|override) "override">
]

```

(b)

Figure 10.8: Greenstone’s file format for storing metadata: (a) example metadata file; (b) document type definition (DTD)

Lighthouse.” Note that in Greenstone, regular expressions are interpreted in the Perl language, which is subtly different from some other conventions. For example, “*” matches zero or more occurrences of the previous character, while “.” matches any character—so *nugget.** matches any string with prefix “nugget,” whether or not it contains a period after the prefix. To insist on a period you would need to escape it, and write *nugget\.** instead.

Metadata elements are processed in the order in which they appear. The second structure in [Figure 10.8a](#) sets *Title* metadata for the file named *nugget-point-1.jpg* to “Nugget Point Lighthouse, The Catlins,” overriding the previous specification. It also adds a *Subject* metadata field.

Sometimes metadata is multi-valued and new values should accumulate, rather than override previous ones. This is what the *mode=accumulate* attribute does. It is applied to *Place* metadata in the first specification in [Figure 10.8a](#), which will therefore be multi-valued if more than one value is specified. To revert to a single metadata element, write *<Metadata name="Place" mode="override">New Zealand</Metadata>*. In fact, you could omit this mode specification because elements override unless otherwise specified. To accumulate metadata for some field, *mode=accumulate* must be specified in every occurrence. Note that all this applies only to metadata specified in *metadata.xml* files, not to metadata specified in the Librarian interface.

Collections of bibliographic information

So far, we have used Greenstone to build collections of documents and to add metadata to them. However, large collections of metadata—that is, bibliographies—are often already available in some standard form. The metadata may be valuable in its own right, just as a library catalog is, and it may contain substantial amounts of text, such as descriptions or abstracts of documents. Full-text documents may be associated with some or all of the metadata records.

Building the collection

We will build a collection of bibliographic records from a file in the MARC format (see Section 6.2). In the Librarian interface, start a new collection called **Papers Bibliography** that will contain some MARC records for working papers published at the Computer Science Department, University of Waikato.

In the **Gather** panel, navigate to the *sample_files*→*marc* folder and drag the file *CMSwp-all.marc* into the collection. A window pops up to ask whether you want to add the MARC plug-in, because it is not one of the standard plug-ins in the default collection. Click **Add Plugin**. Before building, tidy things up in the **Design** panel by selecting **Browsing Classifiers** and removing the classifier for Source metadata. In this collection, all records are from the same file, so this metadata, which is set to the file name, is neither interesting nor useful. Similarly, remove the Source index in the **Search Indexes** section.

Having built the collection, browse through the list of titles and view a record or two. In this one input file there are 234 MARC records. The MARC plug-in has split the file into records and for each one has come up with a separate Greenstone document that lists the field numbers, sub-field identifiers, and contents. It is this plug-in that is responsible for the format that you see when you view a record. The same text is used for searching. Try finding items that include the word *graphics*.

Using the metadata

Back in the Librarian interface, check the **Enrich** panel. There is quite a lot of metadata in this collection, but you can't see any, because the metadata is associated with individual bibliographic records, and there's no way to get at these in the Enrich panel. We will solve this problem shortly.

Go to the **Browsing Classifiers** section of the **Design** panel, select **List** from the **Select classifier to add** drop-down menu and click **Add Classifier**. Look at all the Dublin Core metadata elements that are in the drop-down list. The MARC plug-in has extracted these values from the MARC records themselves. Select **dc.Subject and Keywords** as the metadata item for this classifier, set **bookshelf_type** to **always**, set **partition_type_within_level** to **per letter**, and click **OK**.

Build the collection and preview the result. Records are grouped together by subject. The **bookshelf_type** setting creates a bookshelf even when there is only one record for that subject; the **partition_type_within_level** setting produces the selectable alphabetical list at the top.

Searching

Next we add more fields to search on. Go to the **Search Indexes** section of the **Design** panel. Add a new index based on Subject metadata by clicking **New Index**, selecting **dc.Subject and Keywords**

in the list of metadata, and clicking **Add Index**. Rebuild the collection and preview the result. Recall that you can change the text that is displayed in the drop-down search menu using the **Search** section of the **Format** panel.

So far, all our searching has used a single search box, as most search engines do. Librarians typically use a more sophisticated style of search that involves specifying the content of individual metadata fields. Fielded searching is particularly appropriate for metadata-rich collections. In the Web browser, access Greenstone's **Preferences** page. You will notice a query style option that enables you to switch between normal and fielded search. Change to fielded search, press **set preferences**, and then click the **Search** button. The search page has changed to a form. Notice the extra field corresponding to the search index you have just added. You can do quite complicated queries by searching for words in different fields at the same time.

You can specify in the Librarian interface whether plain search, form search, or both are available for a particular collection, and which is the default. This is done by selecting **Format Features** from the left-hand list of the **Format** panel and choosing **SearchTypes** from the list of assigned formats. If you specify only form search, the collection's home page will no longer include a query box, because the search form is too big to fit nicely. To search, you have to click **Search** in the navigation bar. The Preferences page changes automatically to reflect the search types that the collection makes available.

Controlling the conversion

Greenstone's bridge from MARC to Dublin Core is controlled by a metadata mapping file (called *marc2dc.txt*), part of which appears in [Figure 10.9](#). (The actual file is much larger because it is richly commented.) Each line gives a USMARC/MARC21 field number, or a range of field numbers, and the Dublin Core equivalent. Letters beginning with a dollar sign (*\$a*, *\$b*, etc.) are subfields, which were mentioned briefly in the description of MARC in Section 6.2. The mapping is based on one defined by the Library of Congress.

This text file can simply be edited to use a different correspondence. Alternative mapping files can be specified as an option to the MARC plug-in. In fact, a second file is provided with Greenstone that maps MARC records to *qualified* Dublin Core. You will find both *marc2dc.txt* ([Figure 10.9](#)) and *marc2qdc.txt* in your Greenstone installation in the top-level folder called *etc*. Start a new collection and drag in the file *CMSwp-all.marc* of MARC records, just as above. Configure the MARC plug-in by setting its *metadata_mapping_file* option to *marc2qdc.txt*. It is necessary to build the collection in order to extract the metadata. Now, in the **Enrich** panel, click the **Manage Metadata Sets** button at the lower left. Remove the Dublin Core metadata set, then click **Add** and select **Qualified Dublin Core**. Now you will see the qualified metadata in the drop-down metadata lists on the configuration panels of search indexes or classifiers. If you were to "explode" it, as described in the next section, you would see it in the Enrich panel.

Working with individual metadata records

As you have noticed, the Enrich panel does not show the metadata for this collection because it is associated with records inside the file and not with the file itself. When metadata files are imported

```

245, 246 -> edc.Titl

720 -> dc.Creator
100, 110, 111 -> dc.Creator
700, 710, 711 -> dc.Creator

600, 610, 611, 630, 650, 653 -> dc.Subject

500-505, 507-529 -> dc.Description
531-539, 541-545, 547-599 -> dc.Description

260$a$b -> dc.Publisher

260$c$g -> dc.Date

655 -> dc.Type

340 -> dc.Format
856$q -> dc.Format

020$a, 022$a, 024$a -> dc.Identifier
856$q -> dc.Identifier

786$o$t -> dc.Source
534$t -> dc.Source

041$a$b$d$e$f$g$h$j -> dc.Language
546 -> dc.Language

787$o$t -> dc.Relation
530, 760-785 -> dc.Relation

651, 662 -> dc.Coverage
751, 752 -> dc.Coverage

506, 540 -> dc.Rights

```

Figure 10.9: Part of Greenstone’s MARC to Dublin Core crosswalk file

into Greenstone, their metadata cannot be viewed (let alone edited) in the Librarian interface, although it can be used for browsing, searching, and formatting. To see or edit any metadata you need to go back to the program that created the file.

However, Greenstone does provide a way of “exploding” a metadata database so that each record appears as an individual document whose metadata is both viewable and editable. This process is irreversible: when it is done, the original metadata file is deleted.

Exploding the MARC database

In the **Gather** and **Enrich** panels, the MARC database has a different colored icon. This indicates that it is a metadata database and can be exploded. Right-click it and choose **Explode Metadata Database** from the menu. A window pops up with options for the exploding process, and descriptions of the options appear when the mouse hovers over them. You can specify the plug-in to be used for exploding—in most cases, only one plug-in will process a particular type of file, but in some

cases different file types share the same filename extension. You can specify the encoding of the database. You can specify which metadata set the new fields should be added to. If none is specified, you will be prompted for what to do with each new field in the database: add it as a new element to an existing metadata set, merge with another element, or ignore—just as when you import documents with previously assigned metadata into a collection with a different metadata set. Other options are described in the next section.

Enable the **metadata_set** option and set the value to **Qualified Dublin Core**. Click **Explode** to start the process. This may take a short while, depending on the size of the database. When **Explode** finishes, the MARC file will have been replaced by three folders that together contain an empty file (.nul) for each record in the original database. The metadata can be viewed in the **Enrich** panel. The MARC metadata has been placed into the qualified Dublin Core metadata set, and, unlike extracted metadata, it *can* be edited, because the exploding process cannot be repeated.

Rebuild and preview the collection. Although documents are present in the lists of titles and subjects, clicking them yields the message *This document has no text*. Searching text no longer returns any results (although searching metadata works). There is no text in the documents because previously the MARC plug-in stored the raw record as the “text” of each record, whereas now the metadata is in the Librarian interface and the associated document files (the ones ending .nul) are empty. We need to modify the collection design to account for this change.

Tidying up the search facility

In the **Search Indexes** section of the **Design** panel, remove the text index because it is no longer of any use. Enable combined searching across all indexes at once by clicking **New Index** and ticking **Add combined searching over all assigned indexes**. Move this to the top of the list using the **Move Up** button, so that it appears first in the drop-down list, and click **Set Default Index** so that it becomes the default field for searching. Rebuild and preview the collection. One small problem is the names in the pull-down list on the search page: this is easily dealt with on the **Format** panel under **Search** by typing more appropriate names for the indexes in **Display text** (perhaps *everything*, *titles*, *subjects*).

Formatting the metadata records

The major remaining problem with this collection—and it is a big one—is that the documents have no text. To locate the cause, go to **Format Features** and select the **DocumentText** format statement from the drop-down list. The [text] that appears here displays the text of the document—in this case, a null file. Instead, delete this and replace it by the contents of [Figure 10.10](#). On previewing the result you will see a much more satisfactory document display showing title, subject, and publisher of the metadata record. It could, of course, be improved in many ways. Here are some suggestions if you want to go further.

- Remove the filename from the record title by clearing the **DocumentHeading** format statement: select it in the list of assigned format statements and delete the HTML format string.
- Remove the **Detach** and **No Highlighting** buttons by clearing the **DocumentButtons** format statement.

```
<table>
  <tr><td>Title:</td><td>[dc.Title]</td></tr>
  <tr><td>Subject:</td><td>[dc.Subject]</td></tr>
  <tr><td>Publisher:</td><td>[dc.Publisher]</td></tr>
</table>
```

Figure 10.10: DocumentText format statement that displays metadata

- Remove the filenames from the search results and browsing classifiers by selecting **All Features** and deleting these characters:

```
{ If } { [ex.Source] , <br> <i> ( [ex.Source] ) </i> }
```

Combining metadata and source documents

When Greenstone explodes a file, it creates a new empty document for each record and assigns the record's metadata to the document. The dummy documents are designated with numbers: 000001.nul, 000002.nul, ... These are divided into folders simply to make it easier and more convenient to access their metadata in the Enrich panel. For large databases this accelerates subsequent metadata editing. There is an option on the exploding pop-up to change the number of records per folder. (Incidentally, you can add dummy documents manually using the right-click menu in the Gather or Enrich panels. This creates a new empty file to which metadata can be assigned.)

Having anonymous names for documents makes the collection difficult to work with. If one of the metadata fields contains a more suitable identifier, it can be used by setting the *document_field* option in the exploding pop-up to the field name. Then the value of this field (if present) will be used for the filename.

It is also possible that full-text documents are associated with some or all of the metadata records. If so, the exploding process will bring them into the collection, if it can find them, instead of using null files. This makes the document a first-class member of the collection, with which all the metadata in the record is associated. The exploding pop-up provides *document_prefix* and *document_suffix* options, which can be used to manufacture a file path and file extension from the *document_field* value. They can also be used to make a valid URL, from which Greenstone will automatically download the document.

It is often necessary to add new full-text documents corresponding to existing metadata records in a collection. For example, documents might be digitized and added one by one. This can be done by replacing the existing null file using the right-click menu in the Gather or Enrich panel. Selecting *Replace* brings up a file browser from which you can specify any file to replace the null document. (Files can also be renamed from the same right-click menu. You could drag in the new document, find the null file containing its metadata, and rename the document to this file. However, the document file would end up with the wrong extension.)

10.6 Non-Textual Documents

So far, we have dealt with files containing text—HTML, Word, RTF, PDF, PostScript—and meta-data. However, Greenstone can build collections from many formats. It has plug-ins for other textual formats: Plain text, LaTeX, various kinds of e-mail, and source code. A ZIP plug-in copes with compressed archives. There are plug-ins for other Microsoft Office documents: Excel and PowerPoint. There is one for OpenOffice documents. Entire wikis (in MediaWiki format) can be imported lock, stock, and barrel and built into a Greenstone collection that resembles the original wiki but is equipped with more comprehensive searching facilities.

In this section we describe the use of some multimedia formats. A comprehensive image plug-in processes images in most popular formats, including PNG, GIF, JIF, JPEG, JPEG 2000, and TIFF (see Section 5.3). Audio files can be accommodated by MP3 and Ogg Vorbis plug-ins (see Section 5.2), and there is a generic plug-in that can be configured for audio and video formats like MPEG, MIDI, etc. (see Sections 5.4–5.6).

It is really impossible to describe all these facilities individually. The best way is to learn enough to find your way around the Librarian interface and to explore its facilities interactively. In this section we build three example collections: a very simple one for images, and two advanced collections—one for paged images of the kind that are often produced in scanning projects, the other for a multimedia collection that accommodates many media types. The first example is mostly revision, but the others are quite challenging. Working through them—even if you do not finish—will teach you much about Greenstone that can be applied to many kinds of collection, not just multimedia ones.

Images

It is very easy to create collections of images. Let's just do it. In the Librarian interface, start a new collection (**File→New**) called **backdrop**. Fill out the fields and before closing the pop-up to create the collection, examine the **Base this collection on** menu. This menu shows all existing collections, including ones you have just built. You can base a new collection on any existing one. That means that all its facilities—metadata sets, searching indexes, browsers, and any format statements and other configuration options—are copied. However, none of the documents are copied: what you create is a brand new collection whose structure is a clone of an existing one. Select the **Simple Image Collection (image-e)** from the pull-down menu. (You should obtain this collection from the book Web site before proceeding with this exercise, and place it in your Greenstone *collect* folder).

Drag the nine image files provided in **sample_files→images** into your newly formed collection, switch to the Create panel, and build and preview it. The navigation bar has a single button—Browse—that yields a list of photos, ordered by file name and presented as a thumbnail accompanied by some basic data about the image. (If this is not what you see, you have not based the new collection on the Simple image collection.) The structure of this collection is the same as the Simple image collection, but the content is different.

Changing the thumbnail size

The Image plug-in creates the thumbnail images, and their size can be changed in its configuration settings. Switch to the **Design** panel, select **Document Plugins**, and double-click **ImagePlugin** to pop up its configuration window. Currently all options are off, so standard defaults are used. Select **thumbnailsize** and set it to 50pixels. Now when you build and preview, you will find that the thumbnails are smaller. Once you have seen this, return to the **Design** panel and switch the option off so that the thumbnail reverts to its normal size when the collection is rebuilt.

Adding metadata

In the Librarian interface, change to the Enrich panel and add the Dublin Core metadata set to the collection by clicking **Manage Metadata Sets**. Then add Dublin Core metadata to a few documents. First, set the Title field to the filename, but without the extension. Then add a description of the image as Description metadata. To choose a description, open the file by double-clicking—in Windows, the image will normally be displayed by Microsoft's Photo Editor (although this depends on how your computer has been set up). A suitable description for the first file might be *Bear in the Rocky Mountains*. If you want to use longer descriptions, double-clicking the **Value** field of any metadata element pops up a larger window for the text. Repeat this process for a few more images.

If you build and preview the collection, you will be disappointed to see nothing new. That's because we haven't changed the design of the collection to take advantage of the new metadata.

Changing the format to view the new metadata

In order to see the new metadata, go to the **Format** panel and select **Format Features** from the left-hand list. Leave the feature selection controls at their default values, so that **All Features** is selected for **Choose Feature** and **VList** is selected as the **Affected Component**. Notice that the HTML format string contains four words with underscores: `_ImageName_`, `_Width_`, `_Height_` and `_Size_`. To see the reason for this, return to your Web browser and switch the display language to French or Spanish (on the Preferences page). Now the corresponding text appears in the other language. Greenstone is not translating it, of course; it's pre-stored along with all the other language fragments that are used in the interface. The underscore mechanism invokes a Greenstone macro (Section 12.1) that fetches the fragment in the current interface language.

Return to the Librarian interface, and

Change `_ImageName_` to `Title`

Change `[Image]` to `[dc.Title]`

After `[dc.Title]`
 add `Description: [dc.Description]`

The first change alters the fragment of text that appears to the right of the thumbnail image; the second alters the metadata that follows it; and the third displays the description after the title.

Again, changes in the Format panel take place immediately. When you preview the collection and click on **Browse** in the navigation bar, the presentation has changed to *Title: Bear* and so on. Each image description appears beside the thumbnail, following the title. After the first few items, the title and description probably become blank because you assigned Dublin Core metadata to only a few images. To get a full listing, enter all the metadata.

Adding a classifier and index

To add a new browsing option based on the descriptions, select **Browsing Classifiers** in the **Design** panel, set the menu item for **Select classifier to add to List**, and click **Add Classifier**. In the configuration panel, set the metadata option to **dc.Description** and set **bookshelf_type** to **never** to prevent the classifier from grouping together values that appear multiple times—which is unlikely but not completely impossible. Click **OK**.

Next, add an index so that the collection can be searched by descriptions. Select **Search Indexes** from the left-hand list and click the **New Index** button. Select **dc.Description** from the list of metadata to include in the index and click **Add Index**.

Build the collection, preview it, and choose the new Descriptions link that appears in the navigation bar. Only a few items are shown, because only items with the relevant metadata (*dc.Description* in this case) appear in the list. The original browse list includes all photos in the collection because it is based on *ex.Image*, extracted metadata that contains the image’s filename, which the Image plug-in sets for all images.

There is also a **Search** button in the navigation bar. Try out an example search.

Textual images

Next we build a small replica of a collection of New Zealand historic newspapers published primarily for a Māori audience between 1842 and 1932. We suggest that before beginning you browse around this collection on the Web at www.nzdl.org/niupepa (you will probably want to switch the interface language to English on the first page).

The collection is based on *Niupepa 1842–1933*, a microfiche compendium produced by the Alexander Turnbull Library. Two-thirds of the material is written solely in Māori, a very small amount (3 percent) is in English, and the remainder is bilingual. There are around 30 separate periodicals (called series), each with many newspapers (called issues). They can be searched (full text), browsed (by series), or accessed by date. When a document is viewed, it is presented page by page with **Next-** and **Previous-**page buttons and a “go to page” navigation feature. There are three versions of each page: plain text, screen-view image, and full-size image.

Our small replica uses five issues taken from two series. By the time we are finished it will provide the same access and viewing facilities as the full Niupepa, except that abstracts (and associated formatting details) are missing. Although the collection we build is rather small, it reflects a complex structure that is suitable for far larger volumes of information. If you had more source material, you could just drag it in, rebuild, and reap the rewards of having a life-size digital library collection—with no further work.

The collection involves a mixture of plug-ins, classifiers, and format statements. The bulk of the work is done by the Paged Image plug-in (**PagedImagePlugin**), which is designed for the kind of data we have here. For each document (issue), a file is prepared that specifies the image files that constitute it, tagged with their page number and (optionally) accompanied by a text file containing the machine-readable version of the image—typically produced by Optical Character Recognition

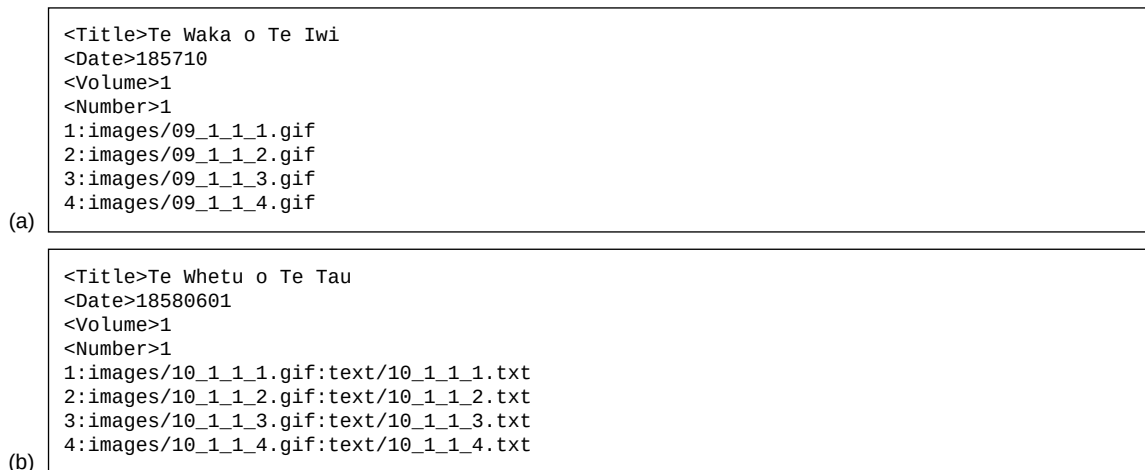


Figure 10.11: Item files for two newspapers: (a) *Te Waka*; (b) *Te Whetu*

(OCR) software (see Section 4.2)—which is used for full text searching. Three issues in our collection have full text and two have images only. These files can also specify metadata, which the plug-in extracts during the building process.

Structure of the newspapers

The source material has a fairly complex structure. There are two series of newspapers, *Te Waka o Te Iwi* (abbreviated below to *Te Waka*) and *Te Whetu o Te Tau* (*Te Whetu*). The former contains two newspaper issues, Vol. 1 No. 1 (four pages, numbered 1 to 4) and Vol. 1 No. 2 (4 pages, numbered 5 to 8). Page numbers increase consecutively through each volume, despite its being split into different Numbers. Each page in *Te Waka* is represented by a single image file. *Te Whetu* has three issues, Vol. 1 No. 1 (pages 1–4), Vol. 1 No. 2 (pages 5–9), and Vol. 1 No. 3 (pages 10–14). Each page is represented by two files, a page image and a text file with its contents.

Small files called *item* files are key to the structure; item files are read by the Paged Image plug-in. There is one item file for each issue. Figures 10.11a and b show these files for *Te Waka* Vol. 1 No. 1 and *Te Whetu* Vol. 1 No. 1, respectively. Both samples contain title, date, volume, and number metadata, and a sequence of image files (.gif) representing the newspaper pages. The second sample (Figure 10.11b) also contains a text file (.txt) for each page. You will never see these files when working with the collection, but they constitute the basic information that is processed by the Paged Image plug-in.

Building a collection

Start a new collection called Paged Images and fill out the fields with appropriate information. In the **Gather** panel, open the *sample_files*→*niupepa*→*sample_items* folder and drag the two sub-folders into your collection. A window pops up asking whether you want to add the Paged Image plug-in to the collection: click **OK** as this is needed to process the item files. Some of the files you have just

dragged in are page images; others are text files that contain the extracted text. We want them all to be processed by the Paged Image plug-in, not the Image or Text plug-in, so switch to the **Document Plugins** section of the **Design** panel and use the **Remove Plugin** button to delete **ImagePlugin** and **TextPlugin**.

Open the configuration window for the Paged Image plug-in by double-clicking it. Switch on its **create_screenshot** option by checking the box. The source images were scanned at high resolution and are large files for a browser to download. This option generates screen-resolution images of each page when the collection is built.

Go to the **Create** panel, **build** the collection, and **preview** the result. Search for *waka* (it means *canoe*) and view one of the titles listed (all three appear in the *Te Whetu* series). Browse by title and view one of the issues of *Te Waka*. Note that only the *Te Whetu* issues have text; *Te Waka* issues do not. This collection was built with Greenstone's default settings. You can locate items of interest, but the information is not presented well.

Grouping documents by series

The title browser repeats issues from the same series without any distinguishing features, such as date, volume, or number. It would be better to group them by series and to display other information within each group. This can be accomplished by using a **List** classifier and tuning the format statement.

In the **Browsing Classifiers** section of the **Design** panel, delete the two existing classifiers. Add a **List** classifier, setting its metadata option to extracted title metadata (*ex.Title*) and **bookshelf_type** option to **always**, and a **DateList** classifier, setting its metadata option to extracted date metadata (*ex.Date*). Build the collection and preview the title and date lists.

Now change the format statement for titles to display more information about the documents. In the **Format Features** section of the **Format** panel, select the *ex.Title* classifier (CL1) in the **Choose Feature** list (and leave **VList** selected beneath). Click **Add Format** to add this format statement to your collection. Delete the contents of the HTML format string box and add the text in [Figure 10.12a](#).

Preview the new titles browser. This classifier shows bookshelves, and the revised format statement has the effect of specifying in brackets how many documents they contain. It works by exploiting the fact that only bookshelf icons define *[numleafdocs]* metadata, which gives the number of leaves under that node of the classifier tree. For document nodes, volume, number, and date information is shown instead of titles.

A date appears at the end of each document title. In format statements, a *[format:]* prefix is added to date metadata to improve the display—otherwise dates appear in Greenstone's internal date format, for example 18580601. You could tidy up the format statement by going to the **Format Features** section of the **Format** panel, selecting the **DateList** (CL2) classifier, click **Add Format**, and replacing the last line

```
<td>{0r}{[format:dc.Date],[format:ex.Date],[format:ex.Date]}</td>
```

- (a)

```
<td valign="top">[link][icon][/link]</td>
<td valign="top">
  {If}{[numleafdocs],[ex.Title] ([numleafdocs]),
  Volume [ex.Volume] Number [ex.Number] Date [format:ex.Date]}
</td>
```
- (b)

```
<table><tr>
<td valign=top>[srclink][screenicon][/srclink]</td>
{If}{[NoText] ne '1',<td valign=top>[Text]</td>}
</tr></table>
```
- (c)

```
<td valign="top">[link][icon][/link]</td>
<td valign="top">
  {If}{[parent:ex.Title],[parent:ex.Title] Volume [parent:ex.Volume]
  Number [parent:ex.Number]: Page [ex.Title],
  [ex.Title] Volume [ex.Volume] Number [ex.Number]}
  <br><i>({Or}{[format:parent:ex.Date],[format:ex.Date],undated})</i>
</td>
```

Figure 10.12: Format statements for the paged image collection: (a) for titles; (b) for document text; (c) for search results

with

```
<td>[format:ex.Date]</td>
```

because this collection does not use the other data metadata. The result should be exactly the same.

Displaying scanned images

When you reach a newspaper issue, only its associated text is displayed. When either issue of *Te Waka* is accessed, the document view reads *This document has no text*. No scanned image information (screen-view resolution or otherwise) is shown, even though it has been computed and stored with the document. This can be rectified by a format statement that modifies the default behavior for document text.

In the **Format Features** section of the **Format** panel, select the **DocumentText** format statement and change it to the text in [Figure 10.12b](#). Including `[screenicon]` has the effect of embedding the screen-sized image produced by the Paged Image plug-in's `create_screenview` option. It is hyper-linked to the original image, which is a large image (although it may be scaled by your Web browser), by `[srclink] ... [/srclink]`. The *If* statement eliminates the unwanted message for documents that lack text. Preview the collection and check this by viewing an issue of *Te Waka*.

Searching at page level

The documents are split into sections, one per page. For large documents, it is useful to be able to search sections as well as entire documents. Go to the **Search Indexes** section of the **Design** panel. Remove the **ex.Source** index, and check **section** to build the indexes on section level as well as document level. Make the former the default using the associated radio button. Build and preview the search facilities. Set the display text used for the “level” drop-down menu by going to the **Search**

section of the **Format** panel. Set the document-level text to *newspaper* and the section-level text to *page*. Compare searching at newspaper level with searching at page level (a useful search term for this collection is *aro**ha*, meaning, approximately, *love*).

In a search for individual pages, the page image is displayed in the search results. Because the images are very large, this is not very useful. Go to the **Format Features** section of the **Format** panel, choose **All Features** in the **Choose Feature** list, and remove this line from the format string (the second line):

```
<td valign="top">[ex.srclink]{0r}{[ex.thumbicon],[ex.srcicon]][ex./
srclink]</td>
```

Also, remove the filename from the display by deleting this text from the last line:

```
{If}{[ex.Source],<br><i>([ex.Source])</i>}
```

Preview the collection and check the result.

Page-level searches show the page title (i.e., its page number) rather than the series title. It would be better if the series title, volume, and number information were shown as well. In the **Format Features** section, select **Search** in **Choose Feature**. Click **Add Format** to add this format to the collection. The previous changes applied to all components of the interface, but the next one will apply only to the search results. The current section's extracted title is *[ex.Title]* and its parent section's title is *[parent:ex.Title]*. Matters are complicated by the fact that the same format statement is used when searching both whole newspapers and individual pages and so must work in both cases. Set the format statement to the text in [Figure 10.12c](#).

Preview the search results. Items display series title, volume, number, and date; and pages also display the page number.

Adding new documents

Another newspaper has been scanned and OCR'd, but has no item file. In the **Gather** panel, drag the folder *sample_files*→*niupepa*→*new_papers*→*12* into the collection. As you can see, it contains four images and four text files. Create an item file for it based on the examples in [Figure 10.11](#). Open a new document in a text editor (e.g., WordPad or Emacs) and add metadata. This newspaper's series title is *Te Haeata 1859–1862*; its Volume is 3; its Number is 6; and the Date is 18610902 (encoded as *yyyymmdd*). Then, for each newspaper page, add a line in the file that specifies it. For example, the first page entry is

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

If there is no text file, simply omit it. Add one line for each page in the document, and be sure to increment the page number for each line. Save the file as *12_3_6.item* (as a plain text document—if you are using Windows, be sure the file name does not have a *.txt* extension). In the Librarian interface's **Gather** panel, locate the new file and drag it into the collection, adding it to the *12* folder. Build the collection and preview it to check that the document has been added.

The item files we have seen use a simple textual format, consisting of metadata for the document followed by a list of pages. An alternative style uses XML and allows a hierarchy of pages, with meta-

data specification at page as well as document level. Here we add two newspaper issues in this style. In the **Gather** panel, drag the folder *sample_files* → *niupepa* → *new_papers* → *xml* to the collection (be sure to add the *xml* folder, not the 23 folder). Open the file *xml* → 23 → 23__2.item and look at the XML. This is No. 2 of the newspaper series entitled *Matariki 1881*. The contents are grouped into two sections: supplementary material, which contains an abstract, and newspaper pages, which comprise the page images and OCR text.

Build and preview the collection. Although the new documents have been included, the display is poor.

Controlling document processing

Paged documents can be presented either with next- and previous-page arrows and a “go to page” box—as we have seen so far—or with a hierarchical table of contents. The display type is specified by an option to the Paged Image plug-in called **documenttype**, with values *paged* or *hierarchy*. The first type suits linear documents, while the second suits hierarchical ones. Ordinarily, a collection has one plug-in per document type, and all documents of that type are treated in the same way. In the present case, the XML-based item files should be treated differently from the text-based ones. This can be achieved by adding two Paged Image plug-ins to the collection and configuring them differently.

In the **Document Plugins** section of the **Design** panel, add a new **PagedImagePlugin**. Enable the **create_screenview** option, set the **documenttype** to **hierarchy** and the **process_exp** option to *xml.*.item\$*. Move this plug-in above the original Paged Image plug-in in the **Assigned Plugins** list.

The XML-based newspapers have been grouped into a folder called *xml*. The *process_exp* value for the first Paged Image plug-in processes item files that include *xml* in their folder name; these will be treated as hierarchical documents. Item files that do not match the process expression will be passed to the second Paged Image plug-in and treated as linear documents. Rebuild and preview the collection. Compare the display for a linear document, e.g., *Te Waka* Vol. 1 No. 1, with that for a hierarchical document, e.g., *Matariki 1881*, No. 1.

Switching between images and text

The document display can be modified to switch between the text version and the screen-view and full-size versions. This requires a combination of format statements and macro files. We met macros briefly when working with the simple image collection above; recall that they are flanked by underscores. Figure 10.13 shows a suitable macro file that defines three substantial macros, called *viewfull-size*, *viewpreview*, and *viewtext*, along with three buttons (the first three) and three snippets of text. (The Greenstone macro system is presented in detail in Chapter 12 and the content of this figure will be clearer after you have read that chapter.)

Click **Collection Specific Macros** on the left-hand side of the **Format** panel and copy the contents of Figure 10.13 into the text box. Under **Format Features**, select **AllowExtendedOptions** in the **Choose Feature** list and click **Add Format**. Tick the **Enabled** check-box. This gives more control over the layout of the page. We will replace the standard **Detach** and **No Highlighting** buttons with ones that switch between text and images.

```

_textfullsize_ {FULLSIZE}
_textpreview_ {PREVIEW}
_textfulltext_ {TEXT}

_textviewfullsize_ {View the fullsize image of this page}
_textviewpreview_ {View the preview image of this page}
_textviewfulltext_ {View the text of this page}

_viewfullsize_{
  <div class="button"><span class="button">
    <a href="_httpcurrentdocument_&p=fullsize" title="_textviewfullsize_">
      _textfullsize_</a>
    </span></div>
}
_viewpreview_{
  <div class="button"><span class="button">
    <a href="_httpcurrentdocument_&p=preview" title="_textviewpreview_">
      _textpreview_</a>
    </span></div>
}
_viewtext_{
  <div class="button"><span class="button">
    <a href="_httpcurrentdocument_&p=text" title="_textviewfulltext_">
      _textfulltext_
    </a></span></div>
}

```

Figure 10.13: Macro file for the paged image collection

- (a)
- ```

<div class="heading_title">{Or}{[parent(Top):ex.Title],[ex.Title]}</div>
<div class="buttons" id="toc_buttons">
 {If}{[srcicon],_document:viewfullsize_}
 {If}{[screenicon],_document:viewpreview_}
 {If}{[NoText] ne '1',_document:viewtext_}
</div>
<div class="toc">[DocTOC]</div>

```
- (b)
- ```

{If}{_cgiargp_ eq 'fullsize',[srcicon],
  {If}{_cgiargp_ eq 'preview',[screenicon],
    {If}{[NoText] ne '1',[Text],[screenicon]}
  }
}

```

Figure 10.14: Formatting for the paged image collection: (a) DocumentHeading; (b) DocumentText

Select the **DocumentHeading** format item and set it to the text in Figure 10.14a. The *Or* statement in the first line outputs the newspaper's title metadata. This is stored at the top level of the document, so if we are at a subsection, it must be retrieved from the top-level parent. (Note: an unqualified *parent* would retrieve it from the immediate parent, which may not be the top level.) The variables starting with *document* and flanked with underscores refer to macros defined in Figure 10.13 that output buttons for preview, full-size, and text versions. The appropriate button depends on the kind of metadata and text in the document. *[DocTOC]* is the document's table of contents or "go to page"

navigation element. This must be explicitly specified if it is to appear in the page, because we are using extended options. The various pieces are surrounded by `<div>` elements so that the appropriate styling information can be used.

Select the **DocumentText** format item and set it to the text in [Figure 10.14b](#). This statement changes the display based on the value of `_cgiargp_`, which is the *p* argument in the URL when the digital library is consulted at run time. We use this argument to switch between the full-size, preview-size, and text versions of each page (`[srcicon]`, `[screenicon]`, and `[Text]`, respectively). Preview the collection. When you reach a newspaper page, you will get full-size, preview, and text options.

Multimedia

As a final example, we build a multimedia collection that contains audio files, pictures, PDF files, Word files, HTML files, plain text files, and bibliography information centered on the Beatles. In truth it is very simple to deal with all the multimedia parts; the main issue is putting them together. Again we will learn about Greenstone facilities that can be applied to other types of collections.

Exploring the Beatles collection

To do this part, you need to know where Greenstone keeps its files. As you will learn in the next chapter, all the collections are kept in a folder called *collect* in your Greenstone installation. This has one sub-folder for each collection. The folder *sample_files* → *beatles* → *advbeat_large* contains an entire new collection about the Beatles. Copy it (with all its contents) into the above *collect* folder. Restart the Windows Local Library server if that is what you are using. (Click the CD icon in the task bar to make the server appear and press **Restart Library**. If it is not running, start it up by selecting Greenstone Server from the Start menu.)

Explore the Beatles collection. Note how the **Browse** button organizes the material into different types. The documents within each category have appropriate icons. Some have an audio icon: when you click an audio icon, you hear the music (assuming your computer is set up with appropriate player software). Others have an image thumbnail: when you click one, you see the images. Each entry under the titles browser has a bookshelf that may include several related items. For example, *Hey Jude* has a MIDI file, lyrics, and a discography item.

Observe the low quality of the metadata. For example, the four items under *A Hard Day's Night* (under *H* in the titles browser) have different variants as their titles. The collection would have been easier to organize had the metadata been cleaned up manually first, but that would be a big job. To build this collection, only a tiny amount of metadata was added by hand—fewer than ten items. The original metadata was left untouched and was cleaned up automatically.

In a file browser (or terminal window), take a look at the files that make up the collection, in the folder *sample_files* → *beatles* → *advbeat_large* → *import*. What a mess! There are over 450 files under seven top-level sub-folders. Organization is minimal, reflecting the different times and ways the files were gathered. For example, *html_lyrics* and *discography* are excerpts of Web sites, and *images* contains various images in JPEG format. For each type, drill down through the hierarchy and look at a sample document.

Building the collection

We proceed to reconstruct this collection from scratch, developing it using a subset of the material to accelerate the repeated rebuilding that is involved. Start a new collection (**File** → **New**) called *small beatles*, basing it on the default New Collection. (Basing it on the existing Advanced Beatles collection would make your life far easier, but we want you to learn how to build it from scratch!). Drag in the files from *sample_files* → *beatles* → *advbeat_small*. Do this by selecting the eight items in this folder (from *discography* to *beatles_midi.zip*) and dragging them across. Because some are in MP3 and MARC formats, you will be asked whether to include the MP3 and MARC plug-ins in your collection; do so. Change to **Enrich** and browse around the files. There is no metadata—yet. (There are no MIDI files in the collection: they require more advanced customization because Greenstone has no MIDI plug-in.) Change to the **Create** panel, build the collection, and preview the result.

Manually correcting metadata

Extracted metadata is unreliable. There are some mistakes in the metadata extracted from the source documents to this collection, such as the misspelling in the title *Magical Mistery Tour*. On the other hand, manually assigned metadata is reliable, but expensive. As we have seen in earlier exercises, using a classifier such as **dc.Title;ex.Title**—which, incidentally, gets included by default when a collection is created—means that any metadata manually assigned to a document is used in preference to the automatically extracted version. This gives the best of both worlds, where manual intervention is necessary only when a piece of automatically generated metadata is incorrect.

Let's correct that misspelling in the title *Magical Mistery Tour*. There are two documents, both in the discography section, with misspelled file names. Locate them in the **Enrich** panel. Open the discography folder, drill down to the individual files, and set the title value. Notice that you cannot correct the extracted title metadata element. Instead, add Dublin Core Title metadata for these files: *Magical Mystery Tour*. Finally, build the collection again, preview it, and check that the correct title is now displayed.

Browsing by media type

Next we remove the filename browser and replace it with a structure that groups documents by category (discography, lyrics, audio, etc.). We will define categories by manually assigning metadata. In the **Enrich** panel, select the discography folder and set its *dc.Format* metadata value to *Discography*. This value is inherited by all files in the folder. Repeat the process, assigning *Lyrics*, *Images*, *Marc*, *Audio*, *Tablature* to their respective folders, and *Supplementary* to the *wordpdf* folder.

In the **Design** panel select the **Browsing Classifiers** section. Delete the filename (*ex.Source*) classifier and add a **List**, selecting *dc.Format* as the metadata field, setting **bookshelf_type** to **always**, and specifying *browse* as the button name. Switch on the **sort_leaf_nodes_using** check-box and select extracted titles in the drop-down list: this will make the classifier display documents in alphabetical order of title. Build the collection again and preview it.

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

Compacting the titles

Sometimes several documents appear with the same title. For example, *All My Loving* appears both as lyrics and tablature (capitalized). The titles browser can be improved by grouping these together under a bookshelf icon. In the **Design** panel, select **Browsing Classifiers**, remove the title classifier (at the top), and add a **AZCompactList** classifier, entering **dc.Title;ex.Title** as its metadata. Move the new classifier to the top of the list (**Move Up** button). Build the collection again and preview it. Both items for *All My Loving* now appear under the same bookshelf. However, many entries have not been amalgamated because of non-uniform titles: for example *A Hard Day's Night* appears as four different variants.

We now clean up the titles browser. One way would be to define proper metadata manually. But that would be a lot of work! Instead we amalgamate variants of titles by stripping away extraneous text. For example, we will use regular expressions to treat *Anthology 1*, *Anthology 2*, and *Anthology 3* the same for grouping purposes. Go to the **List** for titles under **Browsing Classifiers** on the **Design** panel. Activate **removesuffix** and set it to:

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

Build the collection and preview the result. Observe how many more times similar titles have been amalgamated under the same bookshelf. This is difficult to understand, and a bit fiddly to do, but if you can cope with its idiosyncrasies, it provides a quick way to build an impressive collection without much manual labor. Test your understanding of regular expressions by trying to rationalize the amalgamations. (Note: `[[[:punct:]]]` stands for any punctuation character.) The icons beside the Word and PDF documents are not the correct ones, but we will fix that in the next format statement.

The best way to clean up the titles list would be to have proper title metadata. The metadata extracted from HTML files is messy and inconsistent, and this was reflected in the original titles browser. Defining proper metadata would be a simple but tedious task. Instead, we have used a regular expression to clean up the title metadata.

Improving the formatting

Alongside the audio files is an MP3 icon, which plays the audio when you click it, and also a text document that contains some dummy text. Image files also have dummy documents. To remove them, switch to the **Format** panel and select the **Format Features** section. Ensure that **VList** is selected and change the format statement to that in [Figure 10.15a](#).

As well as removing the dummy text, this format statement does two other things. It removes the filename from where it appears after each document. And it makes bookshelves in the titles and browse classifiers show how many documents they contain. Preview the result (you do not need to rebuild the collection.)

Using UnknownPlugin

Now incorporate the MIDI files. Greenstone has no MIDI plug-in. But that doesn't mean you can't use MIDI files! **UnknownPlugin** is a generic plug-in that knows nothing about any given format but

can be tailored to process particular document types—like MIDI—based on their filename extension, and set basic metadata. Add **UnknownPlugin** in the **Design** panel's **Document Plugins** section, set its **process_extension** field to *mid* to make it recognize files with this extension, and set **file_format** to *MIDI* and **mime_type** to *audio/midi*. In this collection, all MIDI files are in a ZIP file. The ZIP plug-in (already in the list of plug-ins) unpacks them and passes them down the list of plug-ins until

(a)

```
<td valign=top>
{If}{[dc.Format] eq 'Audio',
  [srclink][srcicon][\srclink],
  {If}{[dc.Format] eq 'Images',
    [srclink][thumbicon][\srclink],
    [link][icon][\link]
  }
}</td>
<td>{If}{[numleafdocs],([numleafdocs])}</td>
<td valign=top>[highlight]
  {Or}{[dc.Title],[Title],Untitled}
[/highlight]</td>
```

(b)

```
package Style
_collectionspecificstyle_{
  <style>
    body.bgimage \{ background-image: url("_httpcimages_/beat_margin.gif"); \}
    \#page \{ margin-left: 120px; \}
  </style>
}

package Global
_icondisc_{
}
_iconlyrics_{
}
icontab_{
}
iconmarc_{
}
iconmidi_{
}
iconmp3_{
}

_textdisc_ {discography}
_textlyrics_ {lyrics}
_texttab_ {tablature}
_textmarc_ {MARC record}
_textmidi_ {MIDI audio}
_textmp3_ {MP3 audio}

_textdisc_ [l=fr] {discographie}
_textlyrics_ [l=fr] {lyrique}
_texttab_ [l=fr] {tablature}
_textmarc_ [l=fr] {MARC format}
_textmidi_ [l=fr] {MIDI format}
_textmp3_ [l=fr] {MP3 format}
```

Figure 10.15: Improving the Beatles collection: (a) format statement; (b) macro file;

```

<td valign=top>
  {If}{[numleafdocs],[link][icon][link]}
  {If}{[dc.Format] eq 'Lyrics',[link]_iconlyrics_[link]}
  {If}{[dc.Format] eq 'Discography',[link]_icondisc_[link]}
  {If}{[dc.Format] eq 'Tablature',[link]_icontab_[link]}
  {If}{[dc.Format] eq 'MARC',[link]_iconmarc_[link]}
  {If}{[dc.Format] eq 'Images',[srclink][thumbicon][srclink]}
  {If}{[dc.Format] eq 'Supplementary',[srclink][srcicon][srclink]}
  {If}{[dc.Format] eq 'Audio',
    [srclink]{If}{[FileFormat] eq 'MIDI',_iconmidi_,_iconmp3_[srclink]}
  }
</td>
<td>
  {If}{[numleafdocs],[[numleafdocs]]}
</td>
<td valign=top>
  [highlight]
  {Or}{[dc.Title],[Title],Untitled}
  [/highlight]
</td>

```

(c)

Figure 10.15, cont'd: (c) format statement that uses the macros

they reach UnknownPlugin. Back in **Enrich**, click on the ZIP file and assign *Audio* as its format value—do this by clicking on Audio in the existing values list. All files extracted from the ZIP file inherit its settings. Build the collection and preview it to check that you can see the MIDI files, and play them.

Add phrase and collage browsers

In the **Browsing Classifiers** section of the **Design** panel, add a **Phind** classifier (leave the settings at their defaults). This generates a phrase-browsing classifier that sources its phrases from titles and text. Build the collection again and preview it. Select the new *phrases* option from the navigation bar. Enter a single word in the text box, such as *band*. The phrase browser will present you with phrases found in the collection containing the search term. This can provide a useful way of browsing a very large collection. Note that even though it is called a phrase browser, only single terms can be used as the starting point for browsing.

Now we add a special browser for images. In **Browsing Classifiers**, add a **Collage** classifier. Again there is no need to customize the options. Build the collection and preview the effect of the new collage image browser.

Customizing the appearance

Put an image in the top left corner of each page. Go to the **General** section of the **Format** panel. Use the **Browse** button to set the *About* page image to *sample_files* → *beatles* → *advbeat_large* → *images* → *beatlesmm.png*. Preview the collection and check that the image appears.

Now we will use different icons for different media types. Click **Collection Specific Macros** on the left-hand side of the **Format** panel and copy the contents of Figure 10.15b into the text box. Re-edit the **VList** format statement to that shown in Figure 10.15c. (As mentioned earlier, the macro system is described fully in Chapter 12.)

For this to work, you need to move some files into the collection, outside the Librarian interface. In a file browser, go to the folder called *collect* in your Greenstone installation. This has a sub-folder *advbeat_large* that contains the “advanced Beatles” collection that you installed first; the sub-folder *smallbea* contains the small one that you are working on. Each of these contains a sub-folder called *images*. Use a file browser to copy the *images* folder, which contains some useful icons, from *advbeat_large* to *smallbea*. (It’s OK to overwrite the existing *images* folder.)

Preview your collection. Notice that different icons are used for discography, lyrics, tablature, and MARC metadata. Even MP3 and MIDI audio file types are distinguished. If you let the mouse hover over one of the images, a tool tip appears explaining what file type the icon represents in the current interface language. (Note: the macro file defines only English and French.)

Next we change the collection’s background image. In the **Collection Specific Macros** section of the Format panel, change the fourth line (in [Figure 10.15b](#)), replacing *beat_margin.gif* with *tile.jpg*. This line relates to the background image used; the new image was in the folder that you just copied. You will find that the page background in the collection is now the new graphic. Other features—for example, the headers and footers used on each page, and the highlighting style used for search terms—can be altered by editing the macros.

Finally, build a full-size version. Start a new collection called *large beatles* (**File→New**) and base it on the small Beatles collection that you have been working on. Drag the contents of *sample_files→beatles→advbeat_large→import* into this newly formed collection. Build the collection and preview the result. (The collection icon is not copied across when you base a new collection on an old one; you will have to add it in the Format panel.)

10.7 Learning More

If you have gotten this far and done all the exercises, congratulations! You are already a Greenstone expert. Most likely you didn’t complete everything—particularly the last two exercises in [Section 10.6](#), which are exceedingly challenging. Even so, you now have a good grounding in Greenstone.

What you need to continue learning is how to seek help and answer the questions that will inevitably arise when you start building your own collections. The next chapter contains more information about Greenstone’s operation and how to interoperate with other digital libraries, but it will probably not help you answer nitty-gritty questions about the details of collection building.

Sources of information

Four manuals for Greenstone are available. They are:

- Installer’s Guide (36 pp.)
- User’s Guide (90 pp.)
- Developer’s Guide (113 pp.)
- From Paper To Collection (30 pp.).

These manuals predate the information in this chapter. The Installer's Guide covers the different versions of Greenstone, the installation procedure, setting up the Web server, configuring your site, and personalizing the installation. It will be useful if you are having problems installing the software. The User's Guide describes how to use Greenstone collections, the Librarian interface, administration of the installation, and software features—but most of the information has been covered in this chapter. The Developer's Guide summarizes how the collection-building process works, including information about plug-ins, classifiers, format statements, and macros. It also describes the Greenstone runtime system and how to configure your Greenstone site. This may help you with some of the more intricate details. The final guide is not about the software as such, but covers scanners, scanning, and OCR. It outlines the logistics of creating three example collections, from 1,000 to 100,000 pages.

The Greenstone Web site (www.greenstone.org) is a good source of up-to-date information. In particular, it includes a wiki and a list of frequently asked questions. The Greenstone wiki (www.greenstone.org/wiki) brings together all the documentation and is constantly being updated. The section on documentation includes the manuals mentioned above, and much more:

- Frequently Asked Questions
- General documentation
- Tutorial exercises
- Example collections
- Workshops.

The Frequently Asked Questions (FAQ) pages are particularly useful. They cover issues like installing, building collections, formatting, etc. They are divided into sections to make them easy to consult and browse. Some of the contents are user-contributed; anyone can register and add new entries.

The general documentation section is a work in progress that gives access to many miscellaneous documents. Some documents are written by project members in response to specific queries (e.g., there are documents about formatting, macros, and using CSS in Greenstone); others are contributed by users and describe particular aspects of the system or problems they have encountered and solved.

The tutorial exercises are like those in this chapter, except that there are more of them and the steps are set out in more detail. There are several documented example collections whose *About* pages describe how they are constructed. They are a good resource for learning how to build common types of collections. Also, by choosing to base a new collection in the Librarian interface on one of these collections, you can inherit its style and formatting without having to recreate it.

Many workshops have been given all over the world that focus on installing the software and building collections using the Librarian interface. This is an open source project, and all the presentations, lab instructions, screenshots, and test files for several collections are in the wiki. Feel free to use this material for learning—or teaching!—about Greenstone.

The user community

Greenstone user groups have formed in some areas of the world, including South Asia (particularly India) and Southern Africa. There are also language-specific groups for Arabic, Spanish, and French users. Users are encouraged to form their own regional support groups; the wiki contains information about this. There is also a Greenstone blog for announcing significant developments.

There are two mailing lists: a Users List for general discussion and questions, and a Developers List for more technical discussions by people developing or modifying the software. All users are encouraged to join these lists. However, please consult other information sources, particularly the mailing list archive, before posting questions. It is very likely that your question has already been answered!

The mailing list archive is a Greenstone collection that contains all the questions and responses posted to the two mailing lists. Searching this collection is a good way to find answers to questions that haven't yet made it into the FAQ. It is updated regularly and is accessible to everyone.

Here's an example. When describing enhanced PDF document handling in [Section 10.3](#), we mentioned that it would nice, after searching for a document, to display it with the query terms highlighted if the PDF icon is clicked, although we did not describe how to do so. A quick search of the mailing list archive yields a discussion thread on *Passing search terms to PDF files*—exactly what is needed.

When things go wrong ...

Inevitably, things go wrong. Problems often emerge during collection building. Maybe some files can't be processed: the rest of the collection builds fine and can be previewed, but some documents are missing. Or the whole collection is not built properly, in which case a message says "An error has occurred and the collection could not be created."

A simple and useful strategy is simply to close down the Librarian interface and the Greenstone digital library server, and restart it. This doesn't take more than a few seconds, because the Librarian interface comes back with the existing collection already loaded. Often, the collection has actually been built and you can simply preview it; other times a rebuild is necessary.

It is often helpful to switch the Librarian interface into Expert mode and rebuild the collection to see if any error messages appear. In fact, we did this when working with PDF documents in [Section 10.3](#). In this mode a Log History entry appears at the bottom of the Create panel that you can use to review previous attempts to build the collection, whether successful or not—select the log you want by double-clicking on it in the list. A full list of import and build options can be shown by selecting the tabs to the left. A useful option is *verbosity*: set it to the maximum value of 5 for both importing and building to generate a detailed log of what is going on.

Another useful debugging strategy, particularly when your collection doesn't look the way you think it should, is to check what metadata fields the classifiers are built on and actually look at this metadata in the Enrich panel. Remember to scroll down the metadata table to see the extracted metadata! In the Preferences for the Librarian interface there is a check-box called View Extracted Metadata

that allows you to hide the extracted metadata in the Enrich panel (although it is still available during collection design and in the final collection).

Other items in the Preferences panel can be useful for problem solving. The Connection tab lets you alter the path to the locally running Greenstone library server, which is used when previewing collections. You can also set proxy information for connecting to the Internet (e.g., when downloading files). The proxy connection is established when you close the Preferences dialog.

The Librarian interface uses different application programs to open different file types. To alter these, click **File Associations** on the File menu. To modify a launch command, type the new command or click **Browse** to search for an appropriate program. Use the special string %1 (which is appended to the end of a command if not otherwise specified) to indicate where to insert the name of the file to be opened, e.g., `C:\Windows\System\MSPaint.exe %1`. Mac users should always try the Mac-specific command `open [-a <application name>] %1` first. To add a new file type, enter its extension into the *For files ending* box.

Operating and interoperating

As demonstrated in Chapter 10, Greenstone makes it very easy to build collections, although it is a little harder to format them to your own requirements. If the Librarian interface does everything you need, ignore this chapter and start working on your collections! However, if you are going to build and serve digital libraries for public consumption rather than private use, you will probably need to know more. This chapter describes the general structures that make Greenstone work and—most importantly—how to learn more when you need to.

First you need to learn about your Greenstone installation: where to find it on your computer, how to update to new versions, and how to move collections from one computer to another. We also describe how to create self-contained Greenstone CD-ROMs and DVDs, which is very easy—and really belongs in Chapter 10. We have left it until now because it's also useful to know how to take these self-contained collections and move them into your Greenstone installation if necessary.

The next section describes how to run your Greenstone installation. There are many configuration options; what you get when you install the software is just a plain default version. You need to know how to keep user logs (and where they are) and how to add users with different privileges—for, although we have not seen it yet, there is a built-in authentication facility with which you can build protected collections, or collections in which certain documents are protected. The configuration files contain detailed information on the available options: the most important thing is to be able to find them in the first place.

Particularly for large digital libraries, it helps to understand the processes involved in building collections outside the Librarian interface. The Librarian interface gives you access to underlying processes in a convenient form, but you gain extra flexibility by accessing these processes directly. First we walk you through the operations involved in building a collection, and then we describe what happens under the hood.

We call this chapter “Operating and interoperating” because Greenstone has a rich set of facilities for accessing other sources of information, library standards, and digital library systems. The Librarian interface can bring in information by downloading Web pages or whole Web sites. It can retrieve data from protocols like Z39.50 (Section 7.1), the OAI Protocol for Metadata Harvesting (Section 7.2), and SRU (Section 7.4). You can also arrange for any of your collections to be served over OAI so that others can see it using this protocol. And you can export collections to different formats so that they can be loaded into other digital library systems like DSpace and Fedora (Section 7.6).

[Section 11.6](#) is for those who need to work together on more than one computer to build collections and to add documents to them. A remote version of the Librarian interface allows different

workstations to access the same central Greenstone server, with all the facilities of the regular Librarian interface. (Only one person can operate on the same collection at any given time, however.) And an institutional repository subsystem lets authorized users add documents and metadata to collections from a Web browser.

Finally, we consider scalability. How large can Greenstone collections be? This is a difficult question to answer because there are many different dimensions in which collections can be large, and also because it depends on both your computer and your patience. Being a Java application, the Librarian interface is good for only about 10,000 to 20,000 documents, depending on the amount of metadata—and the amount of computer memory. However, the underlying Greenstone programs can cope with a thousand times this amount. We close this chapter by looking inside the National Library of New Zealand's Papers Past collection (described in Section 3.1 and illustrated in Figure 3.4), which is based on Greenstone. It's big: 6.5 million documents, 2 billion words, 20 GB of raw text, and 50 GB of metadata, along with a terabyte of page images.

This chapter, like Chapter 10, describes the production version of Greenstone. Most of what we say here applies equally well to the upcoming new version, Greenstone3, except that its file structure is organized differently. For more information about this, see the last section of Chapter 12.

11.1 Inside Greenstone

Where does the digital library system live on your computer disk? And, why would you want to know? Sometimes you need to find certain configuration files in order to control various options and facilities. Since the software is open source, you can examine any part of the system—including the programs themselves.

Most likely you are not interested in the program code, but you probably will want to know where the collections themselves are kept. Greenstone collections are self-contained nuggets of information. You can give them to others by placing them on a USB flash drive or CD-ROM and having the recipient copy them into the appropriate part of *their* digital library installation.

But suppose they don't have the Greenstone software? It's possible—trivial, in fact—to take any collection or set of collections and write them to CD-ROM or DVD in such a way that users with no Greenstone installation and no Internet connection can access them just as though they were on the Web. These disks contain a mini version of the software that installs on any Windows computer in a matter of moments and includes all the searching, browsing, and multi-lingual interface facilities that the full system offers.

But before we begin, here's a brief plea: Please keep your Greenstone installation up to date!

Updating the software

In contrast to commercial software, open source software is characterized by frequent releases containing bug fixes and new features. New releases of Greenstone are typically issued every few months. If you can't wait to get your hands on a new feature, a snapshot of the system is generated automatically every day.

It is very easy to upgrade to a newer version. Before doing so, however, ensure that the computer is not running the Librarian interface or the Greenstone server. Close the Greenstone server by selecting the disk icon in the task bar (to bring the window to the front) and then clicking the exit hotspot—usually marked as a red cross. The re-installation procedure is exactly the same as the original installation. Greenstone installers check for the existence of previous versions and ask you to uninstall them if necessary—on Windows uninstalling is done either through the Control Panel or by selecting the relevant Greenstone item on the Start menu. At the end of the uninstall procedure you will be asked whether you want all your collections removed: say *no* if you wish to preserve your work.

Occasionally, problems are encountered if older installations are not fully removed. To clean up your system—having already run the installer—move Greenstone’s *collect* folder, which contains all your collections, to the desktop (or somewhere else that is convenient for you, such as *My Documents*). Then check any places where Greenstone has been installed previously and delete them.

Files and folders

Before going any further, you need to learn how to find your way around the software. [Figure 11.1](#) shows the structure of the Greenstone home directory, including one collection, the Demo collection.

First, locate your Greenstone installation. On Windows, by default it is installed in `C:\Program Files\Greenstone` if the user had full administration rights when it was installed, otherwise `C:\Documents and Settings\<username>\Greenstone` is used for Windows XP and `C:\Users\<username>\Greenstone` for Windows Vista. Comparable logic applies to Mac and Linux distributions: `/usr/local/Greenstone` is used by default when a system administrator installs the software, otherwise `/home/<username>/Greenstone` is used. We call this the GSDLHOME folder. It contains several sub-folders, illustrated in [Figure 11.1](#). The most important one is called *collect*, which holds all the collections in your installation. Inside it is a folder called *demo*, and many other folders, too, if you have worked through the exercises in the previous chapter. Each collection is self-contained and corresponds to a single sub-folder of *collect*. We describe the *collect* folder soon.

Before we do so, check out the remaining folders of [Figure 11.1](#). Some hold program code. For example, *bin* (short for executable binary) contains the programs that are used in the building process. Binary programs are held in the folder *linux* for Linux, *windows* for Microsoft Windows, and *darwin* for MacOS X (which is the name Apple gave to its version of Unix). Typically, installations contain binaries only for the platform they are running on—except when an installation, say on a USB flash drive, is set up to run on multiple platforms. [Figure 11.1](#) is based on a Linux installation.

The *script* sub-folder holds the programs used to create, build, and rebuild collections, which are written in the Perl programming language and therefore platform independent. The same is true for the programs in the *java* folder. The *perllib* folder near the end contains program modules that are used by the building scripts. Plug-ins and classifiers are placed in the corresponding sub-folders. (Plug-ins are discussed further in [Section 11.4](#).) The *cgi-bin* folder contains the Greenstone runtime system that works with a Web server. In the Local Library version of Greenstone on Windows, the necessary runtime executable (*server.exe*) is placed in the top-level GSDLHOME folder.

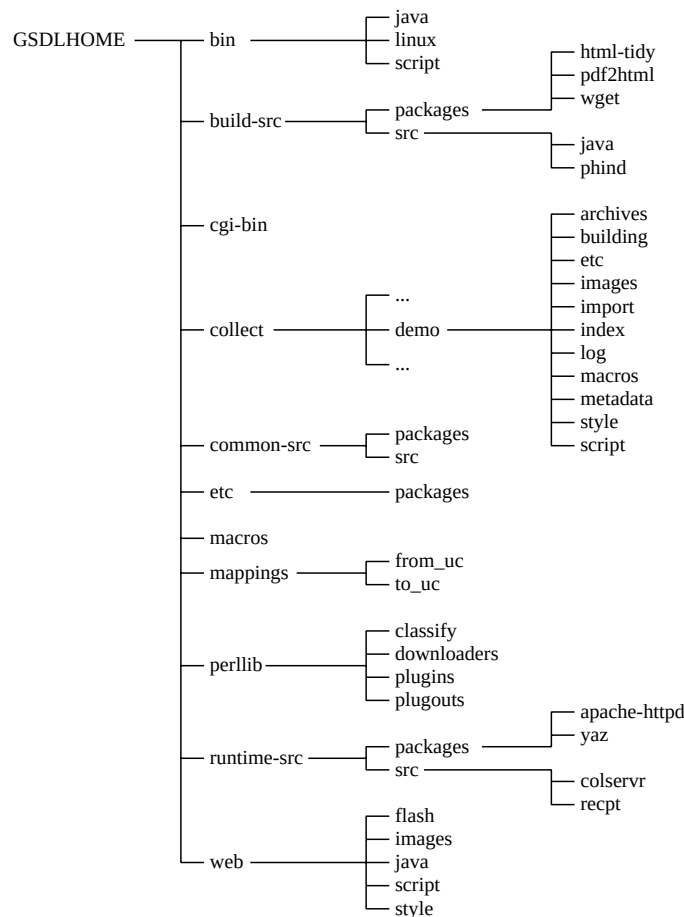


Figure 11.1: Structure of the Greenstone home directory (abridged)

What about the source code? The runtime system is written in the C++ language. The build-time code is primarily written in Perl but calls upon some external C modules. The source code is distributed across three folders: *build-src*, *runtime-src*, and *common-src*, the last containing code that both parts of Greenstone need. Code specifically written for Greenstone is located within each of these folders in a sub-folder called *src*, and any third-party software (all distributed under the GNU Public License and other compatible licenses) is located in a sub-folder named *packages*.

The main Greenstone source code in the *runtime-src* folder comprises the “collection server” (*col-servr* in Figure 11.1), the “receptionist” (*recpt*), and the “protocol” they share (not shown). An example of source code in *build-src* is *hashfile*, used to compute document IDs, and in *common-src* a general purpose library *lib* is provided that reads Greenstone configuration files and represents and manipulates strings in Unicode format, among other things.

The functions of third-party packages vary widely. The *packages* folder of *build-src* contains (among many other things) a program that converts from HTML to XML (*html-tidy* in Figure 11.1), a Web

mirroring program (*wget*), and a utility that converts PDF documents to HTML (*pdf2html*). The *packages* folder of *common-src* contains Expat, a utility for parsing XML, and GDBM, a standard database manager program. The *packages* folder of *runtime-src* contains software related to the Apache Web server (*apache-httpd* in Figure 11.1) and a package dealing with the Z39.50 protocol (*yaz*). Each package is stored in a folder of its own, with a *readme* file that gives more information about the package. Resulting executable programs are placed in *bin*—in the sub-folder corresponding to their underlying operating system—when the software is compiled.

Full-text indexing is central to Greenstone. There are three alternative indexers, called MG, MGPP, and Lucene; near the end of Section 11.4 we show how to switch between them within the Librarian interface. They are stored in a top-level folder of the *common-src* folder called *indexers* (not shown). Within this is a *packages* folder that contains any third-party software that the indexers use.

The *mappings* folder contains Unicode translation tables. The *etc* folder holds configuration files for the system. It also includes initialization and error logs, and the user authorization database. Inside the top-level *web* folder, *images* stores images for the user interface, among them icons like those shown in Table 10.1, and *style* stores the cascading style sheet (CSS) files. The user interface is constructed by small code fragments called *macros*, and these are placed in the *macros* folder. Depending on the type of installation and how it has been configured you may also have the following top-level folders: *apache-httpd* which contains a Web server, *gli* which contains the Librarian interface, *tmp* for storing temporary files, *docs* which contains the documentation for the system, and *packages* which contains a Java runtime environment.

The Librarian interface maintains a small amount of information that is specific to each particular user. This includes the user's Preferences (accessed from its File menu) and the cache that is used when downloading (see Section 11.5). You will find this information in *C:\Documents and Settings\<username>\Application Data\Greenstone\GLI* on Windows XP and *C:\Users\<username>\Application Data\Greenstone\GLI* on Windows Vista. (This is for Windows; on Mac and Unix systems it is in a folder called *.gli* within the user's home directory.)

Collections

Each collection corresponds to a sub-folder of *collect*. Collections are completely self-contained. For example, if you have created a collection—perhaps one of the examples in Chapter 10—and you want to give it to someone else who is also running Greenstone, just locate the appropriate *collect* sub-folder, put it on a USB flash drive, take it to their computer, and transfer it into their *collect* folder. It will appear right away on their Greenstone home page. (Windows Local Library users will have to restart Greenstone first.)

You may also need to move collections around if the software has previously been installed in a non-standard place. Old collections can be transferred to the new installation by moving them from the *collect* folder in the old place into the folder *GSDLHOME\collect*. (We use Windows terminology because this is the most popular platform for Greenstone in practice.)

Figure 11.1 shows the structure of the Demo collection—it's the same for all collections. The *import* folder is where the original source material is placed, and the *archives* folder is where the result of

the import process goes (the first stage of building a collection). The *building* folder is used temporarily during building (the second stage), whereupon its contents are moved into *index*. The *index* folder is where the result of the entire building process is placed, and contains all the information that is served to readers. The *etc* folder contains miscellaneous files, such as configuration and mapping files, and logging information. The metadata sets used by the collection go into a folder of the same name, as do any log files that are written every time the collection is built. Some collections have additional folders: one for images that are used in the collection, another for collection-specific macros, and a third for any special Perl modules that pertain to this collection.

Some of these folders may be absent. Most can be deleted once the collection is built; all the information required to serve the collection is in *index* and *etc* (and, if present, *images* and *macros*). However, then the collection could not be rebuilt—nor loaded into the Librarian interface. [Section 11.3](#) describes the building process in more detail and also gives further information about the role of the various folders.

Just as the top-level *etc* folder in the Greenstone file structure holds configuration information for the system as a whole, there is an *etc* folder for each collection. Configuration information for the collection is recorded in a file there called *collect.cfg*, along with other miscellaneous information. It records the result of the collection design and formatting process—indeed the details displayed in the main panel of the Librarian interface about a collection’s plug-ins, indexes, classifiers, and format statements are exactly what make up this file. It is plain text: choose any collection and take a look at its *etc\collect.cfg*.

Greenstone CD-ROM/DVDs

You can give out a Greenstone collection on a USB flash drive, or even as a zipped e-mail attachment containing the *etc* and *index* folders if the collection is small. Of course, the recipient must be running the Greenstone software in order to make use of it.

Alternatively, collections can also be published as *self-installing* Windows CD-ROMs or DVDs. These are disks that begin the installation process as soon as they are placed in the drive. They do not install the full Greenstone software, just a mini version that allows users to view existing collections but not build new ones. An installation option lets you choose whether to install all the collection files onto your computer disk, or just the software, in which case the CD-ROM/DVD must be present in the disk drive whenever the collections are used. The former option takes more time to install but responds more quickly to the reader’s requests. Either way, interaction—including all browsing and searching, changing preferences, switching languages, etc.—is just the same as on the Web, except that response times are more consistent.

It is very easy to create a self-installing disk containing your own collections. Enter the Librarian interface and choose **File→Write CD/DVD image**. Select the collections you wish to export by ticking their check boxes in the window that pops up. If you enter a name for the disk it will appear in the menu when the CD-ROM/DVD is run, otherwise it will be called “Greenstone Collections.” You can choose whether the CD-ROM/DVD runs directly from the disk drive or installs some files onto the computer first.

Click **Write CD/DVD image** to start the export process. This puts files into a temporary folder called *exported_xxx* (or some such); the interface tells you where it is. The process involves copying many files and may take a few minutes. You need to use your own computer's software to write the generated files to CD-ROM/DVD. On modern computers this capability is built into the operating system: just insert a blank disk into the drive and drag the contents of *exported_xxx* into the folder that represents the disk. It is equally simple to plug in your USB flash drive or portable media player and use it in disk-mode.

Collections installed from prepackaged Greenstone disks do not reside in the standard *collect* folder but in *C:\GSDL\collect*. To amalgamate them with your main Greenstone installation, move them into *GSDLHOME\collect*. After you've done this, the mini version of the software that runs the prepackaged collections is no longer necessary: you can uninstall it from the Greenstone section of the Windows Start menu.

11.2 Operational Aspects

Certain aspects of a digital library site need to be determined individually for each installation. When Greenstone is installed, default values are given to many parameters. Some, like the directory where the software is kept and the HTTP address of key folders, define the whereabouts of the system—what logical space it occupies. Others control what users can do with the system—such as the languages that the user interface makes available—and switch on various components. You need to know what facilities exist and how to turn them on and off.

Whether logs of user activity are kept and whether Web browser cookies are used to identify users are other important operational aspects. Greenstone has a full logging capability, but it is switched off by default to avoid the creation of large, growing files. In many environments it is essential to record usage logs in order to justify the existence of the system.

Greenstone includes a Web-based administration facility that gives information about the entire system, including all collections it offers. This facility is off by default. Perhaps its most useful function is to allow the system administrator to define users and groups of users who are allowed different privileges—such as access to protected collections and documents within collections. Password protection is just one of many techniques used to bolster security when operating in a network environment, especially if the computer is connected to the Internet. Depending on the level of sensitivity of the digital library content, you may wish to consider using additional measures, such as protecting the computer behind a firewall and streaming all communication over HTTPS rather than HTTP.

Configuration files

There are two configuration files: a site configuration file (*gsdlsite.cfg*), which is found in Greenstone's *cgi-bin* folder, and a main configuration file (*main.cfg*), found in the top-level *etc* folder. Both are ordinary text files that can be edited to tailor the individual installation. Moreover, they contain extensive comments that describe what the options are and how to use them.

Once the software has been installed and is working, you are unlikely to need to change the site configuration file—unless you move all the files or change HTTP addresses. The options available in the main configuration file are more interesting. We will not go into details here—you can easily look at the actual file itself for this information—but here is a synopsis of what can be done:

- Log all usage (see next subsection).
- Use cookies to identify users in the logs.
- Enable the administration facility (see next subsection).
- Enable the institutional repository component (see [Section 11.6](#)).
- Enable the Remote Librarian applet (see [Section 11.6](#)).
- Select which languages are enabled in the user interface.
- Determine the encodings to be used for the user interface.
- Set defaults for built-in CGI arguments—for example, the default interface language.

The best way to learn about the configuration options is to experiment with the *main.cfg* file itself. Changes take effect immediately unless you are using the Windows Local Library version (the default for Windows), in which case the server must be restarted before configuration changes take effect.

Encoding statements specify different types of character encoding that can be selected. The UTF-8 version of Unicode (see Sections 4.1 and 8.2), which has standard ASCII as a subset, is handled internally and should always be enabled. But there are many other possible encodings—for example, traditional Chinese text is often represented in “Big-5.” The main configuration file specifies many encodings; most are disabled but can be restored by removing the comment character (#). The main configuration file also contains detailed documentation about the structure of encoding statements.

Logging

Three kinds of log are maintained in the *etc* folder: a usage log, an error log, and an events log. The first is the most interesting. The error log, which is permanently enabled, contains messages relating to initialization and operational errors: it is of interest only to people maintaining the software. The events log relates to an obsolete subsystem and will not be discussed.

All user activity—every page that each user visits—can be recorded in the usage log (*etc\usage.txt*), although no personal names are included. Each action is effectively defined by the arguments in the URL (“CGI arguments”) that characterize it, and these are what are logged. Disabled by default, logging is enabled by switches: one switch turns logging on and off, and another assigns unique identification codes (cookies) to users, which enables their interactions to be traced through the log file.

Each line in the log file records a single page visit. Entries have a time-stamp, the address of the user’s computer, details about Web browser used, and the arguments that the CGI mechanism transmits to Greenstone. The main configuration file also includes a switch that sets the format used for

```

/cgi-bin/library.cgi
its-proxy1.massey.ac.nz
Fri Oct 17 15:57:28 NZDT 2008
a=q, c=demo, l=en, m=50, o=20, p=search, q=snail farming, w=utf-8, z=130.123.128.86-950647871
Mozilla/5.0 [en-US] Firefox/3.0.3

```

```

Library program
User's computer
Time stamp
Arguments
User's browser

```

Figure 11.2: Entry in the usage log

the time-stamp: local time in the format “Fri Oct 17 15:57:28 NZDT 2008,” Greenwich Mean Time (UTC) in the same format, or an integer representing the number of seconds since 01/01/1970 GMT.

Figure 11.2 shows a sample entry, split into these components. On 17 Oct 2008 a user at *massey.ac.nz* displayed a page (action *a=p*) that is the home page (page *p=home*) of the Māori newspaper collection (collection *c=niupepa*). Many of the other arguments have default values—for example, the language is English (*l=en*) and 20 search results will be displayed per page (*o=20*). The user’s browser is Firefox. The last argument, *z*, is a cookie generated by the user’s browser: it contains the computer’s IP number followed by the time that the user first accessed the digital library. (The *z* argument appears only if cookies are enabled in Greenstone.)

When logging is enabled, every action by every user is logged—even the Web pages generated to inspect the log files.

Administration facility

Greenstone’s administration pages display the installation’s configuration files and allow them to be modified. They let you examine the log that records usage and the log that records internal errors. They are available over the Web, so you can use them anywhere. However, the facilities are rudimentary, because Web forms are used for interaction. If you need to edit or examine these files, it is probably best to log into the computer that is serving Greenstone and work with an ordinary text editor. If the hyperlinked button *Administration Page* does not appear on the home page (beneath the available collections), then you must edit the main configuration file *main.cfg* to enable the administration facility—simply locate the appropriate line (search for *Administration*) and change the status value from *disabled* to *enabled*.

The facility is most useful when you need to define user groups with different privileges. For example, it is possible to restrict access to certain documents, or to certain collections, to particular users. Also, the Remote Librarian interface authenticates users before allowing them to alter the structure and content of particular collections, as does the institutional repository facility (both are described in Section 11.6). Of course, the ability to define new users and user groups is restricted to people who have been authorized to act as system administrator.

Authentication

When the Greenstone software is installed, there is the option to create a user called *admin* and set its password. Since this means that a malicious user (from anywhere in the world) who has been able to crack the password has the potential to wreak havoc with your digital library, we advise caution in

using this feature. This is also why the feature is off by default in the installer, requiring a conscious decision by the person installing the software to activate it.

In order to investigate the authentication scheme, enable it by editing the main configuration file, go to the Greenstone home page, and refresh the browser window or restart the Local Library server if that is what you are using. A new line appears on the home page, beneath the collections, that refers to the administration facility. Click the button that leads to the Administration page.

On the left of this page are menu items for configuration files, logs, user management, and technical information. User management is the most useful: it allows you to list users, add new users, and change your password. If you attempt any of these, you will need to sign in as the admin user. If not set at installation time, the default password is *admin*.

Each user can belong to any number of groups. When the software is installed, there are three groups: *administrator*, *all-collections-editor*, and *demo*. Members of the first group can add and remove users and change their groups; the second group is connected with the Remote Librarian facility discussed in [Section 11.6](#); while the third group is mentioned in the next section. Groups are simply text strings, and you can add them at will: just type them into the “groups” box associated with each user. The admin user can also disable users if they misbehave. Information about users, passwords, and groups is recorded in a database in the Greenstone file structure (*etc/users.gdb*); passwords are encrypted using the well-known *crypt* utility.

Protecting a collection

It is sometimes necessary to protect digital library collections, or certain parts of them, from the public eye. For example, it may be necessary to restrict PDF files to use only within an organization but allow open access to the extracted HTML, or to keep images private but provide open access to thumbnails. The safest way to do this is to use the authentication facilities of your Web server. Most popular Web servers (e.g., Apache) can be configured to protect parts of the file system so that private files placed there cannot be accessed unless users authenticate themselves first.

Greenstone’s authentication scheme is another way of controlling access to particular documents or collections. While less powerful than the capabilities provided by a Web server like Apache, it does have the advantage that it is much easier to learn. Authentication works in two stages. First determine what to restrict access to; second, if access is to be restricted, determine which users are to have it. Access can be restricted either to the collection as a whole or to individual documents in it. In the latter case, the documents are specified individually.

The Librarian interface does not yet provide this ability; instead, you have to edit the relevant collection’s configuration file manually (*collect.cfg* in the collection’s *etc* folder). Authentication is activated by a line that begins with the directive *authenticate* and is followed by *collection* or *document* depending on whether it applies to the full collection or to individual documents (the default is *collection*). If authentication is by document, you can specify a list of private documents or a list of public documents. In the former case, all other documents are public; in the latter, all others are private. This is done using directives *private_documents* or *public_documents*—use one or the other, but not both. The documents themselves are specified using identifiers separated by spaces.

```
# Authentication directives

authenticate document
public_documents HASH010f073f22033181e206d3b7 HASHac0a04dd14571c60d7fbfd

auth_group demo
```

Figure 11.3: Authentication lines from a collection configuration file



Figure 11.4: Authentication in Greenstone

Figure 11.3 shows an example of the per-document authentication. It uses Greenstone document identifiers, but other identifiers can be specified when building the collection (see Section 11.4). The easiest way to determine the identifier for a given document is to locate it in the collection and look at the *d* argument in its Greenstone URL.

The second part of the process uses the *auth_groups* directive to specify the groups that are permitted access. It is followed by a group name (or a list of group names separated by spaces). The lines from the collection configuration file in Figure 11.3 restrict access to all documents except two to members of the *demo* group; those two documents are public. As noted above, you can define groups and add members to them from the administration pages.

When users try to access a protected document, they are asked for a user name and password as shown in Figure 11.4. This screenshot, and Figure 11.3, are taken from the Authentication Demo collection, available at nzdl.org. Specifying access control this way is a clunky feature that is little used in practice. Instead of having to specify an explicit list of document identifiers, it would be better to control authentication using a metadata value, and this facility is planned for a future version of the Librarian interface.

11.3 Command-Line Operation

Collections can be built without involving the Librarian interface at all. It simply provides a user-friendly way of working. The Gather panel copies documents into the appropriate *import* folder. The Enrich panel edits metadata files (*metadata.xml*, described in Section 10.5). The Design and Format panels edit the collection configuration file *collect.cfg*. Most importantly, the Create panel invokes standalone programs that do the actual work.

Walking through the operations involved in building a collection without using the Librarian interface at all will show you what is going on behind the scenes. We use the Small HTML collection that we built in Section 10.3 as an example, and on the way we take the opportunity to explain some general features and design principles, as well as the specific steps involved.

Building a collection is the process of taking a set of documents and metadata and creating all the indexes and data structures that support whatever searching, browsing, and viewing operations the collection offers. It breaks down into four phases. First, a skeleton framework is made for the collection. Then documents are imported into a standard representation from whatever format they are provided in. Next the required indexes are built, and finally the collection is installed, so that it becomes operational.

We refer to these operations as *make*, *import*, *build*, and *install*, respectively, and each one is performed by a simple computer command. The terminology is potentially confusing: we say *make* because the English language does not have a specific verb for creating a skeleton framework, and—worse still—*build* is used in two different senses, one that encompasses the whole process and another that refers to the particular sub-process of building indexes. Provided the distinction is kept in mind, it should be clear from the context whether *build* refers to the general process of building-in-the-large or the specific one of building-in-the-small.

Getting started

We recommend that you follow this walk-through on your own computer. We use Windows terminology, but the process for Mac and Linux is nearly identical. Some operations may seem unnecessary, but their role becomes clear later. Remember that our purpose is not to provide a streamlined way of building collections—for that, use the Librarian interface—but to explain the collection-building process. Table 11.1 summarizes the procedure, for reference, and also mentions any differences between Windows and Mac/Linux.

First locate the *command prompt*, which is where you type commands. This differs from one system to another, but on Windows XP look in the Start menu. Invoke the Run entry and type *cmd* in the dialog box. Change into the directory in which the software was installed by typing

```
cd "..."
```

where ... is the actual installation folder, which we have been calling GSDLHOME. (The quotation marks are there to protect any spaces in the folder name, which is necessary on some systems.) Next, type

Table 11.1: The collection-building process

Step	Function
1. <code>cd "C:\Program Files\Greenstone"</code>	Assumes that Greenstone is installed in the default location.
2. <code>setup.bat</code>	This makes Greenstone programs available. On Mac or Linux, use <i>source ./setup.bash</i> instead.
3. <code>perl -S mkcol.pl -creator me@cs.waikato.ac.nz mydemo</code>	Create a skeleton framework of initial files and directories.
4. Copy source files into <code>C:\Program Files\Greenstone\collect\mydemo\import</code>	Populate the collection with sample documents. On Windows, select the files and drag them. On Mac or Linux use the <i>cp</i> command—and if you are copying files from a CD-ROM, you may have to use the <i>mount</i> command first.
5. Edit the file <code>C:\Program Files\Greenstone\collect\mydemo\etc\collect.cfg</code>	Customize the collection by editing the collection-level metadata in the configuration file. Alter <i>collectionname</i> , <i>collectionextra</i> , and <i>collectionicon</i> .
6. <code>perl -S import.pl -removeold mydemo</code>	Convert the source documents and metadata specifications to the Greenstone standard form.
7. <code>perl -S buildcol.pl -removeold mydemo</code>	Build the indexes and data structures that make the collection work.
8. Replace the contents of the collection's <i>index</i> directory with that of the <i>building</i> directory	On Windows, select the files and drag them. On Mac or Linux, use the <i>mv</i> command.

```
setup.bat
```

This batch file (which is quite short—read it if you like) tells the system where to look for programs and other parts of the digital library file structure by setting the system variable *GSDLHOME* to the Greenstone home directory. To return to this place later, type

```
cd "%GSDLHOME%"
```

(again, the quotation marks are there to protect spaces in the file name). If you close the command window and open another one, you must invoke *setup.bat* again.

Making a framework

Now you are in a position to make, import, build, and install collections. The first operation, accomplished by the Perl program *mkcol.pl*—the name stands for “make a collection”—creates an empty framework. Program names have cryptic abbreviations because Greenstone has traditionally run on ancient versions of Windows that impose an eight-character limit on file and folder names.

Run the program by typing *perl -S mkcol.pl*. (If your computer is set up to associate the Perl interpreter with files ending in *.pl*, drop the preamble and simply type *mkcol.pl*.) All Greenstone programs take at least one argument—the name of the collection being operated on—and running them without arguments prints a helpful message on the screen. This convention has the added bonus of providing up-to-date documentation for an evolving system.

As this message explains, *mkcol* requires you to specify the collection name. As for the other programs, there is an extensive list of options, which are preceded by the minus sign (`-`), but all have default values so that only a minimum of information needs to be given explicitly.

Use *mkcol.pl* to create a framework of initial files and folders for the new collection. Assign the collection the name *mydemo* by typing

```
perl -S mkcol.pl mydemo
```

To examine the new file structure, move to the newly created collection directory by typing:

```
cd "%GSDLHOME%\collect\mydemo"
```

List the directory's contents by typing *dir*. The *mkcol.pl* program has created six folders: *etc* (which contains the default collection configuration file), *images* (for any collection specific images), *import* (ready for the collection's source material), *macros* (containing a default collection-specific macro file), *script* (for any Web browser JavaScript enhancements), and *style* (for any collection-specific CSS files). The other files depicted in [Figure 11.1](#) for the Demo collection are created automatically later, when they are needed.

In the new collection's *etc* directory is a collection configuration file called *collect.cfg*, shown in [Figure 11.5](#). The collection name appears in one of the *collectionmeta* lines, which give metadata concerning the collection as a whole. The file shows the same selection of plug-ins that the Librarian interface includes—not surprisingly, because behind the scenes the Librarian invokes exactly the same program.

Importing documents

The next step is to populate the collection with documents. In our case, the source material resides in the *simple_html* folder in the sample files that you downloaded when working through Chapter 10. All the source material should be placed in the new collection's *import* folder. Just copy the *simple_html* folder (or its contents—it doesn't matter which) and paste it into the *mydemo* collection's *import* folder. This is precisely what the Librarian interface does in the Gather panel.

Now you are ready to perform the import process. This brings the documents into the system, standardizes their format, and extracts metadata from them and from any *metadata.xml* files that are present. It invokes plug-ins to process the files and extract metadata.

Type *perl -S import.pl* at the prompt to get a long list of options for the import program, with a brief explanation of each. In fact, if you switch the Librarian interface to Expert mode and go to the Create panel, you will find a list of import options that includes all those implemented by the Perl program—and the tool-tips contain the same explanatory text too. (A few that do not make sense from within the Librarian interface are omitted.) When the build button is pressed, the Librarian interface initiates *import.pl* (followed by *buildcol.pl*, described next) to do the work, with the specified set of options.

Next type

```
perl -S import.pl -removeold mydemo
```

```

public      true
buildtype   mgpp

indexes      text dc.Title,ex.Title Source
defaultindex text
levels       document

plugin       ZIPPlugin
plugin       GreenstoneXMLPlugin
plugin       TextPlugin
plugin       HTMLPlugin
plugin       EmailPlugin
plugin       PDFPlugin
plugin       RTFPlugin
plugin       WordPlugin
plugin       PostScriptPlugin
plugin       PowerPointPlugin
plugin       ExcelPlugin
plugin       ImagePlugin
plugin       ISISPlugin
plugin       NulPlugin
plugin       MetadataXMLPlugin
plugin       ArchivesInfpPlugin
plugin       DirectoryPlugin

classify     List -metadata dc.Title;ex.Title -partition_type_within_level constant_size
classify     List -metadata Source -partition_type_within_level constant_size

format VList "<td valign=\"top\">[link][icon]/[link]</td>
  <td valign=\"top\">[ex.srclink]{Or}{[ex.thumbicon],[ex.srcicon]}[ex./srclink]</td>
  <td valign=\"top\">[highlight]
  {Or}{[dc.Title],[ex.Title],[ex.Title],Untitled}
  [/highlight]{If}{[ex.Source],<br><i>([ex.Source])</i></td>"

format HList "[link][highlight][ex.Title]/[highlight]/[link]"
format DocumentHeading "{Or}{[parent(Top):ex.Title],[ex.Title],untitled}<br>"
format DocumentText "[Text]"
format DocumentButtons "Detach|Highlight"
format SearchTypes "plain,form"

collectionmeta collectionname [l=en] "test collection"
collectionmeta .text [l=en] "text"
collectionmeta .dc.Title,ex.Title [l=en] "titles"
collectionmeta .Source [l=en] "filenames"
collectionmeta .document [l=en] "document"
collectionmeta .section [l=en] "section"

```

Figure 11.5: Collection configuration file created by *mkcol.pl*

Text scrolls past, reporting the progress of the import operation, just as it does when you use the Librarian interface. You do not have to be in any particular folder when the *import* command is issued because the software works out where everything is from the Greenstone home folder and the collection's name. The *-removeold* option forces the collection to be built from scratch. Greenstone also supports incremental building (described below). For now, however, we rebuild collections in their entirety to keep things simple—in fact, this is common practice.

Building indexes

The next step is to build the indexes and data structures that make the collection work. This is building-in-the-small (as opposed to building-in-the-large, which refers to the whole process of making, importing, building, and installing). With a small nod toward the ambiguity of the term *build*, the relevant program is called *buildcol.pl*.

This is the stage at which you would most likely customize the new collection by editing its configuration file, as you have done in the Librarian interface’s Design and Format panels. However, just as in the first exercise of Section 10.3, we’ll go straight ahead and “build” the collection.

First type *perl -S buildcol.pl* at the command prompt for a list of collection-building options (again, a superset of those available to Expert users of the Librarian interface). Then, sticking to the defaults, type

```
perl -S buildcol.pl -removeold mydemo
```

Progress-report text scrolls past again, which under normal conditions can be ignored. (Any serious problem will cause the program to be terminated immediately, with an error message.) The *-removeold* option causes the indexes to be built from scratch.

Installing the collection

Although it has been built, the collection is not yet live—you cannot see it in your digital library. When the *buildcol.pl* command is used, the files that are generated are located in a special area, and the result must be moved to the proper place before the collection can be used. This is because (once you scale up your digital library operations) some collections may take hours—even days—to build, and during that period the existing version of the collection continues to serve users. Building is done in the *building* folder, whereas collections are served from the *index* folder.

To make the collection operational, select the contents of the *mydemo* collection’s *building* folder and drag them into the *index* folder. If *index* already contains some files, remove them first.

The newly built collection can now be invoked from the digital library home page. If you are using the Local Library version, you will have to restart the library program. Otherwise you need only reload the home page (although caching may conceal the change, in which case a forced reload should be sufficient; alternatively you could close and restart the browser). To view the new collection, click on its icon. If it doesn’t appear on the home page, you’ve probably forgotten to move the contents of the *building* folder into *index*.

With the Web Library version, nothing needs to be restarted. What happens if a reader is actually using the previous version of the collection at the very instant the collection is moved from *building* to *index*? Basically, nothing. If she has just done a search, and then repeats it, the results list may change. The worst that can happen is that she clicks on a document in the search results that is absent from the new version of the collection. Then she will see a blank Greenstone page and, after she re-executes the search, the document will have disappeared from the search results.

11.4 Under the Hood

Now that you know how to invoke the underlying programs that build collections, you are in a position to learn a little more about what happens under the hood. This information could prove useful if you need to dig deeper into the structure of the *archives* folder, or to evaluate the pros and cons of different ways of choosing document identifiers, or to learn a little more about plug-ins and what they do, or to understand the many options that are available for searching.

Importing and building

The two main components of the collection-building process are importing (*import.pl*) and building in the narrow sense (*buildcol.pl*). The former brings documents and metadata into the system in a standardized XML format that is used internally. The latter creates the indexes and data structures needed to make the collection operational. Both components have many options, which can be viewed from the command line or in the Create panel in Expert mode.

The import process takes the information in the *import* folder (including *metadata.xml* files, if any), converts it to a standardized XML format (see next section), and puts the result into the *archives* folder. If desired, the original material can then be safely deleted, because the collection can be rebuilt from the archive files. If it is deleted, new material can be added to the collection by placing it in *import* and re-executing the process, this time using *-keepold* instead of *-removeold*: the new material will find its way into *archives* along with what is already there. If following the route of removing source documents from the import folder once they have been processed, then to retain a collection in a form that can be augmented and rebuilt later, do not delete the *archives*.

Each document's archive file occupies its own folder in the archives structure, along with any files associated with the document—such as image files included in it. The folder name is based on the document's object identifier. The structure is rather arcane, because it is designed to work on primitive computers (e.g., early versions of Windows) that restrict the length of file names, the number of files in a folder, and the maximum nesting depth of folders.

You will also notice two files at the top level of the archives folder: *archiveinf-doc.gdb* and *archiveinf-src.gdb*. These are database files that support incremental building. The former stores information about where each document is located within *archives* (and what files comprise the document); the latter records where these files came from in the *import* folder.

Incremental building

Documents can be added, removed, or changed incrementally using the *incremental* option of the import and build processes. In the Librarian interface, this is enabled by clicking the **Minimal Rebuild** check-box in the Create panel. Although the import process can always work incrementally, whether or not the building process can do so depends on the indexer being used. The indexers MG and MGPP are non-incremental, but the Lucene indexer supports true incremental building, and both building and importing are performed incrementally if the Librarian interface's Minimal Rebuild is enabled.

When incrementally building with Lucene, the files generated should no longer be placed in a *separate* area—the indexer needs to be able to find the current indexes and change them incrementally. In other words, there is no “install” step to perform. This can be accomplished by adding the option

```
-builddir "%GSDLHOME%\collect\

```

to the *buildcol.pl* command (as well as *-incremental*), substituting the actual name of the collection for *<collectionname>*. As this is rather tedious to type, Greenstone provides two extra commands: *incremental-import.pl* and *incremental-buildcol.pl*, which set these options for you. A complementary pair, *full-import.pl* and *full-buildcol.pl*, are provided for rebuilding from scratch.

A long-standing historical deficiency of Greenstone, now rectified, has been that the *build* process needed to be re-run from scratch when documents are added, modified, or deleted. This limitation arose because the original MG and MGPP indexers are optimized for maximum compression, which requires non-incremental operation to ensure that the word statistics on which compression is based reflect the collection as a whole. However, Lucene, which Greenstone now incorporates as an option, is capable of operating incrementally.

Scheduled rebuilding

Greenstone incorporates a user-oriented module for scheduled maintenance of collections. This automates the construction of any existing collection and schedules rebuilding to occur periodically. At the command-line level, daily rebuilding of a collection called *pics* is accomplished by invoking

```
perl -S schedule.pl pics "incremental-import.pl pics"
      "full-buildcol.pl pics" daily
```

for MG- and MGPP-based collections, and

```
perl -S schedule.pl pics "incremental-import.pl pics"
      "incremental-buildcol.pl pics" daily
```

for a Lucene-based collection.

This generates a script for rebuilding according to the specified options. It also inserts a record into a configuration file for time-based scheduling that calls the generated script using an operating system service called *cron* (for chronograph), set to be executed daily. (Cron is standard on Unix systems: Greenstone includes a port to Windows to make scheduled rebuilding work there too.) It is important that scheduled builds are completed in their entirety without interference from another scheduled build. To handle this, the Perl script first checks for the presence of a lock file, which indicates that a build is already underway, and ensures that multiple builds do not occur concurrently.

Scheduled rebuilding can be specified within the Librarian interface. In Expert mode, in addition to Import options and Build options, the Create panel contains a Schedule Options tab where scheduling parameters can be specified. Collections can be built hourly (on the hour), daily (at midnight), or weekly (at midnight on Sundays), and the user can request e-mail notification of each build. The output of the script is kept in a log file. Before setting up an automatic scheduled rebuild, the user

can manually build and configure the collection as many times as necessary to confirm that the correct sequence is being performed.

Archive formats

When you import documents and Greenstone puts them in the *archives* folder, there are two standard forms in which they can be represented:

- Greenstone Archive format
- METS (see Section 6.4), using the Greenstone METS profile.

An option to the import process (*saveas*), which is only visible in the Librarian interface's Expert mode, dictates which one is used; the former is the default.

The building (building-in-the-small) process uses plug-ins to process documents just as the import process does, and for this to work, the list of plug-ins must include one that processes whatever archive format is being used. This is why the Greenstone Archive format plug-in (called GreenstoneXMLPlugin) is specified at the top of the list, and it cannot be deleted by Librarian-level users. However, Expert-level users can specify METS as the save format for the import process and replace GreenstoneXMLPlugin in the plug-in list by Greenstone's METS plug-in.

Using METS is a worthwhile experiment. Open any collection in the Librarian interface and switch to Expert mode. In the Create panel, change the import process's *saveas* option to Greenstone METS, and in the Design panel delete GreenstoneXMLPlugin and replace it by GreenstoneMETSPugin. Build the collection, and locate its *archives* folder in the Windows file browser (in *Greenstone→collect→<collection name>→archives*). Two files are generated for each document: *docmets.xml*, the core METS description; and *doctxt.xml*, a supporting file. (Depending on how you view *doctxt.xml*, you may need to be connected to the Internet, because it refers to a remote resource.) Depending on the source documents, there may be additional files, such as images used in a Web page. One of the many features of METS is the ability to reference information in external XML files. This is used to tie the content of the document, which is stored in the file *doctxt.xml*, to its hierarchical structure, which is described in the core METS file *docmets.xml*.

The remainder of this section describes the Greenstone Archive format, which is used by default. (Readers may first wish to reacquaint themselves with Section 4.3, which reviews HTML and XML.) Documents are divided into sections and metadata is stored at the document or section level. One design requirement is the ability to represent any previously marked-up document that uses HTML tags, even if the markup is sloppy. Another is that archive documents must be parsed rapidly. The archive format is an XML-compliant syntax that contains explicit top-level markup for sectioning and metadata and can also embed HTML-style markup that is not interpreted at the top level.

In XML, tags are enclosed in angle brackets for markup, just like HTML tags. The archive format encodes documents that are already in HTML by escaping any embedded left or right angle bracket (<, >), or quote (") characters within the original text using the standard codes *<*, *>*, and *"*. A *<Section>* tag signals the start of each section of the document, and the corresponding closing tag marks its end. Sections begin with a block that defines pertinent metadata. Metadata

specifications give the metadata name and its value. In addition to regular metadata, the file that contains the original document is specified as *gsdlsourcefilename*, and files that are associated with the document, such as image files, are specified as *gsdlassocfile*.

Figure 11.6a shows the XML document type definition for the Greenstone Archive format. Documents are divided into sections, which can be nested. Each section has a description part that comprises zero or more metadata items, and a content part that holds the document's contents. (The content may be null, e.g., for image files, audio files, or the dummy documents that are created by exploding metadata-only files, as described in Section 10.5). A name attribute and some textual data are associated with each metadata element. In XML, PCDATA stands for "parsed character data"—Unicode text in this case.

Figure 11.6b shows a simple document comprising a short book with two associated images. The book has two sections, called *Preface* and *First and only chapter*; the latter has two subsections. Chapters are simply top-level sections. Metadata is stored at the beginning of each section.

This structure serves two purposes. First, it allows readers to browse around inside documents once they have been located. When you open a book, the table of contents shows the section hierarchy. Figure 3.1 (Chapter 3) illustrates browsing within a book that has a hierarchical table of contents showing chapters, sections, and subsections. In some collections, documents are split into pages instead, and Figure 3.2 shows (at the top right) a page selector for such a document. Chapters, sections, subsections, and pages are all "sections."

The second use of document structure is for searchable indexes. There are two levels of index: *document* and *section*, and most collections use them both. The first relates to complete documents—you use it to find all documents that contain a particular set of words. When a *section* index is created, each portion of text that is indexed stretches from one *Section* tag to the next—thus a chapter that immediately begins with a new section will produce an empty document in the index (such documents are hidden from the user). Sections and subsections are treated alike; the hierarchical structure is flattened for the purposes of creating searchable indexes. The MG indexer (but not MGPP or Lucene) provides a third level, *paragraph*, which treats each paragraph as a separate document and is useful for more focused searches.

Document identifiers

The import process assigns object identifiers (OIDs) to documents, which are then stored as a metadata element in the document's archive file (Figure 11.6b). If the import process is re-executed, documents should receive the same identifier. The method of obtaining identifiers can be specified as an option to the import process (this option is available to Librarian-level users). There are four possibilities:

- hash the contents of the file
- use the value of a particular metadata item
- use a simple document count
- use the name of the parent folder.

(a)

```

<!DOCTYPE Archive [
  <!ELEMENT Archive (Section*)>
  <!ELEMENT Section (Description,Content,Section*)>
  <!ELEMENT Description (Metadata*)>
  <!ELEMENT Content (#PCDATA)>
  <!ELEMENT Metadata (#PCDATA)>
  <ATTLIST Metadata name CDATA #REQUIRED>
]>

```

(b)

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE Archive SYSTEM "http://greenstone.org/dtd/Archive/1.0/Archive.dtd">
<Archive>
  <Section>
    <Description>
      <Metadata name="gsdlsourcefilename">ec158e.txt</Metadata>
      <Metadata name="dls.Title">Freshwater Resources in Arid Lands</Metadata>
      <Metadata name="Identifier">HASH0158f56086efffe592636058</Metadata>
      <Metadata name="gsdlassocfile">cover.jpg:image/jpeg:</Metadata>
      <Metadata name="gsdlassocfile">p07a.png:image/png:</Metadata>
    </Description>
    <Content>
      This is the text of the preface
    </Content>
  </Section>
  <Section>
    <Description>
      <Metadata name="dls.Title">Preface</Metadata>
    </Description>
    <Content>
      This is the text of the preface
    </Content>
  </Section>
  <Section>
    <Description>
      <Metadata name="dls.Title">First and only chapter</Metadata>
    </Description>
    <Content>
      This is the first part of the first and only chapter
    </Content>
  </Section>
  <Section>
    <Description>
      <Metadata name="dls.Title">Part 1</Metadata>
    </Description>
    <Content>
      This is the first part of the first and only chapter
    </Content>
  </Section>
  <Section>
    <Description>
      <Metadata name="dls.Title">Part 2</Metadata>
    </Description>
    <Content>
      This is the second part of the first and only chapter
    </Content>
  </Section>
</Archive>

```

Figure 11.6: Greenstone archive format: (a) document type definition (DTD); (b) example document

The first, which is the default method, calculates a pseudo-random number based on the content of the document—called *hashing*. This ensures that identifiers will be the same every time a given document is imported: if the content changes, so does the identifier; if not, the identifier remains the same. Identical copies of a document will be assigned the same identifier and thereafter treated by the system as one. The same document can appear in two different collections: if so, searching both collections will return just one copy of the document. These identifiers are character strings starting with the letters HASH: for example, HASH0158f56086efffe592636058. They are long enough that the chance of different documents receiving the same one is vanishingly small, and this possibility is ignored in the system.

For the second method of assigning OIDs, the user specifies a metadata element that holds the document's unique identifier (havoc will ensue if it is not unique). If that metadata value is unspecified for a particular document, the hash value will be used instead. (The value is preceded by *D* if the identifier is purely numeric, to prevent confusion with document numbers, used internally). The third method is significantly faster than hashing, but does not necessarily assign the same identifier to the same document when the collection is rebuilt from scratch with additional documents. The fourth is intended for situations where there is only one document per folder, and folder names are unique (the folder name is preceded by *J*).

Identifiers are extended to individual sections of a document using integers separated by periods. For example, since the OID of the document in [Figure 11.6b](#) is HASH015...058, the OID of the first subsection of the *First and only chapter* is HASH015...058.2.1—because that chapter is the second in its enclosing section, and the relevant subsection is the first in its enclosing section. Section-level identifiers are not stored explicitly but are used internally to represent individual document sections that are returned as the result of a search. They do not necessarily coincide with the logical numbering of chapters and sections—documents often include unnumbered chapters, such as a Preface—but are only for internal use.

Plug-ins

Most of the import process's work is accomplished by plug-ins. These operate in the order in which they appear in the collection's configuration file. Each input file is passed to the plug-ins in turn until one is found that can process it—thus earlier plug-ins take priority over later ones. Document formats are usually determined by filename extensions—for example, *foo.txt* is processed as a text file, *foo.html* as HTML, and *foo.doc* as a Word file. It is also possible for a plug-in to open a file and inspect its contents before deciding whether to process it. This is the norm for XML files with the generic extension *.xml*, whose type can only be determined by opening them up and examining the root element. If there is no plug-in that can process the file, a warning is printed and attention passes to the next file.

Plug-ins can inherit the functionality of other plug-ins to perform common tasks, such as converting images and extracting key phrases, dates, and e-mail addresses; these are visible as separate parts of the plug-in configuration panel in the Librarian interface. Also, the Word and PDF plug-ins work by converting source documents to an intermediate HTML form and passing this to the HTML plug-in.

Plug-ins are used for both importing and building, and both processes use the same plug-in list. Importing generates files that are processed by a special plug-in (GreenstoneXMLPlugin) that recognizes Greenstone Archive format documents. These do not occur in the *import* folder, but they certainly do occur in the *archives* folder. ArchivesInfPlugin is also used during building only: it processes the document OIDs that were produced during importing and stored in the *archiveinf-doc.gdb* file mentioned earlier.

DirectoryPlugin is another special plug-in that is included in every collection. It traverses the folder structure in the *import* directory. You place the whole file structure containing the source material into *import*, and DirectoryPlugin recurses through this structure. It only processes folders, and it operates by creating a list of all the files they contain (including sub-folders) and passing the name of each back through the plug-in list. The effect is to expand all directories in the hierarchy. DirectoryPlugin is the last member of the list in all collection configuration files. In the Librarian interface, MetadataXMLPlugin, DirectoryPlugin, and ArchivesInfPlugin are all invisible unless you are in Expert mode, and even then they cannot be removed. Librarian-level users are prevented from removing GreenstoneXMLPlugin, because they do not see the *saveas* option to the import process.

Greenstone has over 30 plug-ins—and the number is growing. It is impossible to cover them all here. Descriptions, along with a list of all their options and a brief account of each one, can be obtained using the program *plugininfo.pl*—just specify the plug-in name as the argument. The most useful options can be seen in the Librarian interface, with tool-tips that give the same information as the *plugininfo* program (including translations into different languages).

Here is a summary of the plug-ins for widely used document formats that are included in the default collection configuration file, along with the file names processed by each.

HTMLPlugin (.htm, .html; also .shtml, .shm, .asp, .php, .cgi)

HTMLPlugin processes HTML files. It extracts title metadata based on the <title> tag; other metadata expressed using HTML meta-tags can be extracted too. This plug-in has many options, some of which are discussed in Chapter 10.

WORDPlugin (.doc) and *RTFPlugin* (.rtf)

WORDPlugin and RTFPlugin import Microsoft Word documents. There are many different variants on the Word format—and even Microsoft programs sometimes make conversion errors. As mentioned in Chapter 10, independent programs are used to convert Word files to HTML. For some older Word formats, the system resorts to a simple extraction algorithm that finds all text strings in the input file.

PDFPlugin (.pdf)

PDFPlugin imports documents in PDF, Adobe's Portable Document Format. Like the Word plug-in, it uses an independent program to convert PDF files to HTML. Note that encrypted PDF files cannot be processed.

PostScriptPlugin (.ps)

PostScriptPlugin imports documents in PostScript. It uses an independent program called Ghostscript, which is included as an option when installing Greenstone.

ImagePlugin (.jpg, .jpeg, .gif, .png, .bmp, .xbm, .tif, .tiff)

ImagePlugin handles images of many different kinds, using an external program called ImageMagick. It (optionally) computes thumbnail and screen-size versions of the image and extracts metadata giving its type, size, height, and width.

TextPlugin (.txt, .text)

TextPlugin interprets a plain text file as a simple document. It adds title metadata based on the file's first line.

EmailPlugin (.email)

EmailPlugin imports files containing e-mail and deals with open e-mail formats, such as those used by the Thunderbird, Eudora, and Unix mail readers. Each source document is examined to see if it contains an e-mail, or several e-mails joined together in one file, and if so, its contents are processed. The plug-in extracts *Subject*, *To*, *From*, and *Date* metadata. However, it does not handle MIME-encoded e-mails properly—although legible, they often look rather strange.

ZIPPlugin (.gz, .z, .tgz, .taz, .bz, .zip, .jar, .tar)

ZIPPlugin handles compressed and/or archived document formats: gzip (.gz, .z, .tgz, .taz), bzip (.bz), zip (.zip, .jar), and tar (.tar).

NulPlugin (.nul)

NulPlugin handles the dummy files generated by the metadata database exploding process explained in Section 10.5.

ISISPlug (.mst)

ISISPlug handles metadata in CDS/ISIS format (popular in developing countries but almost unknown elsewhere). Along with the master file (.mst), CDS/ISIS databases also have field definition files (.fdt) and cross-reference files (.xrf). The Greenstone wiki contains a comprehensive document that explains how to deal with CDS/ISIS in Greenstone.

Search indexes

There are three indexers that can be used for full-text searching: MG, MGPP, and Lucene. The MG search engine, implemented in the C programming language, is described in the classic book *Managing Gigabytes* (mentioned in Section 3.7). It can produce separate indexes that operate at the document, section, or paragraph level—meaning that when several search terms are specified, the combination is sought in the entire document, or in an individual section, or in an individual paragraph. Searches can be either Boolean or ranked (but not both at once). A separate physical index is created for each index specified in the collection. For phrase search, MG uses a post-retrieval scan,

which is potentially slow (Section 3.4). It is otherwise very fast and has been extensively tested on large collections.

MGPP, which is the default indexer for new collections, is a reimplementaion of MG in the C++ programming language, with some enhancements. Behind the scenes, it operates at the word level, which allows fielded, phrase, and proximity searching to be handled efficiently. Boolean searches can be ranked. Different document/section levels, and text and metadata fields, are all handled in a single data structure, which reduces the collection size compared with MG for collections with many indexes. Phrase searching is much faster than for MG, but ordinary searching may be slightly slower due to the word-level, rather than section-level, indexing.

Lucene was developed by the Apache Software Foundation and is written in Java. It handles field and proximity searching, but only at a single level—which means that documents and sections require separate indexes. Its range of functionality is similar to that of MGPP, with the addition of single-character wildcards, range searching, and sorting of search results by metadata fields. It also has a “fuzzy search” option that allows approximate searching. It was added to facilitate incremental collection building, which MG and MGPP do not provide.

Adding and configuring indexes

Indexes are added in the Search Indexes section of the Librarian interface’s Design panel. A configuration panel pops up allowing you to determine whether the index should include the full text of documents, along with any selection of metadata fields. (See the discussion of building the Word and PDF collection in Section 10.3.) As well as allowing selection of any set of metadata fields, MGPP and Lucene provide an Add All button as an easy way of adding all metadata and text sources as individual indexes. Also, a special index is available for MGPP and Lucene (called *allfields*) that searches over all specified indexes without having to specify a separate index that contains all sources. To add this, check the **Add combined searching over all assigned indexes** box on the index configuration panel and click **Add Index**.

The top right of the Search Indexes panel shows which indexer will be used when the collection is built. When the indexer is changed, a window pops up with a list of three options: MG, MGPP, and Lucene. Changing indexers affects how the indexes are built and may affect search functionality. The default structure creates three indexes: full text, titles, and file names (the last is rarely useful). Indexes can be at document, section, or (for MG only) paragraph level.

With MGPP and Lucene, the levels are document, section, or both. Levels are determined globally for all indexes, and the two possibilities are shown as check-boxes on the main Search Indexes panel. With MG, each index has its own level—document, section, or paragraph—which is chosen by the Indexing level selector on its configuration panel.

For MGPP, stemming, case-folding, and accent-folding are determined by check-boxes on the main Search Indexes panel. For MG, only stemming and case-folding can be selected. For Lucene there is no choice. There is also a check-box for the Chinese-Japanese-Korean (CJK) languages, which separates ideographs by spaces for searching—it is a place-holder for more sophisticated schemes for word segmentation, described in Section 8.4.

Partitioning indexes

Indexes are built on particular text or metadata sources. As you learned when building the large HTML collection in Section 10.3, the search space can be further controlled by partitioning the indexes. In Section 10.3, you designed a filter based on a metadata value, but indexes can also be partitioned based on the language of the documents.

Note that for MG collections, the total number of partitions generated is a combination of all indexes, sub-collection filters, and languages chosen. Two indexes with two subcollection filters in two languages would yield eight index partitions, which would result in eight separate index data structures (that is, eight separate physical indexes) when the collection was built. For MGPP, all logical indexes are created in one physical index, so there would only be four separate index partitions. For Lucene, the number of physical indexes is determined by the number of levels assigned to the collection, one index per level. In the above situation, if one level were assigned, there would be four physical indexes, while with two levels there would be eight.

Experimenting with MGPP and Lucene

To experiment with the search options for the MGPP indexer, start a new collection (File→New) and base it on the Greenstone Demo collection. In the Gather panel, drag all the folders in *sample_files→demo→HTML* into the new collection. In the Search Indexes section of the Design panel, note that the MGPP indexer is being used. This is because the original Demo collection on which this collection is based uses MGPP. Note the three options at the bottom of the panel—stem, case-fold and accent-fold. Enabled options appear on the collection’s Preferences page. Under Indexing Levels, select section as well as document. Build and preview the collection.

To use the Lucene indexer, click the **Change** button at the top right corner of the panel and select Lucene in the window that pops up. Build the collection and experiment with searching. Lucene provides range searching and single- and multi-letter wildcards. The full query syntax is quite complex (see <http://lucene.apache.org/java/docs>). To get a brief taste, here is how to use wildcards. The character *** stands for any sequence of letters—a multi-letter wildcard—and can be appended to any query term. For example, *econom** searches for words like *econometrics*, *economist*, *economical*, and *economy*. The character *?* is a single-letter wildcard. For example, searching for *economi??* will match words like *economist*, *economics*, and *economies* that have just two more letters. Note that Lucene uses stopwords by default, so searching for words like *the* returns no matches (an appropriate message appears on the search results page).

MGPP does not support wildcards, but unlike Lucene it can optionally stem, case-fold, and accent-fold, as noted above. Both are enabled by default during the building process (but on the collection’s Preferences panel stemming is enabled and case-folding disabled by default). Change the indexer back to MGPP and rebuild the collection. Note that searching for *econom* returns no documents, while searching for *fao* and *FAO* give the same result—78 words; nine matched documents. Go to the Preferences page and change “whole word must match” to “ignore word endings.” When you return to the search page and seek *econom* again, several documents are found. To avoid confusion later on, revert to the original Preferences setting. To differentiate upper-case from lower-case, set the “ignore case differences” option appropriately and search for *fao* and *FAO* again. This time the results are different. Again, revert to the original setting.

Behind the scenes, these effects are achieved by appending modifiers to the query. Appending *#s* to a query explicitly enables the “ignore word endings” option—if you search for *econom#s*, you will find several matches, even though the mode on the Preference page has been restored to “whole word must match.” These modifiers override the Preferences settings. Likewise, appending *#u* makes the current query explicitly match whole words, irrespective of the Preferences setting. Modifiers *#i* and *#c* control case sensitivity. Appending *#i* to a query term ignores case differences, while appending *#c* takes them into account. For example, searching for *fao#c* returns no documents. Modifiers can be used in combination. Appending *#uc* to a query term will match the whole term (without stemming) in its exact form (differentiating upper- from lower-case).

MG also supports stemming and case-folding, but in a more primitive way, and these options cannot be chosen for individual query terms as they can with MGPP.

11.5 Interoperating

Greenstone can bring in information from many other sources apart from the files on your computer disk. It can download pages from the Web—and from wikis. It can harvest information using the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) and use the Z39.50 and SRU protocols. All of these facilities are provided in the Librarian interface’s Download panel.

Greenstone is also able to write out the contents of a collection in many different formats. Perhaps most importantly, a server is included that can serve any collection using the OAI-PMH. As discussed in [Section 11.4](#), METS can be used as the internal format instead of the default Greenstone Archive format. That is one way of exporting the entire content of a collection in a standard form. Collections can also be exported in other forms: MARCXML, the DSpace archive format, and the variant of METS used by the Fedora digital library system.

Downloading Web sites

The Librarian interface’s Download panel gives access to a comprehensive facility for downloading (or *mirroring*) files from the Web, as well as ways of downloading information from other sources.

For Web mirroring, you enter a URL and then set options that govern how the download process proceeds. To see an option’s description, hover the mouse over it until the tool-tip appears. You can determine how many hyperlinks deep to go while downloading, whether to follow all links or just to mirror files below this URL or within this Web site, and whether to download only HTML files or to include associated files, such as images and style sheets, as well.

If your computer is behind a firewall and needs to go through a proxy server, you will need to specify the proxy settings in the Librarian interface. They are available from the Preferences item of the File menu. In the Connection tab, you can switch on the use of a proxy and enter its server address and port number.

For example, suppose you want to download the Citizens section of the Tudor collection that you built in [Section 10.3](#), which is on a Web site called *englishhistory.net*. Enter <http://englishhistory.net/>

tudor/citizens into the Download panel's URL box. The *Server Information* button will obtain information about this Web site. To copy just this section of the Web site, set the depth option to 1 and switch on both *below this URL* and *within this site* by checking their boxes. If you do not, the downloading process may follow links to other areas of the Web site and to other Web sites—including, perhaps, sites with advertisements—and grab those as well.

Now click **Download**. If you have set proxy information in Preferences, a pop-up will request your user name and password. Once the download has started, a progress bar appears in the lower half of the panel that indicates how the process is doing; more detailed information can be obtained by viewing the log. The process can be paused and restarted as needed, or terminated altogether—it is often useful to pause it occasionally and check the log to ensure that nothing untoward is happening.

Web documents are downloaded using a third-party program for mirroring called *wget*, which retrieves files using HTTP, HTTPS, and FTP, the most widely used Internet protocols. It uses local file time-stamps to determine whether documents need to be re-downloaded on subsequent visits. Thus, if you click the **Download** button again, it will probably download fewer files, unless everything has changed in the meantime (or you have cleared the download cache). It will probably download some files—folders, for example, that are necessary in order to explore the Web site—but not all of them. Note that this incremental downloading mechanism only works for files to which the server gives a time-stamp.

Downloading can be a lengthy process involving multiple sites, and so additional downloads can be queued up. When new URLs are entered into the box and Download is clicked, a new progress bar appears beneath those already present in the lower half of the panel. When the currently active download item completes, the next is started automatically.

Downloaded files are stored in an eponymous folder that appears on the left-hand side of the Librarian interface's Gather panel. You choose which ones you want by dragging them into the collection area. The downloaded files can be cleared by clicking **Clear Cache** on the Download panel. Incidentally, the cache is per-user and appears in the user's filesystem, not in the central Greenstone installation. You will find it in *C:\Documents and Settings\<username>\Application Data\Greenstone\GLI\cache* on Windows XP,* and in *~<username>\.gli/cache* on Unix.

The full content of wikis in the MediaWiki format can be downloaded in exactly the same way. Here you can specify a list of URL types that are to be ignored in the download process using regular expressions; you can also exclude certain directories. The MediaWiki plug-in allows you to create a Greenstone collection from the downloaded information in a way that preserves all the wiki's internal links.

Metadata protocols

Metadata and, if available, documents can be downloaded over three different protocols: Z39.50 (Section 7.1), the OAI Protocol for Metadata Harvesting (Section 7.2), and SRU (Section 7.4).

*You may need to change your Windows settings to "show hidden files and folders" to see the *Application Data* folder. On Windows Vista the folder is *C:\Users\<username>\Application Data\Greenstone\GLI\cache*.

For OAI, you specify the source URL, then you can obtain general information about the server by clicking the appropriate button. This includes the repository name, the version of the protocol that is being used, and details of each of the sets of information that are available for download. You can restrict downloading to a certain set in the repository (for this, use the set specification—*setSpec*—in the Server Information rather than the set name). You can specify the maximum number of records to download. If the Dublin Core section of the OAI records follows the standard convention of including an identifier element that contains a URL, you can ask for the documents themselves to be downloaded too. Again, you should ensure that any proxy server is specified in the Preferences panel, if needed. The metadata records, and the documents as well if there are any, are again placed in the Gather panel's Downloaded Files folder.

For Z39.50, you specify the host, port, and database in the boxes on the Download panel. For example, the Library of Congress's Z39.50 server is at

Host: z3950.loc.gov
Port: 7090
Database: Voyager

You specify a search term and the maximum number of records; the information is downloaded as a file of MARC records.

For SRU, the same source specifies that the base URL is

<http://z3950.loc.gov:7090/voyager>

Serving OAI

Greenstone incorporates a server that can serve any collection over the OAI Protocol for Metadata Harvesting. (However, it does not operate on the Windows Local Library server.) The address of the server is the same as the address of your digital library installation, but using the program *oaiserver.cgi* instead of the program *library.cgi*. For example, if your digital library is at

<http://localhost/greenstone/cgi-bin/library.cgi>

the following URL will produce a response from the OAI *Identify* request (Figure 7.2):

<http://localhost/greenstone/cgi-bin/oaiserver.cgi?verb=Identify>

Other basic requests that the server responds to are

```
... cgi-bin/oaiserver.cgi?verb=ListSets
... cgi-bin/oaiserver.cgi?verb=ListIdentifiers&set=xxx (OAI 1.1 only)
... cgi-bin/oaiserver.cgi?verb=ListIdentifiers&set=xxx
    &metadataPrefix=oai_dc (OAI 2.0 only)
... cgi-bin/oaiserver.cgi?verb=ListRecords&set=xxx&metadata
    Prefix=oai_dc
... cgi-bin/oaiserver.cgi?verb=GetRecord&identifier=xxx
    &metadataPrefix=oai_dc
```

```
# the baseURL used by the oaiserver, e.g. "http://your-domain-name/cgi-bin/oaiserver.cgi
baseURL "your-own-baseURL-goes-here"

# The human readable OAI repository name
repositoryName ""

# Administrative email address for the repository
maintainer ""

# Which version of OAI the repository supports, 1.1 or 2.0 (the server supports both)
oaiVersion "2.0"

# How many records to output before needing a resumption token (-1 returns them all)
resumeafter -1

# Collections to made available to OAI clients.
oaiCollection demo oai-e

# Metadata mappings for the demo collection
oaiMapping demo:dls.Title dc.title
oaiMapping demo:dls.Subject dc.subject
oaiMapping demo:dls.Language dc.language
oaiMapping demo:dls.Organization dc.publisher
```

Figure 11.7: Configuration file for the OAI server

The List verbs (ListSets, ListIdentifiers, and ListRecords) all return sequences of items. A server parameter called *resumeafter* indicates how many records to return, and if the request produces more, a resumption token is returned. To retrieve the next set of documents, issue the same request again with an additional parameter: *resumptionToken=xxx*. For example,

```
... cgi-bin/oaiserver.cgi?verb=ListRecords
    &set=demo&metadataPrefix=oai_dc
    &resumptionToken=gsdloai:demo,,11-
```

where *gsdloai:demo,,11-* is the resumption token returned by the previous request.

You configure the server and determine what collections it should serve using the configuration file *oai.cfg* in the main Greenstone *etc* folder. By default, collections are *not* made available. Figure 11.7 shows an extract from this file (the actual file is more liberally commented). The first few lines give administrative information, followed by parameters of the OAI installation. Then comes a list of collections to be served. Each line starts with *oaiCollection*, followed by one or more collection names—the relevant folders in the *collect* directory.

Greenstone's OAI server supports only the (unqualified) Dublin Core metadata format, and metadata in other formats must be mapped to it. Both general and collection-specific mappings can be specified, and mappings can be placed in the OAI configuration file or in an individual collection's own configuration file (*collect.cfg*). In either case, the format is

```
oaiMapping <Greenstone field-name> <OAI field-name>
```

The Greenstone field name can be qualified by a collection name to make the mapping collection-specific, as illustrated in Figure 11.7 for the Demo collection. Mappings specified in a particular collection's configuration name do not need the collection qualifier.

Exporting collections

The documents and metadata in any collection can be exported to several standard formats. This is done from the Export item on the Librarian interface's File menu, which brings up a panel with these choices:

- METS, using the Greenstone METS profile.
- METS, using the Fedora METS profile.
- DSpace archive format.
- MARCXML format (an XML version of MARC 21).

METS using the the Greenstone METS profile is explained in [Section 11.4](#) in the discussion of document formats that can be used for the Archives that the *import* process produces. The next two options are included so that the Librarian interface's facilities can be used to prepare documents and metadata for the Fedora and DSpace digital library systems described in Section 7.6. The final option provides compatibility with systems that use the MARC standard, for MARCXML can easily be converted (losslessly) into the original MARC format.

[Figure 11.8](#) shows the Export panel. You can select any collection for exporting by checking it in the list. You give a folder name for the result, and the files end up in the Greenstone *tmp* directory under that name. If desired, you can specify an XSLT file (see [Section 4.4](#)) that will be applied to the resulting documents in order to transform the exported information into a suitable format.

The other options differ slightly depending on which format you choose. For example, when exporting to MARCXML, the metadata is mapped to Dublin Core, but you can specify a custom file that maps Greenstone metadata to MARC fields in a different way. You can also choose whether records should be exported one at a time or grouped together in a single file.

Interoperating with DSpace

The DSpace repository system uses an internal archive format for batch importing and for exporting documents or collections to other systems. The format is straightforward: one folder per document, with a manifest specifying the files that make up the document, a Dublin Core metadata file in XML format, a handle file containing the DSpace identifier, and the files themselves. This is precisely the form in which Greenstone exports collections when the DSpace option is chosen.

The DSpace plug-in enables DSpace repositories to be digested in their entirety by reading the same archive format. It seeks out manifest files and their associated Dublin Core records. These records are parsed to form a partial document object model. This is attached to a list of the file names that are specified in the manifest as primary forms of the document. Nothing more is necessary: Greenstone's existing mechanism for binding metadata to documents automatically performs the rest of the work.

Consider a DSpace repository that comprises a single PDF document. When exported, the manifest lists just one file: the PDF document. Accompanying it is an XML file that encodes its Dublin Core metadata. To import the repository into Greenstone, the collection is configured with both the DSpace and PDF plug-ins. During the import process, the DSpace plug-in causes the metadata in the



Figure 11.8: Exporting a collection to MARCXML Windows

Dublin Core record to be associated with the named PDF document, but it is not until the PDF plug-in processes the primary document that the information is actually bound to the file and the full document object is formed.

DSpace distinguishes between primary and secondary files. For example, for a Web page with supporting image files, the HTML file is marked as primary and the images are secondary. Secondary files must be blocked to prevent them from being treated as independent documents with the same metadata attached. This is easily arranged in Greenstone by using its block expression feature. To cover the possibility of a DSpace collection including a document type that Greenstone does not support, the UnknownPlug plug-in is used as a catchall.

The DSpace plug-in is able to fuse the alternative forms of a document together. To do this, it uses Greenstone's notion of *associated files*. A document can have any number of associated files, and format statements can be used to allow readers to access them. Optionally, the DSpace plug-in can

treat documents with the same file name but different extensions as a single entity within a collection. The plug-in options provide three ways of determining which version is the primary document:

- treat a document's first occurrence in the metadata file as the primary version
- give a list of file extensions; the first one that matches becomes the primary version
- give a list of mime types; again the first to match is the primary version.

The primary version of a document is the one that is indexed, and—if possible—metadata is extracted from it, while the other versions are treated as associated files. Extracted metadata called *equivlink* gives a list of the associated files in a form that is ready for use in format statements.

The right-hand side of Figure 11.9 shows a Greenstone collection that has been built from information exported from the small DSpace repository on the left. DSpace allows readers to browse by title and author, and the Greenstone collection has been configured to offer the same facilities—although the interface layout differs. The lower screenshot shows the result of clicking the **titles** button in the navigation bar. From here, various source documents can be viewed, in multiple representations where available. For example, the icon for the Word transcript of the Bob Dylan interview is displayed to the left, and the MP3 icon is shown under “also available as.”

11.6 Distributed Operation

Everything we have done so far in terms of building collections has been undertaken on the same computer. Documents may be downloaded from elsewhere using a variety of protocols, but all the collection design, configuration, building, and formatting takes place locally. However, sometimes distributed operation is necessary. In some cases, the full facilities of the Librarian interface need to be made available to different users working in different places. In others, documents and their metadata must be added to collections from remote locations using a Web interface.

Remote Librarian interface

Suppose several librarians need to work on the same set of collections. Typically there is a central Greenstone installation and each librarian wants to update collections from her own office workstation—adding new documents, editing the metadata, adjusting the collection design and formatting, and even creating brand new collections.

Greenstone contains just such a facility. There is a modified version of the Librarian interface that can run on local workstations—say in librarians' offices—and connect with a standard Greenstone installation on a remote server—say in the library's server room. The local version is called the GLI Client; the remote installation is the GLI Server. It is not necessary to have Greenstone running locally: the remote server does all the collection building on command from the client. Multiple users can collaborate on the same collections, although not at the same time. A lot of data may be transferred between client and server, which makes the scheme impractical unless high-speed connections exist. One important caveat is that for all this to work, the server's *collect* folder must be writable by the Web-server user, which is a potential security risk and may be regarded as a show-stopper by system administrators.

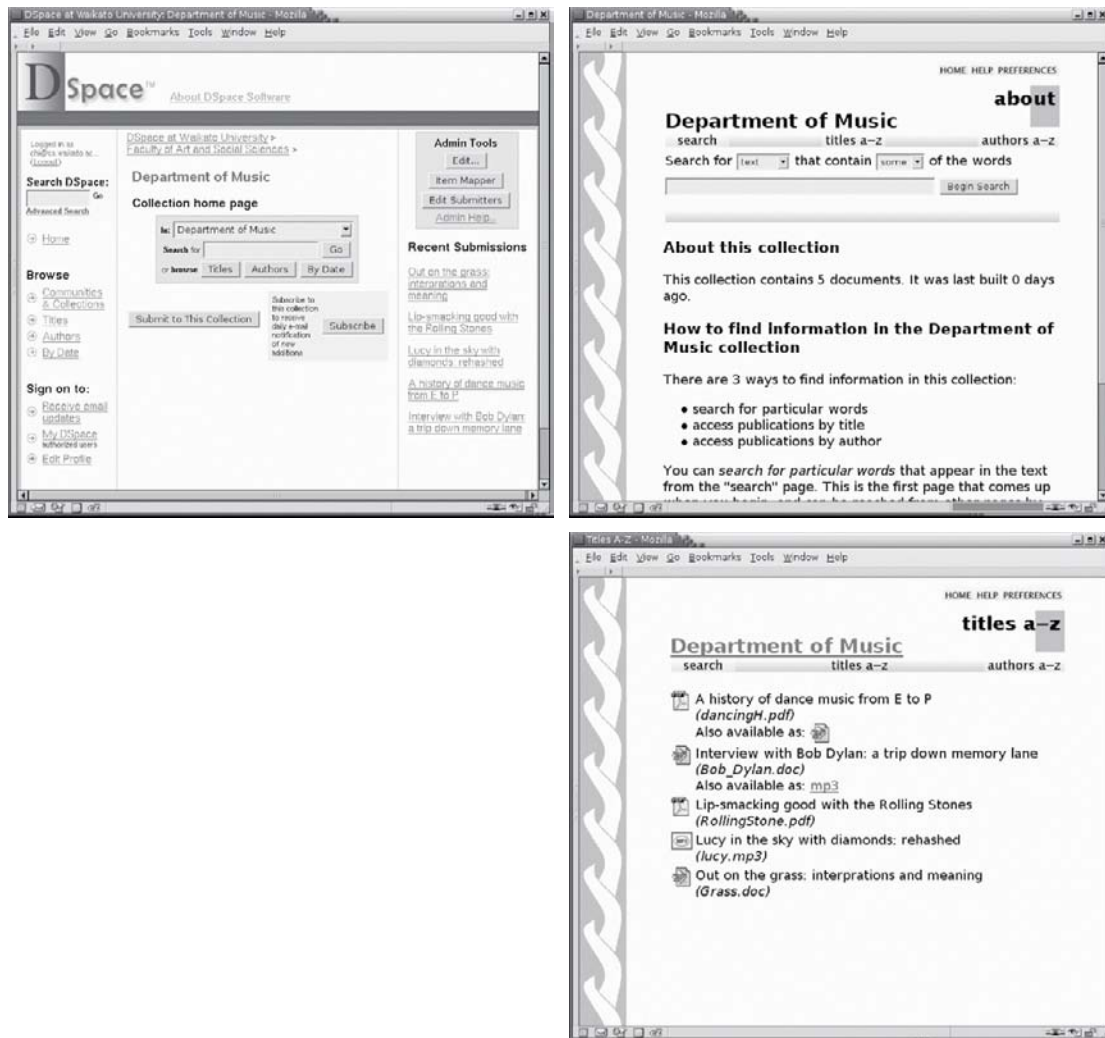


Figure 11.9: Equivalent DSpace and Greenstone collections

Of course, authentication is essential to assure the server that clients who communicate with it are legitimate users with the authority to access individual collections. Greenstone's user account system, described in Section 11.2, serves this purpose, and users are added and removed, and their permissions changed, on the administration pages. The following user groups underpin the remote Librarian interface:

- all-collections-editor
- personal-collections-editor
- X-collection-editor, for any collection X.

Users in the first group can edit all collections and create new ones. Users in the second can create and edit personal collections, that is, ones whose internal collection name begins with their user name. Such collections are created by ticking a “this is a personal collection” box in the New Collection dialog (which only appears on the GLI Client version of the Librarian interface). Users in the third kind of group can create and edit the named collection. For example, someone who needs to create and edit his own collections and collaborate with others on a shared collection called *papers* should be placed in the *personal-collections-editor* and *papers-collection-editor* groups.

For further protection (and, perhaps, for other reasons), the GLI Server can be configured to e-mail the system administrator whenever a collection is built or rebuilt. This is done by editing the *cgi-bin/gliserver.pl* file to set a *mail_enabled* variable, with an associated e-mail address and e-mail server information.

To prevent synchronization problems, each collection can be opened by only one person at a time. When the server receives a request to perform some action on a collection, it checks the collection directory for a lock file (*gli.lock*). This contains the user name of whoever has locked the collection, and it is deleted when the collection is closed. If the collection is locked by someone other than the person making the request, the action fails. The GLI Client reports this information to the user, who is given the option of stealing the lock. This is not recommended, because work may be lost, and it should only be done in cases where the lock file was not deleted because the Librarian interface exited abnormally. To ascertain the situation, consult the user who is locking the collection.

The GLI Client is identical to the regular Librarian interface except for some minor details. The file menu does not contain options to write CD-ROM/DVDs and export collections; and there is no Download panel by default, although this is automatically activated by adding in the Greenstone binary applications for building (see the Greenstone wiki for details).

There is also a Java applet version of the GLI Client, which eliminates the need to install any software in advance on the client computer. However, the GLI Server still needs to be installed and configured, as described below.

Setting up the GLI Server

Install the Greenstone Web Library version (the Local Library does not have this remote facility). Use the Apache v2.0 Web server or higher: v1.3 is not suitable for this application, and IIS is not recommended. Check that Greenstone is communicating properly with the Web server by visiting *http://<your-machine-name>:<port>/greenstone/cgi-bin/library.cgi* (the port number can be omitted from the URL if the default value, port 80, is used).

Next, in a Web browser, visit this URL:

```
http://<your-machine-name>:<port>
/greenstone/cgi-bin/gliserver.pl?cmd=check-installation
```

A message should be returned indicating that Java was found and the installation is OK. However, there are many reasons why it might not be—and there is no point in continuing with these instructions until this check succeeds. If the message says that Java has failed, check that the Java runtime

environment is installed and is on the Web server's path. If it reports a *500 Internal Server Error*, check that Perl has been located correctly by the GLI Server program. Otherwise, on Mac or Linux, failure may indicate that the Web-server user is unable to execute the *gliserver.cgi* program, in which case you must change its file permissions. If all else fails, check the last few lines of the Web server's error log (a file called *apache/logs/error_log*) for the cause of the problem.

Having passed this test, ensure the *collect* folder writable by the Web-server user. Then check the main configuration file (*etc/main.cfg*) to ensure that the Administration page is enabled, and add user accounts as described in [Section 11.2](#).

Using the GLI Client

The GLI Client runs on Windows, Mac, or Linux computers. To use it, download it from the Web (www.greenstone.org/download) or transfer the *gli* folder from an existing Greenstone installation to the required client computer (or computers) and then run *client-gli.bat* (Windows) or *client-gli.sh* (Mac/Linux). The first time it runs it asks for the URL of the remote Greenstone server. This is,

```
http://<your-machine-name>:<port>/greenstone/cgi-bin/gliserver.pl
```

You can now use the GLI Client to edit or update collections on the server and to create new ones. The full facilities of the Librarian interface are available (with the minor exceptions noted above). The first time a collection is opened on a particular machine the plug-in and classifier information is read from the server, which may take a minute or two. If you encounter Premature EOF (end of file) errors when building collections, the Web server has timed out before seeing the next bit of output generated by the building process. The solution is to increase the timeout setting in the Web server configuration file.

Using the GLI Applet

Setting up the applet requires a little more work on the server side. You need to create a trusted version of the program in order to prevent other people from writing rogue applets that spoof the Greenstone server. To do this, use the Java Software Development Kit to generate an appropriate key by running

```
keytool -genkey -alias privateKey -keystore appletstore -storepass  
greenstone
```

in Greenstone's *gli* folder, entering the appropriate details for your organization. Then "sign" the applet by running

```
jarsigner -keystore appletstore -signedjar SignedGLI.jar GLI.jar  
privateKey
```

Move the *SignedGLI.jar* file that this creates into Greenstone's *bin/java* folder. Finally, edit the main configuration file to enable the GLI Applet.

Now visit the home page of your Greenstone installation from a browser on the client computer and click the Librarian interface link part-way down the page. The applet begins loading, and soon a button appears that launches the Librarian interface.

Institutional repositories

A facility called the Depositor lets librarians turn any Greenstone collection into an institutional repository into which authorized personnel can place new items and metadata through an ordinary Web interface. Users belonging to the *all-collections-editor* group have access to this facility. The caveat noted above for the remote Librarian interface applies here, too: in order for the Depositor to work, certain folders must be writable by the Web-server user.

A key issue is the fact that, unlike most repository systems, collections have different metadata sets and there is no restriction on how extensive such metadata can be. Therefore, when metadata is entered through Web pages, the content of these pages and the metadata items that each one requests must be customizable. Whereas a single Web form may suffice for a simple collection, another may require a long sequence of different forms, so the number of pages in the sequence should be customizable too. And to make matters more complicated, this variable amount of data—which is entirely dependent on the metadata set—must be remembered by the Web browser in case the user returns to correct an entry in an earlier step.

The Depositor accomplishes all this and more besides. It is a kind of generalized institutional repository:

- The collection can use any metadata set.
- Depositing an item can involve any number of steps.
- The stages involved in depositing are individually designed.
- The workflow is flexible.
- Roles like reviewer, approver, or editor for deposited items can be integrated into the workflow using, for instance, client GLI.

As a small example, [Figure 11.10](#) shows the submission process for an image collection. To reach this point the user has already logged in. In [Figure 11.10a](#), the collection is selected. Then the user launches a file browser (using the Browse button in [Figure 11.10b](#)) to locate an image and enters its title, artist, date, and notes metadata. Along the bottom is a progress bar with a marker showing the current position, which is Specify Metadata. Clicking **Deposit Item** takes the user to [Figure 11.10c](#), where the new information is digested into the collection—this occurs in a matter of seconds. The final step is to view the collection. In [Figure 11.10d](#), the user is browsing it by title, and the newest addition, *The Bower Meadow*, is at the top. Incidentally, several documents can be added at once by zipping them together—and metadata to be assigned to each document can be included by placing it in a *metadata.xml* file (Section 10.5).

This example is intentionally minimalist, with only two steps in the submission process and hardly any metadata. Developing more sophisticated features, such as checking the content of forms and manipulating their layout (by incorporating previous values, etc.) are typically accomplished by splicing in JavaScript and DOM manipulation. We explain how this is done in Chapter 12.

[Figure 11.11](#) shows edited highlights of the collection-specific macros that produce this example. A macro (called *numsteps*) defines the number of steps in the submission process. In this case it is 1,

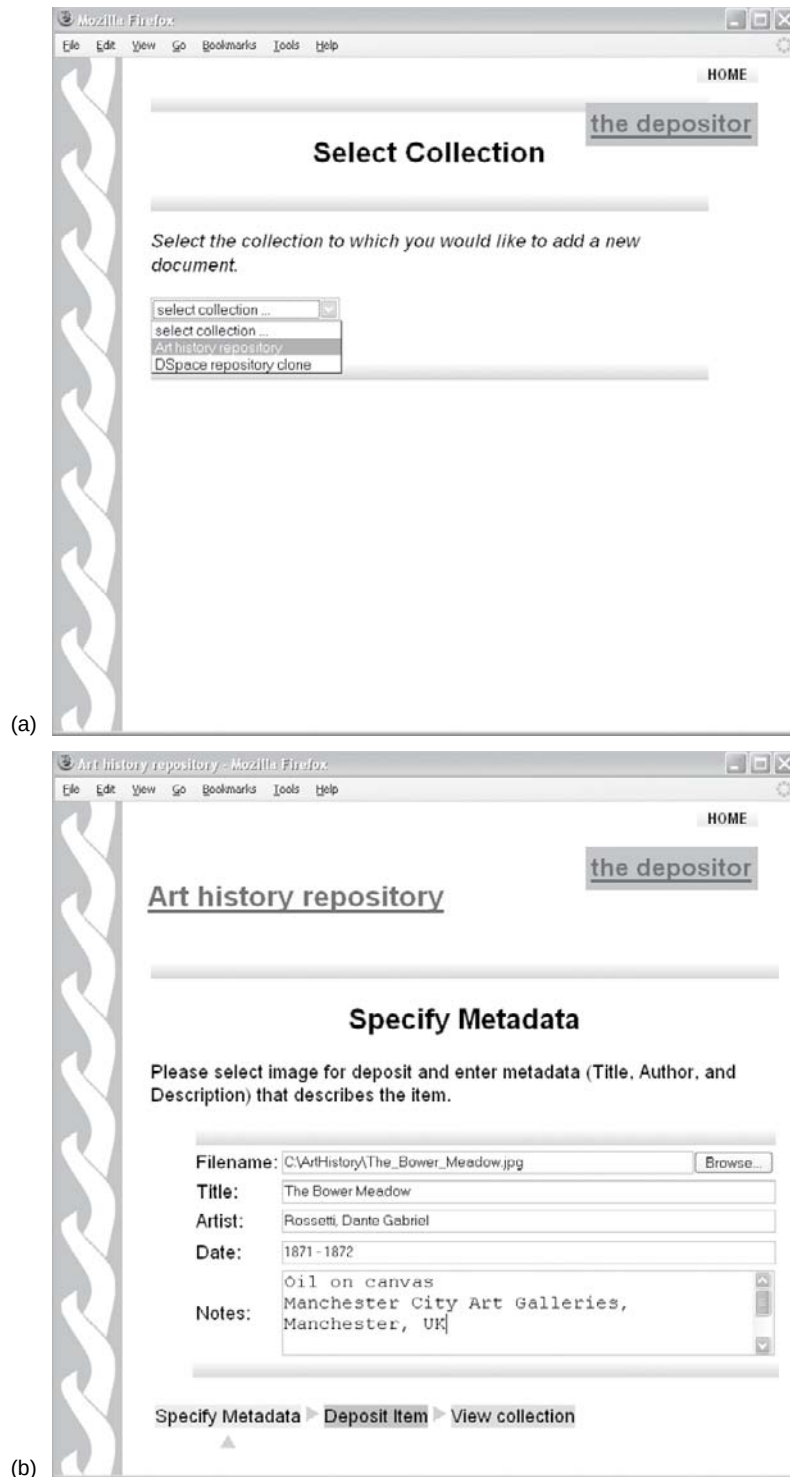


Figure 11.10: Depositor example: (a) selecting the repository; (b) selecting a document and entering metadata;

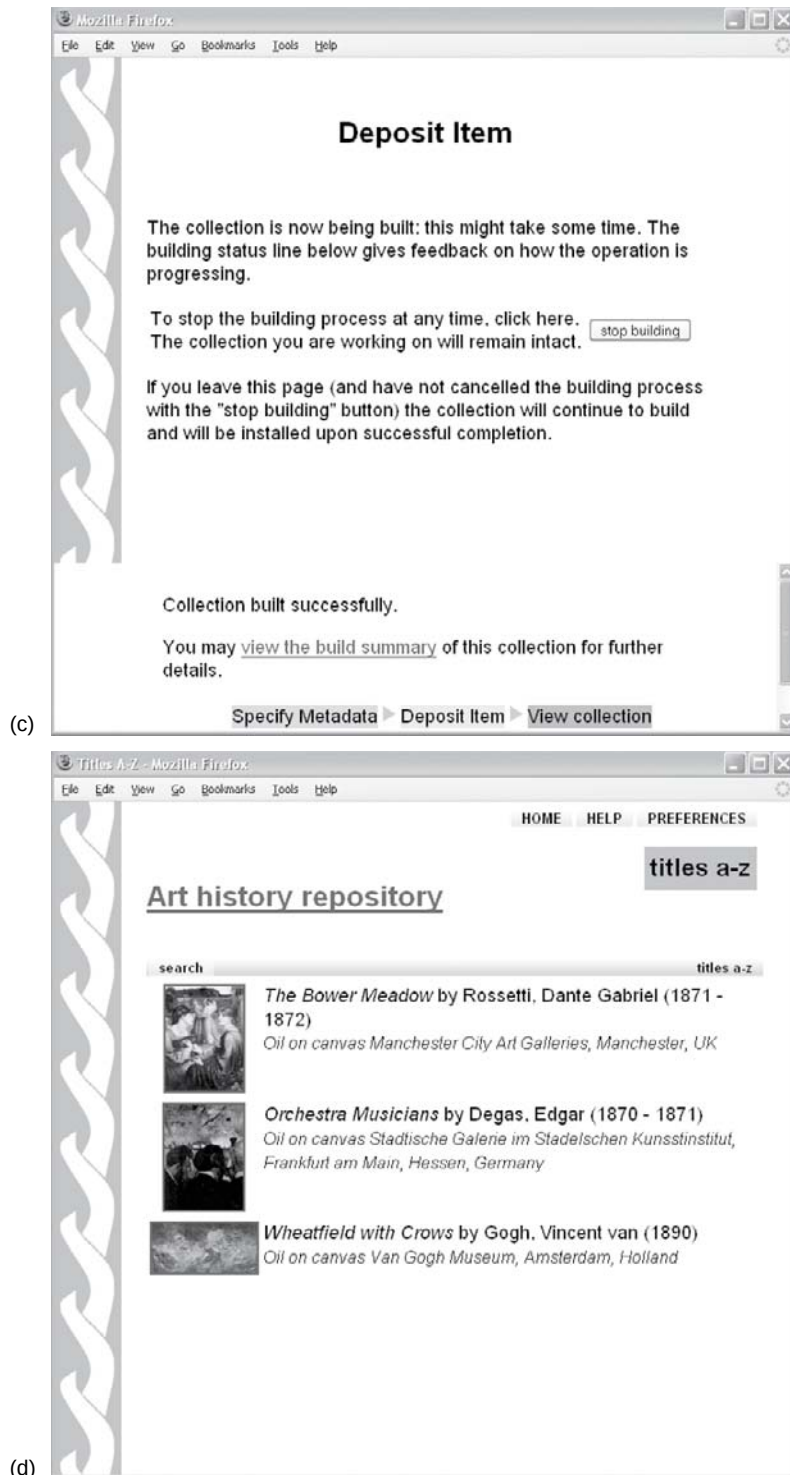


Figure 11.10, cont'd: (c) depositing the item; (d) browsing the collection

```

package depositor

_numsteps_ {1}

_textstep1_ {_textmeta_}
_laststep_ {build}
_textlaststep_ {_textbuild_}

_step1content_ {

<form name="depositorform" method=post action="_gwcgi_" enctype="multipart/form-data">
<input type=hidden name="p" value="_cgiargp_">

<center><h2>_textstep1_</h2></center>
<p>_textimagesimpledesc_</p>

<p><table>
<tr>
  <td>Filename:</td>
  <td> <input type=file name=dauserfile value="_userfile_" size=61></td>
</tr>

<tr>
  <td>Title:</td>
  <td> <input type=text name=damd.dc.Title value="_damd.dc.Title_" size=74></td>
</tr>
<!-- and so on ... -->
</table></p>

<!-- ... -->
<p>_depositorbar_</p>
</form>
}

_textmeta_ [l=en] {Specify Metadata}

```

Figure 11.11: Excerpt from macro file for the first step of the submission process in [Figure 11.10](#)

corresponding to [Figure 11.10b](#), because the final step, [Figure 11.10c](#), is mandatory. For each step, a different macro (called *step1content*, *step2content*, ... *stepNcontent*, and *laststep* for the final one) is used to define the content of the page that is displayed at that step, along with another macro (*step1text*, ...) that dictates what appears in the progress bar. In this example, the progress-bar text (*step1text*) is defined through a further macro (*textmeta*, defined at the bottom of [Figure 11.11](#)) and resolves to the words *Specify Metadata* when viewed in English—you can see this at the bottom of the screenshots in [Figures 11.10b and c](#). The *laststep* macro controls how the workflow ends. For example, the new item may be automatically built into the online collection, or it may be sent to the collection's editor for review.

The progress bar is shown by a macro (*depositorbar*) defined in the Greenstone runtime system. It is formed by composing the information represented in the separate steps (*step1text*, *step2text*, ...). The final step (*laststep*) specifies which of a predefined set of endings terminates the submission process (in this case *textbuild*). Since all this information is stored in the macro files, it can be altered as needed.

DSpace repository clone - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

Describe-1 ▶ Describe-2 ▶ Upload-1 ▶ Upload-2 ▶ Verify ▶ License ▶ Deposit Item ▶ View collection

Submit: Describe Your Item

Please fill in the requested information about your submission below. In most browsers, you can use the tab key to move the cursor to the next input box or button, to save you having to use the mouse each time. ([More Help...](#))

Enter the names of the authors of this item below.

	<i>Last name</i> e.g. Smith	<i>First name(s) + "Jr"</i> e.g. Donald Jr
Author	Holmes	Geoff
	Frank	Eibe

[Add More](#)

Enter the main title of the item.

Title

Enter the series and number assigned to this item by your community.

	<i>Series Name</i>	<i>Report or Paper No.</i>
Series/Report No.		

[Add More](#)

If the item has any identification numbers or codes associated with it, please enter the types and the actual numbers or codes below.

Identifiers

[Add More](#)

Select the type(s) of content you are submitting. To select more than one value in the list, you may have to hold down the "CTRL" or "Shift" key.

Type

- Plan or blueprint
- Preprint
- Presentation
- Recording.acoustical
- Recording.musical
- Recording.oral

Figure 11.12: A more complex Depositor workflow

This scheme is very flexible, although you need a thorough understanding of the macro language to design and implement a new workflow. As a further example, Figure 11.12 shows the first step in a far more complex sequence that mimics the DSpace workflow. DSpace users will be familiar with this sequence. It has eight steps. In this case they are shown at the top of the page, which involves a trivial change to a macro definition. Basic information is entered at the first step (Figure 11.12), while the second is for descriptive metadata. The document file is requested in the next step, following which information is displayed about the file transfer from the user's computer to the server. Then a checksum is shown so that the user can confirm that no transmission errors occurred. (This is accomplished in Greenstone using AJAX technology to retrieve the information from the server in an

extensible manner, as described in Section 12.2, Design pattern 5.) The fifth step provides an opportunity to review and edit all information entered so far.

At any time, the user can return to a previous stage by clicking the progress bar. Making the system remember existing fields—even when they may have arbitrary sets of values, as with authors—is tricky but possible with a little assistance from JavaScript. On the final input page the user decides whether to grant the distribution license. If so, the submitted information (including metadata) is time-stamped and deposited into the collection's *import* folder. The collection's editor is notified by e-mail, and/or the collection is rebuilt, depending on the settings in the collection configuration file.

To use the Depositor, first adjust the file permissions to ensure that the sub-folders of *collect* that contain the relevant collections, and the top-level Greenstone *tmp* folder, are writable by the user account used by the Web-server. Edit the main configuration file and change the relevant line to enable the Depositor. If you are using the Windows Local Library, restart the server. A new button for the Depositor appears on the Greenstone home page. After signing in, you will see a drop-down list of all the available collections, and authorized users may add documents to these. In practice, to get the Depositor to operate according to a particular workflow, it is necessary to design and implement an extensive set of collection-specific macros, which is beyond the scope of this book.

11.7 Large-Scale Usage

How big can Greenstone collections be? It is a little unrealistic to expect general-purpose digital library software, suitable for a wide variety of users and applications, to vie with a purpose-built system designed specifically for a very large information collection (like, say, a Web search engine). However, Greenstone can be pushed reasonably far in the size of collections that can be built.

Limitations of the Librarian interface

For large collections, it is not feasible to use the Librarian interface. For one thing, you may need better control over the files it generates. For another, there is a limit to the amount of metadata that can be handled. With a standard amount of memory allocated to Java (128 MB), the Librarian interface can handle about 8,000 largish (500-character) metadata records; with double the memory, it can handle up to 20,000 records. (These experiments were done in 2006, on a computer with 1 GB of memory.) Larger amounts of memory allocated to the Java process will enable correspondingly larger volumes of metadata.

Without going through the Librarian interface, Greenstone can handle far larger volumes of data—perhaps a thousand times greater. In practice, however, it may be expedient to develop a small-scale version of your collection using the Librarian interface and then build the whole thing manually by issuing *import* and *buildcol* commands as described in [Section 11.3](#).

Large collections

Limits on collection size can be measured in many ways. From a technical point of view, one possible limitation is the full-text indexer. Collections with up to 10 GB of text have been built using the

standard MG and MGPP indexers, while Lucene has been tested on 100 GB of raw text. Note that these sizes are for text alone: this corresponds to a far larger collection of PDF documents. Another possible limitation is the amount of metadata. Collections have been built with many millions of metadata records—for example, 10 to 20 million records totaling 3 GB of text. A further possible limitation is the amount of image data. However, Greenstone only processes images in order to extract metadata and to create other versions of the images, such as thumbnails (which can be switched off), so the file system itself is most likely to be the greatest limitation. The metadata for a collection of 20 million photographs could be handled with modest resources, but the images themselves might occupy 100 TB.

One potential solution for a massive volume of data is to partition it into subcollections and use Greenstone's cross-collection search facility (Section 10.4), which allows users to search in a seamless way across subcollections. However, this only amalgamates the results of *searching* subcollections, not *browsing* them. Of course, alphabetical lists and date lists and the like aren't really much use with very large collections; it is more likely that the metadata will be searched rather than browsed linearly.

As a particular example of a large collection, some years ago the Argentinian organization Archivo Digital built a collection called Archivo Nacional de la Memoria (National Memory Archive) that contains sequences of page images with associated OCR text. They used an old version of Greenstone (2.52) and a 1.8-GHz Pentium processor with 512 MB RAM running Windows XP. There were 20,000 documents, having a total of 1,000,000 pages in TIFF format and 3.2 GB of text files. The Paged Image plug-in was used, and there were five classifiers.

Importing the collection took almost a week (no image conversion was done). Building (building-in-the-small) took 24 hours. The *archive* and *index* folders were placed on separate disks to increase access speed. At run time, it took 2–5 seconds to search the collection regardless of the number of search terms, and 30 seconds to find exact phrases. For unknown reasons, it took an excessive 25 seconds to open a hierarchy node containing 1,000 objects.

A very large collection

Here's a more recent, and far larger, collection. In 2008, DL Consulting (www.dlconsulting.com), a company that specializes in digital library solutions using Greenstone, built a collection called Papers Past for the National Library of New Zealand that contains over a million pages of historic newspapers (<http://paperspast.natlib.govt.nz>). Featured in Chapter 3 (see Section 3.1), it contains nearly 20 GB of raw text: two billion words, with 60 million unique terms, and it is full-text searchable. There are a total of over 6.5 million newspaper articles, each with its own metadata—most of it automatically generated. The total volume of metadata is 50 GB—three times as much as the raw text! Before being built into a digital library collection, the metadata is stored in XML format, which occupies around 600 GB—slightly less than 1 MB per newspaper page.

The collection is growing. Only 60 percent of the newspapers have been OCR'd: when the project is complete, all pages will be searchable. (The company has developed a similar system for Singapore's National Library Board, which is projected to grow to four times the size of the fully realized NZ

collection, or seven times the figures mentioned above.) To operate at this scale, Greenstone software developments were rather modest. All that was necessary was to extend the software to support multiple instances of the flat-file database (GDBM) used for metadata. The extension was needed to surpass an inherent 2-GB limit in the database system. This arrangement adequately satisfies the needs of these projects.

Full-text index

The page images in Papers Past were digitized from microfilm, but the paper originals were of poor quality, which inevitably confounded the OCR software. This yielded a much larger number of unique terms than would normally be expected. The two billion words of running text include 60 million unique terms (2.8 percent), and this is expected to continue to increase linearly as further content is added. By contrast, clean English text typically contains no more than a few hundred thousand terms, and even dramatic increases in size add a relatively small number of neologisms.

Enormous vocabularies challenge search engine performance. Moreover, the high incidence of OCR errors makes an approximate search capability attractive. For this reason it was decided to use the Lucene indexer, because of its fuzzy search feature and proven scalability—it has been tested on collections of 100 GB of raw text.

Metadata database

The Papers Past collection involves a massive amount of metadata. Newspaper articles can be viewed individually as well as in their original context on the page. The 600,000 pages that have been OCR'd so far comprise 6.5 million individual articles, each with its own metadata. The physical coordinates that specify an article's position on the page are stored as metadata to allow article-level images to be clipped from pages. A further requirement is that search terms be highlighted directly within images. In order to do so, the bounding-box coordinates of each and every word in the collection must be stored. These word coordinates represent 50 GB of metadata—nearly three times as large as the collection's full text.

As mentioned previously, Greenstone uses the GNU database management system (GDBM) for storing and retrieving metadata. Fast and reliable, GDBM does this simple job very well. Crucially—and of particular importance for librarian users—GDBM can be installed on Windows, Mac, and Unix computers without requiring any special configuration. However, the design of Papers Past exposed limitations. GDBM files are restricted to 2 GB; moreover, look-up performance degrades noticeably once the database exceeds 500 MB. A simple extension was to modify Greenstone to make it automatically spawn new databases as soon as the existing one exceeds 400 MB: currently, Papers Past uses 188 of them. With this multiple database system, Greenstone can retrieve word coordinates and other metadata very quickly, even when running on modest hardware.

Image server

The archival master source files for Papers Past are compressed bi-tonal TIFF images averaging 800 KB each. The full collection of a million images occupies 850 GB. (The Singapore collection uses grayscale JPEG 2000 master source files, which average 4.5 MB per page; the existing 600,000 pages consume 2.5 TB of storage.) A project goal is that no special viewer software is required other

than a modern Web browser: users should not need browser plug-ins or downloads. However, neither TIF nor JPEG 2000 is supported by all contemporary browsers. Consequently, the source images are converted to a Web-friendly format before being delivered. Also, processing is required to reduce image file size for download, and in the case of individual articles, to clip the images from their surrounding context.

Greenstone normally pre-processes all images when the collection is built, stores the processed versions, and serves them to the user as required. However, it would take nearly two weeks to pre-process all the pages of Papers Past, at a conservative estimate of 1 page/sec. To clip out all articles and save them as pre-prepared Web images would take a further two weeks, assuming the same rate. The pre-processed images would consume a great deal of additional storage. Furthermore, if in the future it became necessary to change the size or resolution of the Web images, they would all need to be re-processed. Hence it was decided to build an image server to convert archival source images to Web-accessible versions on demand and to maintain a cache for frequently viewed items.

A major strength of Greenstone is its ability to perform well on modest hardware. This design philosophy has proven immensely valuable in supporting large collections under heavy load. The software is fast and responsive and on modern computers can service a large number of concurrent users. However, the image server is computation-intensive and consumes significant system resources. The overall system was found to perform satisfactorily with moderately heavy loads using a single quad-core server.

Building the collection

From the original TIFF source files, the OCR process generates information in the METS/ALTO format. ALTO (Analyzed Layout and Text Object) is an XML format for encoding OCR-recognized text and the position of that text in the source image. The METS/ALTO representation includes all the word and article bounding-box coordinates, as well as the full text, article-level metadata, and issue-level metadata. The result includes 92,000 METS files, one per newspaper issue, and 600,000 ALTO files, one per newspaper page. Together, these amount to a total of 570 GB of XML, slightly under 1 MB per page. All this XML is imported into Greenstone, the text is indexed with Lucene, and the metadata and bounding-box coordinates are stored by Greenstone in GDBM databases.

It takes significant time to ingest such a large collection, even without the need to pre-process the source images. [Section 11.4](#) describes the *import* and *build* processes. In this case, the former converts all the METS/ALTO data into the Greenstone Archive format, while the latter creates the Lucene search index and the GDBM metadata databases.

The import phase processes approximately 100 pages/min (on a dual quad-core Xeon processor with 4 GB of main memory), taking four days to import 600,000 pages. This stage of the ingest procedure can be run in batches and spread over multiple servers if necessary. The building phase processes approximately 300 pages/min, taking around 1 1/2 days to build the collection. At the time the collection was originally created, the second step could not be executed incrementally, and the complete collection had to be rebuilt whenever data was added. However, this has been rectified: Greenstone supports fully incremental building providing the Lucene indexer is used, as described in [Section 11.4](#).

Distributed serving

Incremental building improves scalability of the building process. However, the only way to obtain arbitrary scalability of the runtime system is to distribute it over multiple servers. It is already possible for the image server and the core Greenstone system to run on separate computers; indeed, multiple image servers can easily be used. But for true scalability the search index and metadata databases must also be distributed.

Greenstone's cross-collection search facility (Section 10.4) allows users to search in a seamless way across subcollections running on different servers. But this facility is limited and does not fully support distributed operation. In particular, it does not allow distributed metadata databases to be accessed in a simple and uniform way. Buried deep in the code are structures that could be extended to improve support for this. However, here we take a different tack.

The performance of IBM's DB2 platform is being evaluated as both indexer and metadata database for Greenstone. When designing collections, future users may have the option of using this third-party software instead of the default document index and metadata database. DB2 has known scalability and distributed functionality: the ability to distribute a single database over multiple servers is built in. DB2 is a mature, enterprise-scale database application. Implemented as a client/server architecture, a single instance of the server can scale to support 3.5 TB databases with little loss in performance. More importantly, multiple servers can be federated, with a single front-end providing transparent access to a farm of databases. If Greenstone were augmented with an underlying DB2 database, it would be possible to create distributed indexes using this built-in federation technology.

What about the use of a commercial module in an open source system intended for wide distribution? An important factor is the availability of a free version of this software. The DB2 C-Express database has the same basic functionality as the commercial version, but with a restriction on the number of servers. It also allows several key text-search components to be integrated, including the Net Search Extender module, which allows full-text, wildcard, and fuzzy searching. We cannot redistribute this software because it is not issued under the GPL-compatible license, but it should be possible to make it easy for users to download it themselves and hook it into Greenstone.

DL Consulting has demonstrated a proof-of-concept version of Greenstone that uses the DB2 distributed database. Further work is required to make it run smoothly in a production environment. Informal experiments suggest that runtime performance remains unchanged as the size of the underlying collection grows. However, the DB2 version generally took longer to generate a page of results than MG, MGPP, or Lucene, suggesting that optimization is required.

Design patterns for advanced user interfaces

When introducing Greenstone in Section 1.7, we claimed that it is capable of implementing almost all of what is presented in Part I of this book. Take a moment to flip through the illustrations and review the panoply of digital libraries presented here.* Chapters 10 and 11 have taught you most of what you need to know to recreate in Greenstone the functionality described in earlier chapters:

- how to handle different media types and file formats
- how to extract and manually assign metadata
- how to build browsing structures and search indexes based on metadata and text
- how to support different languages
- how to present documents in different ways depending on their media form
- how to add authentication to individual documents or entire collections
- how to interoperate with other digital library systems.

What you have not learned, however, is how to control the presentation of Greenstone collections to give the appearance you want. Chapters 10 and 11 touched on this, but concentrated on functionality rather than appearance. In this chapter presentation is the primary focus.

Greenstone provides two mechanisms to control the presentation of Web pages: format statements and macros (see the examples of Chapter 10). We used a format statement to add volume and date information to the search results list of the paged image collection; we customized the appearance of the Greenstone home page by replacing the macro *home.dm* with *yourhome.dm* (see the end of Section 10.4). Now it is time to develop more advanced user interfaces. We will express examples in the form of *design patterns*, a software engineering term for general, reusable solutions to commonly occurring problems.

Format statements and macros work in concert with cascading style sheets (CSS, described in Section 4.4), but operate in a way that is more tightly coupled with the digital library software itself. As we saw in the branding example at the end of Section 10.4, CSS can be used to alter the overall style of your digital library: the size and typeface of the font used, the width of the margins, and so on. Format statements and macros control far more. For example, macros determine whether the

*Note incidentally that Greenstone supports page flipping too (see Figure 3.5). To turn it on for HTML documents in your own collection, just add the option `-use_realistic_book` in HTMLPlugin.

query page has a single one-line text box, a large query box capable of holding whole paragraphs, or a form with boxes for different metadata elements. Format statements dictate the number of columns in a table and what to put in each one. Large, unwieldy format statements, which can arise when developing complex interfaces, can be stored in macro files. A further degree of control is provided by the programming capabilities of the JavaScript language. For example, a Greenstone collection can be made to respect local conventions for displaying dates, based on the locale set in the user's operating system. We encounter all these in the examples below.

Format statement and macro files are arcane—a fact that stems from their power rather than the whim of their designers!—but it is worth taking time to master them. They are the key to radical customization of a Greenstone digital library and can alter not only presentation but functionality as well. They are what allow collections like Papers Past (Figure 3.4) to be implemented with Greenstone. It is said that the true measure of a software tool's success is when it is used in ways that its designers never even imagined, and that is certainly the case here. An experienced hand can quickly transform a collection from Greenstone's default behavior to something breathtakingly different. You can see the possibilities by browsing the list of sites at www.greenstone.org/examples.

This chapter begins by reviewing and extending the material on format statements introduced in Chapter 10 by presenting some commonly used examples. Next we explain the macro language, which was mentioned in passing when describing Greenstone's institutional repository facility (see Section 11.6).

We follow with a series of five design patterns. The first shows how to use macros to add static content to Greenstone collections, a technique that can be repeated to create as many additional pages as needed—our illustration adds a page containing personal contact details. The second uses JavaScript to further control the format of a Web page—our illustration uses it to format dates more clearly and to convert file sizes from bytes to KB and MB as appropriate. Metadata cannot be represented directly in macros as it can be in format statements; the third design pattern shows how to overcome this by passing metadata as arguments to a macro.

In the late 1990s, dynamic HTML revolutionized the user experience of the Web by allowing pages that crackle with interactivity. The key is real-time manipulation of CSS in tandem with the document object model (DOM) that represents the structure of Web pages. The fourth design pattern illustrates this by showing how to inject header information into alphabetically ordered browsing lists and how to selectively expand and contract sections of a table of metadata. A further significant leap in Web interactivity came with AJAX—Asynchronous JavaScript and XML—and the fifth design pattern starts with a simple example and proceeds to describe a digital music page-turner developed within the Greenstone framework.

12.1 Format Statements and Macros

Greenstone ships with a selection of default macros and format statements. The practical work in Chapters 10 and 11 has familiarized you with these defaults and with the appearance they give digital library collections. However, one shoe does not fit all. This chapter shows, through examples, how to go beyond the default setup.

In broad terms, macros control the structure of pages: where page headers and footers are and what they show, where the results are displayed on the search results page, etc. Format statements fill in details within this structure: for instance, they control just what information the search result list displays. An important difference is that format statements are implicitly associated with documents and consequently can access their metadata (by enclosing the item name in square brackets, such as [dc.Title]; see Section 10.4). Macros are not associated with documents, and therefore cannot access metadata directly. In broad terms:

- Macros express static site-wide information, yielding headers, footers, help text, and so forth.
- Format statements generate dynamic metadata-driven content from a particular collection.

This is an over-simplification, however: for example, it is possible for macros to access metadata if they are invoked from within a format statement. We show how to do this in [Section 12.2](#). But the simplification is useful to get us started. In the next two sections we examine these mechanisms in turn, starting with a series of format statements that demonstrate common ways of tweaking the default presentation, and proceeding to an overview of what macros do and how they are expressed.

Format statements

Format statements introduce document metadata into text marked up in HTML. Additional control is provided by {If} and {Or} statements. These concepts are imported from computer programming languages but are no more difficult than formulas in a spreadsheet, which is familiar territory for many non-programmers. The structure of an Excel *if* expression is the same as Greenstone's: Excel's *If(test,value-if-true,value-if-false)* corresponds to Greenstone's {If}{test,value-if-true,value-if-false}. Unlike Excel, in Greenstone such statements are intermingled with HTML-tagged text, and the role of the curly brackets is to allow the runtime system to sift them out. The {Or} statement is logically unnecessary, providing a syntactic convenience in certain situations: everything it can do can also be done with a more verbose combination of {If}s.

In writing, lists are usually organized vertically, so the most commonly customized format statements are for VLists—vertical lists (see *Format features* in Section 10.4). Here are some common patterns.

If a VList format statement starts with a <td> tag, it is assumed to be a row in a table. Greenstone automatically encloses it with <tr> ... </tr> tags and encloses the entire table with <table> ... </table>. Basic control over document items is provided by

```
<td>[link][icon][link]</td>
<td>[dc.Title]</td>
```

This generates a two-column table, with a link to the document in the first column and title metadata in the second. The original source document—or anything else—could be added in an extra column:

```
<td>[srclink][srcicon][srclink]</td>
<td>[link][icon][link]</td>
<td>[dc.Title]</td>
```

Note that *link*, *icon*, *srclink*, and so on are actually treated as extracted metadata—they are generated by Greenstone and associated with each document. The Librarian interface presents names for extracted metadata as beginning with the prefix *ex*. However, this prefix is omitted internally: Greenstone assumes extracted metadata by default when it encounters unqualified metadata names like *link*, *icon*, and *srclink*.

Another pattern introduced in Chapter 10 is the use of image thumbnails instead of the icon that is automatically generated by *[icon]*:

```
<td>[srclink][thumbicon][/srclink]</td>
<td>[dc.Title]</td>
```

Note the similarity with the first of the two examples above. Exactly the same effect can be achieved by replacing the first table cell (i.e., line 1) above by this more complicated version:

```
<td>[srclink][/srclink]</td>
```

In fact, behind the scenes the ImagePlugin sets *thumbicon* metadata to something equivalent to this expression (it also sets *ThumbWidth* and *ThumbHeight* metadata for the thumbnail image). While thumbnail images are common enough to warrant *thumbicon* metadata as a convenient shorthand, a huge variety of other associated files might be bound to a document, and this more explicit technique shows how to link to such files.

In the excerpt above, the *src* attribute with value *_httpassocdir_/[assocfilepath]/[Thumb]* creates a URL for the associated image. The image plug-in sets up the thumbnail using Greenstone's generic mechanism for a document's associated files. The macro *_httpassocdir_* gives the name of the folder where Greenstone places associated files (in fact, this is the folder *<collectionname>/index/assoc*); within this file area, *assocfilepath* metadata gives the name of the associated-file folder for this document; and *Thumb* metadata gives the name of the actual thumbnail image. The same pattern can be used to construct ** hyperlinks to associated files. There is a range of plug-ins in Greenstone that set up associated files following this pattern, varying only the last part: the metadata name (or perhaps explicit filename) used for the actual resource associated with the document.

Another useful pattern encountered in Section 10.4 (Formatting exercise 1) is

```
<td>[link][icon][/link]</td>
<td>{If}{[numleafdocs]},[Title]([numleafdocs]),[dc.Title]}</td>
```

The first cell is the same as in the first example above. The second uses the variable *numleafdocs* in two different roles. Its first occurrence tests whether the variable has a value, in order to determine which of the two branches of the *If* to take. The metadata *numleafdocs* is only set at internal nodes of a hierarchy (e.g., bookshelves in a classifier), and gives the total number of leaves below this node, which is printed in parentheses by the second occurrence above. In either case, the title is printed: the extracted title *[Title]* for internal nodes, which is set to the metadata value corresponding to that node; and the Dublin Core title *[dc.Title]* for leaf nodes (i.e., documents).

When format statements become overly complicated, they can often be simplified by going back to the source documents and assigning metadata that supports what you want to present. For example, if

```

package Global

_header_ { The New Zealand Digital Library Project}
_content_ { Oops. If you are reading this then an error has occurred in the
           runtime system.}
_footer_ { Powered by <a href="http://www.greenstone.org">Greenstone</a>}.}

package query
_content_ { _If_( _cgiargqb_ eq "large", _largequerybox_, _normalquerybox_ )
           ...}
# ... the macro descriptions for _largequerybox_, _normalquerybox_,
# and other nested macros are omitted for brevity

_header_ [l=en] {Begin search}
_header_ [l=fr] {Démarrer la recherche}
_header_ [l=es] {Iniciar la búsqueda}
# ... and so on.
# Normally grouped together by language; e.g., all the English text fragments
# in english.dm

_matches_ [l=en] { _1_ document(s) matched your query}

```

Figure 12.1: A macro file

different types of document need to be presented in different ways (article, paper, book, ...), you could define a metadata value, say *dc.Format*, that specifies the type. If different types are in different folders in the input material, this can be quickly done by assigning metadata in the Librarian interface at the folder level. This metadata is not necessarily presented to the user, rather it is used as internal metadata that helps simplify the logic of *If* statements.

Macros

Greenstone macros control the structure of all Web pages generated; they also provide language-independent interfaces. Figure 12.1 gives an artificial excerpt that illustrates how macros are defined and used. Macro definitions usually comprise a name flanked by underscores, and a definition placed within braces { ... }. Lines beginning with the hash character (#) are comments. A second form of definition is for macros with arguments (see below). Macro definitions contain a mix of plain text, HTML, and references to other macros. For conciseness, the definitions in Figure 12.1 are short. In practice they often span 20 lines or more, indented to improve readability. Macros do *not* contain references to metadata values like [dc.Title], because macro definitions are not bound to particular documents.

Macros are grouped into packages, and an inheritance scheme is used to determine which definitions are in effect at any given time. This allows the content generated for a page to be embedded within a global formatting style. For instance, most pages have a standard header and footer that enclose page-specific content. Figure 12.1 shows a baseline page defined in the Global package, whose *_content_* macro is not intended to be seen. It is overridden by the immediately following query package that generates a page inviting users to enter search terms and perform queries.

Macros can include parameters interposed in square brackets between name and content. These are known as page parameters because they control the overall generation of a page, and they are expressed as `[x=y]`, which gives parameter *x* the value *y*. The `_header_` macro in Figure 12.1 uses the page parameter *l*, which determines the interface language. Another parameter (not shown in this example) is *v*, which controls whether images are used in the interface. A precedence ordering for page parameters is built into the macro language to resolve conflicting definitions.

Figure 12.1 gives versions of the `_header_` macro in English, French, and Spanish. They set the parameter *l* to the two-letter international standard abbreviation (ISO 639), enabling the system to present the appropriate version when the page is generated. If a macro has no definition for a given language, the parameter is simply omitted, which in standard Greenstone produces an English version because this is the most complete and up-to-date language for the interface. However, it is easy to change the default. It is specified in the main configuration file (see *Configuration files* in Section 11.2) using the `cgiarg` option, specifying (for French) `cgiarg shortname=l argdefault=fr`.

The macro language has its own version of the *If* statement `_If_`—an example appears in Figure 12.1’s `_content_` macro. In this case it uses the macro `_cgiargqb_` to determine whether to display a normal query box or a large one on the search page. The value of `_cgiargqb_` is set at runtime by the Greenstone system; its value is ultimately determined by the user’s setting on the Preferences page, which can be controlled through a pull-down menu that offers the choice of query style as *single line* or *large*. Choosing *large* causes the Web page to set the HTML form field *qb* to *large*, which in turn leads the runtime system to set `_cgiargqb_` to *large*. Several other system-defined macros have values that are determined at runtime—for example, the number of documents returned by a search. We return to this point below.

The final macro definition in Figure 12.1, `_matches_`, illustrates the use of arguments. If invoked by `_matches_(5)`, this generates the text *5 document(s) matched your query*. An *If* statement could be added to separate the singular and plural cases, rather than resorting to the clumsy “document(s)” —but that is not the point this part of the figure is looking to demonstrate. Notice the use of `_1_` in the macro definition to refer to the first argument passed in when the macro is invoked. The pattern continues with `_2_` for the second argument (if required), and so on; and `_matches_(5,3)` would be used for a two-argument macro. It is, however, not normal to call a macro with fixed numeric arguments such as 5 and 3. Values set by the runtime system (such as `_numdocs_`) or metadata from a document are more common.

Figure 12.1 is artificial in both content and layout. Although macro definitions from different packages can be mixed together in a single file, as here, this is not the norm. Usually, different files control the layout of different pages, such as *about.dm*, *query.dm*, and *home.dm* for the Greenstone About, Query, and Home pages. There are two exceptions to this convention. One is a collection-specific macro file called *extra.dm*, which contains any macros defined in the Macros section of the Librarian interface’s Format panel. Another is the set of macros containing textual language fragments, which are grouped into files by language (*english.dm*, *french.dm*, and so on). In fact, each language has two files, for example *english.dm* and *english2.dm*, the former containing phrases used in the core Greenstone system and the latter containing those in auxiliary subsystems. The distinction is made to avoid overwhelming translators and to help them focus on the text used most often.

Table 12.1: Global macros defined in *gsdlsite.cfg*

Definition	Description
<code>_httpprefix_</code>	Specifies the HTTP address of the Greenstone home directory, from which you can build up URLs for images and collections
<code>_httpweb_</code>	Specifies the HTTP address of the top-level <i>web</i> folder in the Greenstone home directory
<code>_gwcgi_</code>	The URL of the Greenstone digital library, which is an executable CGI program named <i>library.cgi</i> (the <i>gw</i> , short for gateway, in <i>gwcgi</i> is an archaic historical hangover)

The standard Greenstone Reader's interface is built entirely out of macros. It is an excellent source of additional examples of how to use them.

Commonly used macros

Table 12.1 lists three values defined in Greenstone's site configuration file (*gsdlsite.cfg*, Section 11.2) that are set as macros `_httpprefix_`, `_httpweb_`, and `_gwcgi_`. The purpose of these values is to map Greenstone into the Web server's space, with increasing levels of refinement. Greenstone is usually installed in a single folder in the file system (say *greenstone-2.83*), which is mapped to a unique prefix in the Web server's configuration file that is also recorded in `_httpprefix_`. For example, setting it to */greenstone* would be a good choice. If a more up-to-date version of Greenstone were installed later, this could be placed in a new folder on the file system, collections moved across as needed, and the Web server's configuration file altered to map */greenstone* to the new location.

The `_httpweb_` macro specifies how the root area for supporting files such as images, CSS, and JavaScript in Greenstone is mapped to the Web server's space. By default, the folder *web*, located in the folder where Greenstone is installed, forms the root, with sub-folders for *images*, *style* and *script*. Continuing the configuration example from before, having set `_httpprefix_` to */greenstone*, `_httpweb_` would need to be set to */greenstone/web*. In fact, on this occasion it can be omitted as this is its default value. The macro `_gwcgi_` provides the URL for accessing the digital library. The norm is to leave this undefined, in which case, for the configuration being described, the library's URL is */greenstone/cgi-bin/library.cgi*.

These three macros are fundamental to Greenstone's operation, and numerous definitions in the macro files are based upon them. Table 12.2 shows some macros that are often used for defining other macros and in format statements. The upper entries define partial URLs that pinpoint general areas within the installation from which full URLs can be formed. The lower part gives fully formed URLs that can be used directly to link to pages like a collection's Preferences or Query page.

Other macros dictate what goes into pages. We have already seen that the overall structure is controlled by `_header_`, `_content_`, and `_footer_`. Table 12.3 summarizes useful macros that splice in further information, be it graphics, text, or anything else that can be embedded into HTML—such as JavaScript. For example, `_pagescriptextra_` is used in the design patterns below to introduce JavaScript functions that extend Greenstone's functionality by improving how items are presented. The

Table 12.2: Macros in *base.dm* that define URLs

	Definition	Description
Partial URLs	<code>_httpimages_</code>	URL prefix for where the site-wide <i>images</i> folder is located in the Greenstone home directory
	<code>_httpstyle_</code>	URL prefix for where the site-wide CSS <i>style</i> folder is located in the Greenstone home directory
	<code>_httpcollection_</code>	URL prefix for where the current collection is located
	<code>_httpcimages_</code>	URL prefix for where the collection-specific <i>images</i> folder is located
	<code>_httpassocdir_</code>	URL prefix for where the collection's associated files directory is located
Full URLs	<code>_httppagex()</code>	Useful macro for forming URLs to specific pages
	<code>_httppagehome_</code>	Link to the home page of the digital library
	<code>_httppageabout_</code>	Link to the <i>About this collection</i> page
	<code>_httppagehelp_</code>	Link to the Help page
	<code>_httppagepref_</code>	Link to the Preferences page
	<code>_httpcurrentdocument_</code>	Link to the current document

Table 12.3: Macros that help generate pages

	Definition	Description
Determined in advance	<code>_pagetitle_</code>	What gets displayed at the top of the browser window
	<code>_pagescriptextra_</code>	Extra JavaScript to include in the header
	<code>_pagebannerextra_</code>	Anything extra to display in the page banner
	<code>_pagefooterextra_</code>	Anything extra to display in the page footer
Determined at runtime	<code>_navigationbar_</code>	Navigation bar populated with search and browse elements
	<code>_builddate_</code>	When the collection was last built
	<code>_numbytes_</code>	Number of bytes indexed
	<code>_numdocs_</code>	Number of documents indexed
	<code>_numsections_</code>	Number of sections indexed
	<code>_numwords_</code>	Number of words indexed
	<code>_httpnextarrow_</code>	Hyperlinks to the next and previous sections of the document
	<code>_httpprevarrow_</code>	being viewed (only set when performing the document action, i.e., <i>a=d</i> in the CGI arguments)
	<code>_numpages_</code>	Number of pages in a document (only set when viewing a document processed by PagedImagePlugin)
	<code>_thisOID_</code>	File name given to document at import time (based on, but not necessarily exactly the same as, its document identifier)
	<code>_versionnum_</code>	Version number of the Greenstone installation

lower part of the table gives macros that are set by the runtime system. For example, `_navigation-bar_` can only be defined once the collection's indexing and browsing structures have been determined.

Another kind of macro definition that is set by the runtime system takes the form `_cgiarg..._`, where the ellipsis gives the name of an argument in the URL. We met this in Section 10.6 when switching between images and text in a collection of paged images. It is a useful way to incorporate new functionality. There are no setup requirements—nothing has to be done in advance to declare a new variable. Greenstone sets a macro of the form `_cgiarg..._` for every CGI argument in the URL, and macro files can immediately start using these to access the variable. As a result, macros can change what is presented to the user. This is how Greenstone was extended to support the display of HTML documents as realistic books (Figure 3.5).

12.2 Design Patterns

Now we study some larger examples, framed as design patterns because the focus is on the general approach rather than what the specific examples actually achieve.

Design pattern 1: Additional static pages

How can you add extra pages to a Greenstone collection? As you already know, the Greenstone home page is created by macros in the file *home.dm*. It starts with the phrase *package home* followed by several macro definitions, the most important being `_content_`, which supplies the page content. There are no definitions for `_header_` or `_footer_` macros, so the default definitions in the Global package will be used. The end result is to sandwich the new `_content_` definition between the standard header and footer (see Section 12.1 for more details). You can repeat this pattern of defining new macro packages and populating them with macro definitions yourself, inventing new package names as you see fit.

We will describe how to add the page in Figure 12.2a, which gives contact details and photos of the three authors of the book, to either a Greenstone digital library site or an individual collection. Figure 12.2b shows the macro file that generates it. It shows the home page of each author as a hyperlink, and each one's e-mail address, hyperlinked using *mailto* to launch the user's mail client. The telephone numbers are formatted by a Skype plug-in that was installed in the browser used for the snapshot: it links values it thinks are telephone numbers to Skype and automatically dials when they are clicked. (As Figure 12.2a illustrates, it includes heuristics to suppress this for fax numbers.)

These macro definitions are placed in a package called *contact*, specified in the new file *contact.dm*. Lines 10–31 define the `_content_` macro, which is made more readable (and maintainable) using a set of support macros, defined above it. Points of interest include:

Line 3. The `_pagetitle_` macro gives the text that appears in the Web browser's title. It overrides the default defined in the Global package that displays the name of the collection.

Line 5. The `_homepage_` macro takes the user's login name as an argument and generates the home page URL in the University of Waikato Computer Science Department.



(a)

```

1 package contact
2
3 _pagetitle_ {Contact the authors}
4
5 _homepage_ {http://www.cs.waikato.ac.nz/~1_/}
6
7 _email_     {1_&#64;cs.waikato.ac.nz}
8 _esubject_  {subject=How%20to%20build%20a%20digital%20library}
9
10 _content_ {
11
12 <table>
13   <tbody>
14     <tr>
15       <td>
16         
17       </td>
18       <td>
19         <ul>
20           <li>Telephone: +64 7 838 4246
21           <li>Extension: 4246, 4577
22           <li>Facsimile: +64 7 858 5095
23           <li>Email:
24             <a href="mailto:_email_(ihw)?_esubject_">_email_(ihw)</a>
25           <li>Home page:
26             <a href="_homepage_(ihw)">_homepage_(ihw)</a>.
27         </ul>
28       </td>
29     </tr>
30   </tbody>
31 </table>
32 }

```

(b)

Figure 12.2: Adding a static page to a collection: (a) viewing the result in a Web browser; (b) macro file

Lines 7–8. The `_email_` macro takes an e-mail name (which does not have to be the same as the user's login name) as an argument and generates a full address. The `@` symbol in the e-mail address is specified using its HTML entity form `@`; in an attempt to defeat spammers from automatically ripping this information from the Web page. The `_esubject_` macro defines the text *How to build a digital library* (encoded for inclusion in a URL using `%20` to represent spaces) and is paired with the `_email_` macro so that if a user clicks on the hyperlink his or her e-mail client will automatically populate the subject line with this text.

To link this new page into the digital library, two things must be done. First, the new macro package must be made visible to the Greenstone software. Second, one or more links to it must be embedded into an existing page. Macro definitions can be added to collections by placing them in a file called *extra.dm* in the collection's *macros* folder or by pasting them into the Macros tab of the Librarian interface's Format panel, which accomplishes exactly the same thing. This file is read automatically. Alternatively, the file *contact.dm* could be placed in Greenstone's top-level *macros* folder, in which case its name must be added to the main configuration file (Section 11.2; see also the end of Section 10.4). It is a good idea to use separate files for significant changes, making things easier to manage in the long run.

The second step, adding a link, can be done using the `_httppagex_(x)` macro in Table 12.2. For instance, this line could be added to the `_content_` macro in *home.dm*:

```
<a href="_httppagex_(contact)">Click here to view contact
information.</a>
```

This would make the new page with contact details available from the digital library's home page.

As noted, Greenstone automatically picks up the file *extra.dm* in a collection's *macros* folder and associates it with that collection. This is also true for other macro files. If desired, you can copy standard files such as *query.dm* and *about.dm* into the collection's *macros* folder and customize them: the definitions included there will override the site-wide definitions. (Note, however, that Greenstone only picks up standard macro files whose names appear in the main configuration file.) If you wanted to put the same information on the *About* page of an individual collection, you could copy *about.dm* into the collection's *macros* folder and add the above line to its `_content_` macro. Delete all the other macros, because they are no different than the original definitions.

Design pattern 2: Using JavaScript to adjust presentation

Format statements allow you to manipulate how things are presented. However, there are limits to what they can accomplish. For example, although they can include conditional tests (*If* statements), they cannot perform numeric calculations or manipulate strings. If you want to add two metadata values together and test the result, you will need to use JavaScript. This design pattern illustrates the use of JavaScript through two examples that take raw data and display it on the Web page in a more readable form.

The first shows file sizes using KB and MB for large values. The second displays dates in a way that respects local conventions, determined by the locale set in the user's operating system. Both use the JavaScript `document.write("....")` function to inject fresh text at that point into an HTML document.

JavaScript must be embedded in `<script>...</script>` tags to ensure that it is not interpreted as HTML. The whole thing can be placed directly into a format statement, but as soon as the JavaScript exceeds a couple of lines (which is invariably the case) it makes more sense to define a function, place it in a macro file, and call it from within the format statement using JavaScript. Greenstone already embeds a set of JavaScript functions into the pages generated. Extra functions can be included in this set by defining the macro `_pagescriptextra_` (listed in [Table 12.3](#)), and that is what we use here.

[Figure 12.3](#) defines two functions to accomplish these tasks: `format_filesize()` on lines 5–17 and `format_date()` on lines 19–29. The snapshot associated with the next design pattern ([Figure 12.4a](#)) shows the result. In the snapshot, a video is playing, and in the last two lines of the metadata tabulated below you can see the file size of the video, 7 MB (rounded to the nearest megabyte and hyperlinked to download the file), and its release date, Wednesday, 20 December 2006, which has been reformatted from the internal Greenstone value of 20061220. The format statement that produces the “Download size” table cells, for instance, simply calls the `format_filesize()` function in JavaScript:

```
<td>Download size:</td>
<td> [srclink]<script>format_filesize(\"[FileSize]\")</script>
[/srclink]</td>
```

Points of interest in [Figure 12.3](#) include

```
1 package document
2
3 _pagescriptextra_ {
4
5   function format\_filesize (gsfilesize)
6   \{
7     var filesize = gsfilesize + " bytes";
8
9     if (gsfilesize > 1048576) \{
10       filesize = Math.round(gsfilesize / 1048576.0) + " MB"
11     \}
12     else if (gsfilesize > 1024) \{
13       filesize = Math.round(gsfilesize / 1024.0) + " KB"
14     \}
15
16     document.write(filesize);
17   \}
18
19   function format\_date (gsdate)
20   \{
21     var year\_num = gsdate.substr(0,4);
22     var month\_num = gsdate.substr(4,2);
23     var date\_num = gsdate.substr(6,2);
24
25     var date = new Date(year\_num,month\_num-1,date\_num)
26     var date\_str = date.toLocaleDateString();
27
28     document.write(date\_str);
29   \}
30 }
```

Figure 12.3: Using JavaScript to present file size and date metadata

Lines 10, 13. Some ambiguity surrounds the values used for kilobytes and megabytes. Is a kilobyte 1000 or 1024 bytes? Is a megabyte 1000 or 1024 kilobytes? We have used 1024 here.

Lines 21–23. The built-in JavaScript function *substr()* extracts a substring: the first argument is the start position and the second is the length of the substring.

Lines 25–26. The built-in JavaScript *Date* object has a member function *toLocaleDateString()*, which converts dates to the appropriate format.

JavaScript uses curly braces { ... } for grouping, but they are reserved characters in Greenstone's macro language. Consequently they are “escaped” by prefixing them with a backslash (\) in order to pass through unaltered to the Web browser's JavaScript interpreter. Underscore is also a reserved character, and is also escaped (for example, in *format_filesize*).

Design pattern 3: Making formats statements reusable through macro definitions

Figure 12.4a demonstrates a video collection whose *DocumentText* format statement has been customized to stream video into the open source Flowplayer, a Flash-based utility giving a similar experience to YouTube. Over a dozen configuration parameters are needed to set this up, and they would



Figure 12.4: A video player in Greenstone: (a) Web browser view;

```

1 package document
2
3 _baseUrl_ {_httpcollection_/index/assoc/_1_}
4
5 _videoplayer_{
6
7 <div class="player">
8   <object type="application/x-shockwave-flash"
9     data="_httpprefix_/ext/video/bin/flash/FlowPlayerLP.swf"
10    width="352" height="290" id="FlowPlayerLP">
11     <param name="allowScriptAccess" value="always" />
12     <param name="movie" value="_httpprefix_/bin/flash/FlowPlayerLP.swf" />
13     <param name="quality" value="high" />
14     <param name="scaleMode" value="showAll" />
15     <param name="allowfullscreen" value="true" />
16     <param name="allowNetworking" value="all" />
17     <param name="flashvars" value="config={\{
18       autoPlay: true,
19       autoBuffering: true,
20       loop: false,
21       initialScale: 'orig',
22       showLoopButton: false,
23       showPlayListButtons: false,
24       allowFullScreen: true,
25       useNativeFullScreen: true,
26       progressBarColor2: '0x308030',
27       progressBarColor1: '0x60d060',
28       videoFile: '_baseUrl_(_1)/_2_',
29       streamingServer: 'lighttpd',
30       thumbsOnFLV: false
31     }\}" />
32   </object>
33 </div>
34 }

```

(b)

```

<object type="application/x-shockwave-flash"
  data="/gsdl/ext/video/bin/flash/FlowPlayerLP.swf"
  width="352" height="290" id="FlowPlayerLP">
  <param name="allowScriptAccess" value="always" />
  <param name="movie" value="/gsdl/bin/flash/FlowPlayerLP.swf" />
  <param name="quality" value="high" />
  <param name="scaleMode" value="showAll" />
  <param name="allowfullscreen" value="true" />
  <param name="allowNetworking" value="all" />
  <param name="flashvars" value="config={
    autoPlay: true,
    autoBuffering: true,
    loop: false,
    initialScale: 'orig',
    showLoopButton: false,
    showPlayListButtons: false,
    allowFullScreen: true,
    useNativeFullScreen: true,
    progressBarColor2: '0x308030',
    progressBarColor1: '0x60d060',
    videoFile: '/gsdl/collect/nzonair/index/assoc/
      HASH7e4d.dir/Duchess_Raglan_City_stream.flv',
    streamingServer: 'lighttpd',
    thumbsOnFLV: false
  }" />
</object>

```

(c)

Figure 12.4, cont'd: (b) macro file; (c) the object created by the macro in (b)

be unwieldy and difficult to manage in a format statement. They can be more conveniently laid out in a macro definition (and decomposed into smaller macros, if desired). But there is a snag—it is necessary to refer to metadata values associated with the video document and, as mentioned previously, this cannot be done directly in macro files.

Figure 12.4b shows how to achieve the best of both worlds by defining a macro to which the metadata values are passed as parameters. It generates the HTML `<object> ... </object>` element shown in Figure 12.4c, which is exactly what is needed to play the video. Most of the code in Figure 12.4b (lines 8–32) defines a Greenstone macro called `_videoplayer_` that generates this HTML. This macro has two arguments, `_1_` and `_2_`; they are used on line 28 and represent metadata values that are needed to construct the video file’s URL. The macro is called from the format statement as follows:

```
format DocumentText " ... _videoplayer_([assocfilepath],
[streamablevideo]) ..."
```

The arguments are two metadata values, `[assocfilepath]` and `[streamablevideo]`. The former is defined for any document that has associated files (such as the video) and provides the name of the folder in which they are stored—in fact, we encountered it in the thumbnail format statement in Section 12.1. The latter metadata is generated by the plug-in that processed the video file, and provides its filename within the associated file directory. These two metadata values are combined with the `_baseurl_` macro defined in Figure 12.4b (line 3) to yield the complete URL of the video. This is then set as the value of the `videoFile` configuration field in the HTML `<object>`. The result is the line near the end of Figure 12.4c that reads

```
videoFile: '/gsdl/collect/nzonair/index/assoc/HASH7e4d.dir/
Duchess_Raglan_City_stream.flv'
```

This macro file is not only easier to create and maintain than embedding all this directly in a format statement, but is in a convenient form to share with other collections and others with Greenstone installations.

Design pattern 4: Dynamic HTML

When a Web page is rendered in a browser, it takes the form of a tree structure in memory that encapsulates the sequencing and nesting of HTML elements. This tree forms an important component of the Document Object Model (DOM), and can be manipulated with JavaScript to dynamically alter what the Web browser displays, usually in conjunction with associated cascading style sheets (CSS). The term “tree” is used in its computer science sense to refer to an abstract structure that is hierarchically composed (as in Figure 11.1).

Figure 12.5 shows an abridged version of the DOM for the introductory HTML example in Chapter 4 (Figure 4.6). HTML elements map to nodes in the tree: nested elements are children of the parent tag that encloses them; sequential elements (such as the `<title>` and two `<meta>` tags at the top left) are siblings, ordered left to right at the same level, with the same parent. Text within an element, such as text to be italicized, also forms a node, shown in the figure as `#text`.

DOM, a World Wide Web Consortium specification, is the key to dynamic HTML. In addition to its hierarchical structuring of HTML elements, it includes methods to access the width and height of the

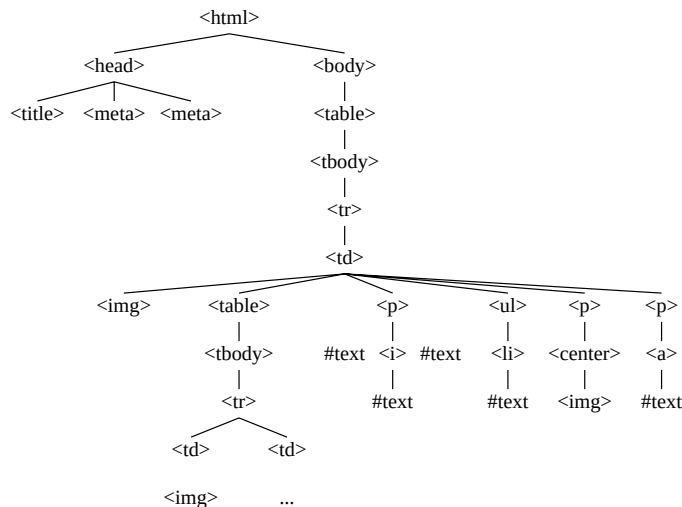


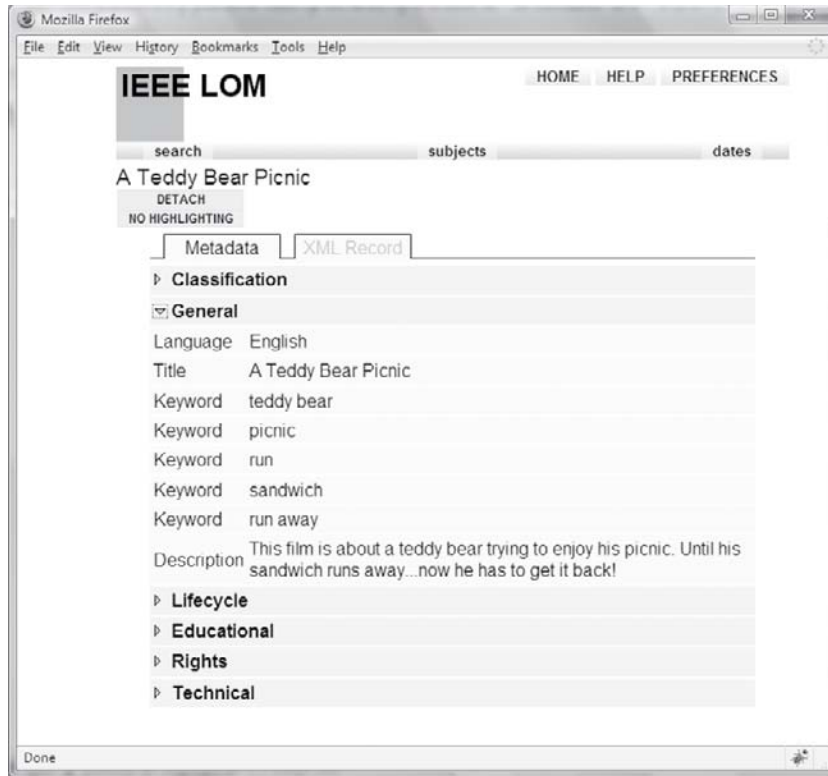
Figure 12.5: Document Object Model for the Web page in Figure 4.6

browser window and to determine whether or not x and y scrollbars are in effect and what their positions are—and it includes ways of changing these values. What we see here is just the tip of the iceberg.

Opening and closing tables interactively

It is often useful to be able to interactively open and close sections of a table, be it a table of contents or a table displaying metadata. For example, Figure 12.6a shows the metadata for an educational resource exported from the University of Calgary’s Learning Commons Educational Object Repository in IEEE LOM format (Section 6.4; this is the same example as in Figure 6.10). Top-level headings appear as emphasized rows in the metadata table—Classification, General, Lifecycle, Educational, Rights, and Technical. Beside each are small triangles ($\triangleright$ and $\triangledown$) that open and close the section. In the figure all are closed except General.

You can place an ID attribute into any HTML tag to mark a salient position in the page with an easily identifiable name that is unique within the file (you can choose whatever name you want); in the example this has been done for each major section of the table in the *DocumentText* format statement. Figure 12.6b shows the JavaScript functions that support the opening and closing operations, *show_description()* and *hide_description()*, respectively, defined using the above-mentioned *_pagescriptextra_* macro. The part of the table to change is passed in as the argument *elem*, and it is located by the built-in DOM function *getElementById()*. The appearance of the part to change (*elem*) is altered by switching its CSS *display* style from *none* (not displayed) to *block* (displayed) and vice versa: lines 7, 9, and 12; and 20, 22, and 25. In fact, there are three things to change. To close a table section, the “open” triangle is located (line 6) and visually suppressed (line 7). Then the “closed” triangle is displayed (lines 8–9). Finally, the section of the table that contains the metadata is suppressed (lines 11–12).



(a)

```

1 package document
2
3 _pagescriptextra_ {
4   function hide\_description(elem)
5     \{
6       var ecl = document.getElementById(elem+"close");
7       ecl.style.display = "none";
8       var eop = document.getElementById(elem+"open");
9       eop.style.display = "block";
10
11       var ete = document.getElementById(elem+"text");
12       ete.style.display = "none";
13
14       return 0;
15     \}
16
17   function show\_description(elem)
18     \{
19       var eop = document.getElementById(elem+"open");
20       eop.style.display = "none";
21       var ecl = document.getElementById(elem+"close");
22       ecl.style.display = "block";
23
24       var ete = document.getElementById(elem+"text");
25       ete.style.display = "block";
26
27       return 0;
28     \}
29 }

```

(b)

Figure 12.6: Opening and closing sections of a table: (a) Web browser view (b) macro file, including JavaScript

For conciseness, the code relies on the convention that the three related items share the same root for the *id* attribute, and it is this root that is passed in as the *elem* argument. With the default being the closed status, the following excerpt of format statement implements the open operation for the section of the table for General metadata:

```
<div id="generalopen" style="display: block;">
  
  <b>General</b>
</div>
```

A similar excerpt is needed for the close operation, and the metadata itself is displayed more simply with:

```
<div id="generaltext" style="display: none;">
<!-- display the various metadata fields General^Language,
General^Title ... -->
</div>
```

Adding table headers

Figure 12.7 shows how to add headers to the information presented when browsing a collection by inserting a row into the table that displays a classifier's VList information. Figure 12.7a shows the result: the headings for Title, Subject, and Description beneath the main banner have been spliced into the document. Figure 12.7b shows the macros that achieve this.

The inclusion of the string *a=d* in the Greenstone digital library URL indicates that the user is either displaying a document or performing a browsing action. Technically, this is called the *document action*: it assigns the value *d* (for document) to the CGI argument *a* (for action). The runtime system can readily distinguish document display from browsing by the other CGI arguments. For the former, the document in question is defined by the *d* argument; for the latter the *cl* argument is used to specify the classifier involved.

The document action (*a=d*) is controlled by macros defined in the package called *document*. Figure 12.7b defines the macros *_pagescriptextra_* and *_pagefooterextra_* (Table 12.3). The first defines two JavaScript functions. The main one, *manipulateVList()*, locates the relevant table in the Web page and adds in the necessary *<th>...</th>* tags for the Title, Subject, and Description table headings. The other, *createVListth()*, avoids repeating the code required to construct the header elements. The second macro, *_pagefooterextra_*, calls *manipulateVList()* to effect the desired transformation. It is placed in the footer to ensure that the main content of the page has been already constructed by the time it is called.

Points of note in the example include:

Line 16. This locates the table in question. Again, *getElementById()* is used, but unlike the tables described above, it relies on the identifier, *group_top*, which is automatically generated by the Greenstone runtime system when creating the top-level table within a VList.

Lines 19–22. This statement overcomes a discrepancy between Firefox and Internet Explorer that showed up during testing. Tables in Internet Explorer contain an extra node (a text node). Unfortunately, such differences are not uncommon when developing DOM-based JavaScript. In this case, it is possible to resolve the issue without needing to test explicitly which browser is being used, which has the added benefit that the code will work on other browsers, whether or not they introduce the extra text node.

There are several software libraries for DOM manipulation that aim to shield programmers from differences between browsers (such as YUI, Yahoo's User Interface library). Our illustrations use unadulterated JavaScript because it is more complicated to explain examples that use libraries.

Lines 41–43. These lines determine if the page has been generated by a classifier by testing whether `_cgargcl_` is defined. Macro code can safely be included within a section of JavaScript (marked with `<script>...</script>` tags) because Greenstone resolves all macros before sending the result to the Web browser. What the browser sees between the tags is correct JavaScript code.

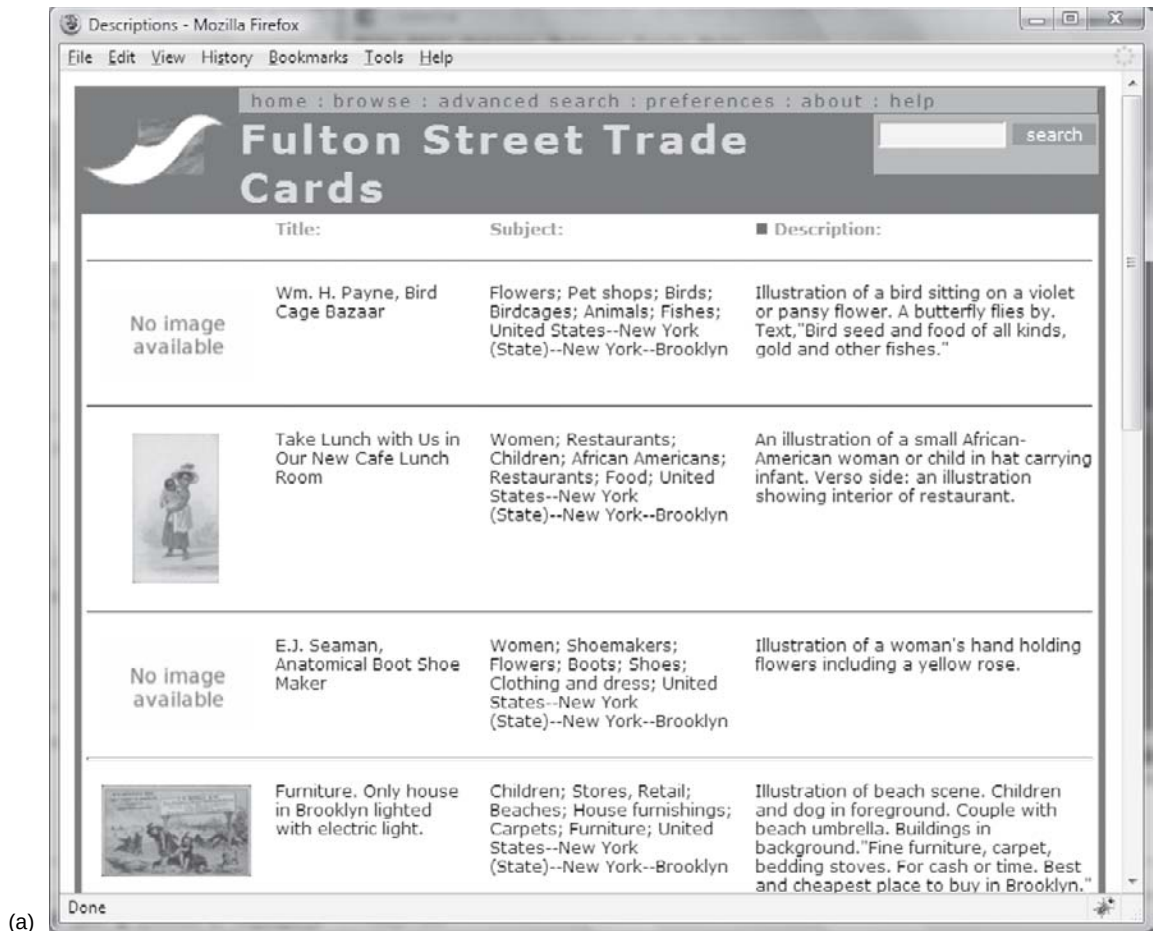


Figure 12.7: Adding header information to a Vlist: (a) Web browser view;

```

1 package document
2
3 _pagescriptextra_ {
4   function createVListth(content)
5   \{
6     var th = document.createElement("th");
7     th.setAttribute("class", "vlistheading");
8
9     th.innerHTML = content;
10
11     return th;
12   \}
13
14   function manipulateVList()
15   \{
16     var vlistTable = document.getElementById("group\_top");
17     var tbody = vlistTable.firstChild;
18
19     if (tbody.nodeName == "#text") \{
20       // cope with Firefox/IE difference
21       tbody = tbody.nextSibling;
22     \}
23
24     var firsttr      = tbody.firstChild;
25     var thIcon       = createVListth("");
26     var thTitle      = createVListth("Title:");
27     var thSubject    = createVListth("Subject:");
28     var thDescription = createVListth("Description:");
29
30     var newtr = document.createElement("tr");
31     newtr.appendChild(thIcon);
32     newtr.appendChild(thTitle);
33     newtr.appendChild(thSubject);
34     newtr.appendChild(thDescription);
35
36     tbody.insertBefore(newtr, firsttr);
37   \}
38 }
39
40 _pagefooterextra_ {
41   <script type="text/javascript">
42     _If_(\_cgiargcl_, manipulateVList());
43   </script>
44 }

```

(b)

Figure 12.7, cont'd: (b) macro file, including JavaScript

Design pattern 5: Exploiting Asynchronous JavaScript and XML (AJAX)

As we have seen, some nifty effects can be created using DOM manipulation. They give Web pages an interactive feel—for example, users can open and close sections of tables without reloading the page. But these adjustments can use only information stored in the page when it was first generated or information provided by the user when interacting with it. AJAX adds the ability to communicate with the server that produced the page. This greatly extends the range of what is possible:

- A user interaction can prompt a request to the server for further information that is used to update the display.
- Changes in the state of the interface can be sent to the server incrementally and can be stored there.
- Multi-way communication channels routed through the server can support simultaneous collaboration with other users.

For example, AJAX could be used to support a more responsive form of query refinement or faceted browsing, avoiding the expense of reloading pages when users narrow the focus of their search. Less data is transmitted overall because the server need only provide information about matching documents that have not already been sent.

Our first example below returns to the Depositor, Greenstone's institutional repository system, and shows how AJAX is used to support the checksum feature. A surprisingly short piece of JavaScript is needed to effect such a radical change in how the Web is used. The second example is a digital music stand that is integrated into Greenstone. Rather than going into technical details, we focus on the bigger picture and describe the role that AJAX plays.

Server-side checksums

Under the discussion of institutional repositories in Section 11.6, we saw how Greenstone can be configured to provide an author-submission workflow comparable to that of DSpace (described in Section 7.6). Most of this is accomplished using features of the macro language that we have studied above. However, it is a little more complicated to calculate the checksum of the uploaded file. To do this, a small server-side CGI program called *checksum* was written; AJAX was used to call it from within the Web page; and the DOM was updated with the data returned. [Figure 12.8](#) shows the page before and after clicking the “calculate ...” hyperlink embedded in the table.

[Figure 12.8c](#) gives the necessary JavaScript. The function *urlGetAsync()*, defined at the beginning, provides a general routine for AJAX communication that can be used elsewhere in the digital library. Its first argument is the URL of the CGI program to communicate with on the server (more than one such program could exist on the server, to augment functionality in different ways). The second argument names the DOM element to be altered as a result of calling the server. With this general routine established, the checksum feature is accomplished by *add_checksum(filename)* (lines 43–46), whose *filename* argument gives the name of the uploaded file on the server. The macro *_httpchecksum_* is defined at the beginning (line 3): it resolves to the URL of the CGI-executable program that computes the checksum based on the file name provided.

There are different ways to transmit data to a Web server and receive the result; the name *urlGetAsync()* reflects the choices made here. The GET method is the standard way to send data on the Web and is suitable for lightweight data items such as numbers or short strings—it produces the CGI arguments we often see in URLs, such as *a=p&p=about*. (There is an alternative called POST that is typically used for transmitting heavyweight data, such as files.) Data can be returned either synchronously or asynchronously. In the former case, the calling program (i.e., the JavaScript in the Web page) makes a connection with the server and awaits a response from it. In the latter case, execution continues and the server returns the result by calling a stipulated “callback function”; this is what *urlGetAsync()* uses.

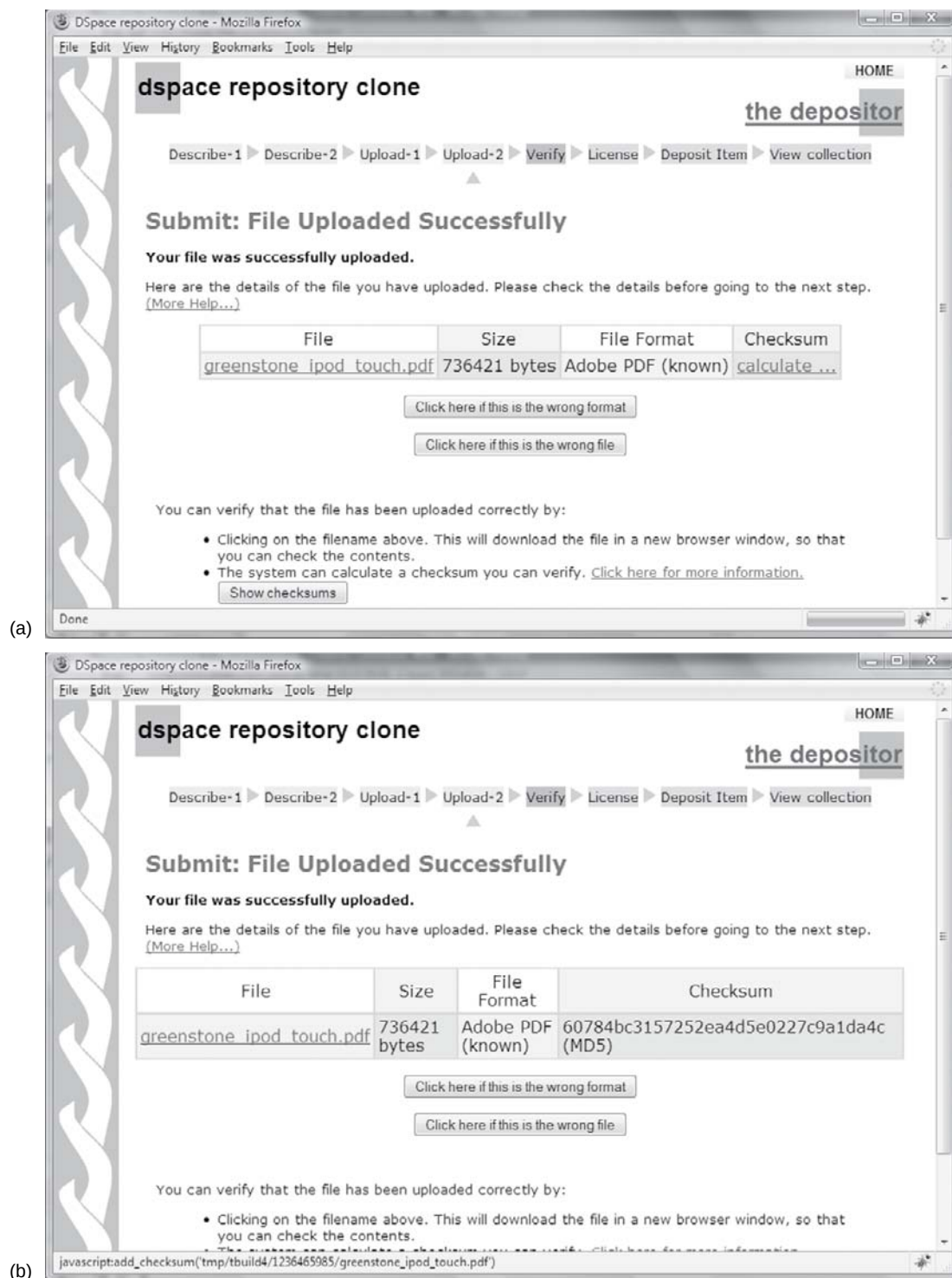


Figure 12.8: Using AJAX to retrieve information from the server: (a) before the checksum is calculated; (b) after the checksum is calculated;


```

1 package depositor
2
3 _httpchecksum_ {_httpprefix_/cgi-bin/checksum.cgi}
4
5 _pagescriptextra_ {
6
7     function urlGetAsync(url,elem)
8     \{
9         var xmlHttp;
10        try \{
11            // Firefox, Opera 8.0+, Safari
12            xmlHttp=new XMLHttpRequest();
13        \}
14        catch (e) \{
15            // Internet Explorer
16            try \{
17                xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
18            \}
19            catch (e) \{
20                try \{
21                    xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
22                \}
23                catch (e) \{
24                    alert("Your browser does not support AJAX!");
25                    return false;
26                \}
27            \}
28        \}
29
30        xmlHttp.onreadystatechange=function() \{
31            if(xmlHttp.readyState==4) \{
32                if (locelem != null) \{
33                    var locelem = document.getElementById(elem);
34                    locelem.innerHTML = xmlHttp.responseText;
35                \}
36            \}
37        \}
38
39        xmlHttp.open("GET",url,true);
40        xmlHttp.send(null);
41    \}
42
43    function add\_checksum(filename)
44    \{
45        urlGetAsync("_httpchecksum_?filename="+filename,"pane1");
46    \}
47 }

```

(c)

Figure 12.8, cont'd: (c) AJAX JavaScript code for asynchronous loading

The function `urlGetAsync()` first primes the variable `xmlHttp` for server-side communication. This object is created in different ways for different browsers, accounting for the bulk of the code to this routine (lines 9–28). Lines 30–37 provide the callback function, and lines 39–40 transmit the data. Callback functions can do many sophisticated things, but this one is simple: it takes the second parameter `elem` and sets its HTML to whatever the server returns. Incidentally, the call could be made synchronous by changing `xmlHttp.open()`'s third argument from `true` to `false` (line 39). The callback function on lines 30–37 would then be unnecessary and should be removed.

All that remains is to set up a hyperlink to invoke the checksum calculation. When discussing Figure 11.11 in Section 11.6, we explained that a different macro is used to define the page content for each step of the process of depositing a new item in the repository. These are called *step1content*, *step-2content*, ... Step 4 is the one that creates Figure 12.8a, and the “calculate ...” hyperlink embedded in the table is generated by

```
<a href="javascript:add\_checksum('tmp/\_cgiargdi1tmp\_/  
\_cgiargdi1timestamp\_/\_diluserfile\_')">calculate ...</a>
```

This calls the JavaScript *add_checksum* function with the appropriate filename as its argument. Files uploaded to the digital library Web server are placed in a subfolder of *tmp* that encodes the time-stamp when they arrived. The macros *_cgiargdi1tmp_*, *_cgiargdi1timestamp_*, and *_diluserfile_* encode the folders and filename used.

Digital music stand

In Section 10.6 (Non-textual documents) we show the basic mechanism Greenstone provides for displaying documents that comprise paged images. The facilities provided are rather austere and also a little cumbersome, because the browser loads a new Web page every time the user turns the page. Figure 12.9 illustrates a digital music stand that is integrated with Greenstone. It is also page based, but uses AJAX to create a smoother interaction with more capabilities. To reach this screen, the user has located a piece of music through the standard browsing and searching features; it is only when she has clicked to view the document that the differences become apparent. For instance, the digital music stand supports textual annotations: a non-destructive—and clearly typed!—analog of the hasty penciled notes that all musicians make. Halfway down the page in Figure 12.9a is a warning to watch the fingering. Notes can be positioned anywhere on the page.

The digital music stand also supports an animated fast-to-slow page wipe. Turning pages is a critical problem for performing musicians, who are already under real-time stress and may be sight-reading the music for the first time. Turning a physical page suddenly obscures the current one, but in this electronic version the next page is overlaid gradually. The transition can be seen as a full-width horizontal bar halfway down the page in Figure 12.9b (colored red, in fact) that wipes downward, carrying the new page with it—so that the top of the next page can be seen along with the bottom of the current one. The motion is fast at first but slows down on reaching the point marked by the scroll-bar (on the right). This gives the musician plenty of time to finish playing the last line of the current page. Even if she reaches the end of the last few notes on the current page before the transition has completed, there is no need to panic because the beginning of the next page has already appeared at the top.

By default, the transition point from fast to slow is between the last two staff systems on the page. If the user changes this, the value is immediately transmitted (via an AJAX call) to the server and stored as metadata with that document, so that on returning to this page (later in the performance, or on a different day) the user’s preference is restored. Likewise, annotations are stored as metadata. AJAX is also used to asynchronously load next and previous pages in the background before they are needed, and the dimensions of the user’s screen window are sent to the server so it can generate an appropriately paginated version that makes full use of the available space.

k523-a4: 1 - Mozilla Firefox

go to page of 5 pages 2

Abendempfindung (KV 523)

für eine Singstimme mit Begleitung des Pianoforte

Wolfgang Amadeus Mozart (1756-1791)

Andante.

Singstimme: A - bend ist's, die Sch - ne ist ver-

Pianoforte: *p*

schwin - den und der Mond strahlt Sil - ber-glantz;

Watch fingering

so ent-flieh'n des-Le-bens-schön-ste-Stun-den, flieh'n vor-ü-ber wie im Tanz.

Bald ent-flieht des-Le-bens-lun-te-Sce-ne, und der Vor-hang rollt her-ab;

Public Domain

Done

(a)

Figure 12.9: An AJAX-based digital music stand: (a) adding an annotation;

k523-a4: 1 - Mozilla Firefox

go to page 2 of 5 pages

2

aus ist un-ser Spiel, des Freun-den Thrä-ne-

file-aset schon auf un-ser Grab. Bald viel-leicht mir

Watch fingering

so ent-flieh'n des-Le-bens-schön-ste-Stun-den, flieh'n vor-ü-ber wie im Tanz.

Bald ent-flieht des-Le-bens-bun-te-Sce-ne, und der Vor-hang rollt her-ab;

Public Domain

Done

(b)

Figure 12.9, cont'd: (b) an animated page-wipe

12.3 The Greenstone Research Project

The Greenstone digital library software grew out of an ongoing project at the University of Waikato in New Zealand. It has been widely used internationally, and (as noted in the Preface) a joint project with UNESCO is promoting its use for practical digital libraries in developing countries. As a result, members of the project have strong motivation and a great deal of personal satisfaction. However, widespread adoption of Greenstone also has the side effect of suppressing innovation. There is a disincentive to introduce new research-led concepts and facilities into a production system because they are less reliable, difficult to test on multiple platforms, and often entail substantial upheavals in the software. Because of Greenstone's wide range of applications and computer platforms, modest changes can introduce bugs that affect users in unforeseen ways. Moreover, the interface and documentation have been translated into many languages, which itself discourages new development.

Research with Greenstone3

Our solution has been to create a new version of the software with the express intention that it constitute a framework for innovative research. We named it Greenstone3 to differentiate it from the production version described in this book, retrospectively dubbed Greenstone2 to coincide with the version number we had already been using. Extensive backward-compatibility was built in right from the start—Greenstone is the digital analog of the physical library building, and you don't have to rewrite the books or card catalogs when libraries move to new premises! Consequently, Greenstone3 can run collections built for earlier versions of Greenstone without any modification whatsoever. Other goals include different levels of customization, software modularity, dynamic services, distributed architecture, future compatibility, integrated documentation, self-describing modules, and integration into the computing environment.

Greenstone3 is a service-based architecture. A network of modules that provide these services communicate with each other via XML messages, which can optionally be set up to communicate in a distributed fashion using the Simple Object Access Protocol (SOAP) described in Section 7.4. Messages can be customized at any point using XSL transformations (mentioned in Section 4.4). Dynamically loadable Web services are layered on top of the communication channel. A “describe-yourself” call is a mandatory fixture that allows services to be discovered in a heterogeneous world of communicating applications; it also assists future compatibility.

From a research viewpoint, Greenstone3 has been a resounding success. We have used it to investigate such things as dynamic text mining, query visualization, alerting services, spatial searching, ontologies, and many more. We are capitalizing on the contents of digital libraries to automate the production and delivery of practice exercises for students who are learning English. We are exploring the role of AJAX in digital libraries to create richly interactive interfaces—for example, it is used for the Koru knowledge-based information retrieval system described in Section 9.3. This research tests the ability to provide future compatibility, because the rise of AJAX postdates Greenstone3's design.

Greenstone3 is an open source project and is under active development. However, demands from users for further enhancements to Greenstone2 have drawn more heavily on our time than we had foreseen when we embarked upon the re-implementation.

Reconciling research and production values

Greenstone3 was envisaged purely as a research framework: its use would require IT skills. It has achieved many of its goals—for example, it is now far easier for external projects to build upon the digital library core, as in the examples mentioned above. By and large, these external projects develop new or enhanced functionality that augments the document searching, browsing, and access already provided. Some projects, notably ones utilizing AJAX, produce new Web-browser interfaces that issue requests to the digital library infrastructure and manipulate the XML messages that are returned. Others develop standalone applications (using the programming language and toolkit that suit their needs) that integrate with the infrastructure at as deep a level as required. In applications that run on the same computer as the digital library system, the communication protocol can be bypassed in favor of direct procedure calls using an application programming interface.

We have found that maintaining two independent versions of the software—in particular, ensuring backward-compatibility when new and enhanced features are still being added to Greenstone2—stretches our resources too thinly. Consequently, we are now developing Greenstone3 to the point that, by default, users will find its installation and operation indistinguishable from those of Greenstone2. For example, the original Greenstone3 was installed by obtaining the code from a software repository and compiling it for your machine. Now, however, it uses Greenstone2's cross-platform installation wizard, which allows end users to install it easily. Greenstone's interface translations into different languages have been migrated to Greenstone3. Enhancements have been made to the default services so that what readers see through a Web browser can be matched, step for step, with what they already experience with Greenstone2. Automated regression testing helps minimize the problem of checking for undesired side effects of recently committed changes. Workshop and tutorial material have been tested to ensure that they work equally well with Greenstone3.

In terms of ingesting documents into a collection, we now utilize Greenstone2's exporting ability, which was originally designed to facilitate interoperability, to output to the registered METS profile utilized by Greenstone3. Greenstone2's collection-building code was upgraded to parse Greenstone3 configuration files, which allows the Librarian interface to operate equally well with either version. Overall, this pattern of gradually moving the two code bases closer together seems to be working well. For example, the realistic book facility illustrated in Figure 3.5 was originally planned for Greenstone3 alone. However, it turned out that an extra optional CGI parameter could be introduced into Greenstone2 that made it generate XML in the Greenstone3 document format, which allowed the work to be “back ported” to Greenstone2 and used in any existing collection.

There is one area where Greenstone3 still differs radically from Greenstone2, and that is in the format statements and macro language described in this chapter. They are based on XSL transformations, a mechanism that is more powerful but, unfortunately, even more difficult to use. However, XSL transformations can more readily support automatic construction of format statements and macros. We are designing an interactive editor for Greenstone digital library pages that will eventually render format statements and macros obsolete—much to the relief of users. However, this is a major undertaking, which is why we have based this book on Greenstone2 rather than the newer version. From the user's point of view, all other changes when upgrading to Greenstone3 will be

invisible. In fact, you might like to download it (from www.greenstone.org/greenstone3-home) and see for yourself.

Looking to the future, our overarching goal is to broaden the realms in which digital libraries are utilized. We are now focusing on multimedia and enhancing Greenstone to incorporate content-based analysis for images, audio, and video, with particular specializations to newspapers, music, and maps. We foresee an extension mechanism as the key to managing this ever-widening and diversifying spectrum of research-inspired functionality, and we will produce an administration tool that facilitates general maintenance of an installation along with the ability to download and incorporate particular extensions.

Closing words

We believe that digital libraries are a key aspect of civil society. Their importance grows with every passing day. People are beginning to recognize the dangers of relying on one or two giant universal search engines for access to our society's treasure-house of information—which constitutes our entire literary, scientific, and cultural heritage. What the world needs, we believe, are focused collections of information, created and curated by people with an intellectual stake in their contents. We mean digital libraries, built by librarians!

We are proud and very pleased that the Greenstone Digital Library Software is playing a part in this international endeavor. Year by year, its technical strength grows and its rate of adoption increases. Yet this success has come at a price: although we are at heart a research group, we feel obliged to provide support for the growing user base of our production system—particularly in developing countries. This final section has given an insight into the development processes of the Greenstone software and conveyed our approach to the difficult problem of reconciling the production values of stability and reliability with a vigorous research framework of innovation in digital library software.

Glossary

AAC	An audio format designed by Apple as a successor to MP3 and used in its iPod and iTunes products
AACR2R	<i>Anglo-American Cataloging Rules</i> (second edition, revised), published by the American Library Association
ASCII	American Standard Code for Information Interchange, a 1968 standard 7-bit code for representing the Roman alphabet plus numerals and special symbols
Authority control	The practice of using a list of preferred terms or proper names for metadata values within a library catalog
BibTeX	Scheme for managing bibliographic data and references within documents in the TeX format
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, and keywords)
Building	Process of creating the indexing and browsing structures that are used to access a collection
Case-folding	Making uppercase and lowercase words look the same, for searching purposes
CD-ROM	Compact disk read-only memory, a 650-MB disk that uses the same technology as audio CDs
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 clicked

Collection	Set of documents that are brought together under a uniform searching and browsing interface
Collection configuration file	File that specifies how a Greenstone collection is to be imported and built, what indexes and language interfaces are to be provided, and so on
Cookie	A short message sent by a Web server to a Web browser and then returned unchanged by the browser each time it accesses that server, the purpose being to retain a unique identifier for the user between invocations
CSS	Cascading Style Sheets, a way of controlling the presentation of HTML and XML documents
DDL	Description Definition Language, a form of XMLSchema used in MPEG-7 to allow users to create their own metadata format
Digital library	Collection of digital objects (text, audio, video), along with methods for access and retrieval, as well as methods for selection, organization, and maintenance
Document	Basic unit from which digital library collections are constructed, which may include text, graphics, sound, and video
DOI	Digital Object Identifier, a scheme for permanent URLs that uses the Handle system and is maintained by the International DOI Foundation
DSpace	Open source institutional repository software, designed by the MIT Library in conjunction with Hewlett-Packard (Document Space)
DTD	Document Type Definition, a specification used in XML (and also SGML) to express the structure of a particular set of documents
Dublin Core	An intentionally minimalist standard for describing metadata, designed to be applied to resources on the Web
DVD	Digital Versatile Disk, a disk format that can hold from 5 to 20 GB of data
Dynamic HTML	Combination of HTML with CSS, Web-page scripting, and the document object model that provides functionality comparable to XSL
EBCDIC	Extended Binary Coded Decimal Interchange Code, an alternative code to ASCII
Encapsulated PostScript	Variant of PostScript designed for expressing graphics of a single page or less that are to be included in other documents

Endnote	Commercial program for managing bibliographic data and references within documents
Faceted classification	Browsing scheme that allows users to navigate the document space using different metadata elements or “facets”
FAO	United Nations Food and Agricultural Organization
Fedora	A repository architecture founded upon a powerful digital object model, which is extremely flexible and configurable (Flexible Extensible Digital Object Repository Architecture)
FLAC	Lossless audio coding scheme (Free Lossless Audio Codec)
FO	Formatting Objects, the component of the XML specification that deals with page layout and related formatting issues
FTP	File transfer protocol
GB-encoding	Standard way of encoding the Chinese language
GIF	Graphics Image Format, a widely used compression scheme for lossless images
GNU Public License	Software license that permits users to copy and distribute computer programs freely, and to modify them, so long as all modifications are made publicly available
Greenstone	The name of the digital library software used as an example in the book (www.greenstone.org)
HTML	HyperText Markup Language, the language in which Web documents are written
HTML Tidy	Software utility that converts older HTML formats to XHTML
HTTP	Hypertext transfer protocol
Hyperlink	Link to another document or to another place within this document
Importing	Process of bringing collections of documents into a digital library system
Index	Information structure that is used for searching or browsing a collection
Institutional repository	A specialized kind of digital library designed to collect, preserve, and disseminate an institution’s intellectual output
ISBN	International Standard Book Number

ISCII	Indian Script Code for Information Interchange, an ASCII extension that incorporates Brahmi-based Indic scripts
JPEG	Standard for (mainly) lossy image compression, named after the Joint Photographic Experts Group
JPEG 2000	Later version of the JPEG image compression standard
LCSH	Library of Congress Subject Headings, a controlled vocabulary for assigning subject descriptors
LZW	Lempel-Ziv-Welch compression scheme, used in GIF, TIFF, PDF, and PostScript Level 2 and above
MADS	Metadata Authority Description Schema, an XML-based format for authority control information
MARC	Machine-readable cataloging format, a metadata scheme designed in the late 1960s for use by professional library catalogers
MARCXML	A standard way of representing MARC data in the XML language
Metadata	Structured information, such as author, title, date, keywords, and so on, that is associated with a document (or document collection)
METS	Metadata Encoding and Transmission Standard for the representation, maintenance, and exchange of digital objects
MIDI	Musical Instrument Digital Interface, a representation of music used by music synthesizers
MIME	Multipurpose Internet Mail Extensions, a standard for including different types of file—text, images, audio, video, or application-specific data—in e-mail messages
Mirror	The process of copying a Web site, or part of a Web site, to another location and making it available there
MP3	A lossy digital audio encoding format (MPEG-1 Audio Layer 3)
MODS	Metadata Object Description Schema, an XML-based format for metadata from the U.S. Library of Congress
MPEG	Set of standards for representing multimedia material, named after the Motion Picture Experts Group
MXF	Material Exchange Format, a container format for video data
New Zealand Digital Library (NZDL) Project	Research project in the Computer Science Department at the University of Waikato, New Zealand, that created the Greenstone software (www.nzdl.org)

OAI-ORE	Standard designed expressly for representing aggregations of digital objects (Open Archives Initiative Object Reuse and Exchange)
OAI-PMH	Protocol designed for the efficient dissemination of digital library metadata (Open Archives Initiative Protocol for Metadata Harvesting; often abbreviated to simply OAI)
OCR	Optical Character Recognition, the process of producing a digital representation of the textual content of a document image
ODF	Open Document Format, an XML-based set of specifications for office documents
Ogg Vorbis	Lossy digital audio encoding format specifically designed to be free of all patent restrictions
Ogg Theora	Lossy digital video encoding format specifically designed to be free of all patent restrictions
OID	Object Identifier, a unique identification code associated with a document or other digital object
OMR	Optical Music Recognition, the musical analog of OCR
OOXML	Office Open XML, an XML-based format for representing documents in Microsoft Office products
Phrase query	Query to a full-text retrieval system that comprises several consecutive words
PDF	Portable Document Format, a page description language designed for interactive use as a successor to PostScript
PNG	Portable Network Graphics, an open standard for lossless images
PostScript	The first widely used page description language
PREMIS	PREservation Metadata Implementation Strategies, a standard for preservation and administrative metadata
Protocol	Set of conventions according to which two systems communicate
PURL	Persistent URL, a particular style of URN
Ranked query	Natural-language query to an information-retrieval system, for which the documents that match the query are sorted in order of relevance
RDF	Resource Description Framework, a scheme designed to facilitate the interoperability of metadata

RSLP collection description	A particular application of RDF by the Research Support Libraries Program based at the University of Bath, United Kingdom
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)
SGML	Standard Generalized Markup Language, a metalanguage for describing markup formats that was standardized in 1986 and forms a precursor to XML
SQL	Structured Query Language, an industry-standard database-query language
SRU	Search/Retrieval via URL, a Web service version of Z39.50
Stemming	Stripping endings off a term to make it more general
SVG	XML-based language for describing two-dimensional graphics
TEI	Text Encoding Initiative, a project founded in 1987 that developed SGML DTDs for representing scholarly texts in the humanities and social sciences
TIFF	Tagged Image File Format, a file format for raster images that incorporates facilities for descriptive metadata
UCS	Unicode Character Set, the set of characters supported by Unicode
UNESCO	United Nations Educational, Scientific and Cultural Organization
Unicode	Standard scheme for representing the character sets used in the world's languages
URI	Universal Resource Identifier, a generic name for URLs and URNs
URL	Universal Resource Locator, a standard way of addressing objects on the Web (but this term is supposed to be superseded by URI)
URN	Universal Resource Name, a way of naming resources instead of specifying their locations
UTF	UCS Transformation Format, a scheme for representing Unicode characters with three variants: UTF-32, UTF-16, and UTF-8
VRML	Virtual Reality Modeling Language, used for presenting virtual reality experiences on the Web
Web server	Standard program that computers use to make information accessible over the World Wide Web

Word	Microsoft Word, a widely used word processing program
XHTML	Modern version of HTML that incorporates the stricter syntactic rules of XML
XLink	XML linking language that provides a more powerful method for connecting resources than HTML hyperlinks
XML	Extensible Markup Language, a metalanguage for describing markup formats for structured documents and data on the Web
XML Schema	Way of specifying the structure of a particular set of documents that provides more expressive facilities for structures and data types than DTDs
XMP	Extensible Metadata Platform, a way of embedding metadata within files like images and PDF documents
XPath	XML specification that allows the selection of parts of a document
XPointer	Development of XPath that provides a finer degree of control over the parts of the document being selected
XQuery	Format used for expressing queries in the style of XML
XSL	Extensible Stylesheet Language, a way of controlling the presentation of XML documents (and XML-compliant HTML ones) that is more expressive than CSS
XSLT	XSL Transformations, a component of the XML specification that allows you to manipulate parts of the document tree
Z39.50	International standard communications protocol developed for use by library catalog systems

References

- ABBYY Software (2000) *FineReader User's Guide*. ABBYY Software, 123015 Moscow, P.O. 72, Russia.
- Adobe Systems Incorporated (1985) *PostScript Language Tutorial and Cookbook*. Addison Wesley, Boston, MA.
- Adobe Systems Incorporated (1999) *PostScript Language Reference*, third edition. Addison Wesley, Boston, MA.
- Adobe Systems Incorporated (2000) *PDF Reference*, second edition (version 1.3). Addison Wesley, Boston, MA.
- American National Standards Institute (1968) *American Standard Code for Information Interchange (ASCII)*, Standard No. X3.4-1968; updated as X3.4-1986.
- Andrews, N. (1987) "Rich text format standard makes transferring text easier." *Microsoft Systems Journal*, Vol. 2, No. 1, pp. 63–67.
- Anderson, C.G. (2006) *Ethical Decision Making for Digital Libraries*. Chandos Publishing, Oxford, UK.
- Apperley, M., Cunningham, S. J., Keegan, T., and Witten I. H. (2001) "Niupepa: A historical newspaper collection." *Communications of the ACM*, Vol. 44, No. 5, pp. 86–87.
- Apperley, M., Keegan, T., Cunningham, S. J., and Witten, I. H. (2002) "Delivering the Maori language newspapers on the Internet." In *Rere Atu Taku Manu! Discovering History Language and Politics in The Maori Language Newspapers*, (Ed.) J. McRae, pp. 211–236. Auckland University Press, New Zealand.
- Arms, W. Y. (2000) *Digital Libraries*. MIT Press, Cambridge, MA.
- Arunachalam, S. (1998) "How the Internet is failing the developing world." Presented at *Science Communication in the Next Millennium*, Egypt; June. Available at www.abc.net.au/science/slab/infopoverty/story.htm.
- Australian Library and Information Association (2006) "She made her MARC." *InCite*, Volume 27, p. 13. Australian Library and Information Association; September.
- Baeza-Yates, R., and Ribeiro-Neto, B. (1999) *Modern Information Retrieval*. ACM Press, New York.
- Bainbridge, D., and Cunningham, S. J. (1998) "Making oral history accessible over the World Wide Web." *History and Computing*, Vol. 10, No. 1/3, pp. 73–81.
- Bainbridge, D., Nevill-Manning, C., Witten, I. H., Smith, L. A., and McNab, R. J. (1999) "Towards a digital library of popular music." *Proc ACM Digital Libraries*, Berkeley, CA, pp. 161–169. ACM Press, New York.

- Bainbridge, D., Cunningham, S. J., and Downie, J. S. (2003) "Analysis of queries to a Wizard-of-Oz MIR system: Challenging assumptions about what people really want." *Proc ISMIR*. <http://ismir2003.ismir.net/papers/Bainbridge.pdf>.
- Bainbridge, D., Jones, S., McIntosh, S., Jones, M., and Witten, I. H. (2008) "Portable digital libraries on an iPod." *Proc. Joint Conference on Digital Libraries*, pp. 333–336. Pittsburgh PA. ACM Press, New York.
- Barbaro, M., and Zeller Jr., T. (2006) "A face is exposed for AOL searcher No. 4417749." *New York Times*, 9th August. <http://www.nytimes.com/2006/08/09/technology/09aol.html>.
- Barnello, I. (1996) "The changing face of reference: a history of the future." In *The Changing Face of Reference*, (Eds.) L. Stuart and D. Hutto, pp. 3–17. JAI Press, Greenwich, CT.
- Bates, M. J. (1989) "The design of browsing and berrypicking techniques for the online search interface." *Online Review* Vol. 13, No. 5, pp. 407–424.
- Beall, J. (2006) "Metadata and data quality problems in the digital library." *J. Digital Information*, Volume 6, Number 3. <http://journals.tdl.org/jodi/article/view/65/68>.
- Bekaert, J., Hochstenbach, P., and Van de Sompel, H. (2003) "Using MPEG-21 DIDL to represent complex digital objects in the Los Alamos National Laboratory Digital Library." *D-Lib Magazine*, Volume 9, Number 11; November. <http://www.dlib.org/dlib/november03/bekaert/11bekaert.html>.
- Benetton, L. (2009) Properly redacting documents crucial, *The Lawyers Weekly*, Vol. 28, No. 1., March 13 2009. <http://www.lawyersweekly.ca/index.php?section=article&articleid=876>.
- Beranek, L., and Burke, L. (2006) "Call us: development of a library telephone enquiry service." *Australian Library Journal*, Vol. 55, No. 3, pp. 211–221.
- Blashill, P. (2001) The Creative Processor, *Wired*, Issue 9.09, Sept. 2001. <http://www.wired.com/wired/archive/9.09/widmer.html>.
- Borgman, C. L. (2000) *From Gutenberg to the Global Information Infrastructure: Access to Information in the Networked World*. MIT Press, Cambridge, MA.
- Bowker, R. R. (1883) "The work of the nineteenth-century librarian for the librarian of the twentieth." *Library Journal*, Vol. 8, pp. 247–250; September–October.
- Bush, V. (1945) "As we may think." *The Atlantic Monthly*, Vol. 176, No. 1, pp. 101–108.
- Cavnar, W. B., and Trenkle, J. M. (1994) "N-Gram-based text categorization." *Proc Symposium on Document Analysis and Information Retrieval*, Las Vegas, NV, pp. 161–175.
- Chang, S. J., and Rice, R. E. (1993) "Browsing: A multidimensional framework." *Annual Review of Information Science and Technology*, Vol. 28, pp. 231–276.
- Chavez, R., Crane, G., Sauer, A., Babeu, A., Packel, A., and Weaver, G. (2007). Services make the repository. *Journal of Digital Information*, 8, (2). <http://journals.tdl.org/jodi/article/view/195/179>.
- Chapman, N., and Chapman, J. (2000) *Digital Multimedia*. Wiley, New York.
- Chen, S. S. (1998) *Digital Libraries: The Life Cycle of Information*. BE (Better Earth) Publisher, Columbia, MO.
- Clinton, W., and Gore, A., Jr. (1993) *Technology for America's Economic Growth: A New Direction to Build Economic Strength*. Executive Office of the President, Washington, DC; February.
- Cole, T.W., and Foulonneau, M. (2007) *Using the Open Archives Initiative Protocol for Metadata Harvesting*. Libraries Unlimited, Westport, CT, USA.

- Committee on Intellectual Property Rights, Computer Science and Telecommunications Board (2000) *The Digital Dilemma: Intellectual Property in the Information Age*. National Academy Press, Washington, DC.
- Cooper, M. D. (1996) *Design of Library Automation Systems*. Wiley, New York.
- Courtney, N. (2003) "Unaffiliated users' access to academic libraries: A survey." *Journal of Academic Librarianship* Vol. 29, No. 1, pp. 3–7.
- Crawford, W., and Gorman, M. (1995) *Future Libraries: Dreams, Madness, and Reality*. American Library Association, Chicago.
- Crossley-Holland, K. (Trans.) (1968) *Beowulf*. Folio Society, London.
- Cundiff, M.V. (2004) "An introduction to the Metadata Encoding and Transmission Standard (METS)." *Library Hi Tech*, Volume 22, Number 1, pp. 52–64.
- Dappert, A., and Enders, M. (2008) "Using METS, PREMIS and MODS for archiving eJournals." *D-Lib Magazine*, Vol. 14, No. 9/10. <http://www.dlib.org/dlib/september08/dappert/09dappert.html>.
- Darlington, J., Finney, A. and Pearce, A. (2003) "Domesday Redux: The rescue of the BBC's Domesday Project videodiscs." *Ariadne*, Issue 36, July. <http://www.ariadne.ac.uk/issue36/tna/>.
- Davidson, C. (1993) "The man who made computers personal." *New Scientist*, No. 1878, pp. 32–35; June.
- DeLong, M. B., Brady, J. G., Bourgeois, L. D., and Niemiec, L. J. (1987) *First Aid in Pictures*. Vade Mecum Press, Sterling, VA.
- Droettboom, M., MacMillan, K., and Fujinaga, I. (2003) The Gamera framework for building custom recognition systems. *Symposium on Document Image Understanding Technologies*, pp. 275–286.
- Dumais, S. T., Platt, J., Heckerman, D., and Sahami, M. (1998) "Inductive learning algorithms and representations for text categorization." *Proc ACM Conf on Information and Knowledge Management*, Bethesda, MD, pp. 148–155. ACM Press, New York.
- Duncker, E. (2002) "Cross-cultural usability of the library metaphor." *Proc Joint Conference on Digital Libraries*, Portland, OR, pp. 223–230. ACM Press, New York.
- Foley, J. D., van Dam, A., Feiner, S. K., and Hughes, J. F. (1990) *Computer Graphics: Principles and Practice in C*. Addison Wesley, Boston, MA.
- Frakes, W. B. (1992) "Stemming algorithms." In Frakes and Baeza-Yates (1992), Chapter 8, pp. 131–160.
- Frakes, W. B., and Baeza-Yates, R. (Eds.) (1992) *Information Retrieval: Data Structures and Algorithms*. Prentice Hall, Englewood Cliffs, NJ.
- Frank, E., Paynter, G. W., Witten, I. H., Gutwin, C., and Nevill-Manning, C. (1999) "Domain-specific keyphrase extraction." *Proc International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, pp. 668–673. Morgan Kaufmann, San Francisco, CA.
- Freire, J., Koop, D., Santos, E., and Silva, C. T. (2008) "Provenance for computational tasks: A survey." *Computing in Science and Engineering*, Volume 10, Number 3, pp. 11–21.
- Gaines, B. R. (1993) "An agenda for digital journals: The socio-technical infrastructure of knowledge dissemination." *Journal of Organizational Computing*, Vol. 3, No. 2, pp. 135–193.
- Giles, C. L., Bollacker, K. D., and Lawrence, S. (1998) "CiteSeer: An automatic citation indexing system." *Proc ACM Digital Libraries*, Pittsburgh, PA, pp. 89–98. ACM Press, NY.

- Gilliland-Swetland, A. J. (1998) "Setting the Stage: Defining Metadata." In *Introduction to Metadata: Pathways to Digital Information*, (Ed.) M. Baca, pp. 1–8. Getty Information Institute, Los Angeles, CA.
- Ginsparg, P. (1997) "Winners and Losers in the Global Research Village." *The Serials Librarian* Vol. 30, No. 3/4, pp. 83–95.
- Gladney, H. M. (2007) *Preserving Digital Information*. Springer, Berlin.
- Gore, D. (Ed.) (1976a) *Farewell to Alexandria: Solutions to Space, Growth, and Performance Problems of Libraries*. Greenwood Press, Westport, CT.
- Gore, D. (1976b) "Farewell to Alexandria: The Theory of the No-growth, High-performance Library." In (Ed.) D. Gore (1976a), pp. 164–180.
- Gorman, M., and Winkler, P. W. (Eds.) (1988) *Anglo-American Cataloguing Rules*, second edition. American Library Association, Chicago.
- Gorn, S., Bemert, R. W., and Green, J. (1963) "American standard code for information interchange." *Communications of the ACM*, Vol. 6, No. 8, pp. 422–426.
- Gurugé, A. (2004) *Web Services: Theory and Practice*. Elsevier, London.
- Gutwin, C., Paynter, G. W., Witten, I. H., Nevill-Manning, C., and Frank, E. (1999) "Improving browsing in digital libraries with keyphrase indexes." *Decision Support Systems*, Vol. 27, No. 1/2, pp. 81–104.
- Heaney, S. (Trans.) (1999) *Beowulf*. Faber and Faber, London.
- Holt, G. E., Elliott, D., and Dussold, C. (1996) "A framework for evaluating public investment in urban libraries." *The Bottom Line: Managing Library Finances* Vol. 9, No. 4, pp. 4–13.
- Hyman, R. J. (1972) *Access to Library Collections: An Inquiry into the Validity of the Direct Shelf Approach, with Special Reference to Browsing*. Scarecrow Press, Metuchen, NJ.
- International Telecommunication Union (1998) *Universal Access to Basic Communication and Information Services*. World telecommunication development report, International Telecommunication Union, Geneva, Switzerland.
- International Telecommunication Union (1999) *Mobile Cellular*. World telecommunication development report, International Telecommunication Union, Geneva, Switzerland.
- Intner, S. S., Lazinger, S. S., and Weihs, J. (2006) *Metadata and its Impact on Libraries*. Libraries Unlimited, Westport, CT.
- Jack, K. (2007) *Video Demystified: A Handbook for the Digital Engineer*, fifth edition. Newnes, Amsterdam, the Netherlands.
- Jerri, A.J. (1977) "The Shannon sampling theorem—Its various extensions and applications: A tutorial review." *Proceedings of the IEEE*, Vol. 65, No. 11, pp. 1565–1596.
- Jones, S. (1999) "Phrasier: An interactive system for linking and browsing within document collections using keyphrases." *Proc IFIP Conference on Human-Computer Interaction*, Edinburgh, Scotland, pp. 483–490.
- Jones, S., and Paynter, G. (1999) "Topic-based browsing within a digital library using keyphrases." *Proc ACM Digital Libraries*, Berkeley, CA, pp. 114–121; August. ACM Press, New York.
- Jones, S., and Staveley, M. (1999) "Phrasier: A system for interactive document retrieval using keyphrases." *Proc ACM SIGIR Conference on Research and Development in Information Retrieval*, Berkeley, California, pp. 160–167. ACM Press, New York.
- Kahle, B. (1997) "Preserving the Internet." *Scientific American*, Vol. 276, No. 3, pp. 82–83.

- Katz, B. (Ed.) (2003) *Digital Reference Services*. Haworth Press, Binghamton, NY.
- Kibbee, J. (2006) "Librarians without borders? Virtual reference service to unaffiliated users." *Journal of Academic Librarianship* Vol. 32, No. 5, pp. 467–473.
- Kientzle, T. (1997) *A Programmer's Guide to Sound*. Addison Wesley, Boston, MA.
- King, G. (1995) "Replication, replication." *PS: Political Science and Politics*, Vol. 28, No. 3, pp. 444–452.
- Knuth, D. E. (1986) *The TeXbook*. Addison Wesley, Boston, MA.
- Koenig, M. E. D. (1990) "Linking library users: a culture change in librarianship." *American Libraries*, Vol. 21, No. 9, pp. 844–849.
- Korfhage, R. R. (1997) *Information Storage and Retrieval*. Wiley, New York.
- Kortekaas, C. G. (2007) "Making Fedora easier to implement with Fez." *Open Repositories*, (Eds.) K. Webster and A. Bennett. San Antonio, TX; January.
- Kuny, T. (1998) "A digital dark ages? Challenges in the preservation of electronic information." *International Preservation News*, No. 17; May.
- Kuny, T. (1998) "A digital dark ages? Challenges in the preservation of electronic information." *International Preservation News*, No. 17; May.
- Lagoze, C., and Payette, S. (2000) "Metadata: Principles, practices and challenges." In *Moving Theory into Practice: Digital Imaging for Libraries and Archives*, (Eds.) A.R. Kenney and O.Y. Rieger, pp. 84–100. Research Libraries Group, Mountain View, CA.
- Lagoze, C., and Van de Sompel, H. (2001) "The Open Archives Initiative: Building a low-barrier interoperability framework." *Proc Joint Conference on Digital Libraries*, Roanoke, VA, pp. 54–62. ACM Press, New York.
- Lagoze, C., Krafft, D., Cornwell, T., Dushay, N., Eckstrom, D., and Saylor, J. (2006a) "Metadata aggregation and 'automated digital libraries': a retrospective on the NSDL experience. *Proc. Joint Conference on Digital Libraries*, Chapel Hill, NC, pp. 230–239. ACM Press, New York.
- Lagoze, C., Payette, S., Shin, E., and Wilper, C. (2006b) "Fedora: an architecture for complex objects and their relationships." *International Journal on Digital Libraries*, Vol. 6, No. 2, pp. 124–138.
- Lamport, L. (1994) *LaTeX: A Document Preparation System User's Guide and Reference Manual*. Addison Wesley, Boston, MA.
- Lankes, R. D., Abels, E. G., White M. D., and Haque, S. N. (Eds.) (2006) *The Virtual Reference Desk: creating a reference future*. Neal Schuman Publishers.
- Lennon, M. D., Pierce, D., Tarry, B., and Willett, P. (1981) "An evaluation of some conflation algorithms for information retrieval." *Journal of Information Science*, Vol. 3, No. 4, pp. 177–183.
- Lesk, M. (1997) *Practical Digital Libraries: Books, Bytes, and Bucks*. Morgan Kaufmann, San Francisco, CA.
- Lessig, L. (2004) *Free Culture: How Big Media Uses Technology and the Law to Lock Down Culture and Control Creativity*. The Penguin Press, New York.
- Library of Congress (2007) *Library of Congress Subject Headings*, 30th edition. Library of Congress Cataloging Policy and Support Office, Washington, DC.
- Licklider, J. C. R. (1960) "Man-computer symbiosis." *IRE Trans Human Factors in Electronics*, Vol. HFE-1, pp. 4–11.

- Liesaputra, V., Witten, I. H., and Bainbridge, D. (2009) "Creating and reading realistic electronic books." *IEEE Computer*, Vol. 42, No. 2, pp. 72–81.
- Lovins, J. B. (1968) "Development of a stemming algorithm." *Mechanical Translation and Computation*, Vol. 11, No. 1-2, pp. 22–31.
- Lynch, C. (1999) "Canonicalization: A fundamental tool to facilitate preservation and management of digital information." *D-Lib Magazine*, Vol. 5, No. 9; September. <http://www.dlib.org/dlib/september99/09lynch.html>
- Mankelow, T. (1999) "The New Zealand School Journal: A digital library for teachers and students." Department of Computer Science, University of Waikato; October.
- Mann, T. (1993) *Library Research Models*. Oxford University Press, New York.
- Marchionini, G., and Geisler, G. (2002) "The Open Video digital library." *D-Lib Magazine*, Volume 8, Number 12, December. <http://www.dlib.org/dlib/december02/marchionini/12marchionini.html>.
- Marshall, C. C. (1998) "Toward an ecology of hypertext annotation." *Proceedings of ACM Hypertext '98*, pp. 40–49. Pittsburgh, PA; June 20–24. ACM Press, New York.
- Marshall, C. C., and Brush, A. J. (2004) "Exploring the relationship between personal and public annotations. *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries*, pp. 349–357. Tucson, Arizona; June 7–11. ACM Press, New York.
- Martell, C. (2008) "The absent user: Physical use of academic library collections and services continues to decline 1995-2006." *Journal of Academic Librarianship* Vol. 34, No. 5, pp. 400–407.
- McCallum, I., and Quinn, S. (2004) "Valuing libraries." *Australian Library Journal* Vol. 53, No. 1, pp. 55–69.
- McNab, R. J., Smith, L. A., Witten, I. H., Henderson, C. L., and Cunningham, S. J. (1996) "Towards the digital music library: Tune retrieval from acoustic input." *Proc ACM Digital Libraries*, Bethesda, MD, pp. 11–18. ACM Press, New York.
- Medelyan, O., and Witten, I. H. (2008) "Domain independent automatic keyphrase indexing with small training sets." *J American Society for Information Science and Technology*, Vol. 59, No. 7, pp. 1026–1040.
- Milne, D., Witten, I. H., and Nichols, D. M. (2007). "A knowledge-based search engine powered by Wikipedia." *Proceedings of the ACM Conference on Information and Knowledge Management*, Lisbon, Portugal. pp. 445–454. ACM, New York.
- Morgan, E. L. (2004) "An Introduction to the Search/Retrieve URL Service (SRU)" *Ariadne*, Issue 40, July. <http://www.ariadne.ac.uk/issue40/morgan>.
- Murray, J. D., and van Ryper, W. (1996) *Encyclopedia of Graphics File Formats*, second edition. O'Reilly and Associates, Sebastopol, CA.
- Nack, F., and Lindsay, A. (1999a) "Everything you wanted to know about MPEG-7: Part I." *IEEE Multimedia*, Vol. 6, No. 3, pp. 65–77.
- Nack, F., and Lindsay, A. (1999b) "Everything you wanted to know about MPEG-7: Part II." *IEEE Multimedia*, Vol. 6, No. 4, pp. 64–73.
- National Security Agency (2006) *Redacting with Confidence: How to Safely Publish Sanitized Reports Converted From Word to PDF*, Report #I333-015R-2005, Information Assurance Directorate. http://www.nsa.gov/ia/_files/vtechrep/I333-TR-015R-2005.PDF.
- Nelson, M. L., and Allen, B. D. (2002) "Object persistence and availability in digital libraries." *D-Lib Magazine* Vol. 8, No. 1; January. <http://www.dlib.org/dlib/january02/nelson/01nelson.html>.

- Nevill-Manning, C. G., Reed, T., and Witten, I. H. (1998) "Extracting text from PostScript." *Software-Practice and Experience*, Vol. 28, No. 5, pp. 481–491.
- Nichols, D. M., Chan, C-H., Bainbridge, D., McKay, D., and Twidale, M. B. (2008) "A lightweight metadata quality tool." *Proc Joint Conference on Digital Libraries*, Pittsburgh PA, pp. 385–388. ACM Press, New York.
- Norris, P. (2001) *Digital Divide? Civic Engagement, Information Poverty and the Internet World-wide*. Cambridge University Press, New York.
- Orwell, G. (1949) 1984. Martin Secker and Warburg, London.
- Payette S., and Lagoze, C. (1998) "Flexible and extensible digital object and repository architecture (FEDORA)." *Proc European Conference on Research and Advanced Technology for Digital Libraries*, Vol. 1513, pp. 41–59. Springer, Berlin.
- Peng, R. D., Dominici, F., and Zeger, S. L. (2006) "Reproducible epidemiologic research." *American Journal of Epidemiology*, Vol. 163, No. 9, pp. 783–789.
- Pennebaker, W. B., and Mitchell, J. L. (1993) *JPEG: Still Image Data Compression Standard*. Van Nostrand Reinhold, New York.
- Pohlmann, K. C. (2000) *Principles of Digital Audio*, fourth edition. McGraw-Hill, New York.
- Pomerantz, J. (2005). "A conceptual framework and open research questions for chat-based reference service." *Journal of the American Society for Information Science and Technology* Vol. 56, No. 12, pp. 1288–1302.
- Porter, M. F. (1980) "An algorithm for suffix stripping." *Program*, Vol. 13, No. 3, pp. 130–137.
- Powell, A., and Recordon, D. (2007) OpenID: Decentralised Single Sign-on for the Web, *Ariadne*, Issue 51, April. <http://www.ariadne.ac.uk/issue51/powell-recordon>.
- Price-Wilkin, J. (2000) "Access to digital image collections: System building and image processing." In *Moving Theory into Practice: Digital Imaging for Libraries and Archives*, (Eds.) A.R. Kenney and O.Y. Rieger, pp. 101–118. Research Libraries Group, Mountain View, CA.
- Ranganathan, S. R. (1931) *The Five Laws of Library Science*. Madras Library Association, Madras.
- Robertson, R. J. (2005) "Metadata quality: Implications for library and information science professionals." *Library Review*, Vol. 54, No. 5, pp. 295–300.
- Rosenthal, D., and Reich, V. (2000) "Permanent web publishing." *Proc USENIX Annual Technical Conference*. San Diego, California, June. USENIX Association, Berkeley, CA. pp. 40. www.usenix.org/events/usenix2000/freenix/full_papers/rosenthal/rosenthal.pdf.
- Rosenthal, D., Lipkis, T., Robertson, T. S., and Morabito, S. (2005) "Transparent format migration of preserved web content." *D-Lib Magazine*, Vol. 11, No. 1, January. <http://www.dlib.org/dlib/january05/rosenthal/01rosenthal.html>.
- Rothenberg, J. (1995) "Ensuring the longevity of digital documents." *Scientific American*, Vol. 272, No. 1, pp. 42–47.
- Rothenberg, J. (1997) "Digital information lasts forever—or five years, whichever comes first." Rand Corporation Video V-079.
- Rothenberg, J. (1999) "Avoiding technological quicksand: Finding a viable technical foundation for digital preservation." CLIR Report pub77, Council on Library and Information Resources, Washington, DC; January.

- Rubin, R. E. (2001) "Ethical aspects of reference service." In *Reference and Information Services: An Introduction*, 3rd Edition, (eds.) Bopp, R. E. and Smith, C. L., Libraries Unlimited, Englewood, CO, pp. 28–46.
- Rusbridge, C. (2006) "Excuse me ... Some Digital Preservation Fallacies?" *Ariadne*, Issue 46, February. <http://www.ariadne.ac.uk/issue46/rusbridge/>.
- Salo, D. (2008) "Innkeeper at the Roach Motel." *Library Trends*, Vol. 57, No. 2. pp. 98–123.
- Salo, D. (2009) "Name authority control in institutional repositories." *Cataloging & Classification Quarterly*, Vol. 47, No. 3/4. pp. 249–261.
- Salton, G. (1989) *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Prentice Hall, Englewood Cliffs, NJ.
- Salton, G., and McGill, M. J. (1983) *Introduction to Modern Information Retrieval*. McGraw-Hill, New York.
- Samuelson, P. A. (1954) "The Pure Theory of Public Expenditure." *Review of Economics and Statistics* Vol. 36, No. 4, pp. 387–389.
- Samuelson, P. (1998) "Encoding the law into digital libraries." *Communications of the ACM*, Vol. 41, No. 4, pp. 13–18.
- Samuelson, P., and Davis, R. (2000) "The digital dilemma: A perspective on intellectual property in the information age." Presented at the *Telecommunications Policy Research Conference*, Alexandria, VA; September.
- Sanders, L. M. (Ed.) (1999) *The Evolving Virtual Library II: Practical and Philosophical Perspectives*. Information Today, Medford, NJ.
- Sanderson, R., Young, J., and LeVan, R. (2005) SRW/U with OAI: Expected and Unexpected Synergies, *D-Lib Magazine*, 11(2), February. <http://www.dlib.org/dlib/february05/sanderson/02sanderson.html>.
- Santayana, G. (1932) *The Life of Reason in the Phases of Human Progress*. Scribner's, New York.
- Schwab, M., Karrenbach, M., and Claerbout, J. (2000) "Making scientific computations reproducible." *Computing in Science and Engineering*, Vol. 2, No. 6, pp. 61–67.
- Shannon, C. E. and Weaver, W. (1949) *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, IL.
- Smith, R. E. (2002) *Authentication: From Passwords to Public Keys*. Addison-Wesley, London.
- Smith, M., Bass, M., McClella, G., Tansley, R., Barton, M., Branschofsky, M., Stuve, D., and Walker, J. (2003) "DSpace: An open source dynamic digital repository." *D-Lib Magazine*, Vol. 9, No. 1, January. <http://www.dlib.org/dlib/january03/smith/01smith.html>.
- Soghoian, C. (2007) "The problem of anonymous vanity searches." *I/S: A Journal of Law and Policy for the Information Society* Vol. 3, No. 2, 299. Available at: <http://ssrn.com/abstract=953673>.
- Spiteri, L. (1998) "A simplified model for facet analysis: Ranganathan 101." *Canadian Journal of Information and Library Science* Vol. 23, No. 1/2, pp. 1–30.
- Stamp, M. (2006) *Information Security: Principles and Practice*. John Wiley & Sons, London.
- Sturges, P., Davies, E., Dearnley, J., Iliffe, U., Oppenheim, C., and Hardy, R. (2003) "User privacy in the digital library environment: an investigation of policies and preparedness." *Library Management* Vol. 24, No. 1/2, pp. 44–50.
- Sugimoto, C. R. (2008) "Evaluating reference transactions in academic music libraries." *Music Reference Services Quarterly*, Vol. 11, No. 1, pp. 1–32.

- Svenonius, E. (2000) *The Intellectual Foundation of Information Organization*. MIT Press, Cambridge, MA.
- Szymanski, G., and Fields, M. A. (2005) "Virtual reference in the music library." *Notes*, Vol. 61 No. 3, pp. 634–658.
- Teahan, W. J. (1997) "*Modelling English text*." Ph.D. thesis, Department of Computer Science, University of Waikato, New Zealand.
- Teahan, W. J., Wen, Y. Y., McNab, R., and Witten, I. H. (2000) "A compression-based algorithm for Chinese word segmentation." *Computational Linguistics*, Vol. 26, No. 3, pp. 375–393.
- Tennant, R., (2004) "Organization and staffing." *Managing the Digital Library*, Chapter 7. Reed Press.
- Thiele, H. (1998) "The Dublin Core and Warwick Framework: A review of the literature, March 1995–September 1997." *D-Lib Magazine*, Vol. 4, No. 1, January. <http://www.dlib.org/dlib/january98/01thiele.html>.
- Thompson, J. (1997) *A History of the Principles of Librarianship*. Clive Bingley, London.
- Thornton, G. A. (2000) "Collection development in an electronic environment." *Library Trends* Vol. 48, No. 4, pp. 842–856.
- Trauth, M. H. (2006). *MATLAB Recipes For Earth Sciences*. Springer, Berlin.
- Turney, P. D. (2000) "Learning algorithms for keyphrase extraction." *Information Retrieval*, Vol. 2, No. 4, pp. 303–336.
- Twidale, M. B., Gruz, A. A., and Nichols, D. M. (2008). Writing in the Library: exploring tighter integration of digital library use with the writing process. *Information Processing & Management*, Vol. 44, No. 2, pp. 558–580.
- Unicode Consortium (2006) *Unicode Standard, Version 5.0*. Addison Wesley, Reading, MA.
- United Nations (1997) *Universal Access to Basic Communication and Information Services*. UN Administrative Committee on Coordination, New York.
- United Nations (1999) *Human Development Report*. UN Development Programme, New York.
- U.S. Congress (1990) *Taking a Byte Out of History: The Archival Preservation of Federal Computer Records*. House Committee on Government Operations Report 101-987, Washington, DC.
- van der Hoeven, J. R., van Diessen, R. J., and van der Meer, K. (2005) "Development of a Universal Virtual Computer (UVC) for long-term preservation of digital objects." *Journal of Information Science*, Vol. 31, No. 3, pp. 196–208.
- van Rijsbergen, C. J. (1979) *Information Retrieval*, second edition. Butterworths, London.
- Wang, X. (2004) "MPEG-21 rights expression language: Enabling interoperable digital rights management." *IEEE MultiMedia*, Volume 11, Number 4, pp. 84–87.
- Weibel, S. (1999) "The state of the Dublin Core metadata initiative." *D-Lib Magazine*, Vol. 5, No. 4, April. <http://www.dlib.org/dlib/april03/weibel/04weibel.html>.
- Weinberger, D. (2007) *Everything is Miscellaneous: The Power of the New Digital Disorder*. Holt, New York.
- Welch, T. A. (1984) "A technique for high-performance data compression." *IEEE Computer*, Vol. 17, No. 6, pp. 8–20.
- Wells, H. G. (1938) *World Brain*. Doubleday, New York.

- Wells, N, Morgan, O., Wilkinson, J., and Devlin, B. (2006) *The MXF Book: An Introduction to the Material eXchange Format*. Focal Press.
- Wendler, R. (2004) "The eye of the beholder: Challenges of image description and access at Harvard." In *Metadata in Practice*, (Eds.) D.I. Hillmann and E.L. Westbrooks. American Library Association, Chicago, IL, pp. 51–56.
- Whitaker, J. (2002) *Master handbook of audio production*. McGraw Hill.
- White, J. (Ed.) (1999) *Intellectual Property in the Age of Universal Access*. ACM Press, New York.
- Wildemuth, B. M., Marchionini, G., Yang, M., Geisler, G., Wilkens, T., Hughes, A., and Gruss, R. (2003) "How fast is too fast? Evaluating fast forward surrogates for digital video." *Proc Joint Conference on Digital Libraries*, pp. 221–230; May. Houston, TX. IEEE Computer Society, Washington, DC.
- Witten, I. H., Loots, M., Trujillo, M. F., and Bainbridge, D. (2001) "The promise of digital libraries in developing countries." *Communications of the ACM*, Vol. 55, No. 5, pp. 82–85.
- Witten, I. H., Moffat, A., and Bell, T. C. (1999) *Managing Gigabytes: Compressing and Indexing Documents and Images*, second edition. Morgan Kaufmann, San Francisco, CA.
- World Bank (1998/99) *World Development Report: Knowledge for Development*. World Bank, Washington, DC.
- World Bank (2000) *World Development Indicators 2000*. World Bank, Washington, DC.
- Wright, E. V. (1939) *Gadsby*. Wetzel, Los Angeles; reprinted by Kassel Books, Los Angeles.
- Wu, J. (Ed.) (1999) *New Library Buildings of the World*. Shanghai Public Library, Shanghai, China.
- Yeates, S., Bainbridge, D., and Witten, I. H. (2000) "Using compression to identify acronyms in text." *Proc Data Compression Conference*, p. 582. IEEE Press, Los Alamitos, CA.
- Yee, K., Swearingen, K., Li, K., and Hearst, M. (2003) "Faceted metadata for image search and browsing." *Proc SIGCHI Conference on Human Factors in Computing Systems*, pp. 401–408. New York: ACM Press.
- Ziv, J., and Lempel, A. (1977) "A universal algorithm for sequential data compression." *IEEE Trans Information Theory*, Vol. IT-23, No. 3, pp. 337–343.
- Zwicker, E. (1961) "Subdivision of the audible frequency range into critical bands (frequenzgruppen)" *J Acoustical Society of America*, Vol. 33, No. 2, p. 248.

Index

Page numbers followed by “f” indicate figures, “t” indicate tables, and “b” indicate boxes and formulas.

- AAC (Advanced Audio Coding) 306
- AAC (Advanced Audio Compression) 229–231
- Abrahamson, David 43
- Abstracts 17
- Access
 - in digital libraries 17, 29, 61–63
 - distributed libraries 29
 - between digital libraries 29
 - Internet 28, 44
 - in physical libraries 11, 54
- ACM (Association for Computing Machiner) 44, 352
- Acquisition stage (interactive OCR) 141–142
- Acrobat PDF reader 192
- Acronym extraction 335–336, 340–341
- Acronym extraction algorithm 340–341
- Acronyms
 - candidate 336
 - defined 335
 - identifying 335–336
- Adams, John 12–13
- Administration, Greenstone Software 521
- Adobe 246, 275, 284
- Adobe Acrobat 192, 195
- Adobe Flash 275–277
- Adobe Reader 19, 190
- ADPCM (adaptive differential pulse code modulation) 226
- Adsorbed, defined 41–43
- AIF-C (compressed) 226
- AIFF 226–228
 - chunks types 305
 - comments chunk 227–228
 - file extensions 226–227
 - marker chunks 228, 305
 - note chunks 227–228
 - text items 227–228
- AJAX (Asynchronous JavaScript and XML) 578–582
- Aldus 246, 283
- Alexandrian library 11, 43
- Alexandrian principle 13–15
- Alphabetic ordering 372, 388
- Alphabetical lists, browsing 111–113, 123
- Alphabets, non-Western 142–145
- American Library Association
 - Bill of Rights 47, 50
 - Code of Ethics 54, 288
 - Web site address 70
- Amru 11
- Anglo-American Cataloging Rules (AACR2R)* 289
- Annotations 67, 71
 - AIFF text item 227–228
 - PDF 193
 - textual 582
- Anonymous use 54–61
- ANSI (American National Standards Institute) 130
- Answers Now* 62–63
- AOL (America Online) 57, 71
- APNG (Animated Portable Network Graphics) 237
- Apple 6–7, 132, 230, 247–248, 421
- Apple Macintosh 134
- Arabic 394
 - case-folding 100
 - collections 375
 - directionality of writing 384
 - Unicode character set 375–380
- Archival copies 217
- Archiving
 - digital documents 413–415
 - Greenstone collections 531–532
 - JPEG 2000 245–248
 - PDF/A documents 190–191
 - TIFF format 245–248
 - World Wide Web 36
 - See also* Preservation
- Archiving images 245–248
 - JPEG 2000 245
 - TIFF format 246
- Arithmetic coding 218
- Arnold, Thelma 57, 59–60
- Artifacts, digitizing 216
- “As We May Think” (Bush) 71
- ASCII (American Standard Code for Information Interchange) 130–132, 387
 - 7-bit 133–134, 179, 385
 - 8-bit ISO extensions 133–134
 - basic character set 130–132, 380
 - character set 130, 133
 - control characters 130
 - frustrations 173
 - Hindi and 371
 - Indic script and 371, 390
 - introduction 127
 - ISCII extension 371

- Latin-1 Supplement characters 380
 notes and sources 211–213
 principle aim of 211
 punctuation characters 380
 round-trip compatibility 380
 UTF-8 encoding 385
 ASF (Advanced Streaming Format) 266
Ask a Librarian 62–63
Ask a Question 62–63
Ask Now 62–63
 ASN.1 (Abstract Syntax Notation One) 344
 ASR (automatic speech recognition) 104, 125
 Assurbanipal 11
 Atom Syndication Format 365–368
 AU format 226–228
 Auckland Museum 71
 Audio 222–224
 AAC 229–231
 AIFF format 226–228
 AU format 226–228
 CDs 221–222
 codecs 258
 converting analog to digital signals 223
 early formats 226–228
 FLAC 229–231
 introduction 221–234
 metadata 304–306
 mixing media 262–263
 MP3 228–229
 MPEG 228–229
 notes and sources 282–283
 Ogg Vorbis 229–231, 305
 post-mp3 formats 229–231, 305
 proprietary formats 264–266
 pulse code modulation (PCM) 222–224
 recording with computers 223
 replaying 231
 sample rates 258
 searching 86–87
 separability in transforming to images 219
 voice recordings 230–231
 WAV format 226–228
 windowing 220
 See also Multimedia; Music
 Audio digital library
 Internet Audio Archive 231–234, 419–420
 Native American language 5
 oral histories 5, 86–87, 124
 August of Wolfenbüttel 12, 410
 Australian Newspapers project 114
 Authentication
 Greenstone 521–522
 of library users 56–57, 62–63, 65, 67, 359–361, 370
 Authority control
 names 324–327
 subjects 327–329
 Authors/authorship
 AIFF text item 227–228
 copyright law 29–30
 in digital world 420
 effect of digital libraries on 29
 scientific/technical 297
 AVI (Audio Video Interleave) 265, 283
 Avram, Henriette 289
 Base fonts
 TrueType 183
 Type I 183
 See also Fonts
 Basic Multilingual Plane 375, 384
 Baudot code 130
 Bengali 375–380
Beowulf 417, 430
 BER (Basic Encoding Rules) 344
 Berne convention 30–32
 Berrybaskets 65–67, 71
 Bézier curve 183
 B-frame (bidirectional frame) 261
 Bibliographic databases 350
 Bibliographic entities 298
 Bibliographic metadata 17, 20, 288–299
 BibTeX 297–298
 building collections 490
 Dublin Core 294–295, 491
 EndNote 298–299
 Greenstone and 490–491
 Greenstone Digital library Software 490–491
 MARC 289–293
 MODS 297–298
 searching 490–491
 Bibliographic references 334
 Bibliographies
 information in 17
 as metadata 20
 Bibliophiles 19
 Bibliothèque du Roi 12
 Bibliothèque Nationale de France 13, 15f
 BibTeX 297–298
 attribute names 298
 database keys 298
 notes and sources 340
 “The big red books” 327
 Big-endian 384–385
 Book of codes 217
Book of Kells 19, 20f, 408–410
 Bookmarks 158
 Acrobat PDF reader 192
 begin-/end-bookmark 201–202
 Books *see also specific books*
 abstract content of 17
 ancient 18f
 bindings 146
 changing nature of 17–20
 copyright law 29–30
 heaviest 9
 preserving 410–411
 realistic 84–86, 124
 scanning 146
 virtual representation of 17
 as works of art 19
 Boolean queries 95, 104
 full-text retrieval system 136
 AND operator 95
 with ranked queries 97
 searches 344
 See also Queries; Searching
 Boolean retrieval systems 97
 Braille codes 217, 375–380
 Brazzaville, Congo, libraries in 26
 British Library 12, 403, 419
 British Museum 13

- Browsing
- alphabetical lists 111–113, 123
 - chapter-by-chapter 78
 - Chinese documents 396–398
 - classification hierarchy 327
 - collage 508
 - defined 110–111
 - Greenstone Digital library
 - Software 438–440
 - hierarchical 309
 - key-phrase 336, 340–341
 - lists in CJK languages 372, 396–398
 - by media type 505
 - metadata 110–116
 - multimedia 309–310, 505
 - notes and sources 125
 - phrase 508
 - in Pinyin 398
 - realistic books 84–86
 - searching vs. 111
 - stroke-based 398
 - structured 309–310
 - unstructured text documents 76–78
- B-spline 183
- Building collections
- See* Collection-building process (Greenstone)
- Bush, Vannevar 16, 43, 68
- C++ 387
- Caesar, Julius 11
- Cambridge University Library 12
- Camera, digital 300–302
- Canonical composition/ decomposition 350–351
- Canonical equivalence 350–351
- Card catalogs, 20th century 286
- Casanova 12
- Case-folding 98–100
- in form search 104
 - highlighting search terms and 100
 - languages and 100
- CCIR 601 259–261
- CD (compact disk) 221–222
- CD-ROM 29
- Cellular phones
- See* Mobile phones
- CGI arguments 520
- Character set 127
- ASCII 130, 133
 - multilingual 184–185
 - Unicode 127
- Character-identifier keyed (CID-keyed) fonts 184–185
- Characters
- ASCII 127
 - combining 381–384
 - composite 381–384
 - conjuncts 392
 - control 130, 134
 - with diacritics 142–145, 387
 - diaeresis 382
 - glyphs 182–183, 382, 389
 - as individual words 137
 - ligature 382, 392–393
 - surrogates 385–387
- Checking stage (interactive OCR) 146
- China
- classic literature
 - collection 375
 - gambling online 38
 - GB and Big-5 standards 130–132
 - Tang poetry collection 375
 - X'ian steles 9, 10f, 19, 408–410
- Chinese
- case-folding 100
 - character set 130–132, 182–183
 - character words 371
 - collections 375
 - full-text indexing 137
 - ordering word lists in 396–399
 - Pinyin browsing 398
 - stroke-based browsing 398
 - Unicode character set 375–380
 - word segmentation 394
 - writing 9, 371
- Chopin Early Editions 279–281
- CIC (Council of Independent Colleges) 248–249, 283
- Citation index 334, 340–341
- CJK (Chinese-Japanese-Korean) family of languages 371, 394
- Classification
- Dewey Decimal 114
 - faceted 114
 - hierarchically structured 114
 - Humanity Development Library 3
 - Library of Congress 114
- Classifiers
- Add 458, 469, 497
 - AZCompactList 506
 - Browsing 458, 461, 499, 505
 - DateList 499
 - Hierarchy 468
 - List 458, 461, 505
 - Remove 458
- Cleanup stage (interactive OCR) 141
- Cleopatra 11
- Codecs 258
- Codex 17–19
- Collection configuration files 125
- Collection-building process (Greenstone)
- archive formats 531–532
 - document identifiers 532–534
 - importing and building 529
 - incremental building 529–530
 - indexers for full-text
 - searching
 - Lucene 537
 - MG 536–537
 - MGPP 537
 - indexes
 - adding and configuring 537
 - partitioning 538
 - scheduled rebuilding 530–531
 - third-party packages 516–517
- Collection-level metadata 316–319
- Collections (Greenstone) 41, 431
- About page 438
 - bibliographic information 490–491
 - branding 483–485
 - browsing 438–440
 - Demo collection 435

- designing 457–458
 - enriching with metadata
 - 456–457
 - exporting 543
 - framework creation 525–526
 - icons and their meaning 440t
 - indexes 536–539
 - indexes, building 528
 - installing 528
 - plug-ins. *see* Plug-ins
 - rebuilding 530–531
 - sharing 514–519
 - viewing and refining 468–469
- Collections, building using the Greenstone Command-line Operation 524–528
- building indexes 528
 - importing documents 526–527
 - installing the collection 528
 - making a framework 525–526
- Collections, building using the Greenstone Librarian interface
- assembling source material 450–451
 - bibliographic metadata 490–491
 - document handling 454–469
 - enriching the documents 451–452
 - example 443–454
 - format strings 473–476
 - formatting 469–485
 - pages 453
 - help 454
 - HTML documents 454–456
 - enhanced handling 464–466
 - extracting document structure 465
 - extracting metadata 464–465
 - metadata for hierarchical documents 465–466
 - images 495–497
 - importing documents 526
 - introduction 435
 - manually 451–452
- metadata extraction 486–487
- metadata files
- combining source documents 494
 - hierarchical organization 486
 - importing documents with 488
 - individual records 491–494
 - reviewing assigned 487–488
 - storing 488–489
- metadata sets 487
- multimedia 504–509
- New Collection example 450–451
- non-textual documents 495–509
- PDF files 456–458
- converting to page images 462–463
 - enhanced handling 461–464
 - formatting exercise 481–482
 - highlighting search terms 464
 - mixed collections 463–464
 - splitting into sections 462
 - trouble-shooting 461–462
- previewing 454
- process 41
- scaling up 466–469
- structure and organization 452–453
- textual images 497–504
- users and functions 442–443
- Word files 456–458
- detecting user-defined styles 459–460
 - enhanced handling 458–461
 - extracting document properties 460–461
 - extracting document structure 459
- formatting exercise 481–482
- Collections, protecting 522–523
- Communications Network Research Institute (CNRI) 351, 353–354
- Companding 224–225
- Compound objects metadata
- collection level 316–319
 - eResearch 321–323
 - LOM (Learning Object Metadata) 319–321
 - METS (metadata encoding and transmission standards) 313–316
 - OAI-ORE (Open Archives Initiative Object Reuse and Exchange) 319
 - RDF (Resource Description Framework) 310–313
 - SCORM 319–321
- Compression
- basic techniques 217–218
 - of images 235
 - introduction 216–221
 - JPEG 2000 245
 - segmented text 395
 - TIFF 247, 300–301
 - video 258
 - See also* Lossless
 - compression; Lossy compression
- Compression utilities 237, 395
- Computer output 19
- Computers
- in developing countries 3, 26, 28–29
 - paper 19
 - recording audio with 223
- Concertina books 17–19
- Configuration files 519–520
- Control characters 130, 134
- Cookies 59–60
- Coordinate matching 97–98
- Copyright 33
- Copyright 29–38
- AIFF text item 227–228
 - digital rights management 33, 64
 - digitization and 34–35

- music on the Web 6
- notes and sources 44
- relinquishing 32–33
- symbol 30
- Web information 35–37
- Copyright Extension Act 31
- Copyright law 29–30, 36
 - illegal and harmful material 37–38
- Copyright Renewal Database 32
- Corrections 68
- CPB (constrained parameter bitstream) 262–263
- Crash Course in Copyright* (Harper) 44–45
- Creative Commons 32–33, 44
- CSS (cascading style sheets) 163–176
 - ambiguity-resolution algorithm 166
 - context-sensitive formatting 168–170
 - format statements and 559–560
 - inheritance mechanism 164
 - introduction 128
 - key features 169
 - media-dependent formatting 168–170
 - notes and sources 211–213
 - pseudo-elements 166
 - selector-declaration pairs 164
 - tables and lists 166–168
 - XSL vs. 170
- Cultural sensitivity 38
- Cultures, indigenous 5, 27, 38
- Dances, digital libraries of 5
- Data, defined 7–8
- Data, user-supplied 67–70
 - annotations 67, 71
 - corrections 68
 - keywords 67–68
 - new documents 68
 - notes and sources 71
 - partial and fluid documents 68–70
 - ratings 68
- Data mining 57
- Data providers 346
- Dates, metadata browsing 113
- DDL (Description Definition Language) 307–308
- Demo collection 125
- Denmark's National Library 13
- Department of Defense, U.S. 210
- Derivational suffixes 99
- Developing countries
 - digital libraries in 3, 25–29, 38, 44
 - disaster relief efforts 26–27
 - Internet penetration 28, 44
 - locally produced information 27–28
 - mobile phone penetration 44
 - preserving indigenous culture 5, 27
 - technological infrastructure 25–26, 28–29
- Dewey Decimal classification 114, 295
- Dictionary (ies) 236
 - language 395
 - OCR systems 145
- Dictionary methods 217, 236
- DIDL (Digital Item Declaration Language) 309–310
- Digital collections 8–9, 35–36
- Digital Dark Ages 413–415
- Digital divide 44
- Digital items 309–310
- Digital libraries
 - accessing content 61–63
 - advantages
 - accessibility 29
 - flexibility 402
 - mobility 402, 420–424
 - age of 11
 - boundaries 8–9
 - collection managers 357
 - cultural sensitivity 38
 - defined 7, 17, 43, 73
 - in developing countries 25–29, 44
 - disseminating humanitarian information 26
 - distributed 29
 - e-mail in 210–211
- environmental conditions required 49
- eResearch in 321–323
- ethical and legal issues for 37
- extracting content 64–65
- HTML in 158
- implementing 41
- for indigenous culture preservation 5, 27, 38
- internationalization of 127
- i-Podbased 424, 430
- killer app for 6–7
- LaTeX in 209–210
- legal liability 35
- for locally produced information collections 27–28
- materials 39t, 40
- METS in 313–316
- multimedia in 267–268
- native Word format in 203
- notes and sources 45–46
- Open Document format (ODF) in 206–207
- for oral cultures 5, 427–429, 431
- oral histories in 86–87, 124
- physical environment 403
- physical libraries compared 7–9, 49, 56, 64, 67
- planning questions 38–41
- PostScript in 189–190
- present-day 402–404
- privacy concerns 71
- problems facing 127
- publishing industry, effect on 29
- purpose 36, 93
- referring to objects in 65
- resource requirements 40–41
- restricting access to 359
- risk management 40
- RTF in 202
- scope of 6–7
- security 359–361
- technology in 39t, 40
- technophile visionaries concepts 15–16
- TIFF format in 247
- usage data, recording 57–61

- user experience 40
 - Web sites vs. 9
 - World Wide Web vs. 9
 - See also* Libraries; *specific digital libraries, e.g. music*
- Digital libraries, future of 310
 - envisioning 404–407
 - notes and sources 429–431
 - physical environment 405–406
 - working inside 407–408
- Digital libraries, trends in
 - knowledge-based information retrieval 424–427
 - portable collections 420–424
- Digital library collections 63–67
- Digital library software
 - authentication infrastructure 361
 - DSpace 361–364
 - evaluation considerations 65
 - Fedora 364–369
 - OAI-PMH support 349
 - Unicode in 387–389
 - XML in 162–163
- Digital library users
 - See* Library patrons
- Digital Millennium Copyright Act (DMCA) 34
- Digital music libraries 279–281
- Digital music stand 582
- Digital objects 7
 - berrybaskets 65–67
 - METS for 313–316
 - persistence of 71
 - referring to 65
- Digital preservation 413–415, 429–430
- Digitization noise 235
- Digitizing
 - of audio tapes 86–87
 - copyright and 34–35
 - costs 147–149
 - decision 127–128
 - donated materials 34
 - fair use and 35
 - images 147–148
 - notes and sources 211–213
 - OCR stage 138, 140–146
 - old technology 413
 - outsourcing 147
 - photographic images 86–87
 - of physical content in libraries 51–52
 - project planning 147–148
 - quality control 61
 - scanning stage 139–140
 - stages of 138
 - See also* Scanning
- Disaster relief 26–27
- Discrete Cosine Transform 219, 239
- Distributed library systems 29
- Distributed proofreading 78–79
- Distribution (Greenstone)
 - institutional repositories 549–554
 - remote librarian interface 545–548
- Distribution right 30
- DMS-1 (Descriptive Metadata Schema) 306
- Document processing plug-ins 457
- Document surrogates
 - metadata 90–93
 - multimedia 93
 - presentation 90–93
- Documents
 - archival copies of historical 217
 - born digital 287–288
 - converting 133
 - digitizing. *see* Digitizing
 - electronic 128
 - evolving concept of 129
 - formatting 469–485
 - Greenstone 41
 - Greenstone archive 532–534
 - hierarchical structure 465–466
 - identifiers 532–534
 - importing (Greenstone) 526–527
 - marked-up, presenting 561
 - metadata extraction 332
 - multilingual 372–375
 - multimedia 93
 - n*-grams 335
 - non-textual 495–509
 - online 129
 - partial and fluid 68–70
 - plain text 133–134
 - PostScript. *see* PostScript
 - replicating 146
 - surrogates 93
 - user-supplied data in 68
 - See also* Bibliographic entities; textual documents; Word-processor documents
- Documents, digitized
 - end-of-line codes 78
 - highlighting search terms 78, 81
 - preservation 413–415
 - printing 81–84
 - proofreading 78–79
- DOIs (Digital Object Identifiers) 352–353
- DOM (document object model) 162, 257, 271–272, 560
- Domesday Project 412, 420, 429
- DRM (digital rights management) 33–34
- DRM technology 33, 51
- DSC (document structuring conventions) 182
- DSpace 361–364, 370, 543–545
- DTDs (Document Type Definitions) 159–160, 162
 - parsed character data 160
- Dublin Core 155
 - encoding schemes 295
 - introduction 294–295
 - MARC vs. 294–295
 - metadata 294t, 349
 - metadata mapping to MARC 491
 - namespaces 205
 - notes and sources 339
 - qualified 295–297, 349
 - resource 295
 - standard 294–295
 - visualization tool 330
 - See also* Bibliographic metadata
- Duncker, Elke 37–38
- DVD's 29
- Dynamic HTML 170

- EBCDIC (Extended Binary Coded Decimal for Information Interchange) 127, 130
 E-Book technology 33
 E-Books (electronic Books) 33
 Education, locally produced information and 27–28
 Education metadata
 LOM 319–321
 SCORM 319–321
 Electronic preprint community 345–346
 E-mail 210–211
 Emulation preservation strategy 416–417
 Encapsulated PostScript 182
 Encoding vector 183
Encyclopedia Britannica 13
 EndNote 298–299
 keywords 299t
 English-language interface 21, 372–375
 Entertainment industry 34, 306, 429
 EPIC (Electronic Privacy Information Center) 71
 EResearch 321–323, 340
 Error log 520
 EScience 323
 Ethiopic 375–380
 European Union, WIPO Treaty approval 34
 Events
 channel 278–279
 copyright notice 279
 cue points 279
 instrument name 279
 lyric 279
 marker 279
 meta 279
 note aftertouch 278–279
 note on/note off 278–279
 pitch bend 278–279
 program change 278–279
 set tempo 279
 system exclusive 279
 time/key signature 279
 track name 279
 Events log 520
 EXIF (exchangeable image file format) 302–304
 EXIF tags 246
 Extensible Access Control Markup Language 365–369
 Extensible markup language
 See XML (extensible markup language)
 Extensible stylesheet language
 See XSL (extensible stylesheet language)
 Facets 114–116
 Fair use 30, 32–33, 35–36, 229
Farewell to Alexandria (Gore) 13, 43
 Fedora (Flexible Extensible Digital Object Repository Architecture) 364–370
 Fedora METS 365–368
 Fez 369
 Fielded queries 104
 Filters
 PNG format 237
 SVG 257
 FineReader 212–213
 Firefox 237
The Five Laws of Library Science (Ranganathan) 47, 70
 FLAC (Free Lossless Audio Codec) 229–231
 Flash video format 88, 275–277, 283–284
 FO (Formatting Objects) 170, 172–173
 Fonts
 CID-keyed 184–185
 composite 184
 hints 183–184
 OpenType 184–185
 PostScript 182–185
 SVG 257
 TrueType 183
 Type I 183
 Unicode 184–185, 389
 Foreign languages
 See specific languages
 Format statements
 CSS and 559–560
 Greenstone 559, 571–573
 making reusable through macro definitions 571–573
 Format strings 473–476
 conditionals 473–474
 document nodes and 475
 Greenstone 473–476
 multiple-valued metadata 475
 predefined 476
 Formatting
 documents 469–485
 pages 453, 469–485
 presentation patterns 469–485
 Fourier transform 219–221
 windowing in 220
 Fox, Ed 43
 FOXML 365–368
 Frames 157–158
 MP3 ID3 305
 Frameset HTML 154, 157–158
 Framework
 creating 525–526
 OAI-PMH 346
 See also RDF (Resource Description Framework)
 France
 1700s, statistics on books published in 12
 illegal and harmful material on the Internet 38
 legal deposit law 12
 scholarly journals 12
 François I 12
Free Culture (Lessig) 32, 44
 Free Software
 Foundation 122–123
 French
 accent folding 387
 character set 130–132
 interface 21, 372–375
 OCR errors and 142–145
 French National Library 12–13
 FTP (file transfer protocol) 134
 Full-text indexing 134–137, 369
 Full-text searching 134–137, 536–537
 See also Searching
Future Libraries 13–15
 FutureWave Software 275

- Gambling online 38
- Games 275–276
- Gasaway, Lolly 44–45
- Generic entity extraction 332–334
 - self-adaptive techniques 334
 - training systems 333
- GeoTIFF 300–301
- German 142–145, 388
- German National Library 13
- GIF (Graphics Interchange Format) 236–237
 - animated image support 237
 - color map 236
 - image-related metadata 299–300
 - progressive refinement 242
 - transparency support 246
- Ginsparg, Paul 41–43
- Glyphs 182–183, 382, 389
- GNU database manager (GDBM) 556
- GNU Free Documentation License 33
- GNU General Public License (GPL) 33
- GNU Project 33, 122–123, 212
- Goethe 11
- Goethe, Johann von 162–163
- Google 212
- GOP (Group of Pictures) 262
- GPL conversion utilities 33
- Great Harris Papyrus 17–19
- Greek interface 21, 375–380
- Greenstone Archive Format 531–532
- Greenstone Digital library
 - Software
 - additional information on 509–510
 - browsing 438–440
 - subjects 439
 - titles 438
 - CD-ROM/DVDs 518–519
 - collection-building process 529–539
 - collections sub-folders 513
 - distributed operation 545–554
 - document handling 454–469
 - documents 41
 - Enrich panel 486–488
 - files and folders 515–517
 - help 511–512
 - icons and their meaning 440t
 - import process 529
 - installation location 515
 - installing 435, 514–519
 - interoperability 539–545
 - introduction 41, 435
 - key features 41, 42t
 - large-scale usage 554–558
 - Librarian Interface 442–454
 - library 41
 - log entry 59
 - manuals 509–510
 - metadata 41, 485–494
 - object identifiers (OIDs) 532–534
 - operating 519–523
 - plug-ins 457
 - Reader's interface 437–442
 - searching 517, 536–539
 - source code 516
 - third-party packages 516–517
 - Unicode translation tables 517
 - updating 514–515
 - user community 511
 - Web site 443
 - See also* Collection-building process (Greenstone); Collections (Greenstone)
- Greenstone Digital library
 - Software, distributed operation 545–554
 - institutional repositories 549–554
 - remote librarian interface 545–548
- Greenstone Digital library
 - Software, interoperability 539–545
 - downloading Web sites 539–540
 - DSpace 543–545
 - exporting collections 543
 - metadata protocols 540–541
 - OAI server 541–542
- Greenstone Digital library
 - Software, large-scale usage
 - distributed serving 558
 - extremely large collections 555–557
 - large collections 554–555
 - Librarian Interface
 - limitations 554
- Greenstone Digital library
 - Software, operating 519–523
 - administration facility 519, 521
 - authentication 521–522
 - collections, protecting 519, 522–523
 - configuration files 519–520
 - error log 520
 - logging 519–521
 - security 519
 - user groups 522
 - user identification 519–520
 - user privileges 519
- Greenstone Reader's interface 437–442
 - browsing 438–440
 - icons and their meaning 440t
 - preferences 440–441
 - searching 440–441
- Greenstone Research Project 585–587
- GSearch 369
- GSM 226, 282
- Gutenberg collection 372–375
- Gutenberg printing press 12
- Gzip 237
- Hamilton, New Zealand oral history project 86–87, 124
- Handle Registry 352
- Handles 351
- Harper, Georgia 44–45
- Hearing in humans 224, 228
- Hebrew 394
 - character set 130–132
 - directionality of writing 384
 - Unicode character set 375–380
- Hewlett Packard 125, 212
- High recall vs. high precision searching 95–97
- Highlighting search terms
 - aiding users by 36–37
 - case-folding and 100

- digitized documents 78, 81
- PDF files 464
- stemming and 100
- Hindi 371, 389–393
- Historic Campus Architecture Project 248–249
- Hong Kong 130–132
- HSV-L-Colour Focus 110
- HTML (Hypertext Markup Language) 152–163
 - basic 155–158
 - in digital libraries 158
 - dynamic 170, 560, 573–577
 - entities 155
 - frameset 154
 - frames/frameset 157–158
 - history of 152
 - hyperlinks 154, 157
 - introduction 128
 - markup 153–155
 - Microsoft Word Save as option 203
 - notes and sources 211–213
 - OCR output 148
 - standards 152
 - stylesheet languages 153–155
 - tag attributes 155
 - tags 155
 - transitional 154
 - XML and SGML relationship 152–153
 - XML files from 152–153
- HTML 5 152
- HTML documents
 - enhanced handling 464–466
 - extracting document structure 465
 - extracting metadata 464–465
 - Greenstone and 454–456
 - metadata for hierarchical documents 465–466
- HTML plug-in 465
- HTML Tidy 154–155
- Http (hypertext transfer protocol) 385
- HTTPS (Hypertext Transfer Protocol Secure) 359–360
- Huffman coding 218, 239
- Human auditory system 224, 228
- Human development 27–28
- Human-assigned metadata 287
- HumanInfo Project 41–43
- Humanity Development Library 3, 29, 41–43, 74–76, 84–86, 114, 123–124
- Hyperlinks 128, 154, 157, 192–193
- IBM 6–7, 419
- Icons
 - folder 76
 - Greenstone Digital library Software 440t
 - text-page 76
 - thumbicon 562
- ID3 (IDentify an MP3) 305
- Identifiers 65
- IFF (Interchange File Format) 226, 279, 282
- I-frame 261
- IISCI (Indian Script Code for Information Exchange) 389–390
- Image digital libraries 248–249, 301
- Image digitization project planning 147–148
- Image metadata
 - EXIF 302–304
 - GIF format 299–300
 - IPTC 302–304
 - MIX 302–304
 - TIFF 300–301
 - XMP 302–304
- Image plug-in 495–496, 562
- Image resolution 139
- ImageMagick 283
- Images
 - animated 237
 - archiving 245–248
 - compression 235
 - converting PDF files to 462–463
 - digitization 147–148
 - GIF format 236–237
 - Greenstone collections 462–463, 495–497
 - introduction 235–257
 - JPEG 237–242
 - JPEG 2000 245–248
 - lossless compression 236–237
 - medical and legal 217
 - notes and sources 283
 - photographic 86–87
 - pixels elements 139, 235
 - PNG format 236–237
 - progressive refinement 242–245
 - quality 235
 - searching 106–110, 125
 - separability in transforming audio to 219
 - space consumed by 29
 - space requirements 235
 - storage 235
 - SVG 252–257
 - switching between textual images and 502–504
 - with text 81–84
 - thumbnails as 504
 - TIFF format 245–248
 - vector graphics 252–257
- Indexers
 - Lucene 537
 - MG 536–537
 - MGPP 537
- Indexes
 - citation 334, 340–341
 - comprehensive 136
 - partitioning 468
 - phrase searching and 100
 - presentation files 210
 - software robots for building 36
 - spreadsheets 210
 - word segmentation 137
 - word-level 100–101, 136
- Indexes (Greenstone)
 - adding and configuring 537
 - building 528
 - partitioning 538
- Indexing
 - automatic citation system 340–341
 - full-text 134–137, 369, 517
 - key phrase 336–339
 - Maori newspapers 372
 - notes and sources 212
- Indexing packages 369

- Indic scripts 384, 389–393
 - ASCII for 371
 - IISCI (Indian Script Code for Information Exchange) 389–390
 - Unicode for 390–392
- Indigenous culture, preserving 5
- Individual metadata records 491–494
- Inflectional suffixes 99
- Information
 - defined 7–8
 - humanitarian, disseminating 26
 - iconic form 427
 - locally produced 27–28
- Information dissemination, power of 27–38
- Information explosion 12–13
- Information overload 71
- Information retrieval
 - knowledge-based 424–427
 - recall-precision tension in 97
- Information revolution 11, 26
- Information technology 28
- Inheritance mechanism 164
- “Innkeeper at the Roach Motel” (Salo) 70
- Institutional repositories 53
 - authority control in 326
 - Greenstone and 549–554
 - MIT 116–123, 125, 361
 - notes and sources 70
 - user-supplied data in 68
- Institutional repository manager 53
- Intellectual property 37, 44
- Interactive OCR
 - acquisition 141–142
 - checking stage 146
 - cleanup stage 141–142
 - Fine Reader 212–213
 - page analysis stage 141–142
 - recognition 142–145
 - saving stage 146
 - steps 141
 - See also* OCR (optical character recognition)
- Interfaces
 - audio recordings 86
 - icons 76
 - languages 21, 372–380
 - multilingual 372–375
 - query interfaces 102–104
 - scroll bar 84
 - search history 73
 - sliders 108–110
 - thumbnails 93, 192–193
- Interlacing 242
- International DOI
 - Foundation 352–354
- Internationalization
 - introduction 371
 - multilingual interfaces and documents 372–375
 - Unicode 375–389
- Internet
 - access to the 28, 44
 - downloading video 88
 - e-mail messages 211
 - illegal and harmful material on the 37–38, 45
 - killer app 6–7
 - search engines 36
- Internet Archive 45, 231–234, 419–420
- Internet Audio Archive 231–234, 419–420
- Internet Explorer
 - APNG supported on 237
 - character sets 127
 - native Word and 196
- Internet file transfer protocol (FTP) 134
- Internet MIME format 210
- Interoperability
 - authentication and security 359–361, 370
 - DSpace 361–364
 - Fedora 364–369
 - introduction 343
 - notes and sources 369–370
 - OAI-PMH (Open Archives Initiative Protocol for Metadata Harvesting) 346–348
 - object identification 350–354
 - Open Archives Initiative (OAI) 345–350
 - Web services 354–359
 - Z39.50 protocol 344–345
- Interoperability (Greenstone) 539–545
 - downloading Web sites 539–540
 - DSpace 543–545
 - exporting collections 543
 - metadata protocols 540–541
 - OAI server 541–542
- Intuitive inheritance 165
- IPTC (International Press Telecommunications Council) 302
- IPTC Photo Metadata 2008 standard 302
- Ireland 13
- ISCI (Indian Script Code for Information Exchange) 371
- ISO (International Organization for Standardization) 130–132, 229, 353–354
- ISO 10646-1 375
- ISO 8859-1 130–132
- ISO-10646 132
- Italy
 - copyright law 31
 - gambling online 38
- Jail breaking 247–248, 424
- Japanese
 - full-text indexing and 137
 - Unicode character set 375–380
- Japanese Text Initiative 162–163
- Java 133, 148, 369
 - String type 387
- JavaScript 190
 - Exploiting Asynchronous JavaScript and XML (AJAX) 578–582
 - using to adjust presentation 569–571
- Joint Photographic Experts Group 237
- Journal des Savans* 12
- Journals, digitizing 81
- Journals, scholarly 12, 26, 29
 - eResearch 321–323
- JPEG 237–242
 - baseline 238
 - compression algorithm 238

- Discrete Cosine Transform
 - 239
 - encoding and decoding 238
 - extended features 237
 - image-related metadata 299–300
 - JPEG 2000 compared 245
 - key features 239–241
 - metadata embedded in 302
 - MPEG compared 260
 - progressive refinement 244
 - transparency support 246
- JPEG 2000 245–248
 - core capabilities 246
 - JPEG compared 245
- JPEG-Lossless 237–242
- Kahle, Brewster 419
- Kataayi Cooperative 3, 41, 403–404
- Kateregga-Ndawulu, Emmanuel 41–43
- Keresan language 5
- Key phrase browsing 336, 340–341
- Key phrase indexing 336–339
- Key phrases 340–341
 - choosing 336
 - in key-phrase extraction 337
 - metadata 336–339
- Key-phrase assignment 336–337
- Key-phrase extractions 337
 - accuracy 337
 - automatic 336
 - key phrases in 337
 - machine learning 337
 - training data 337
- Keywords 67–68, 336
- Khmer language 394, 396
- Killer app 6–7
- Knowledge, defined 7–8
- Knowledge-based information retrieval 424–427
- Korean Hangul ideographs 375–380, 383
- Language
 - dictionaries 395
 - preserving indigenous 5
 - See also specific languages*
- Language identification extraction 334–335, 340–341
- Lao language 394, 396
- LaTeX 129, 207–210
 - benefits 207
 - in digital libraries 209–210
 - notes and sources 213
 - parsing 210
 - problems 210
 - shortcuts 208
- LaTeX commands 207
 - \begin \end 207
 - structural 207–208
 - user defined 207
- Latin 375–380, 394
- LDAP (Lightweight Directory Access Protocol) 360
- Leaf DTDs 160
- Legal images 217
- Lessig, Lawrence 32, 44
- Librarians
 - 19th century 17
 - contemporary vs. medieval 47
 - duty of 13, 43, 54, 401
 - future of 52–54
 - libraries defined by 7
 - philosophy foundations 47
 - roles of vs. digital 51–52
 - value added to libraries 7–8
- Librarians, digital
 - institutional repository managers 53
 - roles of vs. physical 51–52
 - skills required 52–54
- Libraries
 - future of 401
 - for people from oral cultures 427–429, 431
 - preservation of 408–420
 - See also* Digital libraries; Library patrons; Physical libraries; Virtual libraries
- Library catalogs
 - 20th century card type 286
 - errors in 329
 - Library of Congress 289
- Library of Congress
 - classification system 114
 - established 12–13
- legal deposit law 12
- MARC attribute model 293–294
- mission of 63
- Name Authority File 324–327
- notes and sources 43
- online catalog 289
- persistence 353–354
- statistics 13
- Subject Authority File 328
- symbolic nature of 402
- Library of Congress Subject Headings (LCSH) 295, 327–329
- Library patrons 39t, 40, 50
 - anonymous use 54–61
 - authenticating 56–57, 62–63, 65, 67, 359–361, 521–522
 - contributions 67–70
 - contributions by 78–79
 - global users 50–51
 - help and user support services 61–63
 - increasing needs of 343
 - introduction 47
 - logging 519–521
 - notes and sources 70–71
 - from oral cultures 427–429, 431
 - privacy concerns 56
 - restricting services to 56, 62–63, 522
 - tracking usage 288
 - working inside digital libraries 407–408
- Library patrons, contributions
 - annotations 67, 71
 - corrections 68, 78–79
 - keywords 67–68
 - new documents 68
 - partial and fluid documents 68–70
 - ratings 68
- Library science 47, 50
- Licklider, J. C. R. 16, 43
- Ligature 382
- LINCOS project 428
- Link-rot 351

-
- Lists
 - browsing in CJK languages 372
 - metadata browsing 111–113
 - Little-endian 384–385
 - LOCKSS preservation software 416, 430
 - Logging 520–521
 - Logs
 - error 520
 - events 520
 - usage 519–521
 - LOM (Learning Object Metadata) 319–321
 - Loots, Michel 41–43
 - Los Alamos National Laboratory 5
 - Lossless compression 216–217
 - arithmetic coding 218
 - codebook 217
 - dictionary methods 217, 236
 - GIF format 236–237
 - Huffman coding 218, 239
 - images 236–237
 - JPEG-Lossless 242
 - LZW scheme 236
 - PNG format 236–237
 - See also* Compression
 - Lossy compression
 - achieving higher compression 219
 - JPEG 237–242
 - quantization 218
 - See also* Compression
 - Lotus 6–7
 - LPCM (linear pulse code modulation) 226
 - Lucene, Solr 369
 - Lynx browser 158
 - LZW scheme 236
 - Macromedia Shockwave 275
 - Macros
 - additional static pages 567–569
 - commonly used 565–567
 - dynamic HTML 573–577
 - Greenstone 563–565
 - header 564
 - language 564
 - making formats statements reusable 571–573
 - page parameters 564
 - using JavaScript to adjust presentation 569–571
 - MADS (Metadata Authority Description Schema) 326
 - Makasa, Uganda 3
 - Managing Gigabytes* (Witten et al.) 124, 212
 - Mann, Thomas 43
 - Manu 47
 - Maori
 - language 213
 - newspapers 372–375, 398
 - toki* 44
 - MARC (machine-readable cataloging) 289–293
 - Dublin Core vs. 294–295
 - notes and sources 339
 - variants 91
 - MARC records
 - authority control 324–327
 - exchanging between libraries 293
 - field meanings 293t
 - fields 289–291
 - metadata mapping to Dublin Core 491
 - METS compared 313–316
 - producing 289
 - MARC21 297
 - MARXML 293–294
 - Mark Antony 11
 - Markers
 - AIFF 228
 - MIDI 279
 - SVG 257
 - Markup
 - formatting 128
 - presentation 128
 - structural 128
 - Markup languages
 - See* HTML (Hypertext Markup Language); XML (extensible markup language)
 - Massachusetts Institute of Technology institutional repository 116–123, 125
 - Mathematical symbols 375–380
 - McKoi relational database 369
 - Medical images 217
 - Medical Subject Headings (MeSH) 295, 328
 - Meldex collection 125
 - Metadata 89–90
 - administrative 288
 - annotations stored as 582
 - assigning from files 487–488
 - audio 228, 304–306
 - bibliographic. *see* bbibliographic metadata
 - characteristics 286–288
 - classifying 287
 - collection-level 316–319
 - combining source documents 494
 - completeness 330
 - for compound objects 310–323
 - computationally assigned 287–288
 - defined 9, 17, 90
 - descriptive 288
 - early examples of 286
 - for education 319–321
 - embedded 287–288
 - encoding and transmission standards (METS) 313–316
 - Enrich panel 486–488
 - for eResearch 321–323
 - EXIF tags 246
 - Greenstone and 41, 485–494
 - for hierarchical documents 465–466
 - hierarchical organization 486
 - historical features 90
 - HTML documents 465–466
 - human-assigned 287
 - image-related 299–301
 - importing documents with assigned 488
 - individual records 491–494
 - interoperability 310
 - introduction 285
 - JPEG and JPEG 2000 246
 - key-phrase 336–339

-
- MARC 289–293
 - MARXML 293–294
 - moving on the Web 346
 - in MP3 files 231
 - for multimedia 299–310
 - multimedia 307–309
 - multiple-valued 475
 - non-bibliographic 91
 - notes and sources 339–341
 - in ODF files 205
 - Open Archives Initiative
 - Object Reuse and Exchange 319
 - in PDF files 195
 - in plain text documents 134
 - PNG. *see* PNG (Portable Network Graphics)
 - presenting 90–93
 - preservation 288
 - protocols 540–541
 - reviewing assigned 487–488
 - sharing 346, 350
 - space requirements 84
 - standards 91
 - static 288
 - storing 488–489
 - surrogates 93
 - technical 288
 - thumbicon 562
 - usage 288
 - user-supplied 71
 - video 306–307
 - on the Web 350
 - Z39.50 for retrieving 91
 - Metadata browsing 110–116
 - dates 113
 - facets 114–116
 - hierarchies 114
 - lists 111–113
 - notes and sources 123–125
 - Metadata cleaning 330
 - Metadata extraction
 - acronyms 335–336
 - adaptive systems 334
 - bibliographic references 334
 - from documents 332
 - Flash video format 277
 - generic entity 332–334
 - Greenstone 486–487
 - HTML documents 464–465
 - key-phrase 337
 - language identification 334–335, 340–341
 - phrase hierarchies 332
 - from text 330–339
 - training data 333
 - Metadata names 475
 - Metadata quality
 - authority control 324–327
 - dimensions of 323–330
 - error control 329–330
 - notes and sources 340
 - tools 330
 - Metadata sets 487
 - Metalanguages 128
 - METS (Metadata Encoding and Transmission Standards) 297, 313–316
 - notes and sources 340
 - Microfiche 146–147, 372
 - Microfilm 146–147
 - Microsoft Office Open XML (OOXML) 129, 195–196, 203–204
 - Microsoft Windows 134
 - Microsoft Word 129, 148, 195–196
 - Print Preview 19
 - Save as HTML option 203
 - text extraction 203
 - MIDI (musical instrument digital interface) 6, 106, 278–279
 - channel events 278–279
 - headers 278
 - meta events 279
 - system exclusive events 279
 - tracks 278
 - MIDI plug-in 505
 - Miller, Jon 41–43
 - MIME (Multipurpose Internet Mail Extensions) 210
 - Ming Dynasty novels 375
 - MIX (Metadata for Images) 302–304
 - Mixing media 262–263
 - MNG (Multiple-image Network Graphics) 237
 - Mobile phones 28, 61
 - Modified Discrete Cosine Transform 230
 - MODS (Metadata Object Description Schema) 297–298, 326
 - Mongolian 384
 - MONK (Metadata Offer New Knowledge) 407–408
 - Morphological reduction 99
 - Morrison Collection of Chinese Books 317
 - Motion picture industry 34
 - MP3 audio 228–229
 - bit rate 229
 - encoders and decoders 228
 - key technical points 229
 - metadata tags for 305
 - post-mp3 formats 229–231, 305
 - MP3 ID3 305, 340
 - MPEG 259–264
 - CCIR 601 and 259–261
 - HDTV 264
 - JPEG compared 260
 - mixing media 262–263
 - notes and sources 340
 - proprietary formats 264–266
 - MPEG audio 228–229
 - AAC 229–231
 - acoustic masking in 228
 - encoders and decoders 228
 - layers 228
 - mixing media 262–263
 - quality 229
 - standards 228
 - temporal masking 228
 - MPEG Layer III 228
 - MPEG-1 259–264
 - B-frame 261
 - I-frame 261
 - parameter values sets 262–263
 - P-frame 261
 - MPEG-2 259–264, 306
 - MPEG-21 259–264, 309–310, 340
 - MPEG-3 259–264
 - MPEG-4 230, 259–264, 306
 - MPEG-4 Timed Text 306
 - MPEG-7
 - application areas 309
 - components 307–308
 - DDL 307–308

- descriptors/description
 - schemes 307–308
- development of 306
- metadata 307–309
- multimedia features 307–309
- notes and sources 340
- scope 307
- Multilingual interfaces and documents 372–375
- Multimedia
 - application metadata 309–310
 - browsing 505
 - building collections 505
 - collage browsers 508
 - compression 216–221, 259–264
 - correcting metadata 505
 - in digital libraries 267–268
 - document surrogates 93
 - Greenstone collections 504–509
 - introduction 215
 - metadata 307–309
 - MIDI files incorporation 506–508
 - notes and sources 282–284
 - Ogg Theora 266–267
 - phrase browsers 508
 - rich media 271–277
 - searching 104–110
 - streaming 266
 - titles browser 506
 - transforms 216–221
 - See also* Audio; Images; Music; Video
- Multimedia artifacts, digitizing 216
- Multimedia content description interface
 - See* MPEG-7
- Multimedia documents
 - music 90–93
 - notes and sources 123–125
 - presentation 86–90
 - sound and pictures 86–87
 - video 90–93
- Multimedia signals, noise in 217
- Muradora 369
- Music
 - computer generated playback 89–90
 - copyright issues 6
 - digital music stand 582
 - introduction 277–281
 - MIDI 6, 106, 278–279
 - notes and sources 284
 - page turning 582
 - presentation 90–93
 - searching 104–106, 124
- Music digital libraries
 - killer app for 6–7
 - Native American 5
 - popular 6, 41–43
 - prototype 88
 - on the Web 6
- MXF (Material Exchange Format) 306
- MySQL 148
- Nairobi University Medical School Library 26
- Namespaces
 - Dublin Core standard 205
 - prefixes 205
 - RSLP 317
 - URN 204
 - XML 171–172, 346
- Naming with URN 204
- National libraries
 - See specific libraries*
- Native American culture, preserving 5
- Native Word formats 195–196
 - characteristics 202–203
 - in digital libraries 203
 - Fast Save mode 203
 - proprietary nature 202
- Natural order 104
- New York Public Library 8f, 402–403
- New Zealand Digital Library Project 41, 44, 149–152, 155, 372–375, 398
- New Zealand National Library Papers Past collection 81, 124, 418–419
- New Zealand School Journal 79, 124
- N-gram 335
- Noise in multimedia signals 217
- Non-ASCII characters 155–157
- NSF/DARPA/NASA Digital Libraries Initiative 45–46
- NTSC videotapes 261
- OAI (Open Archives Initiative) 345–350
 - Greenstone and 541–542
 - HTTP status code mechanism 348
 - as open protocol 361–369
- OAI-ORE (Open Archives Initiative Object Reuse and Exchange) 319
- OAI-PMH (Open Archives Initiative Protocol for Metadata Harvesting) 343, 369–370
 - data providers 346
 - framework 346
 - GetRecord request 346
 - harvesting 350
 - Identify request 346–348
 - interaction support 346
 - ListIdentifiers request 346–348
 - ListMetadataFormats request 346–348
 - ListRecords request 346–348
 - ListSets request 348
 - metadata element 346
 - record identifiers 346
 - serving 348–349
 - technical goal 346
- Object identification 350–354
 - DOIs 352–353
 - handles 351
 - OpenURLs 353
 - persistence 353–354
- Objects
 - digital 7
 - RTF 201–202
- OCLC (Online Computer Library Center) 32, 71, 350–351
- OCR (optical character recognition) 104, 140–146
 - accuracy 140, 148–149
 - acquisition stage 141–142

- checking stage 146
- cleanup stage 141
- defined 140
- errors 76, 79–81
 - allowing users to see 81
 - correcting 78–79
- example project 149–152
- grayscale image brightness 140
- high-performance products 212
- image resolution requirements 139–140
- interactive. *see* Interactive
- OCR
 - manual typing vs. 141
 - newsgroup 212–213
 - non-Western languages 142–145
 - notes and sources 212
 - output formats 148
 - page analysis stage 141–142
 - page handling 146–147
 - problems generated 141
 - saving stage 146
 - shops 148–149
 - training 145
 - vendors 212–213
- See also* Digitizing; Recognition; Scanning
- OCR engines 212
- OCR shops 148–149, 212–213
- Ocrad 212
- ODF (Open Document format)
 - 129, 204–207, 213
 - in digital libraries 206–207
 - files 205–206
 - formatting 206
 - lossless compression 217
- Ogg Theora 266–267
- Ogg Vorbis 229–231, 305
- OIDs (object identifiers)
 - assigning to documents 532
 - Greenstone and 532–534
 - persistence 57, 71, 351
- OLE (object linking and embedding) 201–202
- Omar, Caliph 11
- Omnipage 151
- OMR (Optical Music Recognition) 284
- OOXML (Office Open XML) 129, 203–204, 213
 - Open Document format (ODF) 204–207
- OPACs (Online public access catalogs) 17, 50
- Open Source projects 33, 125
- Open Video Project 268, 283–284
- OpenID standard 360
- OpenOffice suite 196–197
- OpenURLs 353
- AND operator 95
- Oral cultures 427–429, 431
- Oral histories 5, 86–87, 124
- Oral History collection 86–87, 124
- Ordering
 - alphabetical lists 372, 388
 - lists of words in Chinese 396–398
 - radicals 396
 - stroke-based 396
- Organization for the Advancement of Structured Information Standards 129, 196–197, 213
- OSI (Open System Interconnection) Reference Model 344
- Oxford Text Archive 162–163
- Oxford University Library 12
- Paedophilia/pornography on the Internet 37–38, 45
- Page analysis stage 141–142
 - tables and 142
- Page description languages 177–195
 - introduction 129
 - PDF 190–195
 - PostScript 177–182
- Page handling 146–147
 - books 146
 - deskewing and 147
 - microfilm 146–147
 - See also* Scanning
- Page images 79–81, 462–463
- Pages, turning 582
- PAL videotapes 261
- Paper, computer 19
- Passwords 359
- Patents 32
 - LZW scheme 236
 - MPEG techniques 229
 - multimedia formats unencumbered by 230
 - Web pages 420
- Patriot Act, USA 55, 71
- PCM (pulse code modulation) 222–224
 - adaptive differential 226
 - companding 224–225
 - linear 224, 226
 - variants 224–226
- PDF (Portable Document Format) 177–195
 - Acrobat reader 192
 - annotation 193
 - compression 193–194
 - data types 192
 - document rendering 191
 - features 192–194
 - hyperlinks 192–193
 - interactive display features 190
 - interactive features 193
 - introduction 129
 - linearized 194
 - lossless compression 217
 - metadata in 195
 - notes and sources 213
 - object section 191
 - OCR and 146
 - online display features 195
 - PostScript and 195
 - PostScript vs. 190
 - printing digitized documents in 81–84
 - programming problems 195
 - redaction 194
 - searchable image options 193
 - security and 194–195
 - versions 190
 - white space and 189
 - See also* Page description languages
- PDF files
 - converting to page images 462–463
 - enhanced handling 461–464

- Greenstone and 456–458
- highlighting search terms 464
- mixed collections 463–464
- splitting into sections 462
- trouble-shooting 461–462
- PDF plug-in 462
- PDF/A documents 190–191
- PDF/A-1 190–191
- Peace Corps, U.S. 26
- Pearce, Adrian 413, 420
- Pergamos Digital Library 20–25, 22f–25f
- Pergamum library 11
- Persistence 351, 353–354
- Personal Information Retrieval Assistant 407–408
- P-frame 261
- Phaistos Disk 17–19
- Philosophical Transactions of the Royal Society* 12
- Photographic digital libraries 248–249, 301
- Photographic images 86–87, 302
- Photoshop 148
- Phrase queries 100–101
- Phrase searching
 - postretrieval scan 100–101
 - with punctuation and white space 101
 - thesauri for 424
 - word-level index 100–101
 - See also* Searching
- Physical libraries 7
 - academic, DRM and 34
 - access-preservation tension 418–419
 - acquisitions 51
 - aggressive weeding 13–15
 - aids to information-seeking in 17
 - are for all 43, 50
 - automation systems 45–46
 - changing face of 9–20, 61
 - Chinese 9, 10f, 19, 375
 - classification schemes 114
 - collection development 51, 70
 - destruction of ancient 11
 - digital libraries compared 7–9, 49, 56, 64, 67
 - digitization of physical
 - content in 51–52
 - first 43
 - funding 50
 - future of 52–54
 - growth in 419
 - innovation in 71
 - IT support requirements 52
 - legal deposit law 12
 - mission of 402
 - oldest surviving 9, 10f, 19, 375
 - online reference services 62–63
 - physical and digital elements 49–54
 - physical environment 402–403
 - preservation of 408–420
 - public/open-access 11, 54
 - restricted use of 10
 - role of 63
 - See also* American Library Association; Digital libraries; *specific libraries*
- Physical library catalogs
 - 20th century card type 286
 - history of 17
 - information in 17
 - as metadata 20
 - Trinity College 14f
- Physical library collections 13–15
- Physics archival digital libraries 4, 41–43
- Pinyin browsing 398
- Piracy 34
- Pixels 139
 - PNG format 237
- Plain text 133–134, 137–138
- Plug-ins 88, 457
 - Document Plugins 459
 - HTML 465
 - Image 495–496, 562
 - MARC 459
 - MIDI 505
 - PDF 462
 - Text 501
 - Web browser 275
 - WordPlugin 459
 - ZIP 495
- PNG (Portable Network Graphics) 236–237, 299–300
 - animated image support 237
 - filters 237
 - gamma correction feature 237
- Gzip 237
- image-related metadata 299–300
- pixel restrictions 237
- progressive refinement 244
- transparency support 246
- two-dimensional interfacing support 237
- Polynesia 38
- Portuguese interface 372–375
- Postretrieval scan 100–101
- PostScript 177–195
 - code fragments 185–186
 - in digital libraries 189–190
 - Encapsulated 182
 - evolution 181–182
 - fonts 182–185
 - fundamentals 177–182
 - graphical components 178t
 - introduction 129
 - language 178–181
 - limitations 190
 - lossless compression 217
 - notes and sources 213
 - PDF and 195
 - PDF compared 190
 - rasterization/rendering 177–178
 - RTF vs. 196
 - stack overflow/stack underflow errors 189
 - text extraction 185–189
 - word-processor file versions 189–190
 - word-space threshold 188
 - See also* Page description languages
- PostScript documents 177–178
- PostScript fonts 182–185
 - compatibility with Unicode 184–185
 - composite 182–184
 - formats 183–184
- PostScript Level 2 181

-
- PostScript Level 3 181–183
 - PostScript program 177
 - Practical Digital Libraries* (Lesk) 45–46
 - Presentation
 - audio 86–87
 - document surrogates 90–93
 - documents
 - hierarchically structured 465–466
 - marked-up 561
 - multimedia 86–90
 - textual 74–86
 - format statements 560–567
 - institutional repositories 116–123
 - introduction 73–74, 559
 - macros 559, 563–565
 - markup 128
 - metadata browsing 110–116
 - metadata document
 - surrogates 90–93
 - MIT repository 116–123
 - multimedia 93, 104–110
 - music 90–93, 104–106
 - notes and sources 123–125
 - realistic books 84–86, 124
 - searching 93–110
 - video 90–93
 - Presentation, images
 - page 79–81
 - photographic 86–87
 - searching 106–110
 - with text 81–84
 - Presentation files 210
 - Presentation patterns
 - additional static pages 567–569
 - dynamic HTML 573–577
 - Exploiting Asynchronous JavaScript and XML (AJAX) 578–582
 - features 470–472
 - format panel 469–470
 - format statements 469–485, 559, 571–573
 - format statements, default 472
 - making formats statements reusable through macro definitions 571–573
 - using JavaScript to adjust presentation 569–571
 - Preservation 401
 - acid-free paper 146
 - difficulties 410–411, 413–415, 417
 - digital documents 413–415
 - duplication strategy 416
 - Emacs Manual 125
 - emulation strategy 416–417
 - humans and 416, 419
 - migration strategy 417
 - notes and sources 429
 - of old technology 411–415
 - paper technology and 410–411
 - of physical libraries 408–420
 - in practice 418–419
 - strategies 415–420
 - technology 414
 - Printing
 - computer output 19
 - devices 139
 - history of 12, 389
 - Privacy 54–61, 71, 519–521
 - Progressive refinement 242–245
 - interlacing 242
 - JPEG format 244
 - spectral selection 245
 - successive approximation 245
 - See also* Images
 - Project Gutenberg 123–124
 - distributed proofreading 78–79
 - overview 78
 - volunteers 78–79
 - Public domain 30–32
 - digitizing documents in the 34–35
 - humanitarian information in the 26
 - Public good 54, 68–71
 - Public lending right 30
 - Public library movement 11
 - Publishing industry
 - decline 419
 - DRM and scholarly works 34
 - e-Books 33
 - effect of digital libraries on 29
 - Punctuation 101
 - ASCII 380
 - phrase searching with 101
 - Unicode 388
 - word segmentation 137
 - “The Pure Theory of Public Expenditure” (Samuelson) 71
 - PURL (Persistent URL) 351
 - Quantization 218
 - Queries
 - Boolean 95, 136
 - case-folding 98–100, 104
 - changing search options
 - between 103
 - fielded 104
 - full-text retrieval system 134–137
 - large 102
 - large-query search 103
 - phrase 100–101
 - ranked 97, 104, 136
 - restricting 103
 - sorting 104
 - stemming and 98–100
 - types of 95–98
 - See also* Ranked queries; Searching
 - Query by humming 106
 - Query interfaces 102–104
 - Query ranking 97
 - QuestionPoint software system 71
 - Ramses IV 17–19
 - Ranganathan, S. R. 47, 50, 70
 - Ranked queries 97, 104
 - with Booleans 97
 - coordinate matching 97–98
 - full-text retrieval system 136
 - phrases and 101
 - Ratings 68
 - RDF (Resource Description Framework) 310–313
 - construction 311
 - containers 313
 - modeling graphically with 311
 - XML and 310
 - RDF Schema 310
 - RDF site summary
 - See* RSS feeds

- Realistic books 84–86, 124
 Really simple syndication
 See RSS feeds
 Recognition stage in OCR 142–145
 Redaction 194
 Refer 340
 Registry of Copyright Evidence 32
 Repository, static 349
 Repository management 53, 346
 Repository systems
 DSpace 361–364
 Fedora 361–364
 Fez 369
 Reproduction rights 30
 Research
 in digital libraries of the future 407–408
 electronic 321–323
 electronic preprint community 345–346
 multilingual documents 372–375
 physics archival digital libraries 4
 scholarly journals 12, 321–323
 Research Support Libraries Programme (RSLP)
 collection description 317
 namespaces 317
 Research Support Libraries Programme (RSLP)
 Collection Description project 317
 Researchers, libraries defined by 7
 Resolution 139–140
 RESTful (Representational State Transfer) 356
 Rich media
 Adobe Flash 275–277
 introduction 271–277
 notes and sources 284
 synchronized multimedia integration language (SMIL) 271–275
 Rich site summary
 See RSS feeds
 RIFF (Resource Interchange File Format) 226
 Rights Expression language 309–310
 Robot exclusion protocol 36
 Romania 148–149, 212–213
 Round-trip compatibility 133
 RSLP 317
 RSS 1.0 122
 RSS 2.0 122
 RSS feeds 122
 RTF (Rich Text Format) 197–202
 * command 199
 \ud command 199
 \upr command 199
 annotations 201
 backslash character 197
 backward compatibility 199
 basic types 198–199
 begin-/end-bookmark 201–202
 braces 199
 color table 200
 destination command 198–199
 in digital libraries 202
 document body 200
 field entities 201
 file structure 199–201
 file table 200
 flag command 198
 formatting commands 197
 graphics 201
 headers 198
 introduction 129
 list tables 200
 notes and sources 213
 objects 201–202
 PostScript vs. 196
 revision tables 200
 specifications 198
 style sheet 200
 symbol command 199
 table command 199
 tables 199
 text extraction 202
 toggle command 198
 value command 198
 in Word documents 195–196
 See also Word-processor documents
 Sahel Point Doc collection 372–375
 Salo, Dorothea 53, 70
 SAML (Security Assertion Markup Language) 360
 Samuelson, Paul 54
 Santayana, George 103, 124–125
 Saving stage (interactive OCR) 146
 Scaling up 466–469
 Scanning 139–140
 digitization noise 235
 microfiche and microfilm 146–147
 page handling 146–147
 PostScript documents 186
 quality 139
 resolutions 139
 test pages 147
 See also Digitizing; OCR (optical character recognition)
 Scharfes character 142–145, 213
 Scientific documents 207–210
 Scientific/technical authors 297
 SCORM 321
 Scotland National Library 12
 Search engines 36
 Search terms
 common number of 102
 highlighting 36–37, 78, 81
 Searching
 audio 86–87
 browsing vs. 111
 fielded 104
 full-text 134–137, 536–537
 Greenstone 517, 536–539
 Greenstone Digital library Software 536–539
 high precision 95–97
 high recall 95–97
 images 106–110, 125
 multimedia 104–110
 music 104–106, 124
 notes and sources 123–125, 212
 photographs 86–87
 presentation 93–110
 video 88
 See also Queries

-
- Semantic Web 354
 - Service providers 346
 - SGML 152–153, 159–160
 - Shibboleth system 360
 - Shockwave 275, 284
 - Simple Words* 212–213
 - Single sign-on 360
 - Site configuration file 519
 - Slovak 388
 - SMF (Standard MIDI file format) 278
 - SMIL (synchronized multimedia integration language) 271–275
 - Smith, Lloyd 41–43
 - SOAP (Simple Object Access Protocol) 355–356
 - Society of Motion Picture and Television Engineers (SMPTE) 306
 - Software
 - digital rights management 33
 - GNU Project 33
 - source code 516
 - third-party packages 516–517
 - Software robots 36
 - Sony Bono Copyright Term Extension Act 31
 - Sound and pictures 86–87
 - Source code 516
 - Spanish
 - accent folding 387
 - OCR errors and 142–145
 - Spectral selection 245
 - Spreadsheets 210
 - SRU (Search/Retrieval via URL) 343, 357–359
 - SSL (Secure Socket Layer) 359–360
 - Stack overflow/stack underflow errors 189
 - Stallman, Richard 33, 122–123
 - Static repository 349
 - Static Repository Gateway 349
 - Steles 9, 19
 - Stemming 98–100
 - highlighting search terms and 100
 - Stop words 136
 - Streaming 266
 - Strict HTML 154, 158
 - Stroke-based browsing 398
 - Style sheet languages 128
 - CSS (cascading style sheets). *see* CSS (cascading style sheets)
 - XSL (extensible stylesheet language). *see* XSL (extensible stylesheet language)
 - Style sheets 163–176
 - cascading 166–168
 - development of 163
 - ordering of rules in 166
 - Sumarian clay tablets 286
 - Supplementary Ideographic Plane 384
 - Supplementary Multilingual Plane 384
 - Supplementary Special-Purpose Plane 384
 - SVG (scalable Vector Graphics) 252–257
 - filters 257
 - fonts 257
 - markers 257
 - versions 257
 - Swedish 388
 - SWF 275
 - Synchronized Multimedia Working Group 271–272
 - Tables
 - adding headers 576–577
 - CSS 166–168
 - in interactive OCR 142
 - opening and closing
 - interactively 574–576
 - page analysis stage and 142
 - RTF 199
 - XSL 172
 - Tagged image file format
 - See* TIFF (Tagged Image File Format)
 - Tags
 - anchor element 157
 - DTDs 160
 - EXIF 246
 - Flash video format 275–276
 - HTML 155
 - ID3 305
 - image metadata 302
 - MARC records 289–291
 - regular expressions 158
 - removing 158
 - TIFF 247, 300
 - training data 334
 - XML 154, 158
 - XSL 171–172
 - Taiwan 130–132
 - Tang poetry collection 375, 399
 - Taverna 323
 - TCP (Transmission Control Protocol) 344
 - Technology in digital libraries 39t, 40
 - Technology/technological
 - infrastructure in developing countries 25–26, 28–29
 - Technophile visionaries 15–16
 - TEI (Text Encoding Initiative) 162
 - Telephony 224–225
 - Television, high definition digital 264
 - Television standards 259–261
 - Tesseract 212
 - TeX 297
 - Text
 - annotations 582
 - ASCII 130–132
 - compression 236, 395
 - extracting from HTML documents 158
 - indexing 134–137
 - lossless compression 217, 236
 - mining. *see* Metadata extraction
 - segmenting 394–396
 - surrogates 93
 - Unicode 132–133
 - word segmentation 137
 - Text extraction 185–189
 - based on scanning for strings 186
 - improving output 188–189
 - plain text documents 134
 - PostScript files 185–189
 - RTF documents 202
 - simple program 186–187
 - Word documents 203

- Text mining 330
- Text plug-in 501
- Textual documents
 - documents, chapters, sections 74–76
 - e-mail 210–211
 - images with text 81–84
 - introduction 127
 - notes and sources 123–125, 211–213
 - page images 79–81
 - presentation 74–86
 - presentation files 210
 - realistic books 84–86, 124
 - representing 130–137
 - spreadsheets 210
 - unstructured 76–79
- Textual images 137–152
 - adding documents 501–502
 - building collections 498–499
 - displaying scanned images 500
 - Greenstone collections 497–504
 - grouping by series 499–500
 - scanning 139–140
 - searching at page level 500–501
 - switching between images and 502–504
- Thai language 394, 396
 - Unicode character set 375–380
- Thesaurus 337, 424
- Thimbleby, Harold 43
- Thumbicon 562
- Thumbnails 93, 192–193, 302, 496
- TIFF (Tagged Image File Format) 245–248
 - compression 247, 300–301
 - in digital library use 247
 - file-offset mechanism 247–248
 - image metadata 300–301
 - jail breaking 247–248
 - metadata embedded in 302
 - notes and sources 340
 - security vulnerability 247–248
- software libraries 247
- tags 247, 300
- TLS (Transport Layer Security) 359–360
- Training
 - generic entity extraction 333
 - OCR systems 145
- Transforms
 - Discrete Cosine Transform 219, 239
 - Fourier transform 219–221
 - introduction 216–221
 - inverse versions 219
 - Modified Discrete Cosine Transform 230
 - overview 219
 - Wavelets 219
- Transitional HTML 154, 158
- Transparency 246
- Trinity College Library 12–13, 43
- Typing
 - Indic scripts 389–390
 - OCR vs. 141
- UCS (Unicode Character Set) 133
- UDDI (Universal Description, Discovery and Integration) protocol 355–356
- U.K. Open University 125
- UNESCO 37–38, 45, 372–375
- Unicode 127, 132–133
 - accent folding 133, 387
 - alphabetic ordering 388
 - backward compatibility 132–133, 384, 386
 - Basic Multilingual Plane 375, 384
 - bidirectional processing 133–134, 384
 - big-endian vs. little-endian byte ordering 384–385
 - canonical equivalence/decomposition 383
 - character set 155–157
 - code allocation 375–380
 - code points 382
 - code space zones 380, 389
 - collation algorithm 388
 - comparison string 383
 - compatibility equivalence 383
 - conjuncts 392
 - data structures, writing to disc 385, 388
 - in digital libraries 387–389
 - fonts 184–185, 389
 - Hindi and 389–393
 - for Indic scripts 384, 389–393
 - for integers 384–385
 - internationalization and 375–389
 - ISO 10646-1 standard 375
 - for Korean Hangul ideographs 375–380, 383
 - normalized forms 383
 - notes and sources 398–399
 - official description 399
 - official title 212
 - plain text documents 133–134
 - punctuation 388
 - round-trip compatibility 133, 382–383
 - sorting string 388
 - standard 375
 - Supplementary Ideographic Plane 384
 - Supplementary Multilingual Plane 384
 - Supplementary Special-Purpose Plane 384
 - symbols 383
 - translation tables 517
 - translations tables, Greenstone 517
 - universal character set 381
 - word segmentation 394–398
- Unicode character
 - encodings 384–387
 - byte-order mark 386
 - UTF-16 386–387
 - UTF-16BE variant 386
 - UTF-16LE variant 386
 - UTF-21 384
 - UTF-32 384–386
 - UTF-8 133, 157, 385–388
- Unicode characters
 - combining 381–384
 - composite 381–384
 - deprecated 384
 - glyphs 382, 389

- ligature 382, 392–393
- number of 375
- surrogates 385–387
- Unicode Consortium 132
- Unisys 236
- United Kingdom copyright law 30–31
- United Kingdom National Library 13
- United States
 - copyright law 30–31, 34
 - First Amendment protections 38
 - illegal and harmful material
 - on the Internet 38
 - legal deposit law 12
 - NSF/DARPA/NASA Digital Libraries Initiative 45–46
 - Patriot Act 55, 71
- Unix 134
- URI (Universal Resource Identifier) 311, 351
- URLs (uniform resource locators) 65, 311
 - defined 65
 - reliability 351
 - search/retrieval via 348–349, 357–359
 - success of 350
- URNs (universal resource names) 204, 311, 351
- U.S. National Science Digital Library (NSDL) 350
- User contributions 67–70
 - annotations 67, 71
 - corrections 68
 - Hamilton, New Zealand oral history project 86–87
 - keywords 67–68
 - new documents 68
 - notes and sources 71
 - partial and fluid documents 68–70
 - Project Gutenberg 78
 - proofreading 78–79
 - ratings 68
 - Wikipedia project 69
- Users of digital libraries
 - See Library patrons
- User-supplied metadata 71
- Ussher, Archbishop 12
- UTF (UCS Transformation Format) 133
- UTF-16 386
- UTF-32 385–386
- UTF-8 133, 157, 386–387
- Vanity searches 71
- VBA (Visual Basic for Applications) 148
- Vector graphics images 252–257
- Vector quantization 218
- Video
 - codecs 258
 - compression 258
 - containers 258
 - data container formats 306
 - downloading 88
 - Flash video format 88
 - high definition digital television 264
 - introduction 258–271
 - metadata 306–307
 - MPEG 259–264
 - notes and sources 283–284
 - plug-ins for playing 88
 - presentation 90–93
 - searching 88
 - streaming 266
 - surrogates 93
 - wipes 582
- Video digital libraries 5, 268, 307
- Virtual libraries 16–17
 - defined 16–17
 - notes and sources 45–46
 - reference services 61–62, 71
- VisiCalc 6–7
- VLists 561
 - runtime system support 576
 - specification 576
- Vocabulary
 - controlled 329, 337
 - standardized 327
- Voice recordings 230–231
- Volunteers
 - Hamilton, New Zealand oral history project 86–87
 - Project Gutenberg 78–79
- VRML (Virtual Reality Modeling Language) 157
- Wales National Library 12
- WAV audio 226–228, 247
 - associated data list chunk 227
 - chunks types 226, 305
 - cue chunk 227, 305
 - file extensions 226–227
 - instrument chunks 227
 - labeled text chunks 227
 - playlists 227
 - sampler chunks 227
- Wavelets 219
- Wayback Machine 420
- Web browsers
 - APNG supported on 237
 - backward compatibility 154–155
 - foreign language support 375
 - image formats supported 248
 - JPEG 2000 support 248
 - lynx 158
 - plain text 158
 - plug-ins 275
 - realistic books 84–86
 - security 359–360
 - TIFF support 248
- Web cookies 59–60
- Web culture 153
- Web documents
 - CSS 163–176
 - HTML 152–163
 - XML 163–176
- Web harvesting 350
- Web mirroring 350
- Web servers
 - on iPod-touch 424
 - link structure 350–351
 - privacy of information 359
 - recording user data 58
- Web Service requests
 - HTTP GET/POST requests 356
 - SOAP messages 355–356
 - stateless 356
- Web Services 343, 354–359

- search/retrieval via URL (SRU) 357–359
- self-advertising technology 356
- UDDI (Universal Description, Discovery and Integration) protocol 355–356
- WSDL (Web Services Description Language) 354
- Web sites
 - digital libraries vs. 9
 - downloading 539–540
 - Greenstone 443
 - patents and 420
 - robot exclusion protocol 36
- Wells, H. G. 15–16, 43, 68, 402
- White space 137, 152–153, 162–163, 189
- Wikipedia project 69, 71, 339, 425
- Wilkes, Maurice 43
- William the Conqueror 412
- “Winners and losers in the global research village” (Ginsparg) 41–43
- WIPO (World Intellectual Property Organization) Treaty 33–34
- Wisdom, defined 7–8
- Word boundary 371
- Word files
 - detecting user-defined styles 459–460
 - enhanced handling 458–461
 - extracting document properties 460–461
 - extracting document structure 459
 - Greenstone and 456–458
- Word segmentation 137, 394–398
 - failure 395–396
 - with language dictionary 395
 - punctuation 137
 - sorting Chinese text 396–399
 - space-insertion method 395
 - with text compression model 395
 - Thai/Khmer/Lao 396
- Word-level index 100–101, 136
- Word-processor documents 129, 195–210, 465–466
 - LaTeX 129, 207–210
 - native Word 202–203
 - Office Open XML (OOXML) 129, 196, 203–204
 - Open Document format (ODF) 129, 196–197, 204–207
- transferring between applications 196
- See also* PDF (Portable Document Format); RTF (Rich Text Format); Word files
- Works
 - copyright. *see* Copyright
 - digital representation of. *see* Digitizing
 - illegal and harmful material 37–38
 - in the public domain 30–32
- World Bank 44
- World brain concept 15–16
- World Wide Web
 - anonymous use 57
 - archiving 36
 - copyright and collecting from the 35–37
 - delivery of digital library content 216
 - digital libraries vs. 9
 - GIF format used 236
 - law enforcement on the 36
 - libraries vs. 7
 - metadata scheme for content on 350
 - moving metadata on the 346
 - persistence of information on 57
 - user log displays 59
- World Wide Web 2.0 67
- WorldCat 350
- Writing
 - ancient devices for portable 18f
 - Chinese 9, 10f
 - portable 17–19
- WSDL (Web Services Description Language) 354
- Xandar’s digital library 404–405
- Xerox 132
- XHTML 128, 153–154
- X’ian steles 9, 10f, 19, 375, 408–410
- XLink (XML linking language) 154
- XML (extensible markup language) 163–176
 - Asynchronous JavaScript and (AJAX) 578–582
 - attributes 161
 - basic 159–162
 - context-formatting 173–176
 - DIG35 standard 302
 - in digital libraries 162–163
 - DTDs 159–160
 - header declaration line 164
 - history of 152
 - HTML files and 152–153
 - introduction 128
 - markup 153–155
 - media-dependent formatting 173–176
 - metalanguages 128
 - namespaces 171–172, 346
 - notes and sources 211–213
 - parsing 162
 - power and flexibility 163
 - RDF format and 310
 - rendering documents in 196
 - RSS feeds 122
 - SGML and HTML relationship 153
 - stylesheet languages 153–155
 - SVG in 252
 - syntax 160
 - tags 154, 158
 - using formatting objects 172–173
- XML Schema 154, 307–308, 346
- XMP (Extensible Metadata Platform) 195
- XPath 170, 173
- XSL (extensible stylesheet language) 128, 170–176

-
- context-bases matching 173
 - context-sensitive formatting 173–176
 - CSS vs. 170
 - Formatting Objects (FO) 170, 172–173
 - media-dependent formatting 173–176
 - power and flexibility 154
 - processing in 176
 - specifications 170
 - tables and lists 172
 - tags 171–172
 - template rules 172–173, 175
 - transformations 172
 - XSLT (XSL transformations) 170
 - YouTube, Flash video format 88, 275
 - Z39.50 protocol 343
 - Access Control Facility 344
 - bibliographic registry 344
 - clients 344
 - Explain Facility 344–345
 - Extended Services Facility 344–345
 - facilities 344
 - Initialization Facility 345
 - interoperability 344–345
 - Library of Congress interface 344
 - metadata retrieval 91
 - notes and sources 369
 - query syntax 344
 - registries 344
 - Retrieval Facility 344
 - Search Facility 344
 - Termination Facility 344–345
 - Zebra indexing 369
 - Zia Pueblo digital library 5, 41–43
 - ZIP files 204
 - ZIP plug-in 495