

How to Build a Digital Library



Ian H. **WITTEN**

David **BAINBRIDGE**

How to Build a Digital Library

The Morgan Kaufmann Series in Multimedia Information and Systems

Series Editor, Edward A. Fox, Virginia Polytechnic University

How to Build a Digital Library

Ian H. Witten and David Bainbridge

Digital Watermarking

Ingemar J. Cox, Matthew L. Miller, and Jeffrey A. Bloom

Readings in Multimedia Computing and Networking

Edited by Kevin Jeffay and HongJiang Zhang

Introduction to Data Compression, Second Edition

Khalid Sayood

Multimedia Servers: Applications, Environments, and Design

Dinkar Sitaram and Asit Dan

Managing Gigabytes: Compressing and Indexing Documents and Images,
Second Edition

Ian H. Witten, Alistair Moffat, and Timothy C. Bell

Digital Compression for Multimedia: Principles and Standards

Jerry D. Gibson, Toby Berger, Tom Lookabaugh, Dave Lindbergh, and
Richard L. Baker

Practical Digital Libraries: Books, Bytes, and Bucks

Michael Lesk

Readings in Information Retrieval

Edited by Karen Sparck Jones and Peter Willett

How to Build a Digital Library

Ian H. Witten

Computer Science Department
University of Waikato

David Bainbridge

Computer Science Department
University of Waikato



MORGAN KAUFMANN PUBLISHERS

AN IMPRINT OF ELSEVIER SCIENCE

AMSTERDAM BOSTON LONDON NEW YORK
OXFORD PARIS SAN DIEGO SAN FRANCISCO
SINGAPORE SYDNEY TOKYO

Publishing Director
Assistant Publishing Services Manager
Senior Developmental Editor
Editorial Assistant
Project Management
Cover Design
Text Design
Composition
Copyeditor
Proofreader
Indexer
Printer

Diane D. Cerra
Edward Wade
Marilyn Uffner Alan
Mona Buehler
Yonie Overton
Frances Baca Design
Mark Ong, Side by Side Studios
Susan Riley, Side by Side Studios
Carol Leyba
Ken DellaPenta
Steve Rath
The Maple-Vail Book Manufacturing Group

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. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

Morgan Kaufmann Publishers
An imprint of Elsevier Science
340 Pine Street, Sixth Floor
San Francisco, CA 94104-3205
www.mkp.com

© 2003 by Elsevier Science (USA)
All rights reserved.
Printed in the United States of America

07 06 05 04 03 5 4 3 2 1

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, or otherwise—without the prior written permission of the publisher.

Library of Congress Control Number: 2002107327
ISBN: 1-55860-790-0

This book is printed on acid-free paper.

Contents

List of figures	xiii
List of tables	xix
Foreword	
by Edward A. Fox	xxi
Preface	xxv

1. Orientation: The world of digital libraries 1

Example One: Supporting human development	1
Example Two: Pushing on the frontiers of science	2
Example Three: Preserving a traditional culture	3
Example Four: Exploring popular music	4
The scope of digital libraries	5
1.1 Libraries and digital libraries	5
1.2 The changing face of libraries	8
In the beginning	10
The information explosion	11
The Alexandrian principle	14
Early technodreams	15
The library catalog	16
The changing nature of books	17

- 1.3 Digital libraries in developing countries 20
 - Disseminating humanitarian information 21
 - Disaster relief 21
 - Preserving indigenous culture 22
 - Locally produced information 22
 - The technological infrastructure 23
- 1.4 The Greenstone software 24
- 1.5 The pen is mighty: Wield it wisely 28
 - Copyright 29
 - Collecting from the Web 31
 - Illegal and harmful material 34
 - Cultural sensitivity 34
- 1.6 Notes and sources 35

2. Preliminaries: Sorting out the ingredients 39

- 2.1 Sources of material 40
 - Ideology 41
 - Converting an existing library 42
 - Building a new collection 43
 - Virtual libraries 44
- 2.2 Bibliographic organization 46
 - Objectives of a bibliographic system 47
 - Bibliographic entities 48
- 2.3 Modes of access 55
- 2.4 Digitizing documents 58
 - Scanning 59
 - Optical character recognition 61
 - Interactive OCR 62
 - Page handling 67
 - Planning an image digitization project 68
 - Inside an OCR shop 69
 - An example project 70
- 2.5 Notes and sources 73

3. Presentation: User interfaces 77

- 3.1 Presenting documents 81
 - Hierarchically structured documents 81
 - Plain, unstructured text documents 83

	Page images	86
	Page images and extracted text	88
	Audio and photographic images	89
	Video	91
	Music	92
	Foreign languages	93
3.2	Presenting metadata	96
3.3	Searching	99
	Types of query	100
	Case-folding and stemming	104
	Phrase searching	106
	Different query interfaces	108
3.4	Browsing	112
	Browsing alphabetical lists	113
	Ordering lists of words in Chinese	114
	Browsing by date	116
	Hierarchical classification structures	116
3.5	Phrase browsing	119
	A phrase browsing interface	119
	Key phrases	122
3.6	Browsing using extracted metadata	124
	Acronyms	125
	Language identification	126
3.7	Notes and sources	126
	Collections	126
	Metadata	127
	Searching	127
	Browsing	128
4.	Documents: The raw material	131
4.1	Representing characters	134
	Unicode	137
	The Unicode character set	138
	Composite and combining characters	143
	Unicode character encodings	146
	Hindi and related scripts	149
	Using Unicode in a digital library	154
4.2	Representing documents	155
	Plain text	156

	Indexing	157
	Word segmentation	160
4.3	Page description languages: PostScript and PDF	163
	PostScript	164
	Fonts	170
	Text extraction	173
	Using PostScript in a digital library	178
	Portable Document Format: PDF	179
	PDF and PostScript	183
4.4	Word-processor documents	184
	Rich Text Format	185
	Native Word formats	191
	LaTeX format	191
4.5	Representing images	194
	Lossless image compression: GIF and PNG	195
	Lossy image compression: JPEG	197
	Progressive refinement	203
4.6	Representing audio and video	206
	Multimedia compression: MPEG	207
	MPEG video	210
	MPEG audio	211
	Mixing media	212
	Other multimedia formats	214
	Using multimedia in a digital library	215
4.7	Notes and sources	216
5.	Markup and metadata: Elements of organization	221
5.1	Hypertext markup language: HTML	224
	Basic HTML	225
	Using HTML in a digital library	228
5.2	Extensible markup language: XML	229
	Development of markup and stylesheet languages	230
	The XML metalanguage	232
	Parsing XML	235
	Using XML in a digital library	236
5.3	Presenting marked-up documents	237
	Cascading style sheets: CSS	237
	Extensible stylesheet language: XSL	245

5.4	Bibliographic metadata	253
	MARC	254
	Dublin Core	257
	BibTeX	258
	Refer	260
5.5	Metadata for images and multimedia	261
	Image metadata: TIFF	262
	Multimedia metadata: MPEG-7	263
5.6	Extracting metadata	266
	Extracting document metadata	267
	Generic entity extraction	268
	Bibliographic references	270
	Language identification	270
	Acronym extraction	271
	Key-phrase extraction	273
	Phrase hierarchies	277
5.7	Notes and sources	280
6.	Construction: Building collections with Greenstone	283
6.1	Why Greenstone?	285
	What it does	285
	How to use it	288
6.2	Using the Collector	292
	Creating a new collection	293
	Working with existing collections	300
	Document formats	301
6.3	Building collections manually: A walkthrough	302
	Getting started	303
	Making a framework for the collection	304
	Importing the documents	305
	Building the indexes	307
	Installing the collection	308
6.4	Importing and building	309
	Files and directories	310
	Object identifiers	312
	Plug-ins	313
	The import process	314
	The build process	317

- 6.5 Greenstone archive documents 319
 - Document metadata 320
 - Inside the documents 322
- 6.6 Collection configuration file 323
 - Default configuration file 324
 - Subcollections and supercollections 325
- 6.7 Getting the most out of your documents 327
 - Plug-ins 327
 - Classifiers 336
 - Format statements 342
- 6.8 Building collections graphically 349
- 6.9 Notes and sources 353

7. Delivery: How Greenstone works 355

- 7.1 Processes and protocols 356
 - Processes 357
 - The null protocol implementation 357
 - The Corba protocol implementation 359
- 7.2 Preliminaries 360
 - The macro language 360
 - The collection information database 369
- 7.3 Responding to user requests 372
 - Performing a search 375
 - Retrieving a document 376
 - Browsing a hierarchical classifier 377
 - Generating the home page 378
 - Using the protocol 378
 - Actions 384
- 7.4 Operational aspects 385
 - Configuring the receptionist 386
 - Configuring the site 391
- 7.5 Notes and sources 392

8. Interoperability: Standards and protocols 393

- 8.1 More markup 395
 - Names 395

Links	397
Types	402
8.2 Resource description	408
Collection-level metadata	410
8.3 Document exchange	413
Open eBook	414
8.4 Query languages	419
Common command language	419
XML Query	422
8.5 Protocols	426
Z39.50	427
Supporting the Z39.50 protocol	429
The Open Archives Initiative	430
Supporting the OAI protocol	433
8.6 Research protocols	434
Dienst	435
Simple digital library interoperability protocol	436
Translating between protocols	437
Discussion	438
8.7 Notes and sources	440
9. Visions: Future, past, and present	443
9.1 Libraries of the future	445
Today's visions	445
Tomorrow's visions	448
Working inside the digital library	451
9.2 Preserving the past	454
The problem of preservation	455
A tale of preservation in the digital era	456
The digital dark ages	457
Preservation strategies	459
9.3 Generalized documents: A challenge for the present	462
Digital libraries of music	462
Other media	466
Generalized documents in Greenstone	469
Digital libraries for oral cultures	471
9.4 Notes and sources	474

Appendix: Installing and operating Greenstone	477
Glossary	481
References	489
Index	499
About the authors	517

Figures

- Figure 1.1 Kataaui's information and communication center. 2
- Figure 1.2 The Zia Pueblo village. 3
- Figure 1.3 The New York Public Library. 6
- Figure 1.4 Rubbing from a stele in Xi'an. 9
- Figure 1.5 A page of the original Trinity College Library catalog. 13
- Figure 1.6 The Bibliothèque Nationale de France. 15
- Figure 1.7 Artist's conception of the Memex, Bush's automated library. 16
- Figure 1.8 Part of a page from the *Book of Kells*. 18
- Figure 1.9 Pages from a palm-leaf manuscript in Thanjavur, India. 19
- Figure 1.10 Māori toki or ceremonial adze, emblem of the Greenstone project.
25
- Figure 2.1 Scanning and optical character recognition. 59
- Figure 2.2 (a) Document image containing different types of data;
(b) the document image segmented into different regions. 64
- Figure 2.3 (a) Double-page spread of a Māori newspaper; (b) enlarged
version; (c) OCR text. 71
- Figure 3.1 Finding a quotation in *Alice's Adventures in Wonderland*. 78
- Figure 3.2 Different-looking digital libraries: (a) Kids' Digital Library (b)
School Journal Digital Library. 80
- Figure 3.3 Village-Level Brickmaking: (a) the book; (b) the chapter on
Moulding; (c, d) some of the pages. 82
- Figure 3.4 *Alice's Adventures in Wonderland*. 84

- Figure 3.5 A story from the School Journal collection: (a) “Never Shout at a Draft Horse!”; (b) with search term highlighted (mock-up). 86
- Figure 3.6 A historic Māori newspaper: (a) page image; (b) extracted text. 88
- Figure 3.7 Listening to a tape from the Oral History collection. 90
- Figure 3.8 Finding *Auld Lang Syne* in a digital music library. 92
- Figure 3.9 Foreign-language collections: (a) French (b) Portuguese interface to an English collection. 94
- Figure 3.10 Documents from two Chinese collections: (a) rubbings of Tang poetry; (b) classic literature. 95
- Figure 3.11 An Arabic collection: (a) a document; (b) searching. 96
- Figure 3.12 Bibliography display. 97
- Figure 3.13 Metadata examples: (a) bibliography record retrieved from the Library of Congress; (b) description of a BBC television program. 98
- Figure 3.14 Searching for a quotation: (a) query page; (b) query response. 100
- Figure 3.15 Choosing search preferences. 104
- Figure 3.16 Large-query search interface. 109
- Figure 3.17 Query with history. 110
- Figure 3.18 Form search: (a) simple; (b) advanced. 111
- Figure 3.19 Browsing an alphabetical list of titles: (a) plain list; (b) with A–Z tags. 113
- Figure 3.20 Browsing a list of titles in Chinese: (a) stroke-based browsing; (b) Pinyin browsing. 115
- Figure 3.21 Browsing by date. 117
- Figure 3.22 Browsing a classification hierarchy: (a) the beginning; (b) expanding *Sustainable development*; (c) expanding *Organizations, institutions*. 118
- Figure 3.23 (a) Browsing for information about locusts; (b) expanding on desert locust; (c) document about desert locusts. 120
- Figure 3.24 (a) Browsing for information on *poisson*; (b) INFOPECHE Web page. 122
- Figure 3.25 Browsing interfaces based on key phrases: (a) hierarchical browser; (b) document explorer. 123
- Figure 3.26 Browsing based on information mined from the document collection: (a) acronyms; (b) language identification. 125
- Figure 4.1 Unicode excerpt: Basic Latin and Latin-1 Supplement (U+0000–U+00FF). 142
- Figure 4.2 Unicode excerpts: (a) Latin Extended A (U+0100–U+017F); (b) Cyrillic (U+0400–U+045F). 143

- Figure 4.3 Encoding *Welcome* in (a) Unicode; (b) UTF-32, UTF-16, and UTF-8. 147
- Figure 4.4 Examples of characters in Indic scripts. 150
- Figure 4.5 Devanagari script: (a) ISCII; (b) Unicode (U+0900-U+0970); (c) code table for the Surekh font. 152
- Figure 4.6 Page produced by a digital library in Devanagari script. 155
- Figure 4.7 Entries for the word search in a biblical concordance. 158
- Figure 4.8 Alternative interpretations of two Chinese sentences: (a) ambiguity caused by phrasing; (b) ambiguity caused by word boundaries. 161
- Figure 4.9 (a) Result of executing a PostScript program; (b) the PostScript program; (c) Encapsulated PostScript version; (d) PDF version; (e) network of objects in the PDF version; (f) RTF specification of the same document. 167–169
- Figure 4.10 A PostScript document and the text extracted from it. 174
- Figure 4.11 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. 175
- Figure 4.12 Reading a bookmark-enabled PDF document with Acrobat. 182
- Figure 4.13 Structure of an RTF file. 188
- Figure 4.14 (a) LaTeX source document; (b) printed result. 192
- Figure 4.15 Encoding and decoding processes in baseline JPEG. 199
- Figure 4.16 Transform-coded images reconstructed from a few coefficients. 200
- Figure 4.17 Zigzag encoding sequence. 201
- Figure 4.18 Images reconstructed from different numbers of bits: (a) 0.1 bit/pixel; (b) 0.2 bit/pixel; (c) 1.0 bit/pixel. 202
- Figure 4.19 Progressive versus raster transmission. USC-IPI image database. 204
- Figure 4.20 8×8 tiled template used to generate a PNG interlaced file. 205
- Figure 4.21 (a) Frame sequence for MPEG; (b) reordering for sequential transmission. 211
- Figure 5.1 (a) Sample HTML code involving graphics, text, and some special symbols; (b) snapshot rendered by a Web browser. 226–227
- Figure 5.2 The relationship between XML, SGML, and HTML. 230
- Figure 5.3 Sample XML document. 233
- Figure 5.4 Sample DTD using a parameterized entity. 235
- Figure 5.5 Sample XML document, viewed in a Web browser. 237

- Figure 5.6 (a) Basic CSS style sheet for the United Nations Agencies example;
(b) viewing the result in an XML-enabled Web browser. 239
- Figure 5.7 (a) CSS style sheet illustrating tables and lists; (b) viewing the
result in an XML-enabled Web browser. 241
- Figure 5.8 (a) CSS style sheet illustrating context-sensitive formatting;
(b) viewing the result in an XML-enabled Web browser. 243
- Figure 5.9 Using CSS to specify different formatting styles for different
media. 245
- Figure 5.10 XSL style sheet for the basic United Nations Agencies
example. 247
- Figure 5.11 XSL style sheet illustrating tables and lists. 249–250
- Figure 5.12 XSL style sheet illustrating context-sensitive formatting. 251
- Figure 5.13 XSL style sheet that sorts UN agencies alphabetically. 253
- Figure 5.14 Bibliography item in BibTeX format. 259
- Figure 5.15 Bibliography item in Refer format. 260
- Figure 6.1 Sign at a Tasmanian blowhole. 284
- Figure 6.2 Using the Demo collection. 289
- Figure 6.3 Using the Collector to build a new collection. 295–296
- Figure 6.4 Collection configuration file created by *mkcol.pl*. 306
- Figure 6.5 Collection icon. 307
- Figure 6.6 About page for the *dlpeople* collection. 309
- Figure 6.7 Structure of the Greenstone home directory. 311
- Figure 6.8 Steps in the import process. 315
- Figure 6.9 Steps in the build process. 317
- Figure 6.10 Greenstone Archive Format: (a) Document Type Definition
(DTD); (b) example document. 321
- Figure 6.11 Plug-in inheritance hierarchy. 333
- Figure 6.12 XML format: (a) Document Type Definition (DTD); (b) example
metadata file. 334
- Figure 6.13 Classifiers: (a) *AZList*; (b) *List*; (c) *DateList*; (d) *Hierarchy*;
(e) collection-specific. 337
- Figure 6.14 Part of the file *sub.txt*. 341
- Figure 6.15 Excerpt from the Demo collection's *collect.cfg*. 345
- Figure 6.16 The effect of format statements on (a) the document itself;
(b) the search results. 347
- Figure 6.17 Starting to build a collection. 350
- Figure 6.18 Mirroring a site. 351
- Figure 6.19 Adding new metadata. 352
- Figure 7.1 Overview of a general Greenstone system. 356
- Figure 7.2 Greenstone system using the null protocol. 358
- Figure 7.3 Graphical query interface to a Greenstone collection. 359

- Figure 7.4 (a) About This Collection page; (b) part of the macro file that generates it. 362
- Figure 7.5 Illustration of macro precedence. 366
- Figure 7.6 Greenstone home page. 367
- Figure 7.7 Personalizing the home page: (a) new version; (b) *yourhome.dm* file used to create it. 368
- Figure 7.8 GDBM database for the Gutenberg collection (excerpt). 370
- Figure 7.9 *The Golf Course Mystery*. 371
- Figure 7.10 Browsing titles in the Gutenberg collection. 372
- Figure 7.11 Greenstone runtime system. 373
- Figure 7.12 Searching the Gutenberg collection for *Darcy*. 375
- Figure 7.13 Using the protocol to perform a search. 380
- Figure 7.14 Kids' Digital Library. 381
- Figure 7.15 Implementing the Kids' Digital Library using the protocol. 382
- Figure 7.16 A bibliographic search tool. 383
- Figure 7.17 Entry in the usage log. 389
- Figure 8.1 Adding an XLink to the UN example. 398
- Figure 8.2 Adding extended XLinks to the UN example. 400
- Figure 8.3 Directed graph for the XLink of Figure 8.2. 401
- Figure 8.4 XML Schema for the UN Agency example. 404
- Figure 8.5 XML Schema that demonstrates data typing. 406
- Figure 8.6 Modeling this book graphically using RDF. 408
- Figure 8.7 XML serialization of the example RDF model. 409
- Figure 8.8 RSLP description of the Morrison collection of Chinese books. 412
- Figure 8.9 Reading an eBook of Shakespeare's *Macbeth*. 413
- Figure 8.10 Sample Open eBook package. 416
- Figure 8.11 Inside an Open eBook. 418
- Figure 8.12 Using the Common Command Language. 421
- Figure 8.13 Various FIND commands. 422
- Figure 8.14 XML library of publications: (a) main XML file (*library.xml*); (b) supporting file (*bottle_creek.xml*). 424
- Figure 8.15 XQuery commands. 425
- Figure 8.16 XQuery commands that demonstrate element construction. 426
- Figure 8.17 Interface to the Library of Congress using Z39.50. 430
- Figure 8.18 OAI GetRecord request and XML response. 432
- Figure 8.19 Using the Dienst protocol. 435
- Figure 8.20 Using SDLIP to obtain property information. 437
- Figure 8.21 Mapping SDLIP calls to the Greenstone protocol. 438
- Figure 8.22 Using the SDLIP-to-Greenstone translator. 439
- Figure 9.1 New York Public Library reading room. 446

- Figure 9.2 Digital library in the British National Library. 447
- Figure 9.3 A peek inside the digital library at the Kataayi cooperative in Uganda. 447
- Figure 9.4 Xandar's digital library. 448
- Figure 9.5 Carpenter's workshop. 450
- Figure 9.6 Reading a document in a digital library. 452
- Figure 9.7 Focusing on part of the document and finding pertinent literature. 453
- Figure 9.8 Focusing on part of the document's subject matter. 454
- Figure 9.9 Medieval literature in the library at Wolfenbüttel. 455
- Figure 9.10 Combined music and text search. 464
- Figure 9.11 Application of an optical music recognition system. 465
- Figure 9.12 Home page of the Humanity Development Library. 466
- Figure 9.13 Modeling a book as a physical object. 469
- Figure 9.14 First aid in pictures: how to splint a broken arm. 472
- Figure A.1 The different options for Windows and Unix versions of Greenstone. 478

Tables

Table 2.1	Spelling variants of the name Muammar Qaddafi.	51
Table 2.2	Title pages of different editions of Hamlet.	52
Table 2.3	Library of Congress Subject Heading entries.	54
Table 2.4	An assortment of devices and their resolutions.	60
Table 4.1	The ASCII character set.	135–136
Table 4.2	Unicode Part 1: The basic multilingual plane.	139–141
Table 4.3	Encoding the Unicode character set as UTF-8.	149
Table 4.4	Segmenting words in English text.	163
Table 4.5	Graphical components in PostScript.	165
Table 4.6	International television formats and their relationship with CCIR 601.	209
Table 4.7	Upper limits for MPEG-1's constrained parameter bitstream.	213
Table 5.1	Library catalog record.	254
Table 5.2	MARC fields in the record of Table 5.1.	255
Table 5.3	Meaning of some MARC fields.	256
Table 5.4	Dublin Core metadata standard.	257
Table 5.5	The basic keywords used by the Refer bibliographic format.	261
Table 5.6	TIFF tags.	264
Table 5.7	Titles and key phrases—author- and machine-assigned—for three papers.	275
Table 6.1	What the icons at the top of each page mean.	289
Table 6.2	What the icons on the search/browse bar mean.	289
Table 6.3	Icons that you will encounter when browsing.	290

Table 6.4	The collection-building process.	303
Table 6.5	Options for the import and build processes.	310
Table 6.6	Additional options for the import process.	316
Table 6.7	Additional options for the build process.	318
Table 6.8	Items in the collection configuration file.	323
Table 6.9	Options applicable to all plug-ins.	328
Table 6.10	Standard plug-ins.	330
Table 6.11	Plug-in-specific options for HTMLPlug.	331
Table 6.12	(a) Greenstone classifiers; (b) their options.	339
Table 6.13	The format options.	343
Table 6.14	Items appearing in format strings.	345
Table 7.1	List of protocol calls.	379
Table 7.2	Action.	385
Table 7.3	Configuration options for site maintenance and logging.	387
Table 7.4	Lines in <i>gsdlsite.cfg</i> .	391
Table 8.1	XLink attributes.	399
Table 8.2	Common Command Language keywords, with abbreviations.	420
Table 8.3	Facilities provided by Z39.50.	428
Table 8.4	Open Archive Initiative protocol requests.	433

Foreword

by Edward A. Fox

Computer science addresses important questions, offering relevant solutions. Some of these are recursive or self-referential. Accordingly, I am pleased to testify that a suitable answer to the question carried in this book's title is the book itself! Witten and Bainbridge have indeed provided a roadmap for those eager to build digital libraries.

Late in 2001, with a draft version of this book in hand, I planned the introductory unit for my spring class Multimedia, Hypertext, and Information Access (CS4624), an elective computer science course for seniors. Departmental personnel installed the Greenstone software on the 30 machines in our Windows lab. Students in both sections of this class had an early glimpse of course themes as they explored local and remote versions of Greenstone, applied to a variety of collections. They also built their own small digital libraries—all within the first few weeks of the course.

When the CS4624 students selected term projects, one team of three asked if they could work with Roger Ehrich, another computer science professor, to build a digital library: the Germans from Russia Heritage Society (GRHS) Image Library. After exploring alternatives, they settled on Greenstone. I gave them my draft copy of this book and encouraged them throughout the spring of 2002 as they worked with the software and with the two GRHS content collections: photographs and document images. They learned about documents and metadata, about macros and images, about installation and setting up servers, about user accounts and administration, about prototyping and documentation. They learned how to tailor the interface, to load and index the collection,

and to satisfy the requirements of their client. Greenstone was found useful for yet another community!

Ian Witten has given numerous tutorials and presentations about digital libraries, helping thousands understand key concepts, as well as how the Greenstone software can be of use. Talking with many of those attending these sessions, I have found his impact to be positive and beneficial. This book should extend the reach of his in-person contact to a wider audience, helping fill the widely felt need to understand “digital libraries” and to be able to deploy a “digital library in a box.” Together with David Bainbridge, Witten has prepared this book, greatly extending his tutorial overviews and drawing upon a long series of articles from the New Zealand Digital Library Project—some of the very best papers in the digital library field.

This book builds upon the authors’ prior work in a broad range of related areas. It expands upon R&D activities in the compression, information retrieval, and multimedia fields, some connected with the MG system (and the popular book *Managing Gigabytes*, also in this book series). It brings in a human touch, explaining how digital libraries have aided diverse communities, from Uganda to New Zealand, from New Mexico to New York, from those working in physics to those enjoying popular music. Indeed, this work satisfies the “5S” checklist that I often use to highlight the key aspects of digital libraries, involving societies, scenarios, spaces, structures, and streams.

Working with UNESCO and through the open source community, the New Zealand team has turned Greenstone into a tool that has been widely deployed by *Societies* around the globe, as explained at both the beginning and end of the book. Greenstone’s power and flexibility have allowed it to serve a variety of needs and support a range of user tasks, according to diverse *Scenarios*. Searching and browsing, involving both phrases and metadata and through both user requests and varied protocols, can support both scholars and those focused on oral cultures.

With regard to *Spaces*, Greenstone supports both peoples and resources scattered around the globe, with content originating across broad ranges of time. Supporting virtual libraries and distributed applications, digital libraries can be based in varied locations. Spaces also are covered through the 2D user interfaces involved in presentation, as well as internal representations of content representation and organization.

Structures are highlighted in the chapters on documents as well as markup and metadata. Rarely can one find a clear explanation of character encoding schemes such as Unicode, or page description languages such as PostScript and PDF, in addition to old standbys such as Word and LaTeX, and multimedia schemes like GIF, PNG, JPEG, TIFF, and MPEG. Seldom can one find a clearer discussion of XML, CSS, and XSL, in addition to MARC and Dublin Core. From key elements (acronyms, phrases, generic entities, and references) to collections,

from lists to classification structure, from metadata to catalogs, the organizational aspects of digital libraries are clearly explicated.

Digital libraries build upon underlying *Streams* of content: from characters to words to texts, from pixels to images, and from tiny fragments to long audio and video streams. This book covers how to handle all of these, through flexible plugins and classifiers, using macros and databases, and through processes and protocols. Currently popular approaches are discussed, including the Open Archives Initiative, as well as important themes like digital preservation.

Yes, this book satisfies the “5S” checklist. Yes, this book can be used in courses at both undergraduate and graduate levels. Yes, this book can support practical projects and important applications. Yes, this book is a valuable reference, drawing upon years of research and practice. I hope, like me, you will read this book many times, enjoying its engaging style, learning both principles and concepts, and seeing how digital libraries can help you in your present and future endeavors.

This Page Intentionally Left Blank

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 Shakespeare's Globe Theatre reconstructed 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*. On sitting 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 document management system that stores 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 a dozen 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 this new century. The information revolution not only supplies the technological horsepower that drives digital libraries, but fuels an unprecedented demand for storing, organizing, and accessing information. If information is the currency of the knowledge economy, digital libraries will be 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, and humanitarian applications, as well as technical ones. These collections are formed from different kinds of material, organized in different ways, presented in 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

A comprehensive software resource has been created to illustrate the ideas in the book and form a possible basis for your own digital library. Called the Green-

stone Digital Library Software, it is freely available as source code on the World Wide Web (at www.greenstone.org) and comes precompiled for many popular platforms. It is a complete industrial-strength implementation of essentially all the techniques covered in this book. A fully operational, flexible, extensible system for constructing easy-to-use digital libraries, Greenstone is already 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 being promoted in a joint project in which UNESCO is supporting and distributing the Greenstone digital library software.

Although some parts of the book are tightly integrated with the Greenstone software—for it is hard to talk specifically and meaningfully about practical topics of building digital libraries without reference to a particular implementation—we have worked to minimize this dependence and make the book of interest to people using other software infrastructure for their digital collections. Most of what we say has broad application and is not tied to any particular implementation. The parts that are specific to Greenstone are confined to two chapters (Chapters 6 and 7), with a brief introduction in Chapter 1 (Section 1.4), and the Appendix. Even these parts are generally useful, for those not planning to build upon Greenstone will be able to use this material as a baseline, or make use of Greenstone's capabilities as a yardstick to help evaluate other designs.

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 above-mentioned conference attendee 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 simple 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 rather 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 can support 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 standards used to represent documents digitally, skim Chapter 1; read Chapters 4, 5, and 8 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 converting documents to digital form, read Section 2.4 as well.

To learn how to build a digital library as quickly as possible, skim Chapter 1 (but check Section 1.4) and then turn straight to Chapter 6. You will need to consult the Appendix when installing the Greenstone software. 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 all of the book in sequence but omit Chapters 6 and 7. For a text with a strong practical component, read all chapters in order—and then turn your students loose on the software!

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.

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. One thread that runs through the book is internationalization and the role of digital libraries in developing countries—for we believe that here digital libraries represent a “killer app” for computer technology. After summarizing the principal features of the Greenstone software, the first chapter closes with a discussion of issues involved in copyright and “harvesting” material from the Web.

Recognizing that many readers are itching to get on with actually *building* their digital library, Chapter 2 opens with an invitation to skip ahead to the start of Chapter 6 for an account of how to use the Greenstone software to create a plain but utilitarian collection that contains material of your own choice. This is

very easy to do and should only take half an hour if you restrict yourself to a demonstration prototype with a small volume of material. (You will have to spend a few minutes downloading and installing the software first; turn to the Appendix to get started.) We want you to slake your natural curiosity about what is involved in building digital collections, so that you can comfortably focus on learning more about the foundations. We then proceed to discuss where the material in your library might come from (including the process of optical character recognition or OCR) and describe traditional methods of library organization.

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, material ranging from text to music, and languages ranging from Māori to Chinese. We show many examples of browsing structures, from simple lists to hierarchies, date displays, and dynamically generated phrase hierarchies.

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. 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 also digress to introduce the question of segmenting words in languages like Chinese. We then describe popular formats for document representation: PostScript; PDF (Portable Document Format); RTF (Rich Text Format); the native format used by Microsoft Word, a popular word processor; and LaTeX, commonly used for mathematical and scientific documents. We also introduce the principal international standards used for representing images, audio, and video.

Besides 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. The related term *markup*, which in today's consumer society we usually associate with price increases, has another meaning: it refers to the process of annotating documents with typesetting information. In recent times this has been extended to annotating documents with structural information—including metadata—rather than (or as well as) formatting commands. Chapter 5 covers

markup and metadata and also explains how metadata is expressed in traditional library catalogs. We introduce the idea of extracting metadata from the raw text of the documents themselves and give examples of what can be extracted.

Up to this point the book has been quite general and applies to any digital library. Chapters 6 and 7 are specific to the Greenstone software. There are two parts to a digital library system: the offline part, preparing a document collection for presentation, and the online part, presenting the collection to the user through an appropriate interface. Chapter 6 describes the first part: how to build Greenstone collections. This involves configuring the digital library and creating the full-text indexes and metadata databases that are needed to make it work. Given the desired style of presentation and the input that is available, you come up with a formal description of the facilities that are required and let the software do the rest.

To make the digital library as flexible and tailorable as possible, Greenstone uses an object-oriented software architecture. It defines general methods for presentation and display that can be subclassed and adapted to particular collections. To retain full flexibility (e.g., for translating the interface into different languages) a macro language is used to generate the Web pages. A communications protocol is also used so that novel user interface modules can interact with the digital library engine underneath to implement radically different presentation styles. These are described in Chapter 7.

In Chapter 8 we reach out and look at other standards and protocols, which are necessary to allow digital libraries to interoperate with one another and with related technologies. For example, electronic books—e-books—are becoming popular, or at least widely promoted, and digital libraries may need to be able to export material in such forms.

Finally we close with visions of the future of digital libraries and mention 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. The rest is up to you. Our aim will have been achieved if you actually *build a digital library*!

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 by making the major design decisions that underlie Greenstone. Rodger left our group some time ago, but the influence of his foresight remains—a legacy that this book exploits. Next comes Stefan Boddie, the man who has kept Greenstone going over the years, who steers the ship and navigates 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.

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 in Chapter 3. Through Te Taka's efforts we receive inspiration every day from the magnificent Māori *toki* that resides in our laboratory and can be seen in Figure 1.10, 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) created the music collections that are illustrated here. Steve Jones builds many novel user interfaces, especially ones involving phrase browsing, and some of our key examples are his. Sally Jo Cunningham is the resident expert on library organization and related matters. Stuart Yeates designed and built the acronym extraction module and helped in countless other ways, while Dana McKay worked on such things as extracting date metadata, as well as drafting the Greenstone manuals that eventually turned into Chapters 6 and 7. 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.

Many others in the digital library lab at Waikato have made substantial—nay, heroic—technical contributions to Greenstone. Gordon Paynter, researcher and senior software architect, built the phrase browsing interface, helped design the Greenstone communication protocol, and improved many aspects of metadata handling. Hong Chen, Kathy McGowan, John McPherson, Trent Mankelow, and Todd Reed have all worked to improve the software. Geoff Holmes and Bill Rogers helped us over some very nasty low-level Windows problems. Eibe Frank worked on key-phrase extraction, while Bernhard Pfahringer helped us conceptualize the Collector interface. Annette Falconer worked on a Women's History collection that opened up new avenues of research. 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 came from London for two long and productive spells. He helped develop the communications protocol and built the CD-ROM

writing module, and continues to work with our team. Elke Duncker, also from London, advised us on cultural and ethical issues. Dave Nichols from Lancaster worked on the Java side of Greenstone and, with Kirsten Thomson, helped evaluate the Collector interface. The influence of Carl Gutwin from Saskatoon is particularly visible in the phrase browsing and key-phrase extraction areas. 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, who has encouraged, cajoled, and occasionally bullied us into making our software available in a form designed to be most useful to people in developing countries, based on his great wealth of experience. We are particularly grateful to him 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. We acknowledge the support of John Rose of UNESCO in Paris, Maria Trujillo of Colombia, and Chico Fernandez-Perez of the FAO in Rome. Rob Akscyn in Pittsburgh has been a continual source of inspiration, and his wonderful metaphors occasionally enliven this book. Until he was so sadly and unexpectedly snatched away from us, we derived great benefit from the boundless enthusiasm of Ferrers Clark at CISTI, 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 London has been a constant source of material help and moral support.

We would like to acknowledge all who have translated the Greenstone interface into different languages—at the time of writing we have interfaces in Arabic, Chinese, Dutch, French, German, Hebrew, Indonesian, Italian, Māori, Portuguese, Russian, and Spanish. We are very 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 GNU-licensed packages included in the Greenstone distribution.

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 Technol-

ogy, 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 supports 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.

Diane Cerra and Marilyn Alan of Morgan Kaufmann have worked hard to shape this book, and Yonie Overton, our project manager, has made the process go very smoothly for us. Angela Powers has provided excellent support at the Waikato end. Ed Fox, the series editor, contributed enthusiasm, ideas, and a very careful reading of the manuscript. We gratefully acknowledge the efforts of the anonymous reviewers, one of whom in particular made a great number of 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, including 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, Paris, Calgary, New Orleans, and San Francisco. You all know who you are—thanks! Numerous institutions helped with facilities, including Middlesex University in London, Braunschweig Technical University in Germany, the University of Calgary in Canada, and the Payson Center for International Development and Technology Transfer in New Orleans. 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 and complete the book.

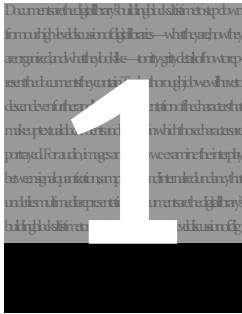
All our traveling has helped spin the threads of internationalization and human development that are woven into the pages that follow. Our families—Annette, Pam, Anna, and Nikki—have supported us in countless ways, sometimes journeying with us, sometimes keeping the fire burning at home in New Zealand.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

About the Web site



You can view the book's full color figures at Morgan Kaufmann's *How to Build a Digital Library* Web site at www.mkp.com/DL. There you will also find two online appendices: a greatly expanded version of the printed appendix, *Installing and Operating Greenstone*, and another appendix entitled *Greenstone Source Code* for those who want to delve more deeply into the system. There is also a novel full-text index to the book that allows you to locate the pages in which words and word combinations appear.

This Page Intentionally Left Blank



Orientation

The world of digital libraries

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 ferrocement 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 are looking for books covering topics such as practical technology,



Figure 1.1 Kataayi's information and communication center.

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 such as 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 miniscule fraction of the cost of paper books.

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 the last 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 had grown to 150,000 papers and processed 150,000 requests per day.

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 36 miles 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,



Figure 1.2 The Zia Pueblo village.

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 a major problem is facing the Zia Pueblo today: the loss of its language and traditional culture. Young people are not learning the Zia Pueblo traditions, nor 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.

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 these tunes 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 rock 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) will become a mass-market consumer product, 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, we will see in Section 1.3 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 of literature, their collection, organization, and access. And not just for books: there are



Figure 1.3 The New York Public Library.

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—or rather, they misguidedly regard the Web as the ultimate library.

But a digital library is not really a “digitized library.” We hope that you, dear reader, are reading *How to Build a Digital Library* because you are thinking of building a digital library. But we do not imagine that you are the director of the New York Public Library, contemplating replacing that magnificent edifice by 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 scenarios above—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 conceived as an organized collection of information,

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 scenarios above. Beyond audio and video we also want to include such things as 3D objects, simulations, dynamic visualizations, and virtual-reality worlds. The second and third parts of the definition deliberately accord equal weight to user (access and retrieval) and librarian (selection, organization, and maintenance). The librarian 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 data. You could go on to define *knowledge* as the accumulation of your set of expectations, and *wisdom* as the value attached to knowledge. All information is not created equal, and it is wisdom that librarians put into the library by making decisions about what to include in a collection—difficult decisions!—and following up with appropriate ways of organizing and maintaining the information. Indeed it is exactly these features that will 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 scenarios 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 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 ones; we return to them in Section 2.1.

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. Most 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, so 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 connects a new acquisition into the 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. We call this information *meta-data*—data about data—and it will figure 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 where 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 about 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!

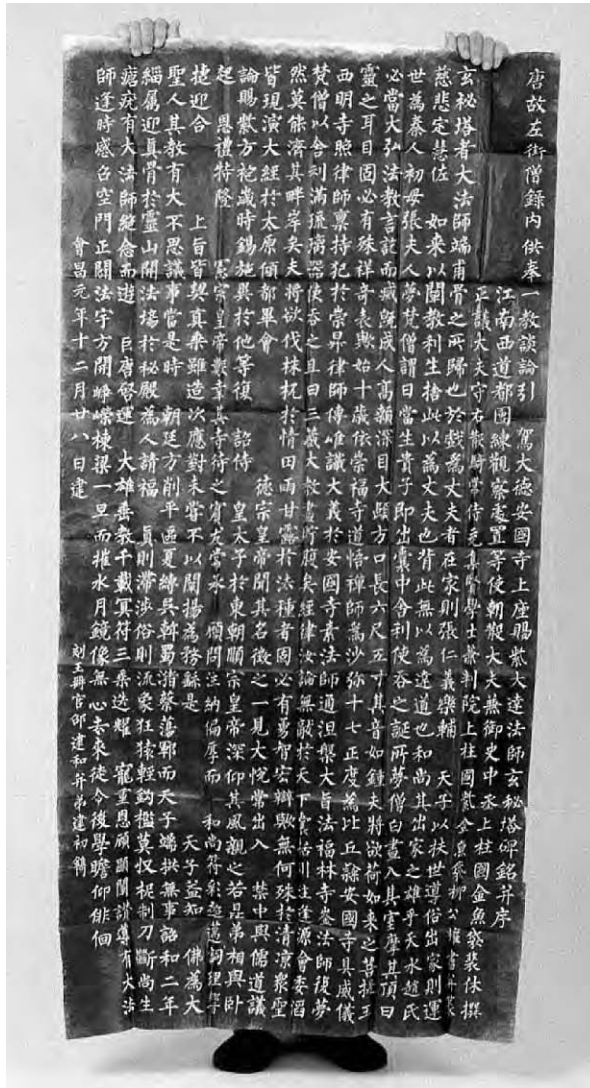


Figure 1.4 Rubbing from a stele in Xi'an.

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 evolved beyond recognition. Originally intended for storage and preservation, libraries have refocused to place users at the center, with increased emphasis on information exchange.

Ancient libraries were only useful to the small minority of people who could read and 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 bookstacks 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 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 which is, for better or worse, 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 which 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 safekeeping.

By the time Julius Caesar set fire to the ships in the harbor of Alexandria in 47 B.C. and the fire spread to the shore, the library had grown 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. Much remained after Caesar's fire, but this 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 1450.¹ 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 of the 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 Duke August's 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, who 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 National Library, the University Libraries of Oxford and Cambridge, and the National Libraries of Scotland and Wales. This "legal

1. The printing press was invented in China much earlier, around five centuries before Gutenberg.

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). Likewise the Library of Congress receives copies of all books published in the U.S. But we digress.

There were no journals in 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, hydralike, into hundreds of thousands of scientific journals today—although, as we have seen, some are being threatened with replacement by electronic archives.

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

In the U.S., President John Adams created a reference library for Congress when the seat of government was moved to the new capital city of Washington in 1800. He began by providing \$5,000 “for the purchase 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. The small congressional library of some 3,000 volumes was lost in the fire. Another fire destroyed two-thirds of the collection in 1851. Unlike Alexandria, however, the Library of Congress has regrown—indeed its rotunda is a copy of the one in Wolfenbüttel two centuries earlier. In fact today it contains approximately 22 million volumes.

The information explosion began to hit home in Ireland in the middle of the 19th century. Work started in 1835 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 National Library), which at the time had nearly 2 million books, and the French National Library in Paris with over 2.5 million.

- Resolutie van de staten generael der Vereenighde Nederlanden, dienende tot antwoord op de memorie by de ambassadeurs van sijne majesteit van Vrankrijck.
'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.
- Nederlandsche absolutie op de Fransche belydenis.
Amsterdam, 1684. 4°. Fag. H. 2. 50. N°. 22.
- Redenen dienende om aan te wijsen dat haar ho. mog. [niet] kunnen verhindert werden een vredige afkomst te maken op de conditien van 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 wijsen 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.

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 past decade the national libraries of the U.K., France, Germany, and Denmark all opened new buildings. The ones in London and Paris are monumental in scale: their country’s largest public buildings of the century. Standing on the bank of the Seine River, the Bibliothèque Nationale de France consists of four huge towers that appear 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, into landfills, because its new building, though lavishly praised by architecture critics, was too small for all the books.

The notion of focused collections is replacing the Alexandrian model that the ideal library 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.



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

Early technodreams

Let us, for a moment at least, give an ear to the technophiles. Over 60 years ago, science fiction writer H. G. Wells was promoting the concept of a “world brain” based on a permanent world encyclopedia which “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.”

Eight years later, 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

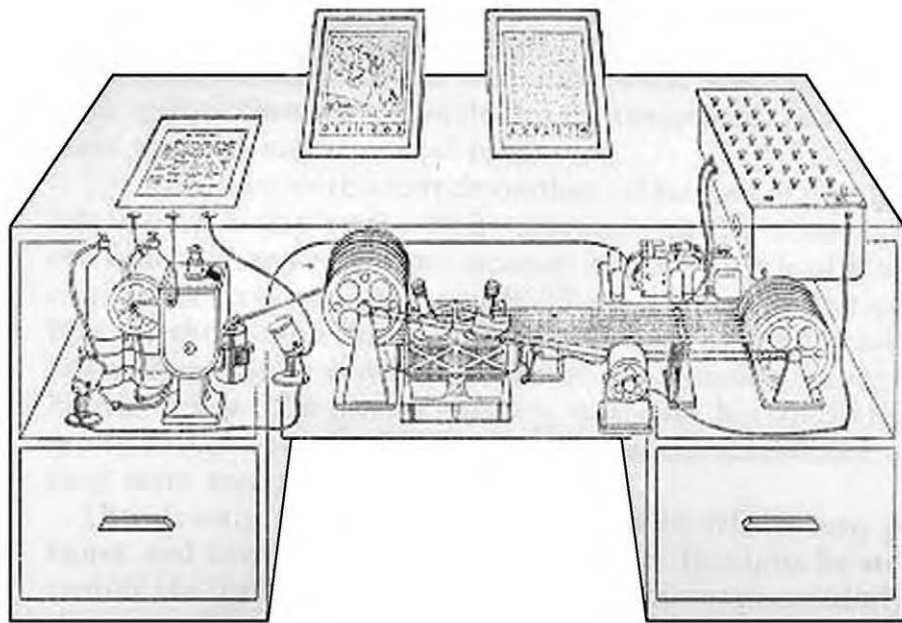


Figure 1.7 Artist's conception of the Memex, Bush's automated library. Courtesy of Mary and Alfred Crimi Estate.

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).

Fifteen years later, 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 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, if not a surrogate for, 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”—which 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. Computer-searchable catalogs are standard in libraries today. And there are many other functions that 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 encountered in the opening scenarios.

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.”

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

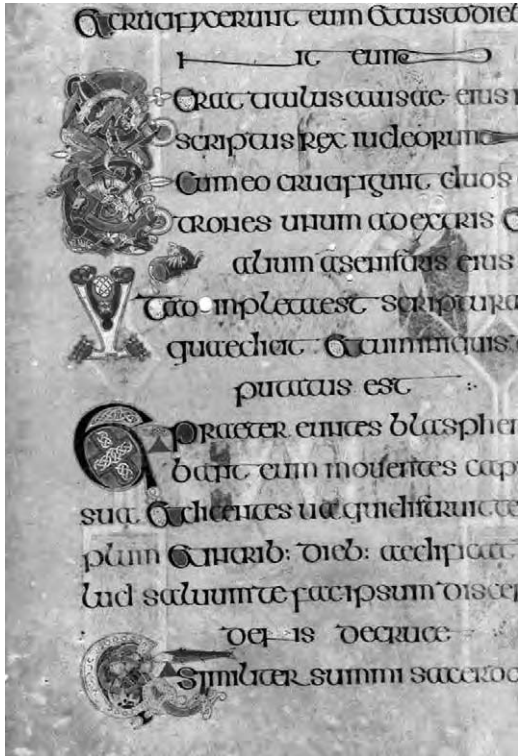


Figure 1.8 Part of a page from the *Book of Kells*.

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 1.8 shows part of a page and illustrates the extraordinary array of pictures, interlaced shapes, and ornamental details. Indeed 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 India one can see ancient books—some 2,000 years old—written on palm leaves, bound with string threaded through holes in the leaves. Figure 1.9 shows an example, which includes a picture of a deity (Sri Ranganatha) reclining with his consort on a serpent (Adishesha). In the castle library of Königsburg 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

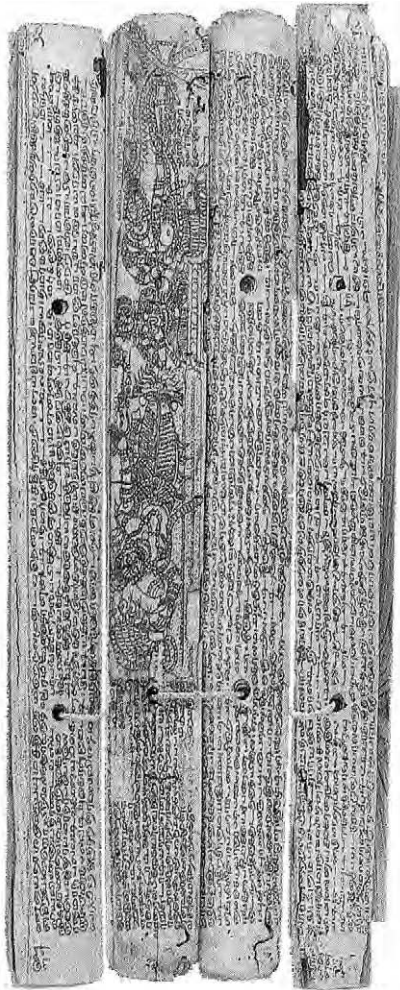


Figure 1.9 Pages from a palm-leaf manuscript in Thanjavur, India. Thanjavur Maharaja Serfoji's Sarasvat Mahal Library, Thanjavur, Tamil Nadu (1995).

in fox skin. History even provides many instances of books bound in human skin. It is hard to resist just one macabre example: a book in the Boston Athenaeum collection.

James Allen, alias George Walton, was a burglar, bank robber, horse thief and highwayman when, in 1833, he attacked John Fenno Jr. on the Massachusetts Turnpike with intent to rob. Fenno resisted his attacker and was shot, but saved by a suspender buckle. Allen fled, was caught and sent to prison where he wrote a boastful autobiographical

account of his life of crime called *The Highwayman*. Admiring Fenno's bravery, he asked that Fenno be given a copy of his book bound in the author's skin.

On July 17, 1837 upon Allen's death, Massachusetts General Hospital "accepted his body for anatomical and pathological studies" and removed enough skin to provide the covering of his book. Bookbinder Peter Low treated the skin to look like gray deer-skin and edged it with gold tooling. It is embossed with the Latin inscription "Hic Liber Waltonis Cute Compactus Est" (This book by Walton is bound in his own skin).

Those who feel nauseous may find this the best argument of all for digital libraries!

Catalogs and bibliographies comprise 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 it is unlikely, perhaps inappropriate, that readers will love simulated books the way that bibliophiles love real ones, and eventually surrogates may become anachronistic and fade away. For what really matters in libraries is information, even knowledge. Ask Kataayi.

1.3 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. Alternative sources such as solar energy are widely used in place of traditional power generation and distribution, while many developing countries have experienced far higher levels of mobile phone growth than developed ones. Digital libraries provide another example, compensating for the failure of traditional distribution mechanisms to address local requirements and deliver information where 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 from before 1993, and the library in a large district hospital consists of a single bookshelf filled mostly with novels.

Digital libraries, by decoupling production and distribution costs from intellectual property charges, offer a desperately needed lifeline. A wealth of essential humanitarian material is produced by various international organizations, such as the United Nations, as well as national units such as 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 copyright on the articles is held 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 such as earthquakes or hurricanes, and man-made ones such as terrorist attacks or nuclear accidents, demand 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 organized collections of information, graced with comprehensive searching and browsing capabilities, to be created rapidly. 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 medical and sanitary information.

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. The scenario described above with the Zia Pueblo provides a good example. The principal participants here are by definition the indigenous people themselves: the technological world assumes the role of catalyst, midwife, and consumer; for 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; 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 there is such a wealth of content on the Internet 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 English or other major languages—but also the fact that there are many local community content issues that contribute toward effective information use.

Teachers prepare educational material that addresses specific community problems, and they adapt published material to employ local examples. Indigenous people have invaluable 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 of 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. Not only does it improve the learning experience, it 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 too.

Effective human development blossoms from empowerment rather than gifting. 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." Disseminating information originating in the developed world is a useful activity for developing countries, as Kataayi members will testify. But a more effective strategy for sustained long-term human development is to disseminate the capability of creating information collections, rather than the collections themselves. This will allow developing countries to participate actively in our information society, rather than observing it from outside. It will stimulate the creation of new industry. And it will help 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 (though 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. With growth predicted at 20 percent per year, we estimate that at the turn of the millennium there were fifty 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. Universities are better equipped, but even there up to 1,000 people can depend on just one terminal. 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. Their 650 Mb capacity 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 such as 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 disk, DVD, which can hold from 5 to 20 Gb of data. A year's supply of those 5,000 medical journals mentioned above could fit, fully indexed, on a single DVD. And save lives.

1.4 The Greenstone software

This is a practical book about *how to build* a digital library. While the concepts developed in the book are quite general, we keep it tightly focused and practically oriented by using a particular digital library system throughout the book: the Greenstone Digital Library software from the New Zealand Digital Library Project at the University of Waikato. Illustrations in the book are nearly all taken from collections built with this software, or with associated software tools: this serves to keep our feet firmly on the ground. For example, three of the four opening scenarios use Greenstone: the Humanity Development Library is built with it (along with many other humanitarian collections), the Zia Pueblo collec-



Figure 1.10 Māori *tōki* or ceremonial adze, emblem of the Greenstone project.

tion is a planned project that will use it (along with other heritage-preservation projects), and the popular music library is a research project that centers around it (along with many other music library activities). The fourth, the physics archives, could use Greenstone quite easily—as similar collections readily testify.

The Greenstone software provides a convenient way of organizing information and making it available over the Internet. A *collection* of information comprises several (typically several thousand, or several million) documents. A *document*, in turn, is any information-bearing message in electronically recorded form. Documents are the fundamental unit from which information collections are built, although they may have their own substructure and associated files. Documents generally comprise text, although they may be images, sound files, or video. A collection may contain many different types of documents. Each collection provides a uniform interface through which all documents in it can be accessed—although the way that documents are displayed will depend on their

medium and format. A *library* generally includes many different collections, each organized differently—although there is a strong family resemblance in how collections are presented.

Making information available using this system is far more than just “putting it on the Web.” The collection becomes maintainable, searchable, and browsable. Prior to presentation, each collection undergoes a building process that, once established, is completely automatic. This process creates all the structures that are used at runtime for accessing the collection. Searching is based on various indexes involving full text and metadata. Browsing is based on various metadata and on phrase structures, and other information, abstracted from the full text of the documents. Support structures for both are created during the building operation. 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 under the GNU General Public License, and, in the spirit of open-source software, users are invited to contribute modifications and enhancements. Only through an international cooperative effort will digital library software become sufficiently comprehensive to meet the world’s needs. Currently the Greenstone software has been used by several United Nations agencies, including the Food and Agriculture Organization in Rome, UNESCO in Paris, the United Nations University in Tokyo, and the Centre for Human Settlements (Habitat) in Nairobi. It is used at sites throughout the world, and interfaces and collections exist in languages ranging from Portuguese to Chinese, from Māori to Arabic. Collections range from newspaper articles to technical documents, from educational journals to oral history, from visual art to folk songs. Further details, and many examples, can be obtained over the Internet from www.nzdl.org.

Throughout the book we will learn about the needs of digital libraries and how the Greenstone software helps to meet them. To give an idea of the breadth of these requirements, here is a list of features of the software.

<i>Accessible via Web browser</i>	Collections are accessed through a standard Web browser (e.g., Netscape or Internet Explorer). The browser is used for both local and remote access.
<i>Runs on Windows and Unix</i>	Collections can be served on either Windows (3.1/3.11, 95/98/ME/, NT/2000, and XP) or Unix (for example, Linux or SunOS); any of these can operate as a Web server.
<i>Permits full-text and fielded search</i>	The user can search the full text of the documents or choose among indexes built from different parts of the documents. Some collections have an index of full documents, an index of sections, an index of titles, and an index of authors, each of which can be searched for particular words or phrases. Results can be ranked by relevance or sorted by a metadata element.

Offers flexible browsing facilities

The user can browse lists of authors, lists of titles, lists of dates, classification structures, and so on. Different collections offer different browsing opportunities, and even within a collection a broad variety of browsing facilities are available. Browsing and searching interfaces are constructed during the building process, according to collection configuration information.

Creates access structures automatically

All collections are easy to maintain. Searching and browsing structures are built directly from the documents themselves. No links are inserted by hand. This means that if new documents in the same format become available, they can be merged into the collection automatically. However, existing links in the original documents, leading both within and outside the collection, are preserved.

Makes use of available metadata

Metadata may be associated with each document, or with individual sections within documents, and forms the raw material for browsing indexes. It must be provided explicitly or derivable automatically from the source documents. Standard schemes for expressing metadata are used, with provision for extensions as necessary.

Capabilities can be extended by plug-ins

“Plug-ins” can be written to accommodate new document types. Plug-ins currently exist for plain text files, HTML documents, Microsoft Word, RTF, PDF, PostScript, Powerpoint, and Excel files, e-mail, some proprietary formats, and for generic tasks such as recursively traversing directory structures containing such documents. A collection may have source documents in different forms. To build browsing indexes from metadata, an analogous scheme of “classifiers” is used: classifiers create browsing indexes of various kinds, based on metadata.

Can handle documents in any language

Unicode is used throughout the software, allowing any language to be processed in a consistent manner. To date, collections have been built containing French, Spanish, Māori, Russian, Chinese, Arabic, and English. On-the-fly conversion is used to convert from Unicode to an alphabet supported by the user’s Web browser.

Can display user interface in multiple languages

The interface can be presented in multiple languages. Currently the system is available in Arabic, Chinese, Dutch, French, German, Hebrew, Indonesian, Italian, Māori, Portuguese, Russian, Spanish, and English. New languages can be added easily.

Can handle collections of text, pictures, audio, and video

Greenstone collections can contain text, pictures, audio, and even video clips. Most nontextual material is either linked to the textual documents or accompanied by textual descriptions (such as figure captions) to allow full-text searching and browsing. However, the architecture permits implementation of plug-ins and classifiers for nontextual data.

Allows hierarchical browsing

Hierarchical phrase and key-phrase indexes of text or any metadata can be created using standard classifiers.

Designed for multi-gigabyte collections

Collections can contain millions of documents, making the Greenstone system suitable for collections up to several gigabytes.

<i>Uses compression techniques</i>	Compression is used to reduce the size of the indexes and text. Small indexes have the added advantage of faster retrieval.
<i>Permits authentication of users</i>	A built-in access control mechanism allows collections, and even individual documents, to be restricted to authorized users using a password protection scheme.
<i>Offers user logging</i>	All queries made to every Greenstone collection can be recorded in user logs.
<i>Provides an administrative function</i>	An “administrative” function enables specified users to authorize other users to build collections, have access to protected collections, examine the composition of all collections, turn logging on and off, and so on.
<i>Updates and adds new collections dynamically</i>	Collections can be updated and new ones brought online at any time, without bringing the system down; the process responsible for the user interface will notice (through periodic polling) when new collections appear and will add them to the list presented to the user. End users can easily build new collections in the same style as existing ones from material on the Web or in local files—or both.
<i>Publishes collections on CD-ROM</i>	Greenstone collections can be published, in precisely the same form, on a self-installing CD-ROM. The interaction is identical to accessing the collection on the Web (Netscape is provided on each disk)—except that response times are faster and more predictable.
<i>Supports distributed collections</i>	A flexible process structure allows different collections to be served by different computers, yet be presented to the user in the same way, on the same Web page, as part of the same digital library. The Z39.50 protocol is fully supported, both for accessing external servers and for presenting Greenstone collections to external clients.
<i>Everything you see, you can get</i>	The Greenstone Digital Library is open-source software, available from the New Zealand Digital Library (www.nzdl.org) under the terms of the GNU General Public License. The software includes everything described above: Web serving, CD-ROM creation, collection building, multilingual capability, and plug-ins and classifiers for a variety of different source document types. It includes an autoinstall feature to allow easy installation on both Windows and Unix. In the spirit of open-source software, users are encouraged to contribute modifications and enhancements.

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.

Copyright

Digital libraries can easily be made far more accessible than physical ones. And the fact that they are widely accessible 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 virtually unlimited audience.

This is great news. For the user, information around the world becomes available wherever you are. For the author, a greater potential audience can be reached than ever before. And for publishers, new markets open up that transcend geographical limitations. But there is a flip side. Authors and publishers ask how many 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 answer is *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?

How will publishers react to this situation? The threat for users is that publishers will adopt technical and legal means to implement restrictive policies governing access to the information they sell—for example, by restricting access to the purchaser (no lending to friends) or imposing expiry dates (no permanent collections). The net result could easily damage the flow of information far beyond the current status quo.

Possessing a copy of a document certainly does not constitute ownership in terms of copyright law. Though 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 redistribute it. That right rests with the copyright owner.

Who owns a particular work? The initial copyright owner is the creator, unless the work is made for hire. Works made for hire are ones created by employees within the scope of their employment, or under a specific contract that explicitly designates the work as being made for hire, in which case it is the employer or contracting organization that owns the copyright. The owner can transfer or “assign” copyright to another party through a specific contract, made in writing and signed by the owner.

Copyright protection begins and ends at different times, depending on when the work was created. In the U.S., older works are protected for 95 years after 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 for hire are protected for 95 years after publication or 120 years after creation, whichever comes first.

Copyright law is complex, arcane, and varies from one country to another. Most countries are signatories to the Berne Convention, which governs international copyright law. Most countries allow material to be copied for research purposes by individuals, a concept known as *fair use*. However, making copies for distribution, or resale, is prohibited. Copyright law applies regardless of whether the document bears the international copyright symbol ©. Unlike patents, it is not necessary to register documents for copyright—it applies automatically.

The legal situation with regard to computer files, and particularly documents published on the World Wide Web, is murky. Lawyers have questioned whether it is legal even to view a document on the Web, since one's browser inevitably makes a local copy, which has not explicitly been authorized. Of course, it is widely accepted that you can view Web documents—after all, that's what they're there for. If we allow that you can view them, next comes the question of whether you can save them for personal use. Or link to them. Or distribute them to others. Note that documents are copied and saved behind the scenes all over the place: to economize on network traffic and 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, work has led people to 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. 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.

Many digital library projects involve digitizing documents. First you must consider: Is the work to be digitized in the public domain, or does it attempt to faithfully reproduce a work in the public domain? If the answer to either question is yes, you may digitize the work without securing anyone's permission. Of course, the result of your own digitizing efforts will not be 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 was the copyright owner, you can certainly go ahead provided the donor gave your institution the right to digitize—perhaps in a written form using words such as “the right to use the work for any institutional purpose, in any medium.” Even without a written agreement, it may reasonably be 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. You cannot, for example, assume permission to digitize letters 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 how things look from the copyright owner's point of view and address those concerns. 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 will have to obtain permission to digitize the work or acquire access to it by licensing it.

If you are building a digital library, you must pay serious attention to the question of copyright. Digital library projects must be undertaken with a full understanding of ownership rights and with full recognition that permissions are essential to convert materials that are not in the public domain. Because of the potential for legal liability, any prudent library builder will 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 1.6) does contain some pointers to sources of further practical information about the copyright question. These sources include information on how fair use can be interpreted and discuss the issues involved when negotiating copyright permission or licensing.

Looking at the situation from an ethical rather than a legal point of view helps to shed light on the fundamental issues. It is unethical to steal: deriving profit by distributing a book on 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 on 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 due 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 though authorship is acknowledged.

Collecting from the Web

All of 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 it in order to organize information into focused collections and 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, provide a good example. They use software "robots" to continually download huge portions

of the Web and create indexes to them. Although they may retain documents on their own computers, users are presented with a summary and directed to the original source documents rather than to local copies. Search engines are commercial operations. Their services are not sold directly to users, however, but revenue is derived from advertising—in effect, a tax on the user’s attention. Although they are widely accepted as a good thing, their legal status is unclear.

Web sites can safeguard against indiscriminate downloading. A generally accepted *robot exclusion protocol* allows individual Web sites to prevent portions of their sites from being downloaded and indexed. Although compliance with this protocol is entirely voluntary, widely used search engines certainly do so. 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, but equally fascinating, copyright issues are raised by projects that are archiving the entire World Wide Web. The reason for doing this is to offer services such as supplying documents that are no longer available and providing a “copy of record” for publicly available documents, in effect supplying the raw material for historical studies. Creating this 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 do search engines. Like archives, they most likely 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 originating 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 of documents, or extracting keywords and key phrases automatically.

Those responsible for such libraries need to consider carefully the ethical issues above. 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 issues of misrepresentation.

The world is changing. Digital libraries are pushing at the frontiers of what is possible by way of organizing anthologies of material. And they are pushing at the frontiers of society's norms for dealing with the 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 find the offending link. On other occasions people 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 find.

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 USEnet, a huge corpus of Internet discussion groups on a wide range of topics, it received many pleas from contributors to retract indiscreet postings from the past because, being easily available for anyone to find, they were now causing their authors considerable embarrassment.

A system that downloads research papers from the Web, extracts citations from them, and compiles them into a citation index receives many complaints about incorrect references, complaints that should be directed to the authors of the citing papers rather than to the extraction service. Indeed it had to be pointed out to one irate user that a particular, incorrect reference to one of his papers that he had noticed on the system had actually been extracted from another paper on the complainant's own Web site.

As a final example, several years ago a researcher was describing to an audience at a major U.S. university an early, noncommercial digital library collection of research reports. The reports had been downloaded from the Internet (using the FTP file transfer protocol, for which there is no established exclusion protocol). One member of the audience indignantly denounced this as theft of information; others volubly defended the system as providing a useful service to researchers using publicly available material. The detractor demanded that all reports from that university be immediately withdrawn from the collection; the others requested that they be retained because they helped publicize their research.

Illegal and harmful material

Some material is illegal and harmful and clearly inappropriate for public presentation. Examples are distasteful. 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 or not often depends on the cultural, religious, and social context in which it is circulated. Standards vary enormously 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. The difficulty of sustaining local legal and cultural standards is a huge challenge that faces society today. It revolves around the dilemma of balancing freedom of expression against citizens' rights to be protected from illegal or harmful material.

A well-publicized example of different views on access to information arose in early 2000 around information concerning online sales of Nazi memorabilia on U.S. Web sites accessed using the Yahoo Internet portal. A Paris judge ruled that the sites are barred under French law and ordered them to be blocked. However, the sites are governed by less restrictive U.S. laws, and a U.S. judge ruled that the First Amendment protects content generated in the U.S. by American companies from being regulated by authorities in countries that have more restrictive laws on freedom of expression. Suit and countersuit followed, and the matter is still not settled as this book goes to press two years later.

Online gambling, where laws are restrictive (or at best muddied) in countries such as the U.S., China, and Italy, provides another example. Some international gambling sites claim to comply with local laws by checking the geographical origin of the user (a difficult and unreliable procedure which is easily circumvented) and refusing to offer their services in countries where gambling is illegal.

Cultural sensitivity

Most digital libraries are international. More often than not they are produced by people from European and North American backgrounds, yet the majority of

people in the world 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 essential for digital library developers to consider how their creations will affect other people.

We pointed out earlier that 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 deeper than this. Particular labels can have strong, and unexpected, connotations: certain car models have failed to sell in certain countries because the manufacturer's name had a serious negative association. So too for icons: dogs, for example, are offensive in Arabic cultures, and users will transfer negative associations if they are adopted as user interface icons. 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 those from Western cultures to appreciate. Many objects have different degrees of *tapu*, and it is rude and offensive to refer to them inappropriately, in the same way that many Westerners find blasphemy rude and offensive. One particular example that can affect digital library design is that representations of people—including pictures—are *tapu*, and it is generally inappropriate for them to be on public display.

1.6 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 interesting information (and, at least at the time of writing, no false hits!). 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 Humanity Libraries 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, 1996); 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, initiated by Rodger McNab and Lloyd Smith and currently led by one of the authors (Bainbridge et al., 1999; Bainbridge, 2000).

The definition of *killer app* is from the online Jargon File (version 4.2.2, at <http://info.astrian.net/jargon/Preface.html>), a comprehensive compendium of hacker slang. Our definition of *digital library* in Section 1.1 is from Akscyn and Witten (1998); it 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/fox/dlib/def.html>). 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 University College, London.

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 and not long ago was still being pursued (Wells, 1938; Goodman, 1987). Vannevar Bush’s vision was described in the year the United Nations was founded (Bush, 1947)—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), while the quotation about Allen’s human-skin book is from Kruse (1994). 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 5. 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 numbers 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*. Some of the examples of the potential uses of locally produced information in Section 1.3 come from an excellent article that Mark Warschauer is writing, entitled “What is the digital divide?”—it’s available on his Web site at www.gse.uci.edu/markw.

The Greenstone software is produced by the New Zealand Digital Library Project and is described by Witten et al. (1999b, 2000). Information about the GNU General Public License can be obtained from www.gnu.org/copyleft/gpl.html. The *toki* (adze) shown in Figure 1.10 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 which 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*.

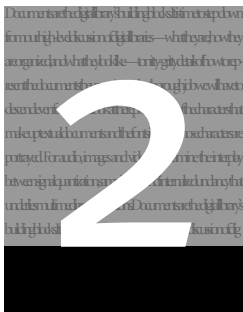
Samuelson and Davis (2000) provide an excellent and thought-provoking overview of copyright and related issues in the information age, which is a synopsis of a larger report published by the National Academy of Sciences Press

(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 that give a wide-ranging discussion of the effect of emerging technologies on intellectual property issues (White, 1999).

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 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/~unclng/public-d.htm.

The Internet archiving project is described in a *Scientific American* article by Brewster Kahle (1997); this is the source for the hypothetical college student scenario. The war stories will remain unattributed. Information on the UNESCO initiative to attack paedophilia on the Internet can be found at www.unesco.org/webworld/child_screen/conf_index.html. 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 (2000).

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 U.S. from 1993–1995. 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.



Preliminaries

Sorting out the ingredients

Building a library is a major undertaking that needs to be carefully planned. Before beginning, you should reflect on the fact that distributing any kind of information collection carries certain responsibilities. There are legal issues of copyright: being able to access documents doesn't necessarily mean you can give them to others. There are social issues: collections should respect the customs of the community out of which the documents arise. And there are ethical issues: some things simply should not be made available to others. The pen is mightier than the sword!—be sensitive to the power of information and use it wisely.

One of the first questions to ask when building a digital library is what technology will be used. You are probably reading this book because you are a practical person and are itching to get on with actually *building* your library. If you plan to use the Greenstone software introduced in Section 1.4, on which the implementation-dependent aspects of this book (Chapters 6 and 7) are based, now is a good time to turn to the first section of Chapter 6 and construct an initial collection to give you a preview of how things will work. The technical part of building a digital library is easy if you know what you want and it matches what can be done with the tools you have available. Section 6.2 leads you through the process of building a collection with Greenstone. First, to download and install the software, you need to consult the Appendix.

Having laid your mind at rest regarding the technology, begin by fixing some broad parameters of your digital library. The other chapters in this book assume

that the raw material for your library is provided in machine-readable form. Of course, one overriding question is where it is going to come from. There are three broadly different answers to this question, leading to three rather different kinds of digital library. We discuss these in Section 2.1.

As well as the raw material, you also need metadata—summary information about the documents in your collection. Librarians are expert in creating metadata and using it to facilitate access to large information collections. If you are building a sizable digital library, the team will include people with training in library science—and you will need to know something about their job. Section 2.2 introduces the principles and practices of bibliographic organization in conventional libraries.

You must also consider what kinds of access your digital library will support, and we review some of the possibilities, and the pitfalls, in Section 2.3.

Probably the single most important issue in contemplating a digital library project is whether you plan to digitize the material for the collection from ordinary books and papers. Because this inevitably involves manually handling physical material, and probably also involves manual correction of computer-based text recognition, it generally represents the vast bulk of the work involved in building a digital library. Section 2.4 describes the process of digitizing textual documents, by far the most common form of 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.

2.1 Sources of material

There are fundamental questions about the nature of the library you are building: what its purpose is, what the principles are for including documents, and when one document differs from another. While we cannot help you in making these decisions, it is essential that you ask—and answer—these questions.

Next we consider three scenarios for where your digital library material originates:

- You have an existing library that you wish to convert to digital form.
- You have access to a collection of material that you want to offer as a digital library.
- You want to provide an organized portal into a focused subset of material that already appears on the Web.

These scenarios are neither exclusive nor exhaustive, and in practice you often encounter a mixture. But they are useful in helping to focus on the questions that should be asked before embarking on a digital library construction project.

Ideology

Begin by formulating a clear conception of what it is that you intend to achieve with your proposed digital library collection—this might be called the “ideology” of your enterprise. The ideology can be formulated in terms of the collection’s *purpose*, the objectives it is intended to achieve, and its *principles*, the directives that will guide decisions on what should be included and—equally important—what should be excluded. These decisions are difficult ones. Section 1.1 introduced “wisdom” as the value attached to knowledge and argued that librarians exercise wisdom when they make decisions about what to include in a collection.

Whenever you build a digital library collection, you should formulate its purpose and state it clearly as an introduction to the collection. You should make it plain to users what principles have been adopted to govern what is included. You should also include a description of how the collection is organized.

We assume that you have established the purpose of the digital library collection and its scope in terms of what works it will include. You will also have to decide what to do about different manifestations of a single work. In the traditional library world, there is an important, but rather slippery, distinction between a *work* and a *document*. A work is the disembodied content of a message and might be thought of as pure information. In traditional libraries a document is a particular physical object (say a book) that embodies or manifests the work. We will elaborate on this distinction in Section 2.2 below.

In the case of digital libraries, a document is a particular electronic encoding of a work. The work/document distinction surfaces when we have to deal with different versions of a document. Digital representations of a work are far easier than printed ones to both copy and change. You will have to decide not only which documents to include and which to exclude, but also when two documents are the same and when they are different. Collections often contain many exact duplicates of documents; should duplicate copies be retained? And when a new version of a document appears, should it supersede the old one, or should both be kept? The answers will depend on the purpose of your collection. Archival or historical records must not be allowed to change, but errors in collections of practical or educational information must be correctable.

Here’s a further complication that affects the identity of documents. Digital libraries are not ephemeral, but have a continued existence over time. For example, they often keep records of the interaction history of individual users to facilitate future interaction. When identifiers are allocated to documents, decisions must be made about whether duplicates are significant and when new versions of documents supersede old ones. For instance, one way of assigning identifiers is to compute a number from the word sequence that makes up the document. This is attractive because exact copies receive the same identifier and are therefore mapped into the same object. However, sometimes you might want to

make an updated version of a document supersede the original by giving it exactly the same identifier even though its content is slightly different, and you cannot do this if identifiers are computed from the content.

Converting an existing library

Converting a conventional library into digital form, often the image that springs to mind when people first mention digital libraries, is the most ambitious and expensive kind of digital library project. It involves digitizing the contents of an existing paper-based collection, which is usually a huge and daunting undertaking.

Before embarking on such a task, you need to consider carefully whether there is really a need. Digital libraries have three principal advantages over conventional ones: they are easier to access remotely, they offer more powerful searching and browsing facilities, and they serve as a foundation for new value-added services. You should look at the customer base for the existing library and assess how advantageous it will be for customers to access the new digital library instead. You should look at the value of the collections in the library and consider whether the customer base will expand if the information can be made available electronically. You should look at what new services the digital library could support, such as automatic notification of new documents that match clients' interest profiles, and assess the demand for them. You need to evaluate the cost/benefit of the proposed digital library.

There are many further questions to ask. Will the new digital library coexist with the existing physical one, or supplant it? Maintaining two separate libraries will be a continual drain on resources. At what rate is the collection growing, or changing? In many situations digitization of material is not just a one-time capital cost, but an ongoing operational expenditure. Should you outsource the whole digital library operation? There exist organizations that can arrange fully searchable Web access to, for example, newspapers of your choice. Such organizations provide a full range of services, including conversion of existing documents and ongoing digitization of new material. Services like these are growing dramatically.

Can user needs be satisfied, or partially satisfied, in alternative ways? For example, it might be possible to buy access to part of your holdings through external organizations that amortize their costs by supplying services to a whole range of other customers. Converting an existing library is such a large and expensive proposition that all alternatives should be carefully explored before making a commitment.

Once you have decided to go ahead, a key question will be how to prioritize material for conversion. Library materials can be divided into three classes: special collection and unique materials, such as rare books and manuscripts; high-use items that are currently in demand for teaching and research; and low-use

items including less frequently used research materials. One set of criteria for digital conversion includes the intellectual content or scholarly value of the material, the desire to enhance access to it, and available funding opportunities. Another concerns the educational value, whether for classroom support, background reading, or distance education. A third may be the need to reduce handling of fragile originals, especially if they are heavily used. Other reasons are institutional: promoting special strengths such as unique collections of primary source material—the jewels in the crown of a research library—or resource-sharing partnerships with other libraries. Cost and space savings may also play a role. Of course copyright will have a crucial influence and is the first thing to consider (see Section 1.5).

Six principles have been identified that drive the development of library collections.

- *Priority of utility:* Usefulness is the ultimate reason behind all collection decisions. Predicting utility is, however, notoriously difficult.
- *Local imperative:* Local collections are built to support local needs, and expenditure of local resources must have a demonstrable local benefit.
- *Preference for novelty:* Although historical collections are essential for research, only limited resources can be devoted to the collection and maintenance of older material.
- *Implication of intertextuality:* To add an item to a collection is to create a relationship between it and other items. Building a collection always creates new textual relationships.
- *Scarcity of resources:* All collection development decisions have to balance scarce resources: funding, staff time, shelf space, and user time and attention.
- *Commitment to the transition:* More and more information will become available in digital form. Libraries are responsible for promoting this transition and assisting users to adjust to it.

These principles apply equally well to the selection of material for digitization.

Building a new collection

We argued in Chapter 1 that digital libraries are about new ways of dealing with knowledge—of achieving new human goals by changing the way that information is used in the world—rather than about deconstructing existing institutions. Many digital library projects build new collections of new material, rather than digitizing existing libraries.

If this is what you plan to do, you should ask—and answer!—the question, “Why you?” Are other organizations better placed to undertake this task? For a start, do you own the material?—or, more to the point, do you own the copyright? If not, you need to quickly ascertain whether you will be able to acquire

permission for your project from the copyright holders before going any further. Copyright holders are naturally cautious about permitting their material to be made available electronically because of the potential for uncontrolled access (whether intended or not). The natural organization to create a digital collection is the copyright holder; if that is not you, you need to have strong arguments as to why your organization is the appropriate one for the job.

The scale of your library-building project will be largely determined by whether the material is already available electronically, or whether material in conventional paper form needs to be digitized. If everything is already electronic, things are immeasurably easier. Even if operations such as collecting and organizing files, and converting formats, are necessary, they will be far cheaper than digitizing the material because once the appropriate procedures have been determined, the computer can be used to apply them rapidly and on a large scale.

The next question to consider is where the metadata will come from. Obtaining the necessary metadata and converting it to electronic form is likely to be a major task. Indeed, in situations where the raw documents are already available electronically, manual input of metadata will usually dominate the cost of the digital library project. When digitizing an existing library the metadata is already available, but when making new collections it imposes a substantial burden.

Virtual libraries

Another kind of digital library provides a portal to information that is available electronically elsewhere. This is sometimes referred to as a *virtual library* to emphasize that the library does not itself hold content. Librarians have used this term for a decade or more to denote a library that provides access to distributed information in electronic format through pointers provided locally. As we noted in Chapter 1, the Web lacks the essential digital library features of selection and organization. But it does contain a vast wealth of useful information. People who sift through this information and build organized subcollections of the Web do a useful job, and an important subclass of digital libraries comprises those that provide access to information that is already available on the Web.

Information portals usually concentrate on a specific topic or focus on a particular audience. Commercial Web search engines are unable to produce consistently relevant results, given their generalized approach, the immense amount of territory they cover, and the great number of audiences they serve. The problem is likely to be exacerbated by the increasing use of the Web for commerce. Search engine companies strive to support themselves by enhancing the level of service they provide for commercial activity, with the aim of becoming attractive vehicles for advertising.

Virtual libraries present new challenges. Clearly the source information is already available electronically, in a form that can be readily displayed with a

Web browser. Some metadata will also be present—notably title and possibly author. The value that is added by imposing a digital library organization on a subset of the Web is twofold: selection of content and the provision of further metadata whereby it can be organized.

First, consider content selection. You need to define a purpose or theme for your library, and then discover and select material on the Web that relates to this theme. You can do this by manually seeking and filtering information, using whatever tools are available—search engines and the like. You can also attempt it automatically, with programs that crawl the Web, following links and seeking out new information that is relevant to the defined theme. Such systems typically start with a small and focused selection of relevant documents and traverse the Web seeking new documents to augment the collection. In a domain-specific search the Web is not explored indiscriminately, but in a directed fashion that seeks relevant documents efficiently. This raises interesting questions of how to direct and control the search. Focused Web crawling is an important topic that is likely to develop into a major technique for information discovery. In practice, to build a high-quality collection it will probably always be necessary to manually filter the results of automatic Web crawling.

Second, consider the provision of further metadata. Like content selection, this can also be done either manually or automatically. Categorizing and classifying Web pages helps to connect researchers and students with important scholarly and educational resources. Of course techniques for automatically assigning metadata are extremely valuable for any digital library, and in Chapter 5 we will encounter various metadata operations, ranging from phrase extraction to acronym identification. In libraries of material gathered from the Web, these techniques assume a special importance since they constitute the bulk of the *raison d'être* for a digital library portal.

A good example of a virtual library is INFOMINE, a cooperative project of the University of California and California State University (among others). Run by librarians, it covers most major academic disciplines through access to important databases, e-journals, e-texts, and other digital collections. It contains descriptions and links to a wealth of scholarly and educational Internet resources, each of which has been selected and described by a professional academic librarian who is a specialist in the subject and in resource description generally. Librarians see this as an important expenditure of effort for their users, a natural evolution of their traditional task of collecting and organizing information in print.

It takes an hour or two to prepare a traditional library catalog entry for a new book or journal. At the other extreme, Web search engines provide no metadata and no access mechanism other than searching the text for particular words and phrases, but automatically recrawl and reindex the entire Web every few weeks. Virtual libraries occupy an intermediate position. INFOMINE asks human catalogers to complete each record in 25 minutes on average. In addition a variety

of semiautomated techniques are used to determine when information has been moved or altered significantly, and to automatically update links, or flag the site for manual reindexing, accordingly.

In general the higher the scholarly or educational value of a resource, the greater the amount of expert time that can be invested in its description. A scenario for semiautomated resource discovery and description might involve three levels of material:

1. Automatically generated, with URL, author-supplied metadata, significant keywords and phrases extracted from the full text, and generalized subjects assigned automatically
2. Manually reviewed by a human expert who edits and enriches the automatically derived metadata, checking it for accuracy and adding annotations and subject headings
3. Intensively described by a human expert who provides extensive metadata from scratch

Information could move from the first to the second level if it is judged to be sufficiently central to the collection's focus on the basis of automatic classification information, and from the second to the third level on the basis of sufficiently high usage.

2.2 Bibliographic organization

We have discussed where the documents in the digital library might originate. Now let's talk about how the metadata is produced: the summary information that provides the hooks on which all library collections are organized.

Organizing information on a large scale is far more difficult than it seems at first sight. In his 1674 Preface to the Catalogue for the Bodleian Library in Oxford, Thomas Hyde lamented the lack of understanding shown by those who had never been charged with building a catalog:

"What can be more easy (those lacking understanding say), having looked at the title-pages than to write down the titles?" But these inexperienced people, who think making an index of their own few private books a pleasant task of a week or two, have no conception of the difficulties that arise or realize how carefully each book must be examined when the library numbers myriads of volumes. In the colossal labor, which exhausts both body and soul, of making into an alphabetical catalogue a multitude of books gathered from every corner of the earth there are many intricate and difficult problems that torture the mind.

In this section we will look at traditional principles of bibliographic organization, and at current practice. Librarians have a wealth of experience in classi-

fying and organizing documents in a way that makes relevant information easy to find. Although some features are less relevant to digital libraries—such as the physical constraint of arranging the library’s contents on a linear shelving system—there is nevertheless a great deal to be learned from their experience, and would-be digital librarians would be foolish to ignore it. Most library users locate information by finding one relevant book on the shelves and then looking around for others in the same area. In fact conventional libraries provide far more powerful retrieval structures.

Objectives of a bibliographic system

It was not until the late 19th century, long after Thomas Hyde bemoaned the difficulty of building a catalog, that the objectives of a bibliographic system were first explicitly formulated. There were three: *finding*, *collocation*, and *choice*.

The first objective, *finding*, was to enable a person to find a book of which either the author, title, or subject is known. In modern terms this involves finding information in a database, confirming its identity, and perhaps ascertaining its whereabouts and availability. The user is assumed to be seeking a known document and has in hand author, title, or subject information. The modern library catalog is designed expressly to support these features.

The second objective, *collocation*, was to show what the library has by a given author, on a given subject, or in a given kind of literature. To “collocate” means to place together or in proper order. This objective is concerned with locating information that is nearby in one of several information spaces. The organization of the library catalog and the spatial arrangement of the books on the shelves help satisfy the objective.

The third objective, *choice*, was to assist in the choice of a book either bibliographically, in terms of its edition, or topically, in terms of its character. The assumption is that the reader is faced with a number of different documents—perhaps several different editions of a work—and must choose among them.

In the last few decades these three objectives have been refined and restated several times. Recently they have been expanded into the following five.

1. To *locate* entities in a file or database as the result of a search using attributes or relationships of the entities:
 - a. To find a single entity, that is, a document
 - b. To locate sets of entities representing
 - all documents belonging to the same work
 - all documents belonging to the same edition
 - all documents by a given author
 - all documents on a given subject
 - all documents defined by “other” criteria

2. To *identify* an entity, that is, to confirm that the entity described in a record corresponds to the entity sought, or to distinguish among several entities with similar characteristics
3. To *select* an entity that is appropriate to the user's needs with respect to content, physical format, and the like
4. To *acquire* or obtain access to the entity described, through purchase, loan, and so on, or by online access to a remote computer
5. To *navigate* a bibliographic database: to find works related to a given one by generalization, association, and aggregation; to find attributes related by equivalence, association, and hierarchy

The Latin word *entity* lacks the Anglo-Saxon directness of *book* and *work*. It serves to broaden the scope of the objectives beyond books and documents to include audio, video, and both physical and digital objects. It also serves to broaden the search criteria from the particular metadata of author, title, and subject. The existence of a catalog of metadata is acknowledged explicitly as the basis of searching, along with the use of attributes and relationships. A new *acquisition* objective has been added, along with a *navigation* objective that greatly generalizes the traditional notion of collocation expressed by objective 1b.

Bibliographic entities

The principal entities in a bibliography are *documents*, *works*, *editions*, *authors*, and *subjects*. Sets of these, such as document collections, the authors of a particular document, and the subjects covered by a document, are also bibliographic entities. Other entities are more directly related to the production process: for example, an *imprint* is the set of printings of a document that preserve the image of a previous printing. We also deal below with *titles* and *subject classifications*, which strictly speaking are attributes of works rather than entities in their own right.

Documents

Documents are the basic inhabitants of the physical bibliographic universe. We characterized them in Section 2.1 as particular physical objects that embody or manifest the intellectual content of a work. Although traditional documents are physical objects, they are not necessarily physically independent of other objects: an article in a journal or a chapter in an edited collection is really a document in its own right.

In the case of digital libraries, a document is a particular electronic encoding of a work. Since one document can form an integral part of another, documents have a fundamental hierarchical structure that can be more faithfully reflected in a digital library than in a physical one. On the other hand, digital documents have a kind of impermanence and fluidity that makes them hard to deal with.

They can be instantly (and unthinkingly) duplicated, they frequently have uncertain boundaries, and they can change dynamically. This creates difficulties that we have already met in Section 2.1: helpful digital libraries need to present users with an image of stability and continuity, as though electronic documents were identifiable, discrete objects like physical ones.

Works

Works are the basic inhabitants of the intellectual bibliographic universe. They can be conceptualized as the disembodied content of documents. In practice an important operational question for traditional librarians is this: when are two documents sufficiently alike that they can be considered to represent the same work? For books, operations such as revision, updating, expansion, and translation into different languages are generally held to preserve the identity of a work. Translation to another medium may or may not. An audiotape of a book would likely be considered the same work; a video production would generally not.

The distinction between a work and a document may appear to be an abstract one, far removed from the world of real libraries. However, it has important practical consequences. A particular work may appear in several different editions. Some editions of children's books may be lavishly illustrated, others plain. New editions of scientific books may contain important revisions or substantial additional material. Editions may appear in different languages. They may be condensed versions ("Reader's Digest" editions) or expanded with scholarly annotations and footnotes. Assuming that different manifestations all have the same, or approximately the same, intellectual content, they may be considered to represent the same work. However, whether the differences are significant or not really depends on the reader's orientation and purpose.

A work's identity is even more seriously compromised when it is reinterpreted in a different medium. Different movie versions of a story usually represent radically different interpretations and are generally considered to be different works. The case of poetry or story readings is less clear-cut. If, as Marshall McLuhan claimed, the medium really is the message, then interpretations in different media certainly do imply different works.

Editions

Editions are the book world's traditional technique for dealing with two difficult problems: different presentation requirements, and managing change of content. The various editions of a work share essentially the same information but differ with respect to both printing details and updated content. A large-font edition for the visually impaired or a pocket edition for the traveler address different presentation requirements. Revised editions correct and update the content.

Electronic documents generally indicate successive modifications to the same work using terms such as *version*, *release*, and *revision* rather than *edition*. These

terms are not standardized, and different people use them in different ways. No matter what term you use, the underlying problem is that new editions of an electronic document may be produced extraordinarily rapidly, unencumbered by traditional processes of physical book production. We cannot even estimate how many versions we have generated of the present book while writing it, if a new version had been declared every time a file was saved. Yet these changes over time could easily have been captured by a dynamic digital library that always presented the current incarnation of the work but allowed you to examine earlier versions too.

Authors

Of all bibliographic entities, authors seem the most straightforward, and authorship has been the primary attribute used to identify works since medieval times. All Western libraries provide an author catalog that places books by the same author together. However, authorship is not always straightforward. Some works appear to emanate from organizations or institutions; are they then “authors”? Modern scientific and medical papers often have numerous authors because of the complex and collaborative nature of the work and institutional conventions about who should and should not be included. Many works are anthologies: is the editor to be regarded as an “author”? If not, what about anthologies that include extensive commentaries by the editors; when is this deemed worthy of authorship? And what about ghostwriters?

For a concrete example of some of the difficulties, turn to the “References” section at the end of the book and locate the entry under “Library of Congress (1998).” Who is the author of this book? A particular person or group of people? The Library of Congress? The Library of Congress Cataloging Policy and Support Office?

In the digital world, documents may or may not represent themselves as being written by particular authors. If they do, authorship is commonly taken at face value. Of course many documents may end up authorless by this criterion. However, even if many works are anonymous, when authorship is clearly identifiable, users of a digital library will expect to be able to locate documents by author.

Taking authorship at face value has significant drawbacks—not so much because people misrepresent authorship, but because differences often arise in how names are written. For one thing, authors sometimes use pseudonyms. But a far greater problem is simply inconsistency in spelling and formatting. Traditional librarians go to great pains to normalize names into a standard form.

Table 2.1, admittedly an extreme example, illustrates just how difficult the problem can become. It shows different ways in which the name of *Muammar Qaddafi* (the Libyan leader) is represented on documents that have been received

Table 2.1 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	Mu amar al-Kad'afi	Qathafi, Mu'Ammar el
Mu ammar Al Qathafi	Kad'afi, Mu amar al-	El Qathafi, Mu'Ammar
Muammar Al Qathafi	Gaddafy, Muammar	Kadaffi, Momar
Mo ammar el-Gadhafi	Gadafi, Muammar	Ed Gaddafi, Moamar
Muammar Kaddafi	Gaddafi, Muammar	Moamar el Gaddafi
Moamar El Kadhafi	Kaddafi, Muamar	

by the Library of Congress. The Library catalog chooses one of these forms, ostensibly the form in which the author is most commonly known—*Qaddafi*, *Muammar* in this case—and then groups all variants under this one spelling—with appropriate cross-references in the catalog from all of the variant spellings. In this case, ascribing authorship by taking documents at face value would end up with 47 different authors!

The creation of standardized names for authors is called *authority control*, and the files that librarians use to record this information are called *authority files*. This is one instance of a general idea: using a *controlled vocabulary* or set of *preferred terms* to describe entities. Terms that are not preferred are *deprecated* (a technical term that does not necessarily imply disapproval) and are often listed explicitly with a reference to the associated preferred term—as in Table 2.1. Controlled vocabularies are to be contrasted with the gloriously uncontrolled usage found in free text, where there are no restrictions at all on how authors may choose to express what they want to say.

Titles

Titles are really attributes of works rather than entities in their own right. Obviously they are important elements of any bibliographic system. In digital collections, titles, like authors, are often taken at face value from the documents

Table 2.2 Title pages of different editions of *Hamlet*.

<i>Amleto, Principe di Danimarca</i>	<i>Montale Traduce Amleto</i>
<i>Der erste Deutsche Buhnen-Hamlet</i>	<i>Shakespeare's Hamlet</i>
<i>The First Edition of the Tragedy of Hamlet</i>	<i>Shakspeare's Hamlet</i>
<i>Hamlet, A Tragedy in Five Acts</i>	<i>The Text of Shakespeare's Hamlet</i>
<i>Hamlet, Prince of Denmark</i>	<i>The Tragedy of Hamlet</i>
<i>Hamletas, Danijos Princas</i>	<i>The Tragicall Historie of Hamlet</i>
<i>Hamleto, Regido de Danujo</i>	<i>La Tragique Histoire d'Hamlet</i>
<i>The Modern Reader's Hamlet</i>	

themselves. However, in the world of books they can show considerable variation, and librarians use vocabulary control for titles as well as for authors. For example, Table 2.2 shows the titles that are represented on the title pages of 15 different editions of *Hamlet*.

Subjects

Subjects rival authors as the predominant gateway to library contents. Although physical library catalog systems boasted card-based subject catalogs, which were sometimes a useful retrieval tool, computer-based catalogs have breathed new life into subject searching and are now widely used in libraries. However, subjects are far harder to assign objectively than authorship, and involve a degree of, well, subjectivity. Interestingly, the dictionary defines *subjective* as both

Pertaining to the real nature of something; essential

and

Proceeding from or taking place within an individual's mind such as to be unaffected by the external world.

Perhaps the evident conflict between these two meanings says something about the difficulty of defining subjects objectively!

As with authorship, digital documents sometimes explicitly represent what they are about by including some kind of subject descriptors. Otherwise there are two basic approaches to automatically ascribing subject descriptors or “key phrases” to documents. One is key-phrase *extraction*, where phrases that appear in the document are analyzed in terms of their lexical or grammatical structure, and with respect to phrases that appear in a corpus of documents in the same domain. Phrases, particularly noun phrases, that appear frequently in this document but rarely in others in the same domain are good candidates for subject descriptors. The second is key-phrase *assignment*, where documents are automatically classified with respect to a large corpus of documents for which subject

descriptors have already been determined (usually manually), and documents inherit descriptors that have been ascribed to similar documents. We return to this topic in Chapter 5 (Section 5.6).

It is far easier to assign subject descriptors to scientific documents than to literary ones, particularly works of poetry. Many literary compositions and artistic works—including audio, pictorial, and video compositions—have subjects that cannot readily be named. Instead they are distinguished as having a definite style, form, or content, using artistic categories such as *genre*.

The Library of Congress Subject Headings (LCSH) are a comprehensive and widely used controlled vocabulary for assigning subject descriptors. They currently occupy five large printed volumes, amounting to about 6,000 pages each, commonly referred to by librarians as “the big red books.” The aim is to provide a standardized vocabulary for all categories of knowledge, descending to quite a specific level, so that books—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.

The red books contain a total of around two million entries, written in three columns on each page. Perhaps 60 percent of them are full entries like the one for *Agricultural Machinery* in the first row of Table 2.3. 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 these three deprecated terms has its own one-line entry that indicates (with a *USE* link) that *Agricultural Machinery* should be used instead; these 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. These are expressed by *BT* (broader topic) and *NT* (narrower topic), respectively, which again are inverses. The *Agricultural Machinery* example of Table 2.3 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 2.3 Library of Congress Subject Heading entries.

Agricultural Machinery		Machinery	
UF	Agriculture—Equipment and supplies	UF	...
	Crops—Machinery	BT	...
	Farm machinery	RT	...
BT	Machinery	SA	...
RT	Farm equipment	NT	...
	Farm mechanization		Agricultural machinery
	Machine-tractor stations		...
SA	subdivision Machinery under names of crops, e.g., Corn—Machinery	Agricultural Implements	
NT	Agricultural implements	UF	...
	Agricultural instruments	BT	Agricultural machinery
	
Agriculture—Equipment and supplies		Farm Equipment	
USE	Agricultural Machinery	UF	...
Crops—Machinery		BT	...
USE	Agricultural Machinery	RT	Agricultural machinery
Farm machinery			...
USE	Agricultural Machinery	...	

Finally, the SA or “see also” entry indicates a whole group of subject headings, often specified by example—like the *e.g., Corn—Machinery* under *Agricultural Machinery* in Table 2.3. 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*.

Subject classifications

Books on library shelves are usually arranged by subject. Each work is assigned a single code or *classification*, and books representing the work are physically placed in lexicographic code order. Note that the classification code is not the same as the subject headings discussed above: any particular item has several subject headings but only one classification code. The purpose of a library classification scheme is

to place works into topic categories, so that volumes treating the same or similar topics fall close to one another. Subject classification systems that are in wide use in the English-speaking world are the Library of Congress Classification (originating from the U.S.), the Dewey Decimal Classification (from England), and the Colon Classification System (used in India).

Physically browsing library shelves is a popular way of finding related material. Of course readers who browse books in this way have access to the full content of the book, which is quite different from browsing catalog entries that give only a small amount of summary metadata. Indeed most readers—even scholars—remain blithely unaware that there is any way of finding related books other than browsing library shelves, despite the existence of the elaborate and carefully controlled subject heading scheme described above. Physical placement on library shelves, being a one-dimensional linear arrangement, is clearly a far less expressive way of linking content than the rich hierarchy that subject headings provide.

Placing like books together adds an element of serendipity to searching. You catch sight of an interesting book whose title seems unrelated, and a quick glance inside—the table of contents, chapter headings, illustrations, graphs, examples, tables, bibliography—gives you a whole new perspective on the subject. Catalog records—even ones that include abstracts or summaries of the contents—are nowhere near as informative as full text, and no mere catalog can substitute for the experience of browsing full texts of related works. This helps to explain why most readers perform a simple author or title search for a specific book, or one with a suitable title, and then follow up by browsing through nearby books—ignoring the whole machinery of controlled vocabularies and subject classifications that librarians have taken such pains to provide.

Whereas physical libraries are constrained to a single linear arrangement of books, digital libraries, of course, are not restricted to such an organization. The entire contents of a digital collection can (in principle) be rearranged at the click of a mouse, and rearranged again, and again, and again, in different ways depending on how you are thinking. The whole reason for the subject classification vanishes: there is no single linear arrangement of works. The only circumstance in which you might need subject classifications is when you reach outside the library to systems whose users are not so fortunate as to have access to full texts.

2.3 Modes of access

The purpose of libraries is to give people access to information, and digital libraries have the potential to increase access tremendously. Now you do not have to go to the library, it comes to you—in your office and home, in your hotel while traveling, even at the café, on the beach, and in the plane.

Most digital libraries are on the Web (although many restrict access to in-house use). This provides convenient and universal access—at least in the developed world, and for people who possess the necessary technology. You can work in the library whenever you want, wherever you are, and so can others.

If a physical library decides to offer a digital library service to the general public, terminals will need to be installed for people to use. Service is no longer confined to a central library building; terminals can be deployed in publicly accessible locations that are closer to individual user groups. Each workstation may need additional equipment for library use, such as printed reference material and a color printer for users, along, perhaps, with general application software and other devices such as scanners. Reliance on external networks may be reduced by configuring clusters of workstations with local servers that contain a copy of the entire digital library—this is often called *mirroring*.

As we learned in Chapter 1, Web access may be infeasible or inappropriate in situations such as developing countries and disaster relief operations. In this case digital libraries may be delivered on read-only mass-storage consumer devices such as CD-ROM or DVD. Even when this is not necessary for logistic reasons, users may like to own a physical, permanent, immutable copy of the library contents. Sometimes this is valuable psychologically. A large private collection of hardly read books makes many people feel that they somehow possess the knowledge therein. Also, on the Web you are always at the mercy of someone else. Finally, one way of controlling access is to distribute the library on tangible physical devices instead of putting it on universal networks.

You can have the best of both worlds. Digital library collections can coexist both on the Web and on read-only mass storage devices, and be accessible in exactly the same way from either medium.

From a user interface point of view, Web browsers implement a lowest common denominator that falls well below the state of the art in interface technology. Apart from a few local operations—scrolling, menu selection, button highlighting—the only possible response to a user action is to display a new page, and response time is unpredictable and often far from immediate. The many small differences among browsers, and different versions of browsers, exacerbate the lowest common denominator effect and suck up inordinate quantities of development effort. Of course the use of scriptable browsers (e.g., JavaScript) and local applets (e.g., in Java) helps to some extent. However, fully reactive user interfaces call for tighter coupling between the user's workstation or "client" and the central digital library server.

Again you can have the best of both worlds. A digital library system can provide a basic end-user service over the Web, as well as a more intimate connection with the digital library server using an established protocol (such as the "common object request broker architecture" or CORBA protocol). Then more

sophisticated user interfaces can be implemented that use the protocol to access the basic library services of searching, browsing, and delivery of documents and metadata. Experimental or special-purpose interfaces can share the same central library system with ordinary users who work on standard Web browsers. Such interfaces might provide services as varied as virtual reality interfaces for physically navigating through a digital library, or query visualization using Venn diagrams, or content visualization via dynamic three-dimensional cluster diagrams. If necessary, network delays can be mitigated by temporary caching or by permanently mirroring the digital library contents locally.

Some digital library applications—notably video libraries and reactive interfaces involving sophisticated visualizations—need higher bandwidth than can be provided by general-purpose network technology. These call for special treatment and are beyond the scope of this book.

Even though the library is delivered by the Web, access may have to be restricted to authorized users. Software firewalls that restrict access to particular named sites are one solution; password protection is another. Since the basic Web infrastructure provides universal access, password protection must be implemented within the digital library system itself. Access can be restricted to the library as a whole, or to certain collections within it, or to certain documents, or even to certain pages or paragraphs. For example, any user may be allowed to present text searches and view the results in the form of extracts or summaries of the target documents, but users may have to authenticate themselves when it comes to viewing the full text of a document—or of particular “classified” documents. The basic technology of reliable password protection is well-developed and easy to provide.

Digital watermarking is another control mechanism: it guards against unauthorized distribution of information taken from the library. While data can be encrypted to prevent unauthorized access during transmission, it is difficult to devise reliable technical means that prevent a user authorized to read a document from copying and distributing it—particularly remote users who are capable of installing their own software on library workstations. Instead documents in the form of pictures and document images can be “watermarked” by embedding a secret, imperceptible, ineradicable code into their structure. This may not prevent copying, but it does ensure that the owner of the copied material can be identified. To help track down the source of the leak, different watermarks can be used that identify the workstation to which the material was served.

Another form of access is communicating with other digital library systems. Conventional library systems use an international standard communication protocol (Z39.50; see Chapter 8) to give individual users and other libraries remote access to their catalog records. Digital libraries may offer their users integrated access to the world’s library catalogs by integrating “client” software

into the user interface. Furthermore, by embedding a “server” for the standard protocol, they can offer their own content to external libraries and their users.

Finally, a digital library system may be implemented as a distributed system. Replicated servers on different computers—perhaps widely dispersed—offer their own collections, or their own parts of a single collection. The servers communicate among themselves and present a single unified view to users. Some collections may be replicated in different places, perhaps to reduce network traffic by permitting local responses to local users, or perhaps to accelerate access by dispatching different queries to different processors. Other collections may be geographically distributed, the results from different servers being integrated before presentation to the user, who perceives a single seamless collection. Such an architecture permits “virtual” digital libraries that do not actually hold their own library material but merely act as a broker to information served elsewhere.

2.4 Digitizing documents

One of the first things to consider when starting to build a digital library is whether you need to digitize existing documents. Digitization is the process of taking traditional library materials, typically in the form of books and papers, and converting them to electronic form where they can be stored and manipulated by a computer. Digitizing a large collection is an extremely time-consuming and expensive process, and should not be undertaken lightly.

There are two stages in digitization, illustrated in Figure 2.1. The first produces a digitized image of each page using a process known as *scanning*. The second produces a digital representation of the textual content of the pages using a process called *optical character recognition* (OCR). In many digital library systems it is the result of the first stage that is presented to library readers: what they see are page images, electronically delivered. The second stage is necessary if a full-text index is to be built automatically for the collection that allows you to locate any combination of words, or if any automatic metadata extraction technique is contemplated, such as identifying the titles of documents by finding them in the text.

Sometimes the second stage may be unnecessary, but this is rare because a prime advantage of digital libraries is automatic searching of the full textual content of the documents. If, as is usually the case, the second stage is undertaken, this raises the possibility of using the OCR result as an alternative way of displaying the page contents. This will be more attractive if the OCR system is not only able to interpret the text in the page image, but can retain the page layout as well. Whether or not it is a good idea to display its output depends on how well the page content and format is captured by the OCR process. We will see examples in the next chapter of collections that illustrate these different possibilities.

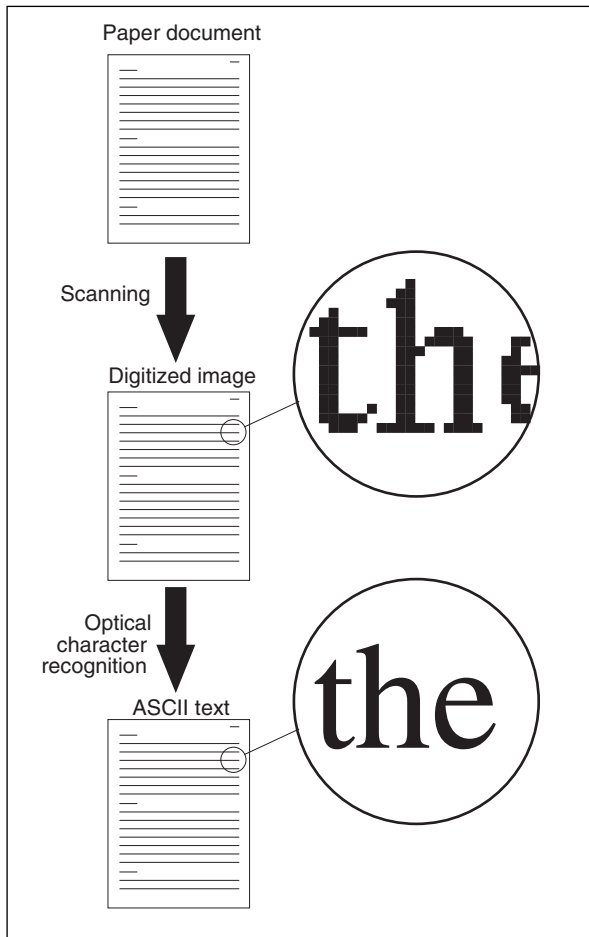


Figure 2.1 Scanning and optical character recognition.

Scanning

The result of the first stage is a digitized image of each page. These images resemble digital photographs, although it may be that each picture element or *pixel* is 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 images include nontextual material such as pictures, or contain artifacts like coffee stains or creases, grayscale or color images will resemble the original pages more closely.

When digitizing documents by scanning page images, you will need to decide whether to use black-and-white, grayscale, or color, and you will also need to determine the resolution of the digitized images—that is, the number of pixels

Table 2.4 An assortment of devices and their resolutions.		
Device	Resolution (dpi)	Depth (bits)
Laptop computer screen (14-inch diagonal, 1024 × 768 resolution)	92 × 92	8, 16, 24, or 32
Fax machine	200 × 100 or 200 × 200	1
Scanner	300 × 300 or 600 × 600	1, 8, or 24
Laser printer	600 × 600	1
Phototypesetter	4800 × 4800	1

per linear unit. For example, ordinary faxes have a resolution of around 200 dpi (dots per inch) in the horizontal direction and 100 dpi vertically, and each pixel is either black or white. Faxes vary a great deal because of deficiencies in the low-cost scanning mechanisms that are typically used. Another familiar example of black-and-white image resolution is the ubiquitous laser printer, which generally prints 600 dots per inch in both directions. Table 2.4 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 many computer screens allow several bits per pixel. Monochrome displays often show 16 or 256 levels of gray, while 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. Grayscale and 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. It corresponds roughly to a 640 × 480 pixel display at normal viewing levels and distances.

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

You will need to obtain scanned versions of several sample pages, choosing the test pages to cover the various kinds and quality of images in the collection, digitized to a range of different qualities—different resolutions, different numbers of gray levels, color and monochrome. You should perform trials with end users of the digital library to determine what quality is 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 for page access by users, particularly remote users. Doubling the linear resolution quadruples the number of pixels, and although this increase is ameliorated somewhat 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 Chapter 4), so that users who need high-quality pictures can be sure that 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” or 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 process involved in creating a digital library from printed material. You might characterize the OCR operation as taking “dumb” page images that are nothing more than images, and producing “intelligent” electronic text that can be searched and processed in many different ways.

As a rule of thumb, you need an image resolution of 300 dpi to support OCR of regular fonts of size 10-point or greater, and an image resolution of 400 to 600 dpi for smaller font sizes (9-point or less). Note that some scanners take up to four times longer for 600 dpi scanning than for 300 dpi. 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, if you scan offline, 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 kind of input that is presented. 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, the result of 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 life rarely presents us with favorable conditions. Problems occur with proper names, with foreign names and words, and with special terminology—like Latin names for biological species. They also occur with strange fonts, and particularly foreign alphabets with accents or diacritical marks, or non-Roman characters. With all kinds of mathematics. With small type or smudgy print. With overdark characters that have smeared or bled together, or overlight ones whose characters have broken up. With tightly packed or loosely set text where, to justify the margins, character and word spacing diverge widely from the norm. With hand annotation that interferes with the print. With water-staining, or extraneous marks such as coffee stains or squashed insects. With multiple columns, particularly when set close together. With any kind of pictures or images—particularly ones that contain some text. With tables, footnotes, and other floating material. With unusual page layouts. When the text in the images is skewed, or the lines of text are bowed from trying to place book pages flat on the scanner platen, or when the book binding has interfered with the scanned text. These problems may sound arcane, but even modest OCR projects often encounter many of 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 seven pages of this book). Such levels are often most easily achievable by keyboarding. Regardless of whether the material is rekeyed or processed by OCR with manual correction, 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 the error rate drops below 95 percent. Moreover, once the initial OCR pass is complete, costs tend to double with each additional percentage increase in accuracy that is required. However, the distribution of errors over pages in a large image conversion project is generally far from uniform: often 80 percent of the errors come from 20 percent of the page images. It may be worth considering having the worst of the pages manually keyed and performing OCR on the remainder.

Interactive OCR

Because of the difficulties mentioned above, OCR is best performed as an interactive process. Human intervention is useful both before the actual recognition,

when cleaning up the image, and afterward, when cleaning up the text produced. The actual recognition part can be time-consuming—times of one or two minutes per page are not unusual—and it is useful to be able to perform interactive preprocessing for a batch of page images, have them recognized offline, and return to the batch for interactive cleanup. Careful attention to such practical details can make a great deal of difference in a large-scale OCR project.

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

Acquisition

In the initial scanning step, 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: this is done using a standard interface specification called *TWAIN*. Your OCR program may be able to scan many page images in one batch and let you work interactively on the other steps afterward; this will be particularly useful if you have an automatic document feeder.

Cleanup

The cleanup stage applies certain image-processing operations to the whole image, or to parts of it. 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.

Page analysis

The page analysis stage examines the layout of the page and determines which parts of it to process, and in what order. Again this can take place either manually or automatically. The result is to segment the page into blocks of different types. Typical types include 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 sorted into correct reading order.

FEATURE ARTICLES GROS TITRES

Visualizing and Understanding Diagnoses

Salvaya Abu-Hakima

Knowledge Systems Laboratory
Institute for Information Technology
National Research Council, Ottawa, Canada
email: salvaya@cit.irc.nrc.ca tel: (613) 994-1351



Sommaire

Le diagnostic de systèmes physiques tels que les automobiles et les moteurs d'avion est une activité complexe. Les techniques affaiblissent les manuels techniques qui incluent des schémas avec l'analyse des données mesurées pour diagnostiquer et réparer les moteurs. Les concepteurs de système à base de connaissance qui ajoutent l'hypermédia pour lier le texte avec les graphiques (hypermedia) pour simplifier la tâche des techniciens, tel qu'imprimé dans le projet JETA (Hakima90). 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 diagnostic qui raisonne en déterminant des explications, les explications sont utilisées pour comprendre le fonctionnement du système (Abu-Hakima90). Cet article argumente que même si ces approches à base de connaissances aident à la visualisation et à 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 actual 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 hypertext 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 (Hakima90). Other systems have followed modelling and simulation techniques that represent the actual physical systems and attempt to diagnose them on the basis of the expected behaviour of the model (MARR91). Some diagnostic systems

also used to 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, explanations is used to understand system reasoning (Abu-Hakima90). This paper argues that although these knowledge-based approaches help in the visualization and understanding of diagnosis in physical systems, they need to be improved and better integrated.

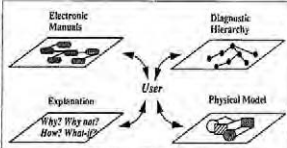


Figure 1: World of user in knowledge-based system

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 hypertext 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 (Hakima90). Other systems have followed modelling and simulation techniques that represent the actual physical systems and attempt to diagnose them on the basis of the expected behaviour of the model (MARR91). Some diagnostic systems

4 / Intelligence Artificielle au Canada automne 1992

(a)

Figure 2.2 (a) Document image containing different types of data; (b) the document image segmented into different regions. Copyright © 1992 Canadian Artificial Intelligence Magazine.

Figure 2.2a shows an example of a scanned document with regions that contain different types of data: text, two graphics, and a photographic image. In Figure 2.2b, 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 such as 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 set up manually a template layout pattern that applies to a whole batch of pages. For example, you may be able to define header and footer regions, and specify that each page contains a double column of text—perhaps even giving the bounding boxes of the columns. Perhaps the whole page analysis process should be circumvented by specifying in advance that all pages contain single-column running text, without headers, footers, pictures, or tables.

FEATURE ARTICLES GROS TITRES

Visualizing and Understanding Diagnoses

Salvaya Abu-Hakima

Knowledge Systems Laboratory
Institute for Information Technology
National Research Council, Ottawa, Canada
email: salvaya@cit.irc.nrc.ca tel: (613) 994-1351



Sommaire

Le diagnostic de systèmes physiques tels que les automobiles et les moteurs d'avion est une activité complexe. Les techniques affaiblissent les manuels techniques qui incluent des schémas avec l'analyse des données mesurées pour diagnostiquer et réparer les moteurs. Les concepteurs de système à base de connaissance qui ajoutent l'hypermédia pour lier le texte avec les graphiques (hypermedia) pour simplifier la tâche des techniciens, tel qu'imprimé dans le projet JETA (Hakima90). 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 diagnostic qui raisonne en déterminant des explications, les explications sont utilisées pour comprendre le fonctionnement du système (Abu-Hakima90). Cet article argumente que même si ces approches à base de connaissances aident à la visualisation et à 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 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 hypertext 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 (Hakima90). Other systems have followed modelling and simulation techniques that represent the actual physical systems and attempt to diagnose them on the basis of the expected behaviour of the model (MARR91). Some diagnostic systems

also used to 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, explanations is used to understand system reasoning (Abu-Hakima90). This paper argues that although these knowledge-based approaches help in the visualization and understanding of diagnosis in physical systems, they need to be improved and better integrated.

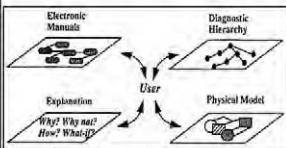


Figure 1: World of user in knowledge-based system

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 hypertext 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 (Hakima90). Other systems have followed modelling and simulation techniques that represent the actual physical systems and attempt to diagnose them on the basis of the expected behaviour of the model (MARR91). Some diagnostic systems

4 / Intelligence Artificielle au Canada automne 1992

(b)

Finally, although word spacing is usually ignored, in some cases spaces may be significant—as when dealing with formatted computer programs.

Tables are particularly difficult to handle. For each one, 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 font type, whether regular typeset text, fixed-width typewriter print, or dot matrix characters. Another is the alphabet or character set, which is determined by the language in question. Most OCR packages only deal with the Roman alphabet; some accept Cyrillic, Greek, and Czech too. Recognizing Arabic text, the various Indian scripts, or ideographic languages like Chinese and Korean is a task that calls for specialist software.

Even within the Roman alphabet there are some character-set variations. While English speakers are accustomed to the 26-letter alphabet, many languages do not employ all the letters—Māori, for 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 such as mathematical symbols. It may also be useful when recognizing 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 an unrecognized character is encountered, the user has an opportunity to enter it as a new pattern. 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 new pattern are supplied. When naming a new 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 can be achieved with training. OCR still does not perform well with more stylized type styles, such as Gothic, that are significantly different from modern ones—and training may not help much.

Obviously, better OCR results can be obtained if a language dictionary is incorporated into the recognition process. It is far easier to distinguish letters such as *o*, *0*, *O*, and *Q* if they are interpreted in the context of the words in which they occur. Most OCR systems include predefined language dictionaries and are able to use domain-specific dictionaries containing such things as 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 particular language determines the basic alphabet, many letter combinations are impossible in a given language. Such information can greatly constrain the recognition process, and some OCR systems allow it to be provided by the user.

Checking

The next stage of OCR is manual checking of the output. The recognized page 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 allow some of this information to be suppressed. The original image itself will be displayed for the user's convenience, perhaps with an auxiliary magnification window that zooms in on the region in question. An interactive dialog, similar to that provided by word processors in spell-check mode, focuses on each error and allows the user to ignore this instance, ignore all instances, correct the word, or add it

to the dictionary as a new word. Other options allow you 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 such as italics and bold, margins, indentation, table operations, and so on. Ideally, general word-processor options will be offered within the same package, to save having to alternate between the OCR program and a standard word processor.

Saving

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 such things as 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 Chapter 4), 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 in the first place and consider some practical issues. Physically handling the pages is easiest if you can “disbind” the books by cutting off their bindings; obviously this destroys the source material and is only possible when spare copies exist. At the other extreme, source material can be unique and fragile, and specialist handling is essential to prevent its destruction. For example, most books produced between 1850 and 1950 were printed on paper made from acid-process wood pulp, 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. (We return to this in Chapter 9.)

Sometimes the source material has already been collected on microfiche or microfilm, and the expense associated with manual paper handling can be avoided

by digitizing these forms directly. Although microfilm cameras are capable of recording at very high resolution, quality is inevitably compromised because an additional generation of reproduction is interposed; 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 using a slightly inferior microfilm to reduce paper handling at the expense of more labor-intensive OCR; perhaps not.

Microfiche is more difficult to work with than microfilm, since 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 be skewed, that is, slightly rotated from their correct orientation on the scanning platen, and a deskewing operation may have to be applied. This can be done either manually or automatically. It may be necessary to split double-page spreads into single-page images; again this may be manual or automatic. 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. As a rough ballpark estimate, you can expect to pay \$1 to \$2 per page for scanning and OCR if the material is in a form that can easily be handled (e.g., books whose bindings can be removed), the text is clear and problem-free, there are few images and tables that need to be handled manually, and you have a significant volume of material. If difficulties arise, costs increase to many dollars per page. Companies that perform image digitization often contract the labor-intensive parts of the process to specialists in other countries.

Using a third-party service bureau eliminates the need for you to become a state-of-the-art expert in image digitization and OCR. However, it will be necessary for you to set standards for the project and ensure that they are adhered to.

Most of the factors that affect image digitization can only be evaluated by practical tests. You should arrange for samples of the material to be scanned and

OCR'd by competing commercial organizations and 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.

Once sample pages have been scanned and OCR'd, you might consider building a small digital library prototype that will allow others to assess the look and feel of the planned collection. This is often a good way to drum up support by getting others excited about the project.

Quality control of the scanned images is obviously an important concern in any image digitization project. The obvious way is to load the images into your system as soon as they arrive from the vendor and check them for acceptable clarity and skew. Images that are rejected are then 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

Being labor-intensive, OCR work is often outsourced from the Western world to developing countries such as India, the Philippines, and Romania. In 1999 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 they were occupied by operators who were clearly working with intense concentration. There are two shifts a day, with about a dozen people in each shift and two supervisors—25 employees in all.

Most of the workers are university students and are delighted to have this kind of employment—it compares well with the alternatives available in their town. Pay is by results, not by the hour—and this is quite evident as soon as you walk into the shop and see how hard people work! In effect, they regard their shift at the terminal as an opportunity to earn money, and they make 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 that are absent in subsequent ones.

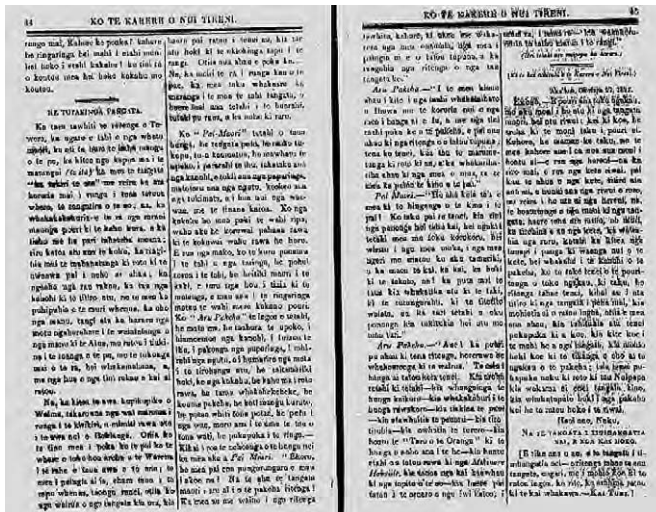
The principal output formats are Microsoft Word and HTML. Again, the latest release of Word is not necessarily the one that is used—obsolete versions have advantages for certain operations. A standalone program is used for converting Word documents to HTML because it greatly outperforms Word's built-in facility. These people 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 written by hand, although they do use an 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 Word Basic to write macros. Although Photoshop is used extensively 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.

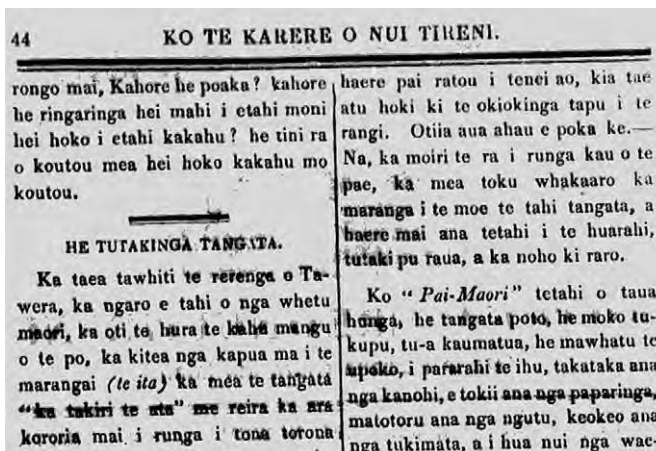
These people have a wealth of detailed knowledge about the operation of different versions of the software packages they use, and they keep their finger on the pulse as new releases emerge. But perhaps their chief asset is their set of in-house procedures for dividing up work, monitoring its progress, and checking the quality of the result. An accuracy of around 99.99 percent is claimed 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 throughput was around 50,000 pages/month, although 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

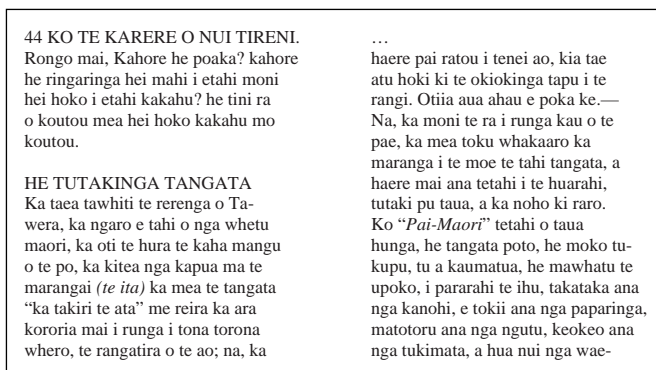
In the New Zealand Digital Library we undertook a project to put a collection of historical New Zealand Māori newspapers on the Web, in fully indexed and searchable form. There were about 20,000 original images, most of them double-page spreads. Figure 2.3 shows an example image, an enlarged version of the beginning, and some of the text captured using OCR. This particular image is a difficult one 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.



(a)



(b)



(c)

Figure 2.3 (a) Double-page spread of a Māori newspaper;
(b) enlarged version; (c) OCR text.

The first task was to scan the images into digital form. Gathering together paper copies of the newspapers would have been a massive undertaking, for the collection comprises 40 different newspaper titles which 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.

Having settled on the image source, the quality of scanning depends on scanning resolution and the number of gray levels or colors. These factors also determine how much storage is required for the information. After conducting some tests, we determined that a resolution corresponding to approximately 300 dpi on the original printed newspaper was adequate for the OCR process. Higher resolutions yielded no noticeable improvement in recognition accuracy. We also found that OCR results from a good black-and-white image were 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 to our specifications by a commercial organization. Because we supplied the images on 35-mm film, the scanning could be automated and proceeded reasonably quickly. We asked for both black-and-white and grayscale images to be generated at the same time to save costs, although it was still not clear whether we would be using both forms. The black-and-white images for the entire collection were returned on eight CD-ROMs; the grayscale images occupied approximately 90 CD-ROMs.

Once the images had been scanned, the OCR process began. Our first attempts used Omnipage, a widely used proprietary OCR package. But we encountered a problem: this software is language-based and insists on utilizing one of its known languages to assist the recognition process. Because our 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.² The result was that recognition accuracies of not much more than 95 percent

-
2. In previous versions of Omnipage one can subvert the language-dependent correction by simply deleting the dictionary file, and we know of one commercial OCR organization that uses an obsolete version for precisely this reason.

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 of our sample pages with impressive results. The same firm offers a bureau service and was prepared to undertake the basic OCR form for only \$0.16 per page (plus a \$500 setup fee). Unfortunately this did not include verification, which we had identified as being the most critical and time-consuming part of the process—partly because of the Māori language material.

Eventually we did locate a reasonably inexpensive software package that had high accuracy and allowed us to establish our own language dictionary. We determined to undertake the OCR process in house. This proved to be an excellent decision, and we would certainly go this route again. 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. We produced our own software for segmentation and skew detection and used a semiautomated procedure in which the system displayed segmented and deskewed pages for approval by a human operator.

2.5 Notes and sources

A useful source of information on criteria for selecting material for digitization is de Stefano (2000), who is specifically concerned with digital conversion. The problem of selecting for preservation raises similar issues, described by Atkinson (1986) in the predigital era. The six principles for the development of library collections are Atkinson's.

McCallum et al. (2000) describe methods for using machine learning techniques to automate the construction of Internet portals, that is, virtual libraries. Their techniques, which are still under development, help to automate the creation and maintenance of domain-specific virtual libraries. As an example, a virtual library of computer science research papers is available on the Web at www.cora.justresearch.com.

The term *virtual library* was characterized in 1993 as “remote access to the contents and services of libraries and other information resources, combining an on-site collection of current and heavily used materials in both print and electronic form, with an electronic network which provides access to, and delivery

from, external worldwide library and commercial information and knowledge sources” (Gapen, 1993, p. 1). The pioneering INFOMINE project is described by Mason et al. (2000), an inspiring paper from which much of our information about virtual libraries was taken. Begun in January 1994, INFOMINE now provides organized and annotated links to over 20,000 scholarly and educational Internet resources, all selected and described by professional librarians. This project shows how librarians and librarian-designed finding tools can play a welcome role in making the Web a more useful environment for researchers and students.

The person who first formulated the objectives of a bibliographic system was Charles Ammi Cutter, an outstanding late Victorian library systematizer (Cutter, 1876). He was a great champion of users who astonished dyed-in-the-wool librarians with radical opinions such as “the convenience of the user must be put before the ease of the cataloger.” According to Svenonius (2000), he practiced what he preached, rejecting the traditional European classified catalog, designed for scholars, in favor of a new alphabetic subject approach more suitable for the person in the street.

Section 2.2 on bibliographic organization has been strongly influenced by two classic works: *The Intellectual Foundations of Information Organization* by Svenonius (2000), and *Library Research Models* by Mann (1993). It is difficult, perhaps, to make books on library science racy, but these come as close as one is ever likely to find. The five objectives of a bibliographic system are from Svenonius. Mann’s book has already been cited in Chapter 1 as a wonderful source of information on libraries and librarianship.

Development of the Library of Congress Subject Headings began in 1898, and the first edition came out in 1909. A recent edition of the big red books was published in 1998 (Library of Congress, 1998).

A good source of further information to help stimulate thought on the basic structure and parameters of your digital library is *The Digital Library Toolkit* from Sun Microsystems (2000). This provides a series of questions that people undertaking a digital library construction project should consider. It also raises issues that affect the planning and implementation of a digital library, summarizes a host of technological resources (with Web pointers), and briefly reviews existing digital library systems.

Steganography takes one piece of information and hides it in another. Digital watermarking is a kind of steganography where what is hidden is a trademark or identification code. Brassil et al. (1994) wrote an early article on watermarking textual images; Katzenbeisser and Petitcolas (1999) have collected a number of papers on all aspects of digital watermarking; and Cox, Miller, and Bloom (2001) have written a comprehensive reference book on the subject.

High-performance OCR products are invariably proprietary: we know of no public-domain systems that attain a level of performance comparable to com-

monly used proprietary ones. However, at least two promising projects are underway. One is GOCR (for “Gnu OCR”), which aims to produce an advanced open-source OCR system; its current status is available at <http://jocr.sourceforge.net>. Another is Clara OCR, which is intended for large-scale digitization projects and runs under X Windows; it is available at www.claraocr.org/.

The interactive OCR facilities described in Section 2.4 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* (www.sw.ro), a well-organized and very successful private company that specializes in high-volume work for international and nongovernment organizations.

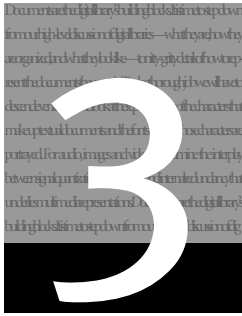
The Māori language has fifteen sounds: the five vowels *a*, *e*, *i*, *o*, and *u*, and ten consonant sounds written *h*, *k*, *m*, *n*, *p*, *r*, *t*, *w*, *ng*, and *wh*. Thus the language is written using fifteen different letters. The first eight consonant sounds are pronounced as they are in English; 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 in German-speaking countries. Indeed in August 2000 Germany’s leading daily newspaper, the *Frankfurter Allgemeine Zeitung*, returned to traditional German spelling. Acting on its own and virtually alone among Germany’s major newspapers, *FAZ* suddenly announced that it was throwing out the new spelling and returning to the previous rules. Today there are ever-increasing calls for a “reform of the reform.”

TWAIN is an image-capture application programming interface, originally released in 1992 for the Microsoft Windows and Apple Macintosh operating systems, which is typically used as an interface between image processing software and a scanner or digital camera. The TWAIN Working Group, an organization that represents the imaging industry, can be found at www.twain.org. According to *The Free On-Line Dictionary of Computing* (at www.foldoc.org), the name comes from the phrase “and never the twain shall meet” in Kipling’s *The Ballad of East and West*. It reflects the difficulty, at the time, of connecting scanners and personal computers. On being uppercased to TWAIN to make it more distinctive, people incorrectly began to assume that it was an acronym. There is

no official interpretation, but the phrase “Technology Without An Interesting Name” continues to haunt the standard.

The design and construction of the “Niupepa” (the Māori word for “newspapers”) collection of historical New Zealand newspapers sketched at the end of Section 2.4 is given by Keegan et al. (2001). A more accessible synopsis by Apperley et al. (2001) is available, while Apperley et al. (in press) give a comprehensive description. 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.



Presentation

User interfaces

How do you build a digital library? Where do you start? How do you *explain* how to build a digital library? “Begin at the beginning,” the King of Hearts said gravely, “and go on till you come to the end: then stop” (Figure 3.1). But we will ignore his advice and begin our story at the end: what you might expect the final library system to look like. Then in the next two chapters we jump to the beginning and look at the forms in which the library material might be provided. Chapters 6 and 7 fill in the middle by explaining how it’s all done. Despite the Red King’s words, this is quite logical. It corresponds to looking first at the goal, what you hope to achieve; then at the starting point, what you have to work with; and finally in between, how you get from where you are now to where you want to be.

Our *digital library* definition from Chapter 1 begins

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

and a good place to start is with the objects themselves. We will mostly be concerned with textual objects—we call them *documents*—and how they appear on the user’s screen. The next section illustrates how different documents can appear within a digital library system. There are many possibilities, and we include just a smattering: structured text documents, unstructured text documents, page images, page images along with the accompanying text, speech

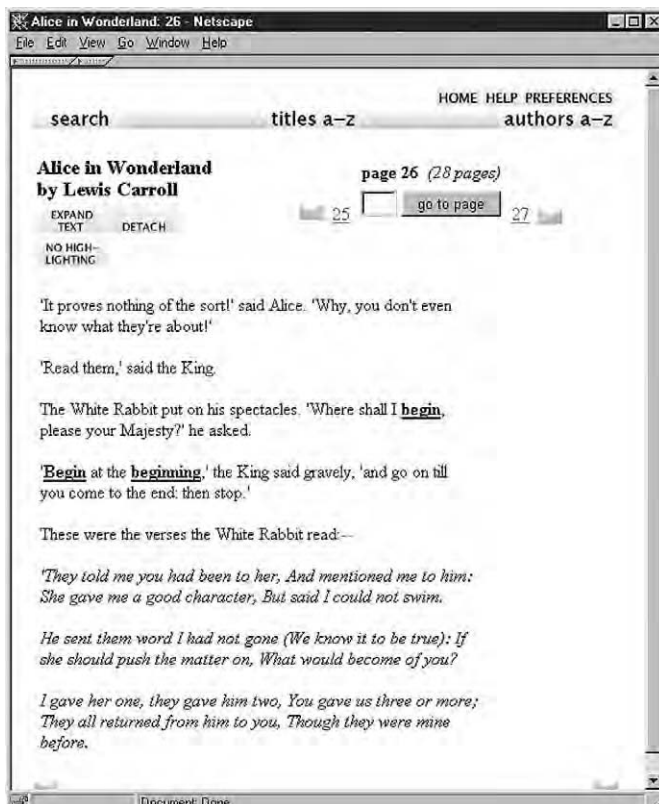


Figure 3.1 Finding a quotation in *Alice's Adventures in Wonderland*.

audio, photographic pictures, and videos. A digital library might include “non-standard” information such as music, which has many different representations—synthesized audio, music notation, page images, recorded MIDI performances, recorded audio performances. We also take the opportunity, when surveying examples of documents, to illustrate the wide range of uses to which digital libraries are being put.

In addition to documents, digital libraries include metadata of the kind used conventionally for bibliographic organization as discussed in Section 2.2, although metadata potentially involves a wider range of information and formats. We next examine some examples of metadata display, which will help convey the idea of what kind of metadata might accompany a digital library collection.

The definition goes on to say

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

The second part of this chapter illustrates different methods for access and retrieval. Conventionally these are divided into *searching* and *browsing*, although in truth the distinction is not a sharp one. We first examine interfaces that allow you to locate words and phrases in the full text of the document collection. Searching is also useful for metadata—such as finding words in title and author fields—and we look at digital library 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, such as titles, authors, dates, and subject hierarchies, and show how these relate to the structure that is implicit within the metadata itself.

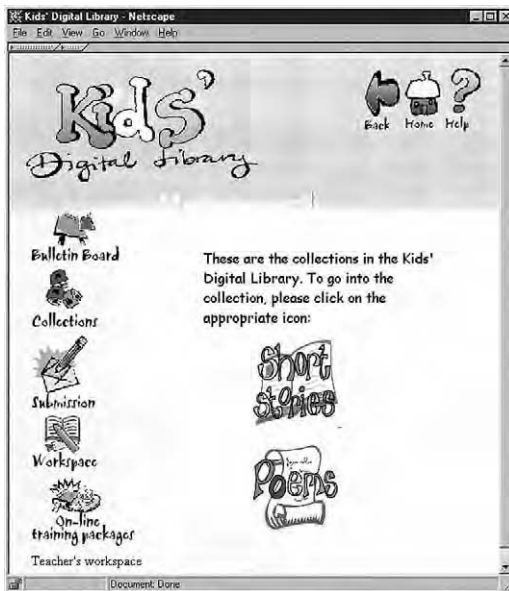
But searching and browsing are not really different activities: they are different parts of a spectrum. In practice people want to be able to interact with information collections in different ways, some involving searching for particular words, some involving convenient browsing—perhaps browsing the results of searching. We examine two schemes that effectively combine searching and browsing. Both are based on phrases extracted automatically from the documents in a digital library: one on an enormous—almost exhaustive—set of phrases that appear in the text; the other on carefully selected sets of key phrases for each document.

The final part of the definition is

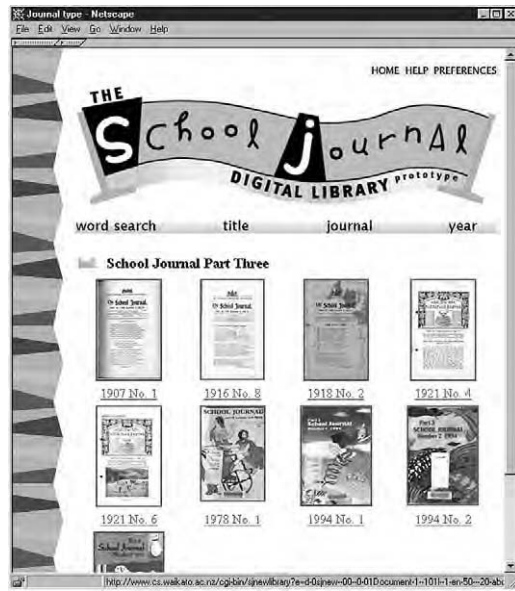
... and for selection, organization, and maintenance of the collection.

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

Figure 3.1, like most of the illustrations here, shows a screen shot from a Web browser. However, the browser does not show a static Web page: the digital library software has constructed this page dynamically at the time it was called up for display. The navigation bar at the top, the book title at the top left, the buttons underneath, and the page selector at the top right are all composed together with the text at display time, every time the page is accessed. The information is actually stored in compressed form in a textual database system. If you look on the computer that is the Web server for this digital library, you will not find this page stored there. Likewise in the next examples, in Figure 3.2, these pages are also put together, along with all images and links, at the time requested—not before. And in Figures 3.3a and b, the cover picture at the top left and the table of contents at the top right are stored internally in such a way that they can be accessed, and searched, and browsed, independently of the text.



(a)



(b)

Figure 3.2 Different-looking digital libraries: (a) Kids' Digital Library (Middlesex University, London, England); (b) School Journal Digital Library (Learning Media Limited, Wellington, New Zealand).

This is why in Chapter 1 we distinguished 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.

The general look and feel of a digital library system can easily be altered by changing the way the screen is laid out and the icons used on it. Figure 3.2a shows the front page of the Kids' Digital Library, which uses a hand-printed font, chunky, hand-drawn icons, and bright pastel shades to promote a feeling of friendliness and informality—contrasting strongly with the austere, businesslike image conveyed by Figure 3.1. Figure 3.2b shows a page of a collection taken from a School Journal (we will return to this collection and give more information about it shortly): again, colorful pictures are available and are used wherever possible in place of textual descriptions, and the logo at the top is designed to help communicate the intended feeling. Exactly the same underlying technology can be used to support many different styles of digital library. Because all Web 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—immediately, at runtime, without regenerating or even touching the content of the collection. None of the underlying information need be altered, just the output processing routines.

3.1 Presenting 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?

Hierarchically structured documents

Figure 3.3 shows a book in the Humanity Development Library entitled *Village Level Brickmaking*. A picture of the front cover is displayed as a graphic on the left of Figure 3.3a, and the table of contents appears to its right. Below this is the start of the main text, which begins with title, author, and publisher. The books in this collection are generously illustrated: many pictures are included in the text. On the screen these images appear in-line, just as they did in the paper books from which the collection was derived. Figures 3.3c and d show some of the images, obtained by scrolling down from Figure 3.3b.

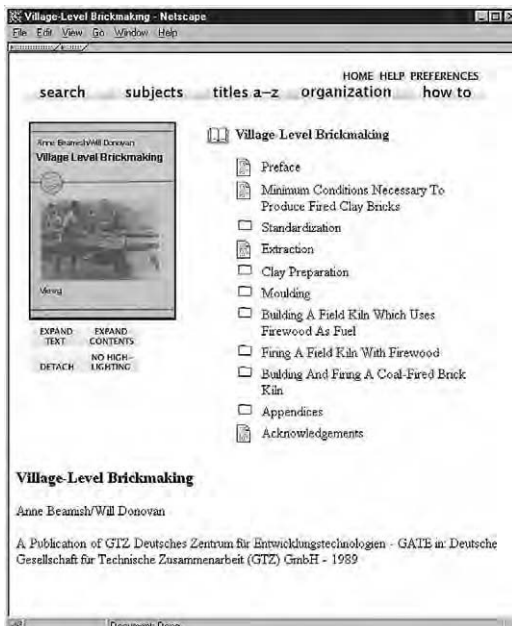
All books in this collection have front-cover images, and the appropriate image always appears at the top of any page where a book, or part of a book, is displayed. This ever-present picture gives a feeling of physical presence, a constant reminder of the context in which you are reading. The user interface look and feel may be a poor substitute for the real look and feel of 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 book-mark—but it's a lot better than nothing.

The books in the Humanity Development Library are structured into sections and subsections. The small folder icons in Figure 3.3a 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.

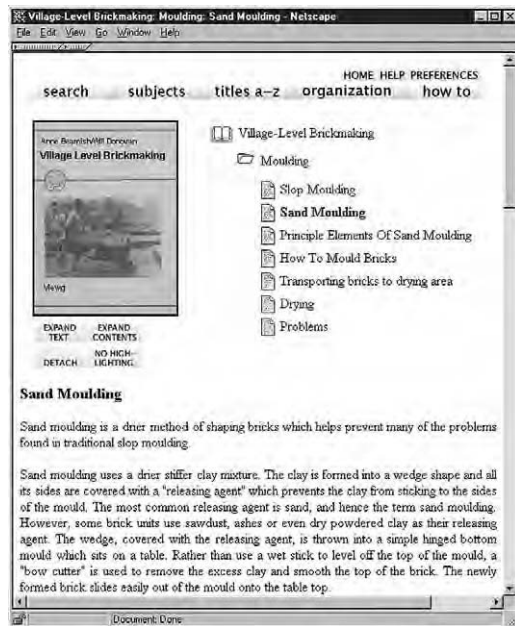
Clicking on *Moulding* in Figure 3.3a produces the page in Figure 3.3b, which shows the chapter's structure in the form of a table of contents of its sections. Here the user has opened the book to the *Sand moulding* section by clicking on its text-page icon; that section heading is shown in bold and its text appears below. By clicking on other headings the reader can learn about such topics as *Slop moulding*, *How to mould bricks*, and *Drying*.

You can read the beginning of the *Sand moulding* section in Figure 3.3b: the scroll bar to the right of the screen indicates that there is more text underneath. Figures 3.3c and d show the effect of scrolling further down the same page.

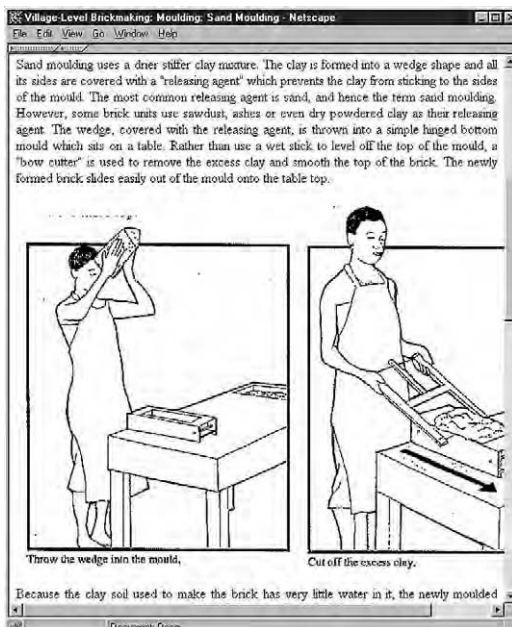
Sometimes you want to see the full table of contents, with all chapters and their sections and subsections included. Sometimes you want to see the text of



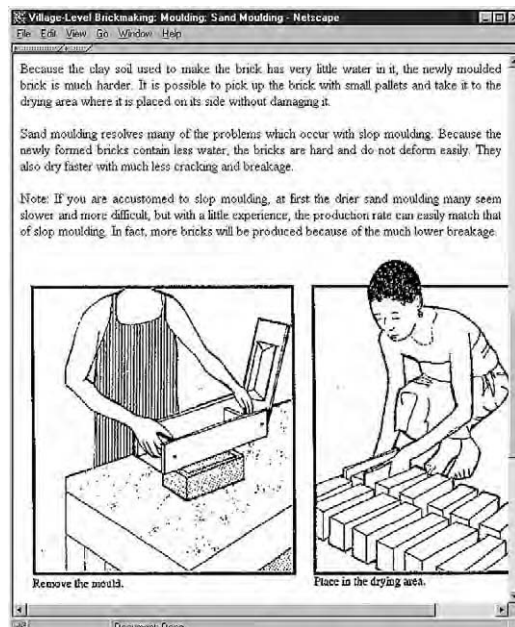
(a)



(b)



(c)



(d)

Figure 3.3 *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.

the whole book—apart from getting a complete view, a common reason for doing this is in order to print it out. In Figure 3.3 the button labeled “Expand contents” expands the table of contents into its full hierarchical structure. Likewise the Expand Text button expands the text of the section being displayed. If we pressed Expand Text in Figure 3.3a we would get the text of the entire book, including all chapters and subsections; if we pressed it in Figure 3.3b we would get the complete text of the *Moulding* chapter, including all subsections. Finally, the Detach button duplicates this window on the screen, so that you can leave the text in it while you go and look at another part of the library in the other window—invaluable when comparing multiple documents.

The Humanity Development Library is a large compendium of practical material that 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. It contains 1,230 publications—books, reports, and magazines—totaling 160,000 pages, which as noted in Chapter 1 would weigh 340 kg in print form and occupy a small library bookstack.

This material was carefully selected and put together by a dedicated collection editor, who acquired the books, arranged for permission to include each one, organized a massive 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 3.3c 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 this one.

Plain, unstructured text documents

Figure 3.4, and Figure 3.1 earlier, show screen shots from a far plainer collection, a set of documents that have been treated as unstructured text. There is no hierarchical structure here—at least none that is known to the digital library system. Neither are there front-cover images. In place of Figure 3.3’s cover picture and table of contents at the top of each page, Figure 3.4 shows a more prosaic display: 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



Figure 3.4 *Alice's Adventures in Wonderland*.

made by the computer every few hundred lines—that's why *Alice's Adventures in Wonderland* appears to have only 28 pages! The only reason for having pagination at all is to prevent your Web browser from downloading the entire book every time you look at it.

In fact, this book *does* have chapters—in Figure 3.4 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. At some extra cost in effort when setting up the collection, it would have been possible to identify the beginning of each chapter, and its title, and incorporate this information into the library system to permit more convenient chapter-by-chapter browsing—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 structure, or any given book, it is easy to do; but in real life any large collection (of, say, thousands of books) will exhibit considerable variation in format. As we mentioned before, the task of proof-reading thousands of books is not to be undertaken lightly!

Another feature of this collection is that the books are stored as raw ASCII text, with the end of each line hard-coded in the document, rather than (say) HTML. That is why the lines of text in Figure 3.4 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 the end-of-line codes would be easy for the text visible in Figure 3.4, but a simple removal algorithm would destroy the line breaks in 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 3.1 shows another feature: 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 3.3), to find the quotation with which this chapter begins. This digital library system highlights search terms in (almost) every collection: 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 target documents—partly because they do not store the target documents but instead direct the user to the document held at its original source location.

Alice's Adventures in Wonderland is a book in the Gutenberg collection. The goal of Project Gutenberg is to encourage the creation and distribution of electronic text. Although conceived in 1971 with the exceedingly ambitious aim of a trillion electronic literature files by the year 2001, work did not begin in earnest until 1991, and the aim was scaled back to 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 was never released because of copyright restrictions. The growth rate of 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; at this rate the final goal should have been reached in 2001. At the time of writing the project was a little behind schedule, with nearly 4,500 books entered by the end of 2001 and a rate of increase of perhaps 100 per month.

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 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 serious problem. However, for a vision that dates back more than two decades before the advent of the World Wide Web, Gutenberg is remarkably farsighted and gives an interesting perspective on the potential role of volunteer labor in placing society's literary treasures in the public domain.

In keeping with the Project Gutenberg spirit, little effort has been made to “pretty up” this digital library collection. It represents the opposite end of the spectrum to the Humanity Development Library. It took a few hours for a person to download the Project Gutenberg files and create the collection and a few

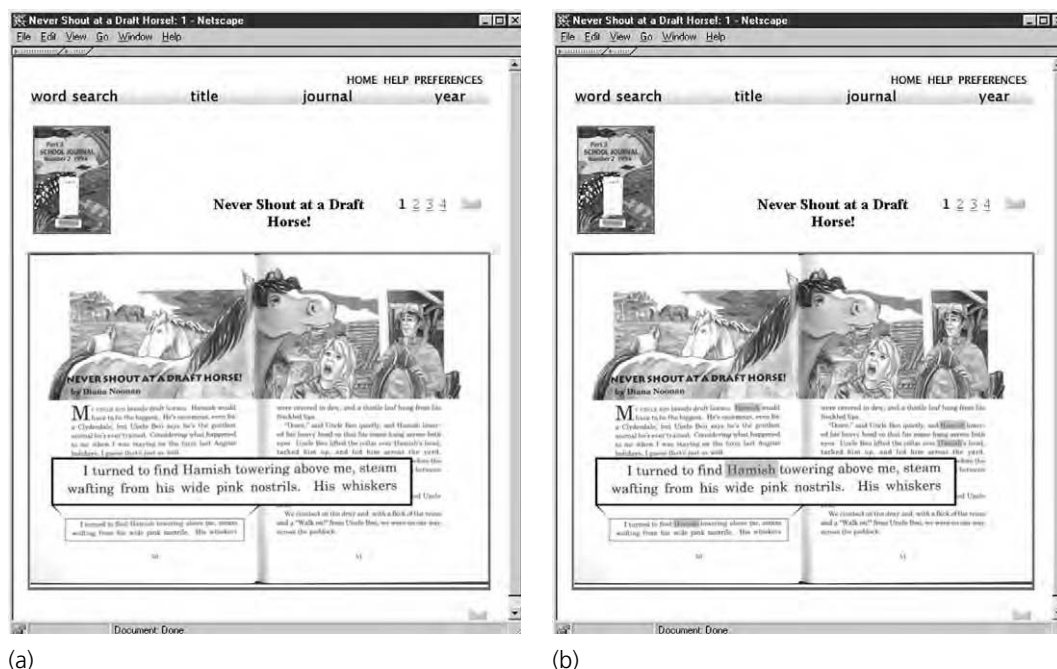


Figure 3.5 A story from the School Journal collection: (a) “Never Shout at a Draft Horse!”; (b) with search term highlighted (mock-up). “Never Shout at a Draft Horse” by Diana Noonan. New Zealand School Journal, 1994, Part 3, No. 2.

hours of computer time to build it. Despite the tiny amount of effort spent constructing it, 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

The page of the children’s story “Never Shout at a Draft Horse!” shown in Figure 3.5 is represented not as text but as a facsimile of the original printed version. This example is taken from a collection of literature written for children, the New Zealand School Journal. The collection’s designer decided to show digitized images of the books’ pages rather than text extracted from them. From a technical point of view there is a big difference: a textual representation generally occupies only about one-twentieth as much storage space as a page image,

greatly reducing the space required to store the collection and the time needed to download each page. However, the picture of the horse would have to be represented as an image, just as the pictures are in Figures 3.3c and d, sacrificing some of the storage space gained.

One good reason for showing page images rather than extracted text in this collection is because the optical character recognition (OCR) process that identifies and recognizes the text content of page images is inevitably error-prone. When schoolchildren are the target audience, special care must be taken not to expose them to erroneous text. Of course errors can be detected and corrected manually, as in the Humanity Development Library, 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 the reader never sees 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 was misrecognized, then a search for that word will not return that page. It is also possible that a word on the page is misinterpreted as a different word, in which case a search for *that* word will incorrectly return the page, in addition to other pages on which the word really does appear. However, neither of these errors was seen as a big problem—certainly not so serious as showing children corrupted text.

Figure 3.5 shows the journal cover at the top left and a page selector at the right that lets you browse around the story more conveniently than the numeric selector in Figure 3.2. These stories are short: “Never Shout at a Draft Horse” (Figure 3.5) has only four pages.

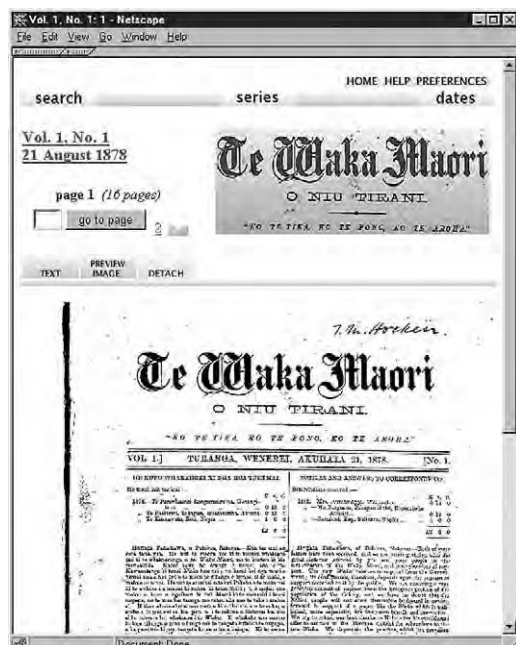
A disadvantage of showing page images is that it is hard to find search terms on the page. In Figure 3.1 the terms *begin* and *beginning* are in boldface. While it is easy to highlight words in a page of text, it is more difficult to highlight them in a page *image*. It is not impossible—Figure 3.5b shows the same page with the word *Hamish* highlighted as though it had been marked with a yellow marker pen. It looks impressive, and indeed it is not difficult to process the page image in this way to simulate yellow highlighting. However, it is necessary to find out exactly where the word occurs in the page in order to know what areas to highlight. It is difficult to infer from the page image the information required to do this, although some OCR systems generate information on word positions.

The *School Journal* is a magazine for New Zealand schoolchildren, delivered free to schools throughout the nation by the Ministry of Education. Its purpose is to provide quality writing that is relevant to the needs and interests of New Zealand children, and the real aim is to foster a love of reading. The material it contains is widely used throughout the school curriculum. Not only is it used for teaching reading, but also for social studies, science, and many other subjects.

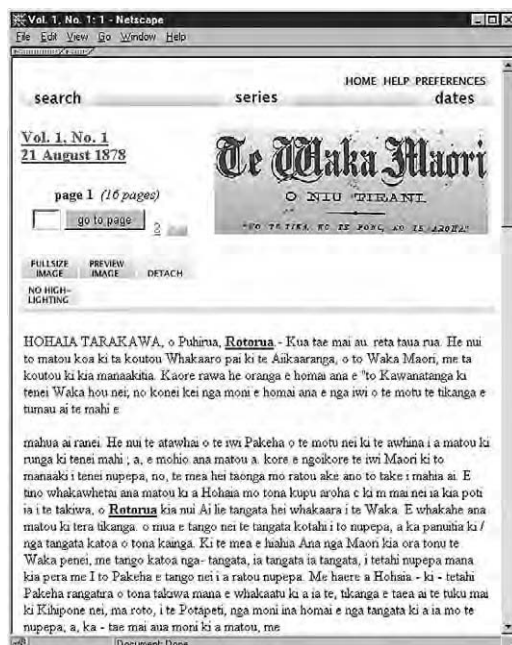
The magazine originated over 90 years ago, and the cover of the very first issue is the top left image in Figure 3.2b. Like most children's literature of the time, the content of early issues was based around conveying attitudes and values of society at large. During the 1930s the journal began to encourage children's intellectual curiosity by relating the material to the student's environment. This collection of a small sample of School Journals gives a fascinating historical perspective on the development of attitudes toward children's education—and indeed the development of society in general—throughout the 20th century.

Page images and extracted text

While readers of the School Journal digital library collection see page images only, in other situations it is useful to provide a facility for viewing the extracted text as well. Figure 3.6 shows an example from a collection of historical New Zealand Māori newspapers, in both 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. Because it is the text version that supports searching, this information is not visible 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.



(a)



(b)

Figure 3.6 A historic Māori newspaper: (a) page image; (b) extracted text.

Both forms are provided in this collection because the advantage of being able to locate search terms in these rather long pages was deemed to outweigh any damage that might be wrought by showing readers incorrect versions of the text. The reader can choose whether to view text or image and can move quickly from one to the other. A magnified version of each page image is also available, should it be required—this facility is provided in most image collections.

The Māori newspapers record fascinating historical information that is useful from many points of view. They 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, for 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, though 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.

Construction and distribution of the microfiche was an enormous effort and was undertaken to make this national treasure available to scholars. However, the material was entirely unindexed. Although a research project has begun that is producing 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 over the Web, which makes them readily accessible to a far wider audience. Second, they can be fully searched, which makes almost any kind of research immeasurably easier. Third, the collection does not require any research skills to use, so that ordinary people can discover things that they would never otherwise know about their heritage, ancestry, or home town.

Audio and photographic images

Some years ago the public library in the small New Zealand town where we live began a project to collect local history. Concerned that knowledge of what it was like to grow up here in the 1930s, 1940s, and 1950s would soon be permanently lost, they decided 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 interviewer developed a brief typewritten

summary of each interview, dividing it into sections representing themes or events covered in the interview. And then the collection sat in a cardboard box behind the library's circulation desk, largely unused.

Recently all the tapes and photos were digitized and made into a digital library collection, along with the summaries. Figure 3.7 shows it in use. The small control panel in the center is being used to replay a particular recording, using a standard software audio-player that sports regular tape-recorder functions—pause, fast forward, and so on. You don't have to wait until the whole audio file is downloaded: the software starts playing the beginning of the file 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

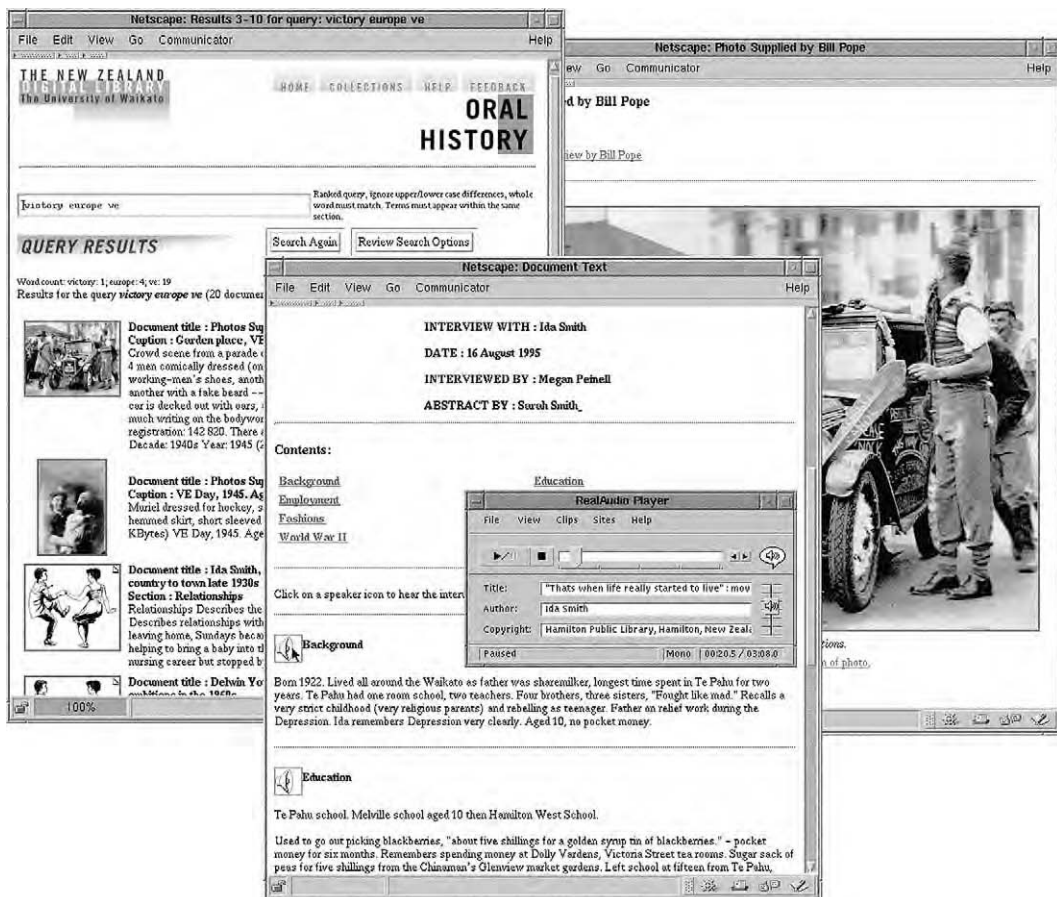


Figure 3.7 Listening to a tape from the Oral History collection. Hamilton Public Library, Hamilton, New Zealand.

case of the celebrations in our town on VE Day 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 interview; interviews also can be played back in full using buttons at the top of the page (not visible in Figure 3.7). When the tapes were digitized, timings were generated for the beginning and end of each section. Flipping through a tape in this way, scanning a brief textual synopsis and clicking on interesting parts to hear them, is a far more engaging and productive activity than trying to scan an audiotape with a finger on the fast-forward button.

It is the contents of the interview pages that are used for text searching. Although they do not contain a full transcript of the interview, most keywords that you might want to search on are included. In addition, brief descriptions of each photograph were entered, and these 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—a so-called controlled vocabulary—were not used. Nevertheless it seems to be easy for people to find material of interest.

Imagine the difference in access between a box of cassette tapes at the library's circulation desk and the fully indexed, Web-accessible digital library collection depicted in Figure 3.7. Text searching makes it easy to find out what it was like at the end of the war, or to study the development of a particular neighborhood, or to see if certain people are mentioned—and you can actually hear senior citizens reminisce about these things. Casual inquiries and browsing immediately become 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 tape player. In fact, although this collection can be accessed from anywhere on the Web, the audio files are only available to users on terminals in the local public library because when the interviews were made the subjects were not asked for consent to distribute their voices worldwide. There is a message here for those engaged in local history projects—think big!

Video

Including videos as documents in digital libraries is just as easy as including audio or photographic images. Web browsers, suitably equipped with plug-ins, are capable of playing video in a variety of formats. Of course a great deal of storage space will be required for a large collection of videos. And the feasibility of downloading video material over the Internet depends on numerous technical factors, such as the bandwidth of the connection.

Just as the oral history audiotapes are accessed through textual summaries of the interviews and descriptions of the photographs, so with videos it is possible to supply appropriate descriptive text so that they can be located. Summaries and reviews are readily available and provide a good start.

Music

As Chapter 1 mentioned, digital collections of music have the potential to capture popular imagination in ways that more scholarly libraries will never do. Figure 3.8 shows the process of trying to find the tune *Auld Lang Syne*. The player in the front window is controlling playback of audio generated by a music synthesizer program from an internal representation of the music. Also visible is the musical notation for the tune, which is generated from the same internal representation, this time by a music-typesetting program that produces an image suitable for display in a Web browser. In this collection the same internal representation supports musical searching: we return to this in Chapter 9.



Figure 3.8 Finding *Auld Lang Syne* in a digital music library.

The music representation for this collection was produced by an optical music recognition (OMR) program—similar to an OCR program but working in the domain of printed music—from a scanned page of a music book that includes the tune. Not shown in Figure 3.8, 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. In other music collections it is easy to locate and listen to prerecorded renditions of the tune that people have keyed into their computer and stored in the widely used MIDI (musical instrument digital interface) format, which is a standard adopted by the electronic music industry for recording and controlling electronic instruments. It is even possible to click through to music sites that contain actual recordings and play those too.

The twin keys to creating a rich digital library music collection that is interesting and entertaining to search and browse are (1) being able to convert between the different forms that music takes, and (2) making resourceful use of the Web to locate relevant information. There are several possible representations of music:

- printed notation
- human-produced MIDI file
- audio replayed from a human MIDI file
- audio synthesized from an internal representation
- audio file representing a human performance
- internal representation, suitable for searching

An internal representation can be generated without difficulty from a human-produced MIDI file, though not from a human audio performance (at least not without excessive difficulty). And it is certainly not possible for a computer to synthesize a human performance! All other conversions are possible, with varying quality of results. For example, using optical music recognition to convert from printed notation to an internal representation is by no means a perfect process; printed music generated from a MIDI file is nowhere near as legible as professionally typeset music.

Foreign languages

We have already seen an example of a foreign-language collection: the Māori newspapers in Figure 3.6. Māori uses the standard Roman character set, although it does include one diacritical mark, a bar that occurs over vowels to lengthen them. In Chapter 5 (Section 5.1) we show how to represent such characters in Web documents.

Figure 3.9 shows interfaces in French and Portuguese. The French illustration is from a UNESCO collection called Sahel Point Doc that contains information

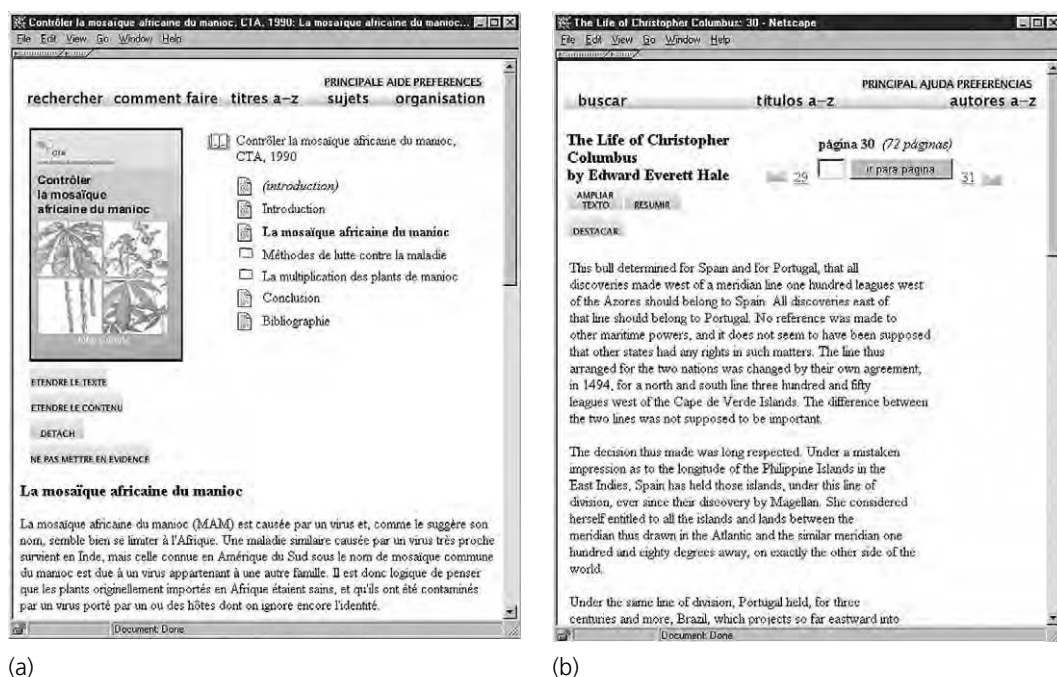


Figure 3.9 Foreign-language collections: (a) French (*Contrôler la mosaïque africaine du manioc* by J. Guthrie, CTA, 1990); (b) Portuguese interface to an English collection (*The Life of Christopher Columbus* by Edward Everett Hale, Electric Umbrella Publishing).

about the Sahel region of sub-Saharan Africa. Everything in this collection is in French: all target documents, the entire user interface, and the help text. Figure 3.9b 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 such as German, Dutch, Spanish, and Māori too.

Figure 3.10 shows documents from two different Chinese collections. The first comes from a collection of rubbings of Tang poetry—not unlike those stone steles described in Chapter 1, the world’s oldest surviving library. These documents are images, not machine-readable text. Like the School Journal collection, 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. Because of the difficulty of locating particular words in the images, search terms are not highlighted in this collection.

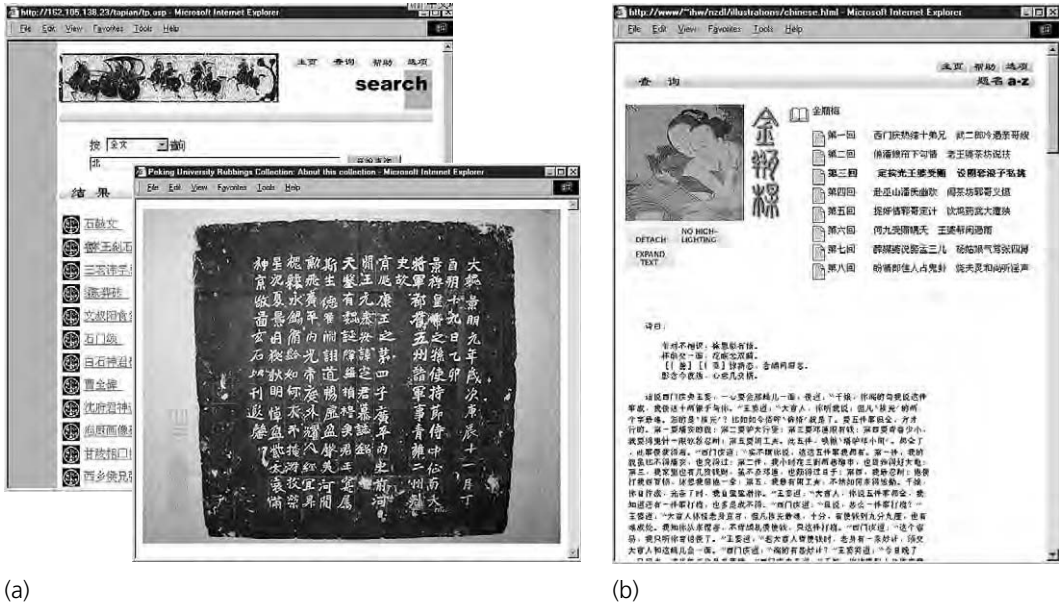


Figure 3.10 Documents from two Chinese collections: (a) rubbings of Tang poetry; (b) classic literature.

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, with chapters shown by folders. The work illustrated is here *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 imbroglis, 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. In this collection, being textual, search terms are highlighted by putting them in bold. Boldface characters (and indeed italics) are used in Chinese just as they are in Western languages.

It's easy to display documents in exotic languages 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 3.11 shows pages from an Arabic collection of information on famous mosques, displayed using an ordinary browser. If your browser does not support a particular character set, you may need to download an appropriate plug-in.

As Figures 3.10a and 3.11b imply, the text of both the Chinese and Arabic collections (as well as all other languages) is fully searchable. Again, the browser does the hard part, facilitating character entry and ensuring that Arabic text is

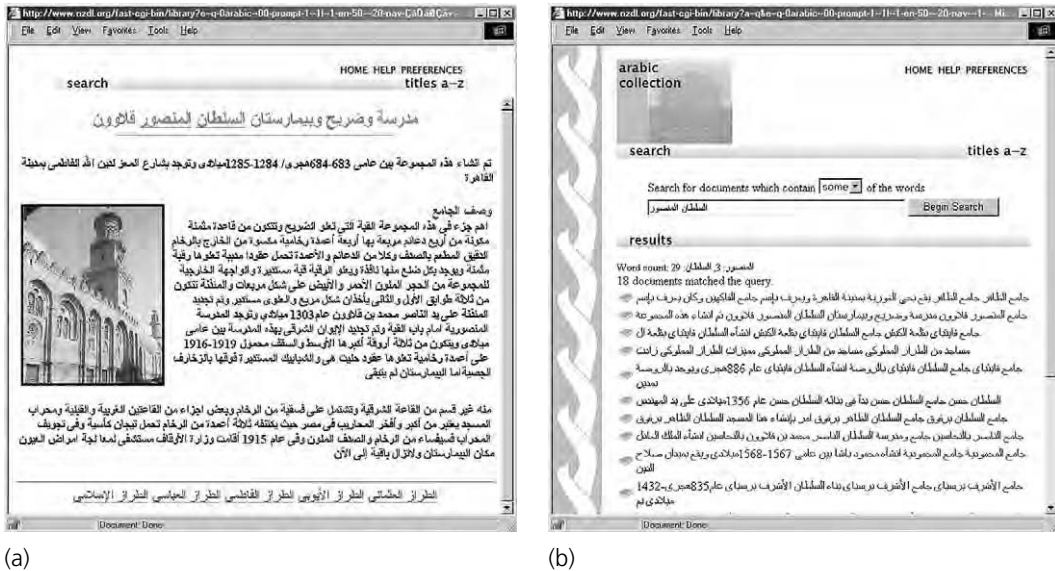


Figure 3.11 An Arabic collection: (a) a document; (b) searching.

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). We discuss how to handle different character sets in Chapter 4 (Section 4.1), while Chapter 5 (Section 5.1) mentions how to embed them in Web documents.

3.2 Presenting metadata

As we saw in Chapter 2, 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 recent term for such information. Metadata is information in a structured format, whose purpose is to provide a description of other data objects to facilitate access to them. Whereas the data objects themselves—the books, say—generally contain information that is not structured, or (as in the Gutenberg collection in Figure 3.4) whose structure is not apparent to the system, 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.

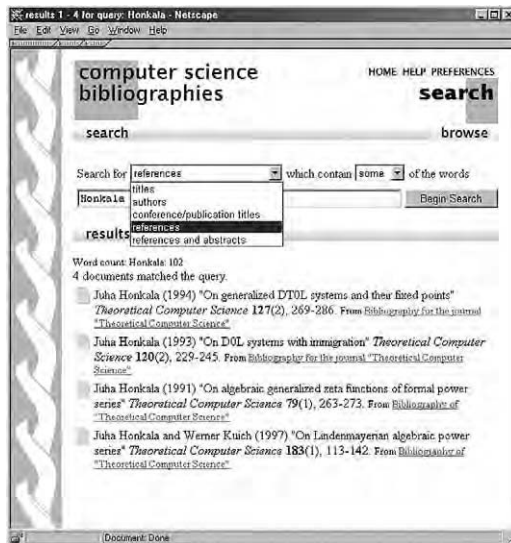


Figure 3.12 Bibliography display.

In the computer's representation of a book it may not be obvious what is the title, the author, the publisher, the source of the original copy, and so on. However, if this information is represented in metadata, it is done so in a standard way using standard elements, so that the computer can identify these fields and operate on them.

Figure 3.12 shows metadata presented as a conventional bibliographic listing. These items have been located in a large collection of computer science references: the hyperlinks at the end of each reference link to the source bibliographies. Many of the bibliographic entries have abstracts, which are viewed by clicking the page icon to the left of each entry. In this case all the icons are grayed out, indicating that no abstracts are available. The metadata here includes title, author, date, the title of the publication in which the article appears, volume number, issue number, and page numbers. These are standard bibliographic metadata items. Also included are the URL of the source bibliography and the abstract—although you may argue whether this is structured enough to really constitute metadata.

Metadata has many different aspects, corresponding to different kinds of information that might be available about an item, that are not included in the abbreviated reference format of Figure 3.12. 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.

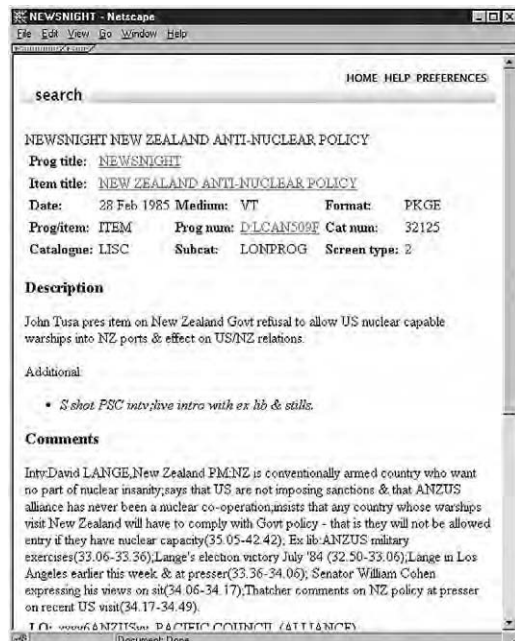
Metadata provides assistance with search and retrieval; gives information about usage in terms of authorization, copyright, or licensing; addresses quality issues such as authentication and rating; and promotes system interoperability.

Figure 3.13a shows a metadata record retrieved over the Internet from the Library of Congress information service and displayed within a simple interface that shows all the fields in the record (only half of which are visible). The more 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 a subfield of the title; the other authors also appear further down the record in separate *Author Note–Name* fields. This metadata was retrieved using an information interchange standard called Z39.50 that is widely used throughout the library world (described in Section 8.5) and is represented in a record format called MARC, for “machine-readable cataloging,” that is also used by libraries internationally (described in Section 5.4).

Library metadata is standardized—although, as is often the case with standards, there are many different ones to choose from. (Indeed MARC itself comes in more than 20 variants, produced for different countries.) Metadata is important in contexts other than bibliographic records, but these areas often lack any



(a)



(b)

Figure 3.13 Metadata examples: (a) bibliography record retrieved from the Library of Congress; (b) description of a BBC television program.

accepted standard. Figure 3.13b shows a record from a BBC catalog of radio and television programs and gives information pertinent to this context—program title, item title, date, medium, format, several internal identifiers, a description, and a comments field. This database also includes many other fields.

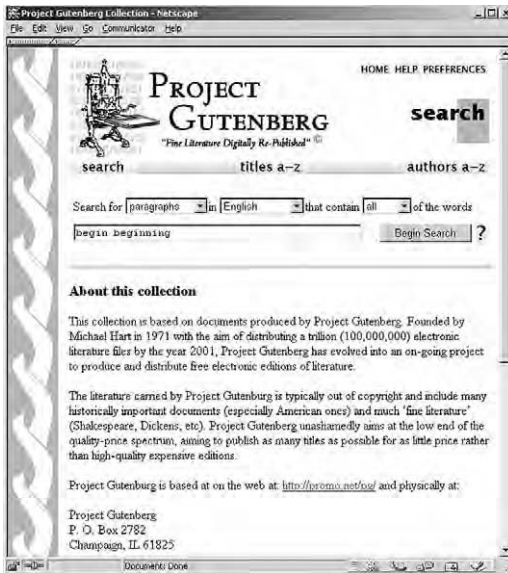
Metadata descriptions often grow willy-nilly, in which case the relatively unstructured technique of text searching becomes a more appropriate way of locating records than the conventional way of searching a structured database with predefined fields. Because of increased interest in communicating information 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.

3.3 Searching

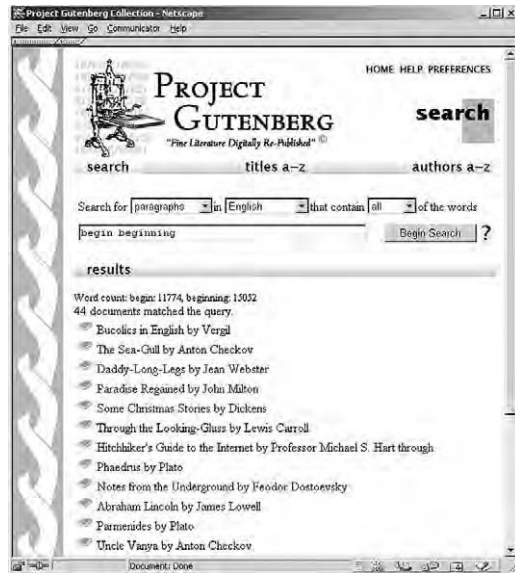
Electronic document delivery is the primary *raison d'être* for most digital libraries. But searching comes a close second—in particular, searching the full text of the documents' contents. Whereas conventional automated library catalog searches are restricted to metadata, digital libraries have access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

Figure 3.14a shows a request for the search that was used to find this chapter's opening quotation: it seeks paragraphs in English that contain both the words *begin* and *beginning*. Figure 3.14b shows the computer's response: a list of documents that contain paragraphs matching the query. These pages provide a deliberately plain, stripped-down, unadorned search mechanism with only rudimentary functionality. In digital libraries—particularly ones targeted at nonacademic users—simple facilities should be available to satisfy the basic things that most people want most of the time. Options may be provided for more advanced interactions, but not at the expense of simplicity for the casual user. As Alan Kay, a leading early proponent of the visually based paradigm of human-computer interaction, said, “Simple things should be simple, complex things should be possible.”

The screens shown in Figure 3.14 allow readers to choose the unit that is searched, the language, and the type of search. The units that can be chosen vary from collection to collection. They typically include such items as *paragraphs*, *sections*, *documents*; also *section titles*, *document titles*, and *authors*. The first group of items involves the full text. The second group is quite different in that it involves metadata elements—but it is unnecessary to point this out explicitly to the user, at least for casual users.



(a)



(b)

Figure 3.14 Searching for a quotation: (a) query page; (b) query response.

What is returned, as shown in Figure 3.14b, is a list of the titles of the documents. Even though the search may be for *paragraphs*, not just the relevant paragraph is returned, but also the enclosing document in its entirety. In this digital library system, the units that the user finally sees on the screen as the result of searching or browsing operations are defined to be the “documents.” If you wanted the system to present paragraphs as individual units, you would need to define paragraphs to be the “documents”—and arrange for paragraphs to include internal links, perhaps to preceding and succeeding ones. This is not hard to do.

In multilingual collections it is useful to restrict searches to a specified language in order to avoid false hits on words in other languages. The type of search, in this interface, can be either *all of the words* (as chosen in Figure 3.14) or *some of the words* (not shown). Technically these are interpreted as a Boolean AND query and a ranked query respectively.

Types of query

In the field of information retrieval, an important distinction is made between *Boolean* and *ranked* queries. Both comprise a list of *terms*—words to be sought in the text. In a Boolean query, terms are combined using the connectives AND, OR, and NOT, and the *answers*, or *responses*, to the query are those units of text

that satisfy the stipulated condition. In a ranked query, the list of terms is treated as a small document in its own right, and units of text that are “similar” to it are sought, ranked in order of the degree of match. It may be more logical to view ranking as a separate operation that can be applied to any kind of query; from this perspective what we are calling a *ranked query* is usually interpreted as an OR query, which seeks documents containing *any* of the specified words, followed by a ranking operation. In both Boolean and ranked retrieval, what is searched may be units of text such as *paragraphs* or metadata elements such as *authors*, whichever the reader chooses. The unit returned is not necessarily the same as the unit searched—in the above examples the entire enclosing document is returned regardless of what search unit is specified.

AND is the most common Boolean query type, and this is how *all of the words* is interpreted in Figure 3.14. A query such as

digital AND library

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

Retrieval systems inevitably return some answers that are not relevant, and users must filter these out manually. There is a difficult choice between casting a broad query to be sure of retrieving all relevant material, albeit diluted with many irrelevant answers, and a narrow one, where most retrieved documents are of interest but others slip through the net because the query is too restrictive. A broad search that identifies virtually all the relevant documents is said to have *high recall*, while one in which virtually all retrieved documents are relevant has *high precision*.

An enduring theme in information retrieval is the tension between these two virtues. When searching you must decide whether you prefer high precision or high recall and formulate your query appropriately. In typical Web searches, for example, precision is generally more sought after than recall. There is so much out there that you probably don’t want to find *every* relevant document—you likely couldn’t handle them all anyway—and you certainly don’t want to have to pay the price of checking through a lot of irrelevant documents. 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 checked *every*

relevant precedent, because the last thing you want is for the prosecutor to spring a nasty surprise in court.

Another problem is that small variations of a query can lead to quite different results. Although you might think the query *electronic AND document AND collection* is similar to *digital AND library*, it is likely to produce a very different answer. 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 access to online information in commercial and scientific applications. However, Internet search engines have taught us that they are not the only way a database can be queried.

Rather than seeking exact Boolean answers, people—particularly nonprofessional and casual users—often prefer 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, a list of terms such as

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.

Identifying documents relevant to a list of terms is not just a matter of converting the 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 answers.

The solution is to use a *ranked query*, which applies some kind of artificial measure that gauges the similarity of each document to the query. Based on this numeric indicator, a fixed number of the closest matching documents are returned as answers. If the measure is good, and only a few documents are returned, they will contain predominantly relevant answers—high precision. If many documents are returned, most of the relevant documents will be included—high recall. In practice, low precision invariably accompanies high recall since many irrelevant documents will almost certainly come to light before the last of the relevant ones appears in the ranking. Conversely, when the precision is high, recall will probably be low, since precision will be high only near the beginning of the ranked list of documents, at which point only a few of the total set of relevant ones will have been encountered.

Great effort has been invested over the years in a quest for similarity measures and other ranking strategies that succeed in keeping both recall and precision reasonably high. Simple techniques just count the number of query terms that appear somewhere in the document: this is often 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. An obvious drawback is that long documents are favored over short ones since by virtue of size alone they are more likely to contain a broader selection of the query terms. Furthermore common terms appearing in the query tend to discriminate unfairly against documents that do not happen to contain them, even ones that match on highly specific ones. For example, a query containing *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*.

Many ranking techniques assign a numeric weight to each term based on its frequency in the document collection. Common terms receive low weight. These techniques 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. It is difficult even to describe the *idea* of ranking to end users in a way that is both succinct and comprehensible. That is why the digital library system illustrated ducks the issue, in the simple form of interface illustrated in Figure 3.14, by mentioning only that answers should match “some of the words.” Hopefully the reader will not be too confused by the responses he or she receives.

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. These are especially appropriate if it is meta-data that is being searched, and particularly if professional catalogers have entered it, for the terms that are used are relatively unambiguous and predictable. Casual users, on the other hand, may not want to grapple with the complex matter of 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. These are especially appropriate if full text is being searched, for term usage is relatively unpredictable.

A compromise between Boolean and ranked queries emerged in early Internet search engines. By default they treated queries as ranked, but allowed users more precise control by indicating certain words that must appear in the text of every answer (usually by a preceding + sign) and others that must not appear (preceded by -). Of course there are many other possibilities, and as the Web

grew and the quest for precision began to dominate recall, some search engines began to return only documents that contained all of the search terms, corresponding to a ranked Boolean AND. An obvious generalization of these ideas is to undertake a full Boolean search and rank the results. And many other schemes have been proposed. Some use nonstandard logic, such as fuzzy logic, instead of the standard Boolean operators; this provides an alternative rationale for ranking the results.

It is difficult to design querying methods that scale up satisfactorily to hundreds of millions of documents—particularly given the additional constraint that it must be possible to implement the scheme so that queries are answered almost immediately. The usual one- or two-word query soon becomes completely inadequate except for very special queries. Long Boolean expressions are hard to enter, manage, and refine, tipping the balance toward automatic methods of ranking to assist users in their quest for satisfactory means of information retrieval.

Case-folding and stemming

Two operations that are often needed when querying are *case-folding* and *stemming*. Figure 3.15 shows a Preferences page that allows users to choose between different types of query interface and different query options. In the lower part, case-folding and stemming options are selected. An attempt has been made in the interface to describe these operations in three or four words whose meaning

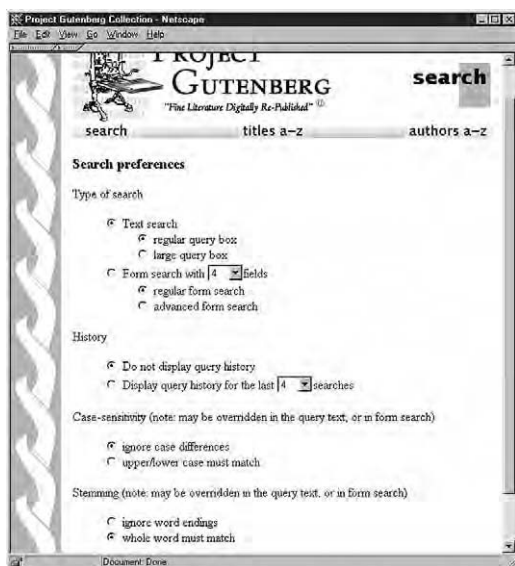


Figure 3.15 Choosing search preferences.

will hopefully be grasped immediately by casual users; here we discuss them in more detail.

In our earlier example queries, uppercase versions of words—such as *Digital* and *DIGITAL*, *Library* and *LIBRARY*—should probably be considered equivalent to the query terms *digital* and *library*, respectively. This is easily accomplished by case-folding—in other words, replacing all uppercase characters in the query with their lowercase equivalents. There will be situations where users wish to disable this feature and perform an exact-match search, perhaps looking for documents containing *Digital AND Equipment AND Corporation* (the now-defunct computer manufacturer), to avoid being flooded with answers about *corporation policy on capital equipment for digital libraries*. Users need to be able to specify either option in their query.

The second process, *stemming*, relaxes the match between query terms and words in the documents so that, for example, the term *libraries* is accepted as equivalent to *library*. Stemming involves reducing a word by stripping off one or more suffixes, converting it to a neutral *stem* that is devoid of tense, number, and—in some languages—case and gender information.

The process of altering words by adding suffixes (and, in some languages, affixes) is governed by many linguistic rules. Converting the *y* in *library* to the *ies* in *libraries* is one simple example; another is the final-consonant doubling when the word *stem* is augmented to *stemming*. The process of reducing a word to its root form, which is called *morphological reduction*, is correspondingly complex and typically requires a language dictionary that includes 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. So long as all variants of a word are reduced to the same stem, and no other words reduce to that stem, the desired effect will be obtained. There is a further simplification: 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 such as *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.

Like case-folding, stemming is not necessarily appropriate for all queries, particularly those involving names and other known items.

Stemming is language-dependent: an algorithm for English will not work well on French words and vice versa. If no stemmer is available for the language under consideration, it is probably best not to stem. Indeed the very concept of stemming differs widely from one language to another. Many languages use prefixes

as well as suffixes to indicate derivational forms; others involve complex compound words that should be split into constituent parts before taking part in a query. Case-folding, too, is not relevant to certain languages, such as Chinese and Arabic.

Stemming and case-folding complicate the task of highlighting search terms in the retrieved documents. You cannot simply find the stem and highlight all words that contain it, for this would often highlight the wrong word. For example, a particular system might stem both *library* and *libraries* to *librar* so that they match, but avoid stemming *librarian* because this is a derivational suffix that changes the meaning of the word. (In practice, many stemmers are not so sophisticated.) However, the form *librarian* does contain the letters *librar* and so will be highlighted if this is based on a simple textual match—incorrectly, for this system. And this method may fail to highlight a correct term—a different stemming algorithm might stem *libraries* to the root form *library*, and then fail to match when the text is highlighted.

Correct results would be obtained by stemming each word in the retrieved document using the same stemming algorithm and seeing if the result matched the stemmed query word, but this is likely to be prohibitively time-consuming. An alternative is for the information retrieval system to record the stemmed form of each word and expand the query by adding each stemmed form of these terms prior to the highlighting operation.

Phrase searching

Users often want to specify that the search is for contiguous groups of words, or *phrases*. 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 putting the phrase in quotation marks.

Although from a user's point of view phrase searching is a simple and natural extension of the idea of searching, it is more complex from an implementation perspective. Users think phrase searching is as easy and natural as, say, Boolean searching, because they visualize the computer looking through all the documents, just as they would, but faster—a lot faster! Given that you can find the individual words in all those documents, surely it doesn't make much difference if you have to check that they come together as a phrase.

Actually it does. When computers search, they don't scan through the text as we would: that would take too long. Computers are not all that fast at working through masses of text. For one thing the documents will be stored on disk, and access to the disk 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, to get a list of document

numbers. Then the query is answered by manipulating these lists—in the case of a Boolean AND query, by checking which documents are in all the lists. (The process is described in a little more detail in Section 4.2.)

Phrase searching changes everything. No longer can queries be answered simply by manipulating the lists of document numbers. There are two quite different ways of proceeding. The first 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. We refer to this as a *postretrieval scan*. The second 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 resulting list of word positions if the words occur together in a phrase by checking whether they are numbered consecutively.

The mechanism that is used to respond to phrase queries greatly affects the resources required by an information retrieval system. With a postretrieval scan, only a document-level index is needed. But it can take a great deal of time to respond to a phrase query because—depending on how common the terms are—many documents might have to be scanned. Phrases containing unusual words can be processed quickly, for few documents include them and therefore few need 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 each 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 several times in any given document and each 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 a phrase that includes punctuation and even white space. A word-level index treats the documents as sequences of words, and punctuation and spacing are generally ignored. Distinguishing between differently punctuated versions of phrases requires a postretrieval scan even if a word-level index is available—unless the index also includes word separators.

Phrases complicate ranked searching. Ranking uses the frequency of a query term throughout the corpus as a measure of 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 a phrase is used in the query, it should really be the frequency of the phrase, not the frequency of the individual words in it, that determines its influence. For example, the English words *civil*, *rights*, and *movement* are used in many contexts, but the phrase *civil rights movement* has a rather 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 of the constituent words.

As we have seen, the simple idea of including phrases in queries complicates things technically. In practice, building digital library systems involves pragmatic decisions. A word-level index will be included if it is feasible, if phrase searching is likely to be common, and if the space occupied by the system is not a significant constraint. In simple systems a postretrieval scan will be used, particularly if phrase searching will be rare or if phrases might contain punctuation. In either case ranking will be based on individual word frequencies, not phrase frequencies, for practical reasons.

One of the problems in digital libraries is ensuring that users grasp what is happening. As we have seen, it is difficult to fully understand what is happening in phrase searching! An alternative is the kind of phrase browsing technique described in Section 3.5: this provides a natural interface whose workings are easy to grasp.

Different query interfaces

Different query interfaces are suitable for different tasks, and users can choose different search preferences, as shown in Figure 3.15, to suit what they are doing.

For example, the search pages we have seen (e.g., Figure 3.14) have miniscule query boxes, encouraging users to type just one or two terms. In reality many queries contain only a few words. Indeed studies have shown that the most common number of terms in actual queries to actual Web search systems is—zero! Most often people just hit the search button, or the Return key, without typing anything in, presumably either by accident or without understanding what is going on. The second most common type of query has just one term. Note that for single-term queries, the difference between Boolean and ranked retrieval is moot. There *is* a difference—most Boolean systems will return the retrieved documents in a particular, predetermined, order—say, date order—whereas ranked systems will return them in order of how often they contain the query term (normalized by document length). But the user probably doesn't focus on this difference and is just as likely to enter the query term in either mode—or far more likely, in practice, to use whatever default the system provides. The third

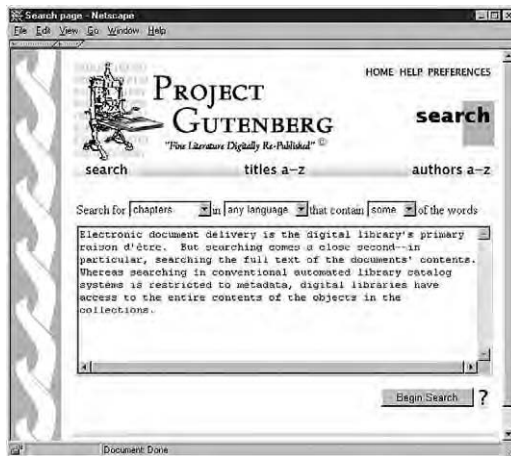


Figure 3.16 Large-query search interface.

most common query has two terms; after that, frequency falls off rapidly with query length.

Most search systems implicitly encourage short queries by not providing enough space on the screen for users to specify long ones! But modern search engines are capable of dealing with large queries: indeed they can often be processed more efficiently than small ones because they are more likely to contain rare words, which greatly restricts the scope of the search. Figure 3.16 shows a large query box into which users can paste paragraph-sized chunks of text—and it is scrollable to facilitate even larger queries.

A useful feature for all kinds of search is to allow users to examine and reuse their search history. “Those who ignore history,” to adapt George Santayana’s famous dictum, “will be forced to retype it.” New queries are routinely constructed by modifying old ones—adding new terms if the query is too broad, to increase precision at the expense of recall, or removing terms if it is too restrictive, to increase recall. Figure 3.17 shows an interface that presents the last four queries issued by the user. If one of the buttons to the left is clicked, the corresponding query is reentered into the search box, where it can be further modified. For example, clicking on the button to the left of the field containing *begin beginning* will place those words in the search box. The interface, and the number of history items displayed, is selected by making appropriate choices on the Preferences page in Figure 3.15.

A slightly awkward situation arises when the user changes search options, or even collections, between queries. It could be misleading to display query history in a way that ignores this. And it is possible that the user is experimenting with these options to ascertain their effect on a query—to see if the number of results changes if stemming is enabled, for example, or determine whether one

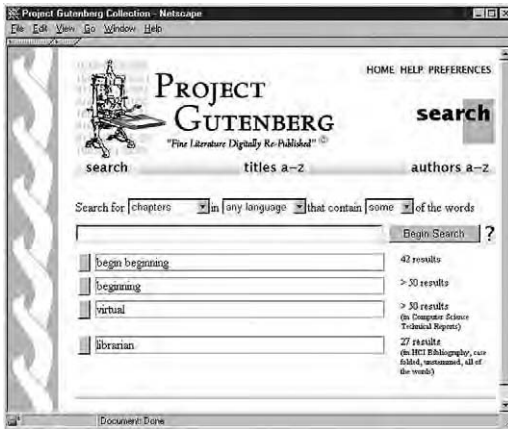


Figure 3.17 Query with history.

collection or the other contains more potential answers. This is why details are given alongside the history item when such changes occur, as can be seen in Figure 3.17. In most situations these details never appear, because users rarely alter their query options. The details, when they do appear, clarify the context within which the query history should be interpreted.

The query facilities shown so far let users search the full document text, or certain metadata fields which were chosen when the collection was created. Often, particularly for academic study, searches on different fields need to be combined. You might be seeking a book by a certain author with a particular word in the title or with a particular phrase in the main text. Library catalog systems provide a search form that consists of several individual fields into which specifications can be entered.

Figure 3.18a shows a form for a fielded search. The user types each field specification into one of the entry boxes, uses the menu selector to the right of the box to select which field it applies to, and decides whether to seek documents that satisfy *some* or *all* of these conditions. If you need more than four entry boxes, the Preferences page (Figure 3.15) gives further options.

Using the form in Figure 3.18b, more complex fielded searches can be undertaken. Again, up to four specifications are placed in the entry boxes, and the fields to which they apply are selected. Case-folding and stemming can be determined for each field individually, rather than being set globally using the Preferences page as before. The selection boxes that precede the fields allow Boolean operators: they give the three options *and*, *or*, and *and not*. In fact, completely general Boolean queries cannot be specified using this form because the user cannot control the order of operation—there is no place to insert parentheses. The specifications are executed in sequential order: each one applies to all the fields beneath it.

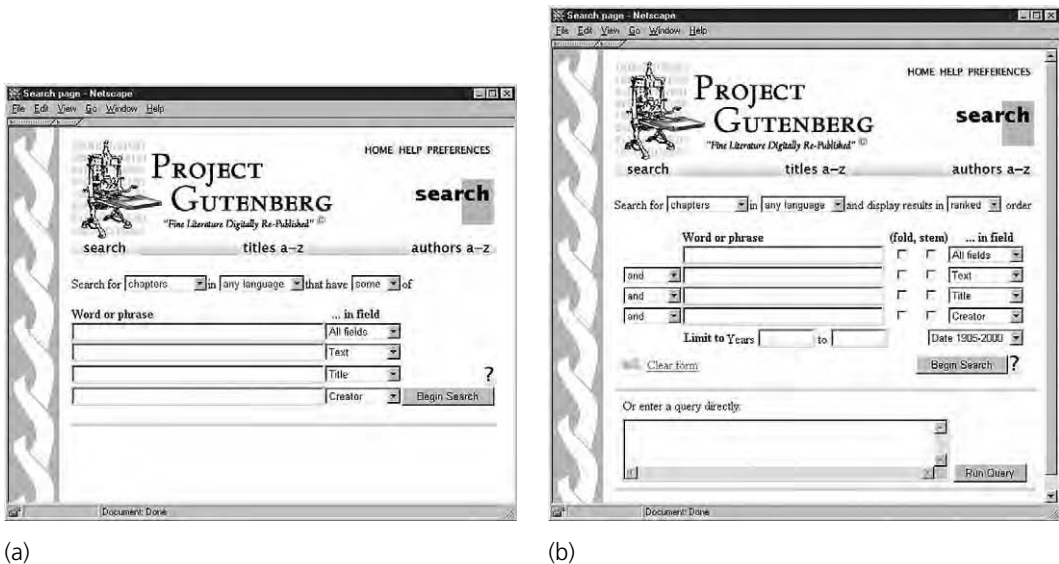


Figure 3.18 Form search: (a) simple; (b) advanced.

The line near the top, just beneath 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 date order). Below the field specification boxes there is an opportunity to limit the search to certain date ranges. In some collections more than one date may be associated with each document, corresponding to different kinds of metadata. For example, in a historical collection there is a big difference between the period that a document covers and the date it was written. If the metadata involves multiple date fields, a particular one is chosen using the selection box to the right of the year specifications—and here the beginning and end year is shown as well, to clarify what it is sensible to type in as a “limit” specification.

This interface is intended for expert users. However, experts often find it frustrating to have to specify what they want by filling out a form: they prefer to type textual queries rather than having to click from one field to another. Behind the form-based interface is an underlying textual specification of queries, and users may prefer to enter their queries directly in this query language.

The box at the bottom of Figure 3.18b allows queries to be entered textually. Whenever a form-based query is executed, its textual equivalent appears in this box, above the query results. The query specification language used is CCL, standing for “common command language.” This was a popular language for expressing queries in the library world, especially before the widespread availability of graphical interfaces—which are universally preferred by nonexpert users. CCL is described in Section 8.4.

This system makes it particularly easy to learn the language syntax because queries entered on the form are automatically converted to CCL and displayed underneath when the query is executed. Users who are unsure about the precedence order in which their field specifications are interpreted can look at the CCL equivalent (which is fully bracketed) to find out—and alter the interpretation order by inserting brackets into the textual query and reexecuting it. This provides a relatively easy path for frequent users of the form interface to evolve into power users of CCL.

It is difficult to provide a useful history facility with a forms-based interface because the representation of previous queries is cumbersome—several copies of a form consume a lot of screen space and are only rarely informative. However, the textual query language provides a natural way of utilizing history within the advanced search interface. Previous queries can be expressed in CCL and displayed in boxes just like those of Figure 3.17. They can be recalled into the query specification box, and reexecuted, or modified, there.

3.4 Browsing

Browsing is often described as the other side of the coin from searching, but really there is an entire spectrum between the two. Our dictionary describes browsing as “inspecting in a leisurely and casual way,” whereas searching is “making a thorough examination in order to find something.” Other dictionaries are more loquacious. According to Webster’s, browsing is

- to look over casually (as a book), skim;
- to skim through a book reading at random passages that catch the eye;
- to look over books (as in a store or library), especially in order to decide what one wants to buy, borrow, or read;
- to casually inspect goods offered for sale, usually without prior or serious intention of buying;
- to make 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 1.3.

In contemporary information retrieval terms, 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 the lack of a specific plan of action for reaching a goal—if indeed one exists.

Searching implies that you know what you're looking for, whereas in browsing the most you can say is that you'll know it when you see it—and the activity is often far less directed than that, perhaps even no more than casually passing time. But the distinction 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 and discuss various forms or degrees of browsing.

The metadata provided with the documents in a collection offer handles for different kinds of 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 browsing structures are present. The structure that is implicit in metadata is the key to providing browsing facilities that are based on it. And now it's time for some examples.

Browsing alphabetical lists

The simplest and most rudimentary of structures is the ordered list. In Figure 3.19a a plain alphabetical list of document titles is presented for browsing. The list is quite short: if it were not, it might take a long time to download over a network connection, and the user would have to scroll through it using the browser's scroll bar—which is an inconvenient way to find things in extremely long lists.

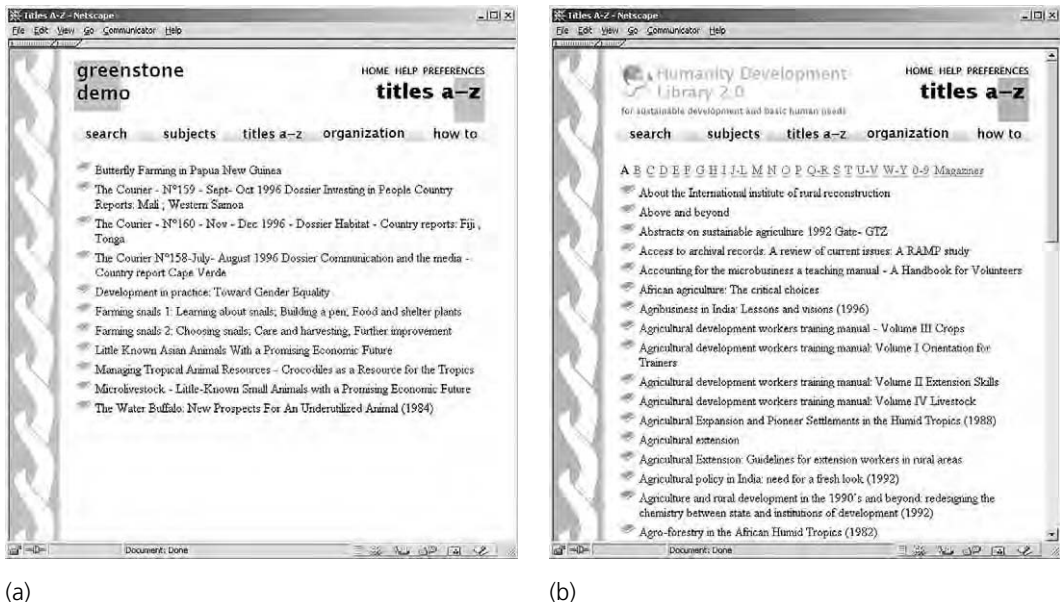


Figure 3.19 Browsing an alphabetical list of titles: (a) plain list; (b) with A–Z tags.

Notice incidentally that the ordering is not strictly alphabetical: the article *The* (and also *A*) is ignored at the beginning of titles. This is common practice when presenting sorted lists of textual items. For special kinds of text the ordering may be different yet again: for example, names are conventionally ordered alphabetically by surname even though they may be preceded by first names and initials.

Figure 3.19b shows part of a much larger list, with tags at the top to select alphabetic ranges. The user clicks on a tag, and the list of titles beginning with that letter appears underneath. The letter ranges are automatically chosen so that each one covers a reasonable number of documents. That is, letters in the three ranges *J–L*, *Q–R* and *U–Y* have been merged, because the number of documents that fall under each of these letters is rather small. In fact Figure 3.19a was generated by exactly the same scheme—but in this case there were so few documents that no alphabetic ranges were generated at all.

With very many documents, this interface becomes cumbersome. It is inconvenient to use tabs with multiletter labels, such as *Fae–Fal*. Although we are used to seeing these at the top of the pages of dictionaries and telephone directories, we are more likely to estimate the place we are trying to find on the basis of bulk, rather than going strictly by the tabs. Moreover, if the interface presents a sequence of such decisions during the process of locating a single item (e.g., *F*, *Fa–Far*, *Fae–Fal*, . . .) it is a tedious way of narrowing down the search. It forces you to think about what you are doing in a rather unnatural way.

The final tab, *0–9*, indicates another snag with this scheme. There is no knowing what letters titles may start with!—they may even include punctuation characters or arithmetic operators. This is not a big problem in English because such documents rarely occur, and when they do they can be included in a single Miscellaneous tab at the end.

Ordering lists of words in Chinese

Some languages are not alphabetic. Chinese has no single universally used way of ordering text strings analogous to alphabetic ordering in European languages. Several different ordering schemes are used as the basis of printed dictionaries and telephone directories. For example, characters, or *ideographs*, 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 users prefer Pinyin (not all Chinese people know Pinyin). Browsers for these languages call for special design.

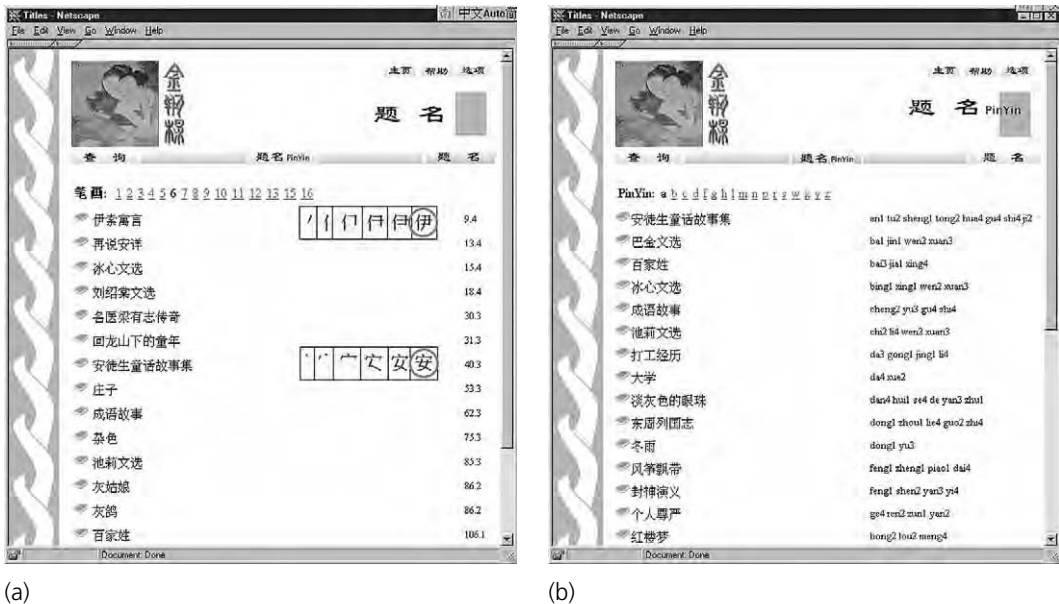


Figure 3.20 Browsing a list of titles in Chinese: (a) stroke-based browsing; (b) Pinyin browsing.

For an appreciation of the issues involved, see Figure 3.20, which shows a list for browsing titles in a large collection of Chinese books and articles. In Figure 3.20a, which is invoked by the rightmost of the three buttons on the access bar near the top, titles are ordered by the number of strokes contained in their first character when written. The number of strokes is given across the top: the user has selected six. The first character in all the book titles that follow has six strokes. If you don't know Chinese this will not be obvious from the display: you can only count the number of strokes in a character if you know how the character is written. 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 are easily able to work out the number of strokes required to write any particular character.

In most cases there are more than 200 characters corresponding to a given number of strokes, almost any of which could occur as the first character of a book title. Hence the titles in each group are displayed in a particular conventional order, again determined by their first character. This ordering is a little more complex. With each character is associated a radical, or basic structure that is contained in it. For example, in the first example singled out in Figure 3.20a (the first title), the radical is the pattern corresponding to its 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 second example singled out in Figure 3.20a (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 3.20a; they would not form part of the actual Web page 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 these cases the tie is broken randomly.

The stroke-based ordering scheme is quite complex. In practice, Chinese readers have to work harder to identify an item in an ordered list than we do. It is easy to decide on the number of strokes. Once a page like Figure 3.20a is reached, however, people often simply scan it linearly. One strength of computer displays is that they can at least offer a choice of access methods.

The central button in the navigation bar near the top of Figure 3.20 invokes the Pinyin browser shown in Figure 3.20b. This orders characters alphabetically by their Pinyin equivalent. The Pinyin codes for the titles are shown to the right of the figure; they would not form part of the actual Web page display. Obviously this kind of display is much easier for Westerners to comprehend.

Browsing by date

Figure 3.21 shows newspapers being browsed by date. An automatically generated selector at the top gives a choice of years; within each range the newspapers are laid out by month. Just as Figure 3.19 was created automatically based on *Title* metadata, so Figure 3.21 was created automatically based on *Date* metadata. Again the year ranges are chosen by the computer to ensure that a reasonable number of items appear on each page.

Hierarchical classification structures

The browsers we have seen are essentially linear, and this restricts them to situations where the number of documents is not excessive. Hierarchical classification structures are standard tools in any area where significant numbers of objects are being considered. In the library world, the Library of Congress classification and the Dewey Decimal classification are used to arrange printed books in categories, the intention being to place volumes treating the same or

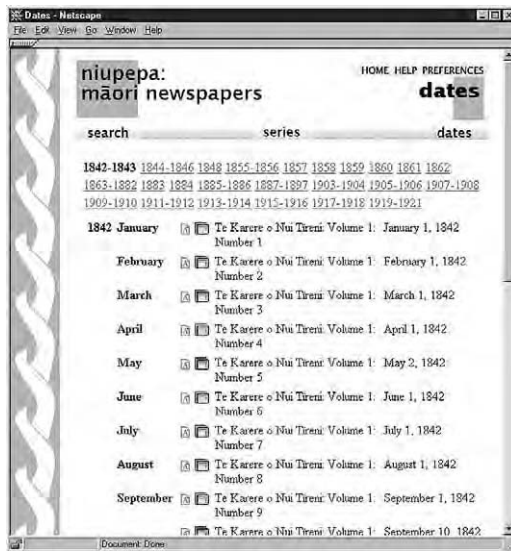


Figure 3.21 Browsing by date.

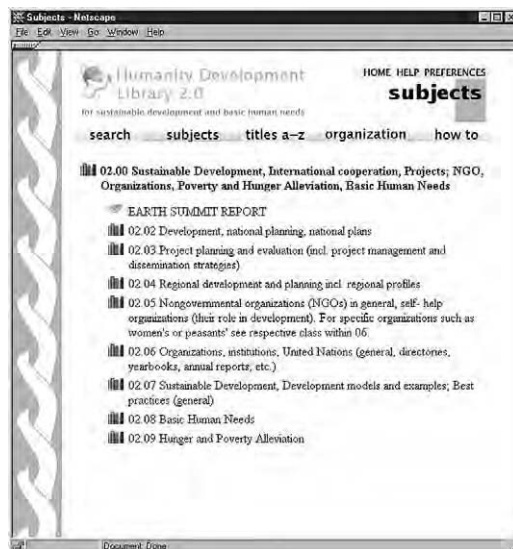
similar subjects next to each other on the library shelves. These schemes are hierarchical: the early parts of the code provide a rough categorization that is refined by the later characters.

Figure 3.22 shows a hierarchical display according to a particular classification scheme that was used for the Humanity Development Library. Nodes of the hierarchy are represented as bookshelves. Clicking one opens it up and displays all documents that occur at that level, as well as all nodes that lie underneath it. For example, node 2.00, shown in Figure 3.22b, contains one document and eight subsidiary nodes, of which one—node 2.06—is shown in Figure 3.22c. Just as bookshelf icons represent internal nodes of the hierarchy, so book icons represent documents, the leaves of the classification tree. One is visible in Figure 3.22b for the *Earth Summit Report*, which is the only document with classification 2.00.

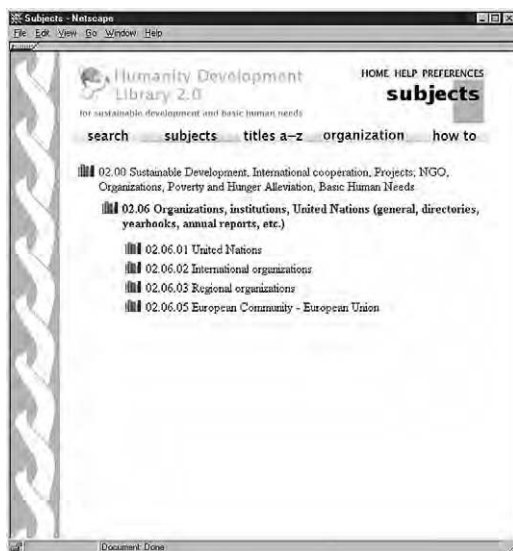
This browsing structure was generated automatically from metadata associated with each document. Consider the information that must be given to the system to allow the hierarchical display to be built. Each document must have associated with it its position in the hierarchy. In fact, in this classification scheme an individual document may appear in different places, so there needs to be a way to specify classification information multiple times for a single document. In addition to the hierarchical information, names must be provided for the interior nodes—so that all the “bookshelf” nodes in Figure 3.22 can be labeled appropriately. This involves a separate file of information, distinct from the document collection itself.



(a)



(b)



(c)

Figure 3.22 Browsing a classification hierarchy: (a) the beginning; (b) expanding *Sustainable development*; (c) expanding *Organizations, institutions*.

The classification scheme in Figure 3.22 is nonstandard. It was chosen by the collection editor as being the most appropriate for the collection's intended users. Implementers of digital library systems have to decide whether to try to impose uniformity on the people who build collections, or whether instead to provide flexibility for them to organize things in the way they see fit. Opting for the latter gives librarians freedom to exercise their professional judgment effectively.

3.5 Phrase browsing

Naturally people often want to browse information collections based on their subject matter. As we have seen, this kind of browsing is well supported by displays based on hierarchical classification metadata that is associated with each document. But manual classification is expensive and tedious for large document collections. What if this information is not available? To address this problem, one can build topical browsing interfaces based on phrase metadata, where the phrases have been extracted automatically from the full text of the documents themselves.

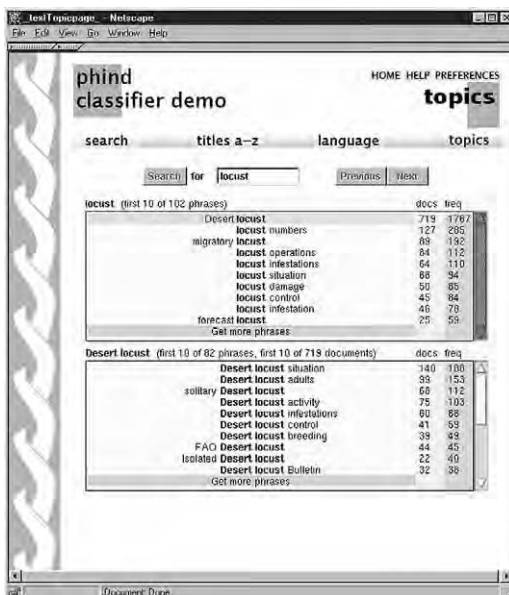
A phrase browsing interface

The browser illustrated in Figure 3.23 is an interactive interface to a phrase hierarchy that has been extracted automatically from the full text of a document collection. In this case the collection is simply a Web site: the site of the United Nations Food and Agriculture Organization (FAO). As we pointed out earlier, Web sites are not usually digital libraries. Even though they may be well organized, the organization is provided by a manual insertion of links, which conflicts with an essential feature of libraries: that new material can be added easily and virtually automatically, merely by supplying appropriate metadata. However, digital libraries can certainly be created from Web sites by adding automatically generated organization, and this collection is a good example. Figure 3.23c (and also Figure 3.24b) shows typical target documents. They are just Web pages, complete with all internal and external links, which are fully functional.

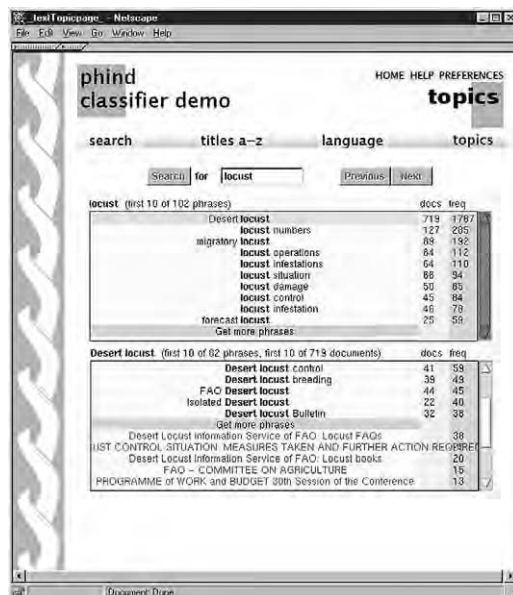
This interface is designed to resemble a paper-based subject index or thesaurus. In Figure 3.23a the user enters an initial word in the search box at the top—in this case the word *locust*. On pressing the Search button the upper of the two panels appears. This shows the phrases at the top level in the hierarchy that contain the search term. The list is sorted by phrase frequency; on the right is the number of times the phrase appears in the entire document collection, and beside that is the number of documents in which the phrase appears.

Only the first 10 phrases are shown, because it is impractical with a Web interface to download a large number of phrases, and many of these phrase lists are huge. At the end of the list is an item that reads *Get more phrases* (displayed in a distinctive color); clicking this will download another 10 phrases, and so on. The interface accumulates the phrases: a scroll bar appears to the right for use when more than 10 phrases are displayed. The number of phrases appears above the list: in this case there are 102 top-level phrases that contain the term *locust*.

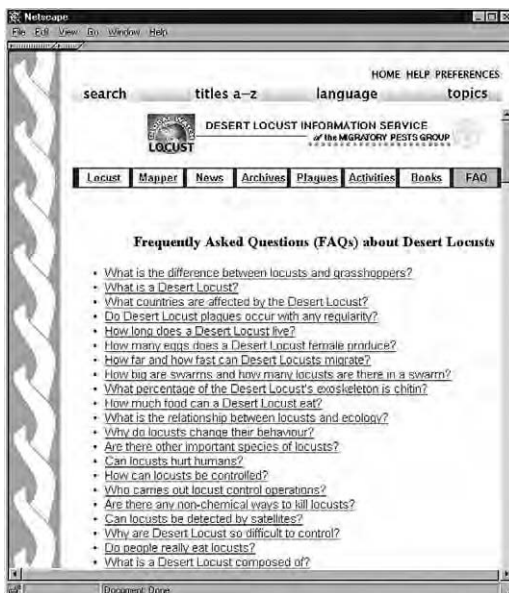
So far we have only described the upper panel in Figure 3.23a. The lower one appears as soon as the user clicks one of the phrases in the upper list. In this case



(a)



(b)



(c)

Figure 3.23 (a) Browsing for information about locusts; (b) expanding on *desert locust*; (c) document about *desert locusts*. United Nations Food and Agriculture Organization.

the user has clicked *Desert locust* (which is why the first line is highlighted in the upper panel), causing phrases containing the string *Desert locust* to be displayed in the lower panel.

If you continue to descend through the phrase hierarchy, eventually the leaves will be reached. In this system any sequence of words is a “phrase” if it appears more than once in the entire document collection. Thus a leaf corresponds to a phrase that occurs in a unique context somewhere in the collection (although the document that contains that contextually unique occurrence may include several other occurrences too). In this case the text above the lower panel shows that the phrase *Desert locust* appears in 82 longer phrases and also in 719 documents. These 719 documents each contain the phrase in some unique context. The first 10 documents are visible when the list is scrolled down, as is shown in Figure 3.23b.

In effect both panels show a phrase list followed by a document list. Either list may be empty: in fact the document list is empty in the upper panel because every context in which the word *locust* occurs appears more than once in the collection. The document list displays the titles of the documents.

In both panels of Figures 3.22a and b, you can click *Get more phrases* to increase the number of phrases that are shown in the list. In the lower panels you can also click *Get more documents* (again it is displayed at the end of the list in a distinctive color, but to see that entry you must scroll the panel down a little more) to increase the number of documents shown.

Clicking on a phrase expands that phrase. The page holds only two panels, and if a phrase in the lower one is clicked, the contents of that panel move up to the top to make space for the phrase’s expansion. Alternatively clicking on a document opens that document in a new window. In fact the user in Figure 3.23b has clicked on *Desert Locust Information Service of FAO: Locust FAQs*, and this brings up the page shown in Figure 3.23c. As Figure 3.23b indicates, that document contains 38 occurrences of the phrase *Desert locust*.

Figure 3.24 shows another example of the interface in use. In this case a French user has entered the word *poisson*, exposing a weakness of the phrase extraction algorithm. The FAO Web site contains documents in French, but the phrase extraction system is tailored for English. The French phrases displayed are of much lower quality than the English ones shown earlier; of the ten phrases in the upper panel, only four are useful. Phrases like *du poisson* (meaning “of fish”) are not useful and can obscure more interesting material. However, the system is still usable. Here the user has expanded *commercialisation du poisson* and, in the lower panel, has clicked on a document titled INFOPECHE which is shown in Figure 3.24b.

Utilizing phrases extracted automatically from the document text, as these example phrases are, has the great advantage that no manual processing is needed

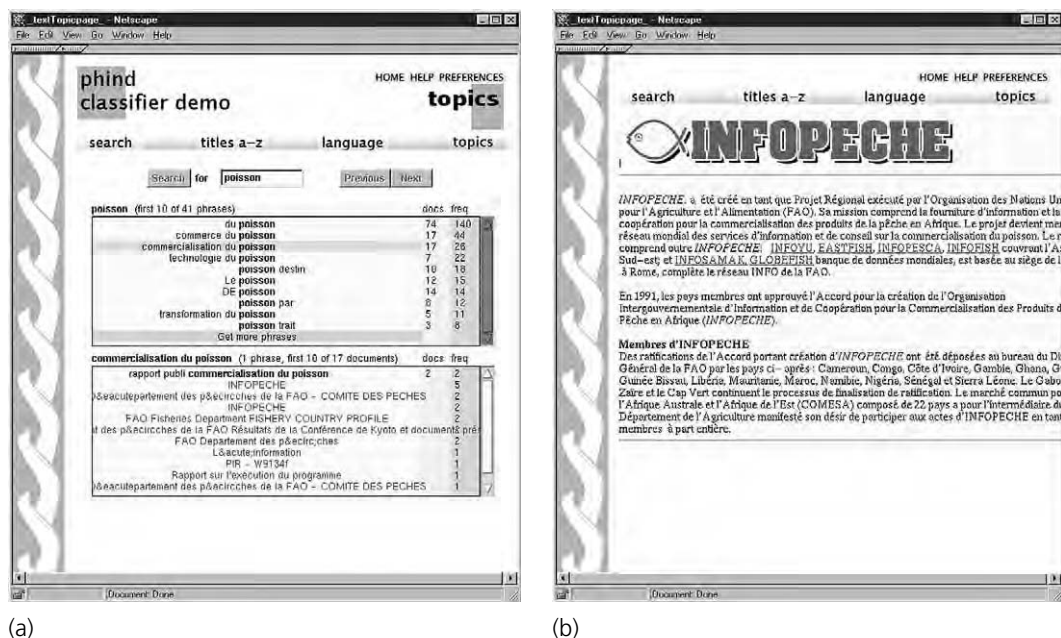


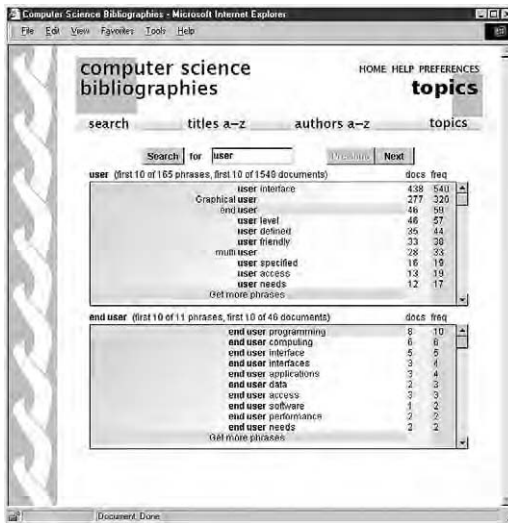
Figure 3.24 (a) Browsing for information on *poisson*; (b) INFOPECHE Web page.

to generate these indexes. However, the amount of computer processing required is substantial. This phrase interface has the advantage that it is readily comprehensible by users. The notion of browsing documents based on phrases that they contain has great intuitive appeal because you are in direct contact with the raw material of the documents themselves, without any intermediary operations that you may only dimly understand.

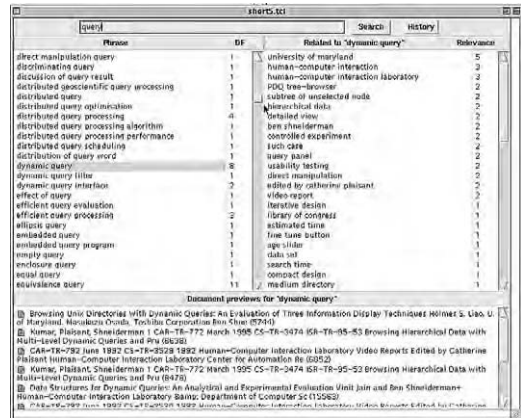
An important feature of this interface is that it scales up to large collections. Other browsing techniques, such as alphabetic lists of titles and authors, do not scale well: ranges of letters are uncomfortable to use for large collections. Hierarchical classification schemes scale well, but require manual classification of each document. This phrase interface is easy to browse even for colossal collections, and users are not overwhelmed with information, nor with difficult choices. It is unique in being both scalable and automatically generated. We return to the question of how to generate the phrase hierarchy at the end of Section 5.6.

Key phrases

The phrase browsing interface is based on a staggeringly large number of phrases—essentially all those that appear in the full text of the documents. Sometimes it helps to be more selective, choosing perhaps half a dozen representative phrases for each document. Such phrases are usually called *key phrases*.



(a)



(b)

Figure 3.25 Browsing interfaces based on key phrases: (a) hierarchical browser; (b) document explorer.

Authors of scholarly or academic documents often provide accompanying key phrases, particularly in scientific and technological disciplines.

Figure 3.25a shows a hierarchical phrase browser just like that in Figures 3.23 and 3.24—except that in this case the phrases are author-assigned key phrases, rather than being taken automatically from the full text of the documents. Not surprisingly these key phrases are of much higher quality. In fact this example shows a collection of bibliographic entries in the computer science area: the full text is not available, just the bibliographic references and author-supplied key phrases.

The person using this interface has entered the term *user*, and the key phrases that contain this word are shown in the upper panel. There are 165 of them in total; only the 10 most frequent are shown. Of these, *end user* has been selected, and the phrases that appear below give a good synopsis of the way the term is used in the computer science literature. The key phrases are all high quality because they have been carefully chosen by the papers' authors, and the result is an excellent interactive taxonomy of terms and concepts in the field of computer science. Although the key-phrases metadata is supplied manually, the hierarchy is constructed automatically.

Figure 3.25b shows a different style of interface to a document collection, also based on key phrases. At the very top the user has typed the word *query*. Underneath are three large panels. The left-hand one displays all key phrases in the document collection that contain the word *query*. Unlike the earlier displays,

this is not hierarchical—the number of key phrases is not so great that a hierarchy is necessary, although, as Figure 3.25a illustrates, it could be helpful. The user has selected one of these phrases, *dynamic query*; the number beside it indicates that it has been identified as a key phrase for eight documents. In the bottom panel appear the titles of these eight documents (only six are visible). Clicking on a document icon displays that document.

Whereas hierarchical phrase browsers allow users to focus on a particular area by selecting more and more specific terms from the choices that are offered, this interface provides a good way of broadening the concepts being considered. The panel at the right of Figure 3.25b gives the *other* key phrases that are associated with these eight documents. In a sense these key phrases “share a document” with the selected phrase *dynamic query*. The fact that *Library of Congress* appears in this list, for example, indicates that there is a document—namely, one of the eight displayed—that has both *dynamic query* and *Library of Congress* assigned to it as key phrases. Clicking on *Library of Congress* would transfer these words to the query box at the top of the screen, and then any key phrases containing them would be displayed in the panel on the left, with associated documents underneath.

The scheme in Figure 3.25b provides another convenient way to explore a document collection. Unlike the other interfaces in this chapter, it is not Web-based and does not operate in a Web browser. However, it could easily be reengineered to do so.

3.6 Browsing using extracted metadata

The browsing methods we have examined all rely on metadata that must be provided for all documents in the collection. With the exception of the phrase hierarchy metadata, which is extracted from the document text, metadata must be entered manually for each document. This is a tedious and time-consuming chore, which makes you wonder whether metadata can be automatically extracted from the documents’ full text.

For example, titles may be identified by seeking capitalized text close to the beginning of documents—and if information about font size is available, it provides extra clues.

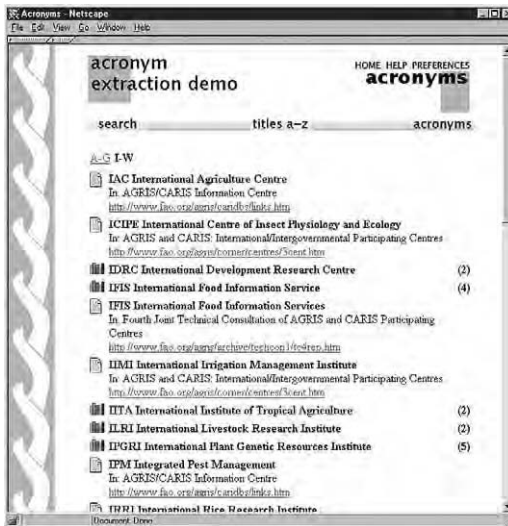
Names may be identified in the text of all documents by looking for the capitalization and punctuation patterns that characterize forms such as *Surname*, *Forename* and *Forename Initial*. *Surname*. Once this metadata is available, a browsing index could be provided that allows you to look through all people who are mentioned in the document collection and check references to them. Even if the method used to extract name metadata had some deficiencies, so

that some names were omitted and some non-names included, the browsing facility could still be useful.

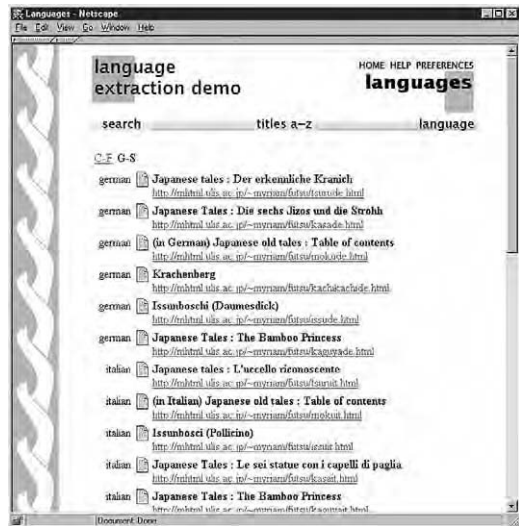
Dates that appear in documents could be identified. In historical collections it is useful to be able to restrict searches to documents that describe events within a certain time period—searching for information about *Trade Unions* in the 1930s, for example. Of course, dates of historical events must be distinguished from other dates—particularly ones that give the publication date of references. In many cases this distinction can be made automatically on the basis of punctuation, such as parentheses, that typically appears in references (e.g., dates are often included in brackets). Again, some errors in date identification are not necessarily disastrous.

Acronyms

We end our tour of browsing interfaces with two unusual examples based on metadata extraction. Figure 3.26a shows a list of acronyms in a document collection, along with their definitions. The tabs at the top allow you to select alphabetic ranges—although in this case there are only two. Where an acronym is defined in more than one document, a bookshelf is shown, with the number of documents in parentheses; clicking on the bookshelf will bring up the list of documents.



(a)



(b)

Figure 3.26 Browsing based on information mined from the document collection: (a) acronyms; (b) language identification.

These acronyms were extracted automatically from the full text of the document collection using a technique described in Chapter 5 (Section 5.6). Essentially the idea is to look for a word containing (mostly) uppercase letters—the acronym—preceded or followed by a sequence of words that begins with those letters, generally capitalized. Although the results are not perfect, the acronym browser is particularly useful for political and scientific documents. Because the information is mined automatically, no human effort at all goes into creating browsers for new collections.

Language identification

Figure 3.26b shows another example of metadata extraction in which documents are grouped by language, the language of each document having been identified automatically. This collection comprises a set of Japanese folktales that have been translated into a variety of languages. The same language metadata could be used to provide a separate subcollection for each language, in which the language could be selected when searching.

The language of each document was automatically identified by a system that looks at letter patterns, described in Section 5.6. For example, *yes* is characteristic of English, whereas *oui* is French and *ja* is German. However, instead of looking at complete words, the system looks at groups of up to five letters—because these are more general and sensitive indicators of language. Particular words may not occur in a given document (*yes* does not occur often in this book), yet it is hard to imagine any stretch of English writing that does not include the characters *the*. (Hard, but not impossible—surprising as it may seem, a complete book has been written that does not use the letter *e*!)

3.7 Notes and sources

“Begin at the beginning” is what the King of Hearts said to the White Rabbit in the trial scene in *Alice in Wonderland*. The screen shot in Figure 3.1, like most of the figures in this chapter, came from a collection in the New Zealand Digital Library Project, which is publicly accessible on the Web at www.nzdl.org. All these collections are created and served using the Greenstone Digital Library Software.

Collections

The Kids’ Digital Library in Figure 3.2a is from a project at Middlesex University, London, that also uses Greenstone (<http://kidsdl.mdx.ac.uk/kidslibrary>). The Humanity Development Library in Figure 3.3 is mentioned in the “Notes and sources” of Chapter 1 (Section 1.6); it is available both on CD-ROM and at

www.nzdl.org. More information about the Gutenberg Project from which Figure 3.1 comes can be found at www.promo.net/pg.

The *New Zealand School Journal* is produced by the Ministry of Education and published by Learning Media Ltd. in Wellington, New Zealand. Figures 3.2b and 3.5 are taken from a prototype collection containing a few sample journals across the entire range of years from 1907 to 1998; it was designed by Mankelow (1999). For copyright reasons, this collection is not publicly available. Note that Figure 3.5b is a fake: while we have worked on software to highlight words in images, this is not at present incorporated into the School Journal collection.

Information on the “Niupepa” (the Māori word for “newspapers”) collection of historical New Zealand newspapers in Figure 3.6 is given in Chapter 2’s “Notes and sources” (Section 2.5). The Oral History collection in Figure 3.7 was originally produced on audiotapes by Hamilton Public Library; Bainbridge and Cunningham (1998) describe its conversion into digital form. Work on music collections and music searching, some of which is illustrated in Figure 3.8, is described by McNab et al. (1996) and Bainbridge et al. (1999).

The Sahel Point Doc collection shown in Figure 3.9a is on a CD-ROM of the same name distributed by UNESCO; it also appears at www.nzdl.org. The collection of Tang Dynasty rubbings in Figure 3.10a was produced in conjunction with the Chinese Department of Peking University; it can be found on the Web at <http://162.105.138.23/tapian/tp.htm>. The collection of Chinese literature in Figure 3.10b, and the Arabic collection in Figure 3.11, are small demonstrations built out of material gathered from the Web.

Metadata

Figure 3.12 comes from a large collection of computer science bibliographies assembled and maintained by Alf-Christian Achilles at the University of Karlsruhe, Germany. The original source is at <http://linwww.ira.uka.de/bibliography>; this material has been mirrored as a Greenstone collection. Figure 3.13a was obtained from the Library of Congress using the Z39.50 client that is embedded in the Greenstone software. This feature makes it possible to reach any Z39.50 server from a Greenstone installation. Figure 3.13b is from a large collection of the BBC radio and television archives with around one million entries. It was produced for the BBC 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.

Searching

Much of the material on searching is taken from the book *Managing Gigabytes* (Witten, Moffat, and Bell, 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 to the topic of text searching, briefly, in Chapter 4 (Section 4.2).

Alan Kay, quoted near the beginning of Section 3.3, is a guru in the interactive computing world; an interesting article about him appeared in *New Scientist* (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!

Browsing

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 at the beginning of Section 3.4 about the “patron’s random examination of library materials” is his. The brief characterization of browsing is from our favorite dictionary, 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 3.19a shows the Demo collection, which is supplied with every Greenstone installation and contains a small subset of the Humanity Development Library; Figure 3.19b is from the full Humanity Development Library, as is Figure 3.22. The Chinese title browsers in Figure 3.20 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 Sections 6.6 and 6.7 of Chapter 6). The date list in Figure 3.21 was generated within the Niupepa collection noted earlier.

Phrase browsing is a topic of lively current interest. The interface shown in Figures 3.23 and 3.24 has developed over several years; the current version is described in Paynter et al. (2000). Much work has gone into algorithms for extracting hierarchies of phrases from large document collections, and we return to this topic in Chapter 5 (Section 5.6). The collection is built from the FAO Web site at www.fao.org, which contains the documents shown.

Key-phrase browsing is another burning issue. The illustration in Figure 3.25a is from Alf-Christian Achilles' Computer Science Bibliography collection mentioned earlier. Figure 3.25b is taken from a collection of Computer Science Technical Reports that includes author-assigned key phrases. The interface is described by Gutwin et al. (1999); similar ones are described by Jones and Paynter (1999). Although these collections have manually assigned key-phrase metadata, key phrases can be extracted automatically from the full text of documents with a surprising degree of success, as we shall see in Section 5.6.

The acronym extraction software, and the interface shown in Figure 3.26a, is described by Yeates, Bainbridge, and Witten (1999); the collection used to illustrate it is a subset of the FAO Web site. The methods used to identify languages and extract acronyms are described further in Chapter 5 (Section 5.6). The illustration in Figure 3.26b is built from the material at <http://mhtml.ulis.ac.jp/~myriam/deuxgb.html>, a site that contains a charming collection of old Japanese folktales in different languages (Dartois et al., 1997). The book without any *e*'s is *Gadsby*, by E. V. Wright (1939).

This Page Intentionally Left Blank



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 we will have to descend even further and look at the representation of the characters that make up textual documents and the fonts in which those characters are portrayed. For audio, images, and video we examine the interplay among signal quantization, sampling rate, and internal redundancy that underlies multimedia representations.

A great practical problem we face when writing about how digital libraries work inside is the dizzying rate of change in the core technologies. Perhaps the most striking feature of the digital library field, at least from an academic point of view, is the inherent tension between two extremes: the very fast 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 decade 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!

Internationalization is an important component of the vision of digital libraries that motivates this book, and in the last chapter we saw examples of collections and interfaces in many different languages. It is all too easy from the native English speaker's perspective (which includes most of our readers and both this book's authors) to sweep under the carpet the many challenging problems of representing text in other languages. The standard ASCII code used on computers (the *A* stands for "American," of course) has been extended in dozens of different and often incompatible ways to deal with other character sets—including, for example, ISCII for Hindi and related languages, where the initial *I* stands for "Indian." To take a simple example, 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*. Or suppose the character set is specified explicitly: Web pages often do this. Internet Explorer recognizes over 100 different character sets, mostly extensions of ASCII and EBCDIC, some of which have half a dozen different names. Without a unified coding scheme, search programs must know about all of this to work correctly under every circumstance.

Fortunately there is an international standard called Unicode which aims to represent all the characters used in all the world's languages. Unicode emerged over the last 10 years and is beginning to be widely used. It is quite stable, although it is still being extended and developed to cover languages and character sets of scholarly interest. It allows the content of digital libraries, and their user interfaces, to be internationalized. Using it, text stored in digital libraries can be displayed and searched properly. We describe Unicode in the next section. Of course it 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 4.2 discusses the representation of documents in text form—plain text. Even ASCII presents ambiguities in interpretation (you might have wondered about the difference between the ASCII and binary modes in the FTP file transfer protocol, or encountered the junk characters *^M* at the end of each line in a TELNET window). We also sketch how an index can be created for textual documents that contains for each word a list of documents it occurs in, and perhaps even the positions where it occurs, in order to facilitate rapid full-text searching. Before creating the index the input must be split into words. This involves a few mundane practical decisions—and introduces some deeper issues for some languages, such as Chinese and Japanese, that are not traditionally written with spaces between words.

Desktop publishing empowers ordinary people to create carefully designed and lavishly illustrated documents and publish them electronically, often dispensing with print altogether. In a short space of time we have become accus-

tomed to reading publication-quality documents online. It is worth reflecting on the extraordinary sea change in our expectations of online document presentation since, say, 1990. This revolution has been fueled largely by Adobe's PostScript language and its successor, PDF or Portable Document Format. 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. As society's notion of *document* has become more fluid—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—it seems inevitable that more and more documents will be handed around in word-processor formats.

As examples of word-processor documents we describe 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 strongly on the exact version of Microsoft Word that was used and is not always backward compatible. RTF is more portable, 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. We also describe the format used by the LaTeX document processing system, which is widely used to represent documents in the scientific and mathematical community.

The final part of the chapter describes image, audio, and multimedia formats. There is less variation here in the basic representation of the data. But because of the raw, quasi-analog nature of these media, file size bloats quickly and so compression schemes are often built into the formats. However, as well as simply making files smaller, compression has side effects that need to be considered when designing digital libraries.

For image data we describe the GIF, PNG, and JPEG formats. The first two are suitable for representing artificially produced images such as text, computer-generated artwork, and logos. JPEG is designed for continuous-tone images such as photographic portraits and landscapes. Multimedia encompasses both video and audio formats: we focus principally on the open MPEG standard, which includes the MP3 scheme that is widely used for music representation. We also mention Apple's QuickTime and Microsoft's AVI formats for multimedia, and WAV, AIFF, and AU for audio.

We warned at the outset that this chapter gets down to details, the dirty details (where the devil is). You will probably 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 such as hyperlinks. Converting images from GIF to JPEG often reduces file size but degrades picture quality irreversibly. 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 document 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 characters

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. Table 4.1 shows the 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 a rude 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

Table 4.1 The ASCII character set.

Dec	Oct	Hex	Char	Dec	Oct	Hex	Char
0	000	00	NUL	35	043	23	#
1	001	01	SOH	36	044	24	\$
2	002	02	STX	37	045	25	%
3	003	03	ETX	38	046	26	&
4	004	04	EOT	39	047	27	'
5	005	05	ENQ	40	050	28	(
6	006	06	ACK	41	051	29)
7	007	07	BEL	42	052	2A	*
8	010	08	BS	43	053	2B	+
9	011	09	HT	44	054	2C	,
10	012	0A	LF	45	055	2D	-
11	013	0B	VT	46	056	2E	.
12	014	0C	FF	47	057	2F	/
13	015	0D	CR	48	060	30	0
14	016	0E	SO	49	061	31	1
15	017	0F	SI	50	062	32	2
16	020	10	DLE	51	063	33	3
17	021	11	DC1	52	064	34	4
18	022	12	DC2	53	065	35	5
19	023	13	DC3	54	066	36	6
20	024	14	DC4	55	067	37	7
21	025	15	NAK	56	070	38	8
22	026	16	SYN	57	071	39	9
23	027	17	ETB	58	072	3A	:
24	030	18	CAN	59	073	3B	;
25	031	19	EM	60	074	3C	<
26	032	1A	SUB	61	075	3D	=
27	033	1B	ESC	62	076	3E	>
28	034	1C	FS	63	077	3F	?
29	035	1D	GS]	64	100	40	@
30	036	1E	RS	65	101	41	A
31	037	1F	US	66	102	42	B
32	040	20	SPAC	67	103	43	C
33	041	21	!	68	104	44	D
34	042	22	"	69	105	45	E

(continued on the following page)

Table 4.1 The ASCII character set (continued).

Dec	Oct	Hex	Char	Dec	Oct	Hex	Char
70	106	46	F	99	143	63	c
71	107	47	G	100	144	64	d
72	110	48	H	101	145	65	e
73	111	49	I	102	146	66	f
74	112	4A	J	103	147	67	g
75	113	4B	K	104	150	68	h
76	114	4C	L	105	151	69	i
77	115	4D	M	106	152	6A	j
78	116	4E	N	107	153	6B	k
79	117	4F	O	108	154	6C	l
80	120	50	P	109	155	6D	m
81	121	51	Q	110	156	6E	n
82	122	52	R	111	157	6F	o
83	123	53	S	112	160	70	p
84	124	54	T	113	161	71	q
85	125	55	U	114	162	72	r
86	126	56	V	115	163	73	s
87	127	57	W	116	164	74	t
88	130	58	X	117	165	75	u
89	131	59	Y	118	166	76	v
90	132	5A	Z	119	167	77	w
91	133	5B	[120	170	78	x
92	134	5C	\	121	171	79	y
93	135	5D]	122	172	7A	z
94	136	5E	^	123	173	7B	{
95	137	5F	_	124	174	7C	
96	140	60	`	125	175	7D	}
97	141	61	a	126	176	7E	~
98	142	62	b	127	177	7F	DEL

(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 bit) 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 Standards Organization (the international counterpart of the American standards organization, ANSI), extends ASCII for Western European languages. For example, it represents *é* as the single decimal value 233 rather than the clumsy ASCII sequence “*e* followed by backspace 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 such as Hebrew and Chinese, ASCII is irrelevant. Here other schemes have arisen: for example, GB and Big-5 are competing standards for Chinese, the former 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, a consortium 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 Standards Organization 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 hieroglyphics and Indo-European languages, and notations such as music. There is a steady stream of additions, clarifications, and amendments which eventually lead to new published versions of the standard. Of course backwards compatibility with the existing standard is taken for granted.

A standard is sterile unless it is adopted by vendors and users. Recent programming languages—notably Java—have built-in Unicode support. Earlier ones—C, Perl, Python, to name a few—have standard Unicode libraries. All principal operating systems 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 Unicode. 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 there is an existing character set that includes *é* as a single character, it must be represented as a single Unicode character—even if an identical glyph can be generated using a sequence along the lines of “*e* followed by backspace followed by ‘.’” The idea of 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, as we shall see.

The Unicode character set

The Unicode standard is massive. It comes in two parts (ISO 10646-1 and ISO 10646-2), specifying 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 4.2 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 covers Western and Middle Eastern languages such as Latin, Greek, Cyrillic, Hebrew, and Arabic; the so-called CJK (Chinese-Japanese-Korean) ideographs comprising Chinese, Japanese, and the Korean Hangul characters; and other scripts such as (to name just a few) Bengali, Thai, and Ethiopic. Also included are Braille, mathematical symbols, and a host of other shapes.

Table 4.2 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 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 4.1 shows the Basic Latin (or standard ASCII) and Latin-1 Supplement (an ASCII extension) char-

Table 4.2 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

(continued on the following page)

Table 4.2 Unicode Part 1: The basic multilingual plane (continued).

Zone	Area	Code	Script	Number of codes
Alphabetic	General scripts (continued)	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)
	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)
		2800	Braille Pattern Symbols	256
			—	(1536)

(continued on the following page)

Table 4.2 Unicode Part 1: The basic multilingual plane (continued).

Zone	Area	Code	Script	Number of codes
Ideographic	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
			—	(90)
Other		A000	Yi	1225
			—	(1847)
		AC00	Hangul Symbols	11172
Surrogates			—	(92)
		D800	High Surrogates	1024
		DC00	Low Surrogates	1024
Reserved	Private use Compatibility and specials	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

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	RI	SS2	SS3
090	DCS	PU1	PU2	STS	CCH	MW	SPA	EPA	SOS	XXX	SCI	CSI	ST	OSC	PM	APC
0a0	NB/SP	ı	ç	£		¥	—	§	"	©	ª	«	¬	SHY	®	-
0b0	°	±	—	—	´	µ	¶	·	¸	¸	º	»	—	—	—	¿
0c0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
0d0	—	Ñ	Ò	Ó	Ô	Õ	Ö	—	Ø	Ù	Ú	Û	Ü	—	—	ß
0e0	à	Á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
0f0	—	Ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	—	—	ÿ

Figure 4.1 Unicode excerpt: Basic Latin and Latin-1 Supplement (U+0000–U+00FF).

acters, which are used for most European languages. The capital A in French is the same character as the one 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 4.1 is used in Greek and Cyrillic text too. However, periods in languages such as Armenian, Arabic, Ethiopic, Chinese, and Korean are shaped differently and have their own Unicode representations. The multifunction ASCII hyphen is retained for the purpose of round-trip compatibility, but new codes are defined under “general punctuation” (codes 2000 and up in Table 4.2) 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 4.2).

Unicode does distinguish letters from different scripts even though they may look identical. For example, the Greek capital alpha looks just like the Roman capital A, but receives its own code in the Greek block (codes 0370 and up in Table 4.2). This allows you to downcase 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 essentially 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

100	Ā	Ā	Ā	ā	Ą	ą	Ć	ć	Ĉ	ĉ	Ċ	ċ	Č	č	Ď	ď
110	Đ	Đ	Ē	ē	Ě	ě	É	é	Ė	ė	Ê	ê	Ĝ	ĝ	Ğ	ğ
120	Ġ	Ġ	Ģ	ģ	Ĥ	ĥ	Ħ	ħ	Ĩ	ĩ	Į	į	İ	ı	Ĳ	ĳ
130	Ĵ	ĵ	Ķ	ķ	κ	Ĺ	ĺ	Ł	ł	Ł	ł	Ł	ł	Ł	ł	Ł
140	Ŧ	ŧ	Ũ	ũ	Ů	ů	Ű	ű	Ų	ų	Ŵ	ŵ	Ŷ	ŷ	Ÿ	Ź
150	Ž	ž	Ž	ž	Ž	ž	Ž	ž	Ž	ž	Ž	ž	Ž	ž	Ž	ž
160	Š	š	Š	š	Š	š	Š	š	Š	š	Š	š	Š	š	Š	š
170	Ų	ų	Ų	ų	Ų	ų	Ų	ų	Ų	ų	Ų	ų	Ų	ų	Ų	ų

(a)

400	Ё	ѐ	ѐ	ѐ	ё	ѕ	і	ї	ј	љ	њ	ѣ	ќ	ѝ	џ
410	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О
420	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю
430	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о
440	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю
450	ё	ђ	ѐ	ѐ	ё	ѕ	і	ї	ј	љ	њ	ѣ	ќ	ѝ	џ

(b)

Figure 4.2 Unicode excerpts: (a) Latin Extended A (U+0100–U+017F); (b) Cyrillic (U+0400–U+045F).

along the way, what languages it is used in, how it relates to other characters, how it can be transliterated or transcribed. We cannot tell these stories here: to hear them you will have to refer to other sources.

As an example of diversity, Figure 4.2a shows some of the Extended Latin characters, while Figure 4.2b 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 still 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.

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 COMBINING 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 combining 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*—the original dot in the base letter is omitted. 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 also has single code points that correspond to precisely the same units as character combinations—in fact, they are part of the Latin-1 supplement shown in Figure 4.2a. 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 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 such as Hangul, the Korean syllabic script. In Unicode, Hangul is covered by 11,172 precomposed symbols (codes AC00 and up in Table 4.2), 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 4.2). 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 *ï*. *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 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 (for 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 Angstrom 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 32 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 comes 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, where UTF stands for “UCS Transformation Format” (a nested acronym: UCS is Unicode Character Set)—so called because Unicode characters are “transformed” into this encoding format. 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 also is complicated by the endian question, which is resolved in the same way. For almost all text the UTF-16 encoding requires half the space needed for UTF-32.

It is convenient to define 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. ASCII characters are automatically 1-byte UTF-8 codes; existing ASCII files are valid UTF-8. Being byte-oriented, this scheme avoids the endian issue that complicates UTF-32 and UTF-16. We explain these more fully in the following sections.

UTF-32

In Figure 4.3 the word *Welcome* in five different languages has been converted to Unicode (Figure 4.3a) and encoded in the three different UTF methods (Figure 4.3b). UTF-32 maps the universal character set to 4-byte integers, and Unicode is turned into UTF-32 by dropping the *U+* prefix.

Unicode		
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 ...
Akwāba	(Fante from Ghana)	U+0041 U+006B U+0077 U+00E4 ...

(a)

	UTF-32	UTF-16	UTF-8
Welcome	00000057000000650000006C00000063 ...	00570065006C0063 ...	57656C63 ...
Haere mai	00000048000000610000006500000072 ...	0048006100650072 ...	48616572 ...
Wilkommen	00000057000000690000006C0000006B ...	00570069006C006B ...	57696C6B ...
Bienvenue	0000004200000069000000650000006E ...	004200690065006E ...	4269656E ...
Akwāba	000000410000006B00000077000000E4 ...	0041006B007700E4 ...	416B77C3A4...

(b)

Figure 4.3 Encoding *Welcome* in (a) Unicode; (b) UTF-32, UTF-16, and UTF-8.

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. 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 4.3b. However, two variants are defined: UTF-32BE and UTF-32LE for big-endian and little-endian, respectively.

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 4.2, is defined as “zero-width no-break space.” Its byte-for-byte transpose is carefully 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 4.3b. 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 4.2, 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.

Table 4.3 Encoding the Unicode character set as UTF-8.

Unicode value	21-bit binary code	UTF-8 code
U+00000000 – U+0000007F	0000000000000000wwwwww	0wwwwww
U+00000080 – U+000007FF	0000000000wwwwxxxxxx	110wwww 10xxxxxx
U+00000800 – U+0000FFFF	00000wwwxxxxxxxxyyyyyy	1110www 10xxxxxx 10yyyyyy
U+00010000 – U+001FFFFF	wwwxxxxxxxxxxxxxxxxzzzzz	11110www 10xxxxxx 10yyyyyy 10zzzzz

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 4.3. 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 4.3b 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 4.3.

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.

Hindi and related 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 used by some 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. These languages raise subtle problems that are difficult for people of European background to appreciate, and we can only give a glimpse of the complexities involved. As Table 4.2 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,

Devanagari	अ आ इ ई उ ऊ ऋ ॠ एँ ऐँ ओँ औ क ख ग घ ङ च छ ज झ
Bengali	অ আ ই ঈ উ ঊ ঋ ৠ ঐ ঐ ঔ ঐ ঔ ঋ ঌ ঍ ঎ এ ঐ ঐ ঔ ঐ ঔ
Gurmukhi	ਅ ਆ ਇ ਈ ਉ ਊ ਏ ਐ ਓ ਔ ਐ ਐ ਐ ਐ ਐ ਐ ਐ ਐ ਐ ਐ ਐ
Gujarati	અ આ ઇ ઈ ઉ ઊ ઋ ઋ ઐ ઐ ઐ ઐ ઐ ઐ ઐ ઐ ઐ ઐ ઐ ઐ ઐ
Oriya	ଅ ଆ ଇ ଈ ଉ ଊ ଋ ଋ ଐ ଐ ଐ ଐ ଐ ଐ ଐ ଐ ଐ ଐ ଐ ଐ ଐ
Tamil	அ ஆ இ ஐ உ ஊ எ ஏ ஐ ஒ ஓ ஔ கஙசஜடதந
Telugu	అ ఆ ఇ ఈ ఉ ఊ యు ఎ ఏ ఐ ఒ ఓ ఔ క ఖ గ ఘ జ చ ఛ జ యు
Kannada	ಅ ಆ ಇ ಈ ಉ ಊ ಯು ಎ ಏ ಐ ಒ ಓ ಔ ಕ ಖ ಗ ಘ ಜ ಚ ಛ ಜ ಯು ಇ
Malayalam	അ ആ ഇ ഊ ഉ ഊ ഊ ഊ ഊ ഊ ഊ ഊ ഊ ഊ ഊ ഊ ഊ ഊ ഊ

Figure 4.4 Examples of characters in Indic scripts.

Oriya, Assamese, Tamil, Telugu, Kannada, Malayalam, Urdu, Sindhi, and Kashmiri. The first twelve 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, Malayalam (southern or *Dravidian* ones). Figure 4.4 gives some characters in each of these. As you can see, the characters are rather beautiful—and the scripts are quite different from one another. Unicode also includes a script for Sinhalese, the official language of Sri Lanka.

Hindi, the official language of India, is written in Devanagari,³ which is used for writing Marathi and Sanskrit too. (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 there is one additional glyph and another glyph 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.

3. Pronounced *Dayv'nagri*, with the accent on the second *a*.

Since the 1970s various committees of the Indian Department of Official Languages and Department of Electronics have worked on devising codes and keyboards that cater to all official Indic scripts. A standard keyboard layout was developed that provides a uniform way of entering them all. A common code was defined so that any software that was developed could be used universally. This is possible because, despite the very different scripts, the alphabets are phonetic and have a common Brahmi root that was used for the ancient Sanskrit. The simultaneous availability of multiple Indic languages was intended to accelerate technological development and 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 the ASCII code set that, like other extensions, places new characters in the upper region of the code space. The code table caters for all the characters required in the Brahmi-based Indic scripts. Figure 4.5a shows the ISCII code table for the Devanagari script; code tables for the other scripts in Figure 4.4 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 4.5a), and 18 accents and combining characters. There are also three special escape codes, but we will not go into their meaning here.

The Unicode developers adopted ISCII lock, stock, and barrel—they had to, because of their policy of round-trip compatibility with existing coding methods. 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—which in the original ISCII design were supposed to be formed from combinations of other keystrokes. Figure 4.5b 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)

Although not part of the ISCII character set, this can be created from the keyboard by typing the sequence of characters

उ ँ (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

A0		०	:	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	ढ	ण	त	थ	द	ध	न	त	प	फ	ब	भ	म	य	र
D0	ॠ	ॡ	ल	ळ	व	श	ष	स	ह	INV	।	।	।	।	।
E0	ॢ	ॣ	।	॥	०	१	२	३	४	५	६	७	८	९	०
F0	EXT	०	१	२	३	४	५	६	७	८	९				ATR

(a)

0900		०	:	अ	आ	इ	ई	उ	ऊ	ऋ	ॠ	ऐ	ए	ऐ	ँ
0910	ऐ	ऑ	ओ	औ	क	ख	ग	घ	ङ	च	छ	ज	झ	ञ	ट
0920	ठ	ड	ढ	ण	त	थ	द	ध	न	त	प	फ	ब	भ	म
0930	र	ॠ	ल	ळ	व	श	ष	स	ह			।	।	।	।
0940	ॢ	ॣ	।	॥	०	१	२	३	४	५	६	७	८	९	०
0950	ॠ	ॡ	ॢ	ॣ	।	॥	०	१	२	३	४	५	६	७	८
0960	ॠ	ॡ	ॢ	ॣ	।	॥	०	१	२	३	४	५	६	७	८
0970	ॠ	ॡ	ॢ	ॣ	।	॥	०	१	२	३	४	५	६	७	८

(b)

20		!	'	०	ॠ	ॡ	ॢ	ॣ	।	॥	०	१	२	३	४
30	०	१	२	३	४	५	६	७	८	९	:	:	इ	उ	ऊ
40	ऋ	ॠ	ए	व	क	ख	ग	घ	ङ	च	छ	ज	झ	ञ	ट
50	ठ	ड	ढ	ण	त	थ	द	ध	न	त	प	फ	ब	भ	म
60	ॠ	ॡ	ॢ	ॣ	।	॥	०	१	२	३	४	५	६	७	८
70	ॠ	ॡ	ॢ	ॣ	।	॥	०	१	२	३	४	५	६	७	८
80															
90															
A0		फ	फ	फ	ल	ळ	व	श	ष	स	ह	।	।	।	।
B0	ॠ	ॡ	ॢ	ॣ	।	॥	०	१	२	३	४	५	६	७	८
C0	ॠ	ॡ	ॢ	ॣ	।	॥	०	१	२	३	४	५	६	७	८
D0	ॠ	ॡ	ॢ	ॣ	।	॥	०	१	२	३	४	५	६	७	८
E0	ॠ	ॡ	ॢ	ॣ	।	॥	०	१	२	३	४	५	६	७	८
F0															०

(c)

Figure 4.5 Devanagari script: (a) ISCII; (b) Unicode (U+0900–U+0970); (c) code table for the Surekh font.

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 the Indic scripts contain numerous conjuncts, which are clusters of two to four consonants without any intervening vowels. Conjuncts are the same idea as the ligatures discussed earlier, characters represented by a single glyph whose shape differs from the shapes of the constituent consonants—just as the sequence *f* followed by *i* is joined together into the single ligature *fi* in the font used for this book. 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

ल

ृ (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 ISCII 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.

Figure 4.5c 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 4.5b 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 using the font 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—this 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 feel it is 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 4.5c. We will learn in Section 4.3 how such rules can be embodied in “composite fonts.” 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 such as 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 4.5c, 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 for the Malayalam language diverse font-specific codes are used. To read a new Malayalam newspaper in your Web browser often involves downloading a new font!

To accommodate such documents in a digital library that represents documents internally in Unicode, it is necessary to implement several mappings:

1. from ISCII to Unicode, so that ISCII documents can be incorporated
2. from various different font representations (such as ISFOC, used for the Surekh font) to Unicode, so that documents in other formats can be accommodated
3. 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 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 4.6 shows an example page produced by such a scheme.

Using Unicode in a digital library

Unicode encoding may seem complex, but it is straightforward to use in digital library software. 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*. Read-

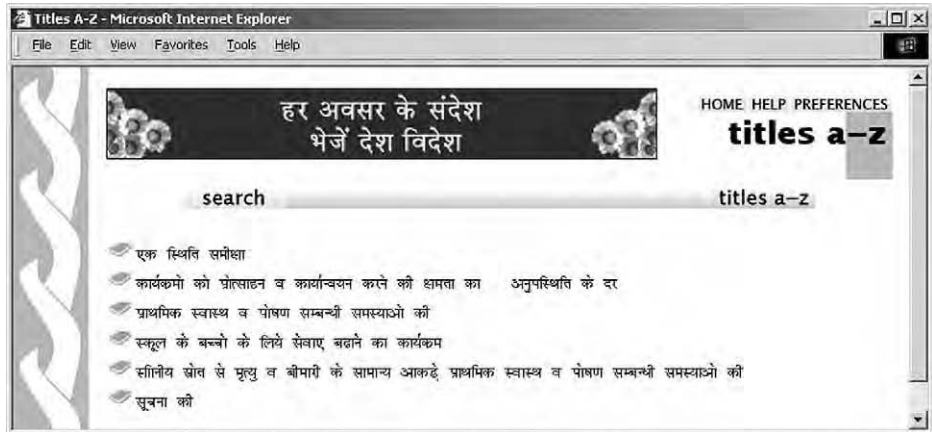


Figure 4.6 Page produced by a digital library in Devanagari script.

ability 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, such as the type `wchar_t` defined in the ANSI/ISO standard to represent “wide” characters, can be used.

Further work is necessary to treat surrogate characters properly. But usually operation can safely 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 work with Unicode level 1—which makes it far easier to implement string matching operations. For higher Unicode levels you should seek library support for normalization and matching.

When writing Unicode data structures to disk, it is easy to convert the 16-bit characters to UTF-8, reversing the process when reading. In most common situations this greatly reduces file size and also increases portability because the files do not depend on endianness.

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 Unicode character to a displayable character in a known font. Complications include composite Unicode characters and the direction in which the character sequence is displayed. By working through a modern Web browser, digital libraries can avoid having to deal with these issues explicitly.

4.2 Representing documents

Unicode provides an all-encompassing form for representing characters, including manipulation, searching, storage, and transmission. Now we turn our 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.

Plain text

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, http, or FTP—the various programs involved may make different assumptions. These alphabet mismatches often mean that character values in the range 128–255 are displayed incorrectly.

Formatting within such a document is rudimentary. Explicit line breaks are usually included. Paragraphs are separated by two consecutive line breaks, or 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 such as dashes (two hyphens in a row). Headings are underlined manually using rows of hyphens, or equal signs for double underlining. Emphasis is often indicated by surrounding text with a single underscore (like this), or by 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 DOS (and Windows) use 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.⁴ While the meaning of the message is not obscured, the effect is rather distracting.

4. ^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 ^.

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 become mystified.

In a digital library, plain text is a simple, straightforward, but impoverished representation of documents. 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 this 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 document collections.

Indexing

Rapid searching of the full text of large document collections for particular words, sets of words, or sequences of words is a core function of digital libraries that distinguishes them from physical libraries. The ability to search full text adds great value to documents used for study or reference, although it is rarely applicable to recreational reading, which normally takes place sequentially, in one pass.

Before computers, full-text searching was confined to highly valued—often sacred—works for which a concordance had already been prepared. For example, some 300,000 word appearances are indexed in Cruden's concordance of the Bible, printed on 774 pages. They are arranged alphabetically, from *Aaron* to *Zuzims*, and any particular word can be located quickly using a binary search. Each probe into the index halves the number of potential locations for the target, and the correct page for an entry can be located by looking at no more than 10 pages—fewer if the searcher interpolates the position of an entry from the position of its initial letter in the alphabet. A term can usually be located in a few seconds, which is not bad considering that only elementary manual technology is being employed. Once an entry has been located, it gives a list of references that the searcher can follow up. Figure 4.7 shows some of Cruden's concordance entries for the word *search*.

In digital libraries searching is done by a computer rather than a person, but essentially the same techniques are used. The difference is that things happen a

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 to within a unit such as a paragraph, chapter, or document, yielding a coarser index—partly because pointers can be smaller, but chiefly because if 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 (Section 4.7).

The basic function of a full-text index is to provide, given 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 be paragraphs or chapters 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 enough 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 Chapter 3 (Section 3.3), 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 satisfies 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 must be sorted according to the ranking heuristic that is in place. As Section 3.3 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; 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 creating an index 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 to a far smaller size—perhaps four numeric characters, only indexing numbers less than 9,999. 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 impacts certain technical aspects of the indexing procedure. Years, however, at four digits, should be preserved as single words.

In some languages plain text presents special problems. Languages such as Chinese and Japanese 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 these 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.

This causes obvious problems in full-text indexing: to get a bag of words we have to identify the words first. 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 you 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.

Readers unfamiliar with Chinese can gain an appreciation of the problem of multiple interpretations from Figure 4.8a, which shows two alternative interpretations of the same character sequence. The text is a joke that relies on the ambiguity of phrasing. Once upon a time, the story goes, a man set out on a long journey. But before he could return home the rainy season began, and he had to take 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 4.8a. The intended interpretation is shown in the second line, which means "It is raining, the god would like the guest to stay. Although the god wants you to stay, I do not!" On seeing the note, the visitor added the punctuation shown in the third line, making three sentences whose meaning is totally different—"The rainy day, the staying day.

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>

(a)

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>

(b)

Figure 4.8 Alternative interpretations of two Chinese sentences: (a) ambiguity caused by phrasing; (b) ambiguity caused by word boundaries.

Would you like me to stay? Sure!” (Nevertheless, according to the story he did take the hint, leaving the amended note as a joke.)

This example relies on ambiguity of phrasing, but the same kind of problem can arise with word segmentation. Figure 4.8b 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 Chinese readers are accustomed to inferring the corresponding sequence of words as they read. Accordingly machine-readable versions are invariably stored in unsegmented form. If they are to be incorporated into a digital library that offers full-text retrieval, a segmentation scheme should be used to insert word boundaries at appropriate positions when the text is indexed.

One way of segmenting text 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 more than one valid segmentation, and heuristics must be sought to resolve ambiguities.

Another segmentation method is based on the insight that text divided into words is more compressible than text that lacks word boundaries. You can see 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% 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 training data that has been segmented by hand. Between every pair of characters lies a potential space. Segmentation can be achieved by training a text compression model on presegmented text and using a search algorithm to interpolate spaces in a way that maximizes the overall compression of the text. The “Notes and sources” section at the end of the chapter (Section 4.7) points to a fuller explanation of the technique.

For non-Chinese readers, the success of the space-insertion method can be illustrated by applying it to English text. Table 4.4 shows, at the top, some original text, with spaces in the proper places. Below is the input to the segmentation procedure. Underneath that is the output of a dictionary word-based segmentation scheme and of a character-based one that uses the compression method. The training text was a substantial sample of English, although far smaller than the corpus used to produce the dictionary for the word-based method.

Word-based segmentation fails badly when the words are not in the dictionary. In this case neither *crocidolite* nor *Micronite* were, and they are segmented incorrectly. In addition, *inits* is treated as a single word because it occurred that

Table 4.4 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.

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. This method makes two errors, however. First, a space was not 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.

This brings to a close our discussion on plain text documents. We now move on to richer document representations that cater to combined text and graphics.

4.3 Page description languages: PostScript and PDF

Page description languages allow typeset pages to be expressed in a way that is independent of the particular output device being used. Early word-processing programs and drawing packages incorporated code for output to particular printers and could not be used with other devices. With the advent of page description languages, programs can generate graphical documents in a device-independent format which will print on any device equipped with a driver for that language.

Most of the time digital libraries can treat documents in these languages by processing them using standard “black boxes”: generate this report in a particular language, display it here, transfer it there, and print. However, to build coherent collections out of 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 now describe some page description languages in detail.

PostScript

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.

PostScript files comprise a sequence of graphical 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 compose 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 300 to 600 dpi, and publishing houses use 1,200 dpi and above (see Table 2.4 in Chapter 2). This very book is an example of what can be described using the language.

Modern computers are sufficiently powerful 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 how it was intended. 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 we saw with 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, a scheme called the Portable Document Format (PDF) has been devised as the successor to PostScript (see subsection “Portable Document Format: PDF”) for online documents.

PostScript graphics

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 4.5 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*, a type of well-defined curved line whose shape is 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 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 and can be rotated, translated, scaled, and colored just like any other object. However,

Table 4.5 Graphical components in PostScript.

Component	Description
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

because of its importance, text comes in for some special treatment. The PostScript interpreter stores information about the current font: font type, typeface, point size, and so on, and operators such as *findfont* and *scalefont* are provided to manipulate these components.

There is also a special operator called *image* for sampled images.

The PostScript language

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 4.9b shows a simple PostScript program that, when executed, produces the result in Figure 4.9a, which contains the greeting *Welcome* in the five languages we used earlier to illustrate Unicode. The first line, which is technically a 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 many 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 4.9b 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, ISO Latin1 Encoding or composite fonts, but they require syntax beyond the scope of this simple example.

Levels of PostScript

Standards and formats evolve. There is a tension between stability, an important feature for any language, and currency, or the need to extend in response to the

Welcome Haere mai Willkommen Bienvenue Akwäba

(a)

```
%!PS-Adobe-3.0
/Helvetica findfont
14 scalefont
setfont
10 10 moveto
(Welcome ) show
(Haere mai ) show
(Wilkommen ) show
(Bienvenue ) show
(Akw) show 2 0 rmoveto (\310) show -6 0 rmoveto (aba ) show
showpage
```

(b)

```
%!PS-Adobe-3.0 EPSF-3.0
%%Creator: Dr David Bainbridge
%%Title: Welcome example
%%BoundingBox: 0 0 350 35
%%DocumentFonts: Helvetica
%%EndComments
/Helvetica findfont
14 scalefont
setfont
10 10 moveto
(Welcome ) show
(Haere mai ) show
(Wilkommen ) show
(Bienvenue ) show
(Akw) show 2 0 rmoveto (\310) show -6 0 rmoveto (aba ) show
showpage
```

(c)

Figure 4.9 (a) Result of executing a PostScript program; (b) the PostScript program; (c) Encapsulated PostScript version; (d) PDF version; (e) network of objects in the PDF version; (f) RTF specification of the same document. (continued on the following pages)

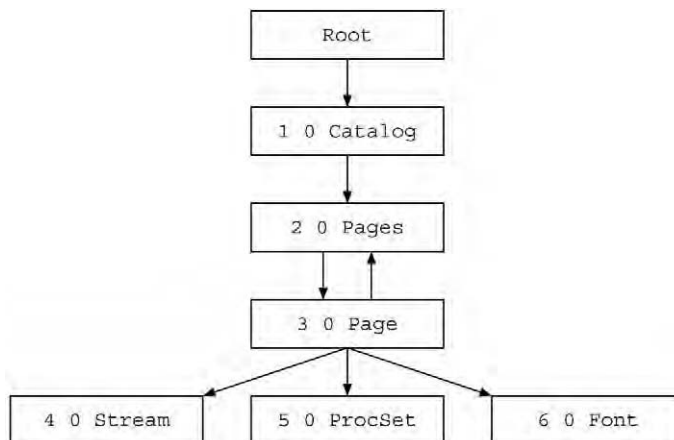
ever-changing face of computing technology. To help resolve the tension, *levels* of PostScript are defined. The conformance level of a file is encoded in its first line, as can be seen in Figure 4.9b (*PS-Adobe-3.0* means Level 3 PostScript). Care is taken to ensure that levels are backward compatible.

What we have described so far is basic Level 1 PostScript. Level 2 includes

- improved virtual memory management
- device-independent color
- composite fonts
- filters

<pre> %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 </pre>	<pre> 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 </pre>
--	--

(d)



(e)

Figure 4.9 (continued)


```
{\rtf1\ansi\deff0{\fonttbl{\f0\froman Times;}{\f1\fswiss Helvetica;}}
{\info{\title Welcome example}{\createm\yr2001\mo8\dy10}{\nofpages1}
}\pard\plain\f1\fs28\uc0
Welcome
Haere mai
Wilkommen
Bienvenue
Akw\u228ba
\par}
```

(f)

Figure 4.9 (continued)

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 certain 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

Document structuring conventions

While PostScript per se does not enforce an overall structure to 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 prologue, 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 such as word processors that generate PostScript commonly use the prologue 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 of PostScript designed for expressing documents of a single page or less. It is widely used to incorporate artwork created using one software application, such as 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 4.9c 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 drawing size 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 4.9c 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—which contains raw PostScript along with a few special comments—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 composite character—making them suitable for languages with large alphabets, such as Chinese, and with frequent character combinations, such as Korean.

In the *Welcome* example of Figure 4.9b, 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 such as */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.

Base font formats

The original specification for PostScript included a means for 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 the company that introduced PostScript, Adobe, kept the font specification secret. This spurred Apple to introduce a new font description format six years later, which 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 problem 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 more important for screen displays.

Composite fonts

The essence of composite fonts, which became standard in Level 3 PostScript, boils down to two key concepts. First, instead of mapping character values through a single dictionary as base fonts do, there is now a hierarchy of dictionaries. At its root a composite font dictionary directs character mappings to subsidiary dictionaries. These can either contain 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 font 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 which 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 such as paragraph boundaries and font characteristics must be identified from PostScript.

Although PostScript allows complete flexibility in how documents are described (for example, the characters do not even have to be in any particular order), 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 4.10 shows an example, along with the text extracted from it. Characters to be placed on the page appear in the PostScript file as parenthesized 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 together 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 wide 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.

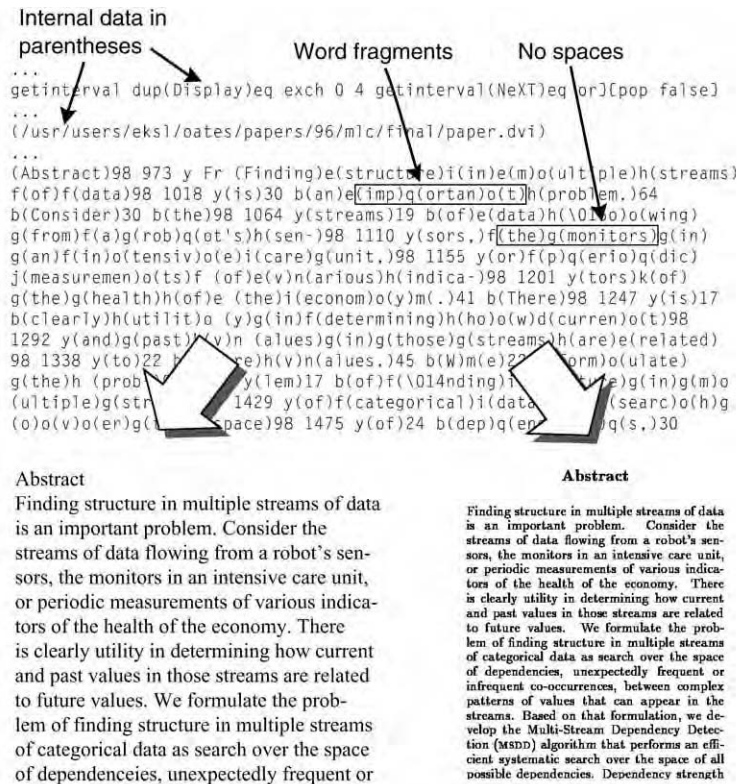


Figure 4.10 A PostScript document and the text extracted from it.

By prepending a PostScript code fragment to a document and then running it 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 the PostScript *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 4.11a, to the document of Figure 4.10 redefines the *show* operator. The effect is to define the name *show* to read *print* instead—and therefore print the characters to a file. The result appears at the right of Figure 4.11a. One problem has been solved: winnowing the text destined for a page from the remainder of the parenthesized text in the original file.

```
/show { print } def
```

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

(a)

```
/show { print ( ) print } def
```

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)

```
/X 0 def
/show {
  currentpoint pop
  X sub 5 gt { ( ) print } if
  dup print
  systemdict /show get exec
  currentpoint pop /X exch def
} def
```

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)

```
/X 0 def
/protoshow {
  currentpoint pop
  X sub 5 gt { ( ) print } if
  dup print
  systemdict exch get exec
  currentpoint pop /X exch def
} def
```

```
/show      { /show      protoshow } def
/kshow     { /kshow     protoshow } def
/widthshow { /widthshow protoshow } def
/ashow     { /ashow     protoshow } def
/awidthshow { /awidthshow protoshow } def
```

(d)

Figure 4.11 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.

The problem of identifying whole words from fragments must still be addressed, for the text in Figure 4.11a contains no spaces. Printing a space between each fragment yields the text in Figure 4.11b. 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, whereas if a space intervenes the distance is larger. An appropriate threshold will depend on the type size and should be chosen accordingly; however, a fixed value will be used for illustration.

The program fragment in Figure 4.11c 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.

In order to record the new *x* coordinate, the fragment must actually be rendered. Unfortunately Figures 4.11a 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 and executes it 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 at the right of Figure 4.11c, 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 4.11d 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 returns, the code for the appropriate command is located in the system dictionary and executed.

Improving the output

Notwithstanding the use of fixed thresholds for word and line breaks, this scheme is quite effective at extracting text from many PostScript documents. However, several enhancements can be made to improve the quality of the output. First, fixed thresholds fail when the text is printed in an unusually large or small font. With large fonts, interfragment gaps are mistakenly identified as interword gaps, and words break up. With small ones, interword gaps are mistaken for interfragment gaps, and words run together.

To solve this problem the word-space threshold can be expressed as a fraction of the average character width. This is calculated for the fragments on each side of the break by dividing the rendered width of the fragment by the number of characters in it. As a rule of thumb, the interword threshold should be about 30 percent greater than the average character width.

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.

Sometimes paragraphs are distinguished by horizontal indentation rather than by vertical spacing. Treating indented lines as paragraph breaks sometimes fails, however—for example, quotations and bulleted text are often indented too. Additional heuristics are needed to detect these cases. For example, an indented line may open a new paragraph if it starts with a capital letter; if its right margin and the right margin of the following line are at about the same place; and if the following line is not also indented. Although not infallible, these rules work reasonably well in practice.

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.

Fourth, when documents are justified to a fixed right margin, words are often hyphenated. Output will be improved if this process is reversed. But simply deleting hyphens from the end of lines inadvertently removes hyphens from compound words that happen to straddle line breaks.

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 should be 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

Digital libraries are often built from PostScript source documents. 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 with printing—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 such as 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 a 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. Even though 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 sampled 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 digital library repositories commonly include legacy documents in Level 1 PostScript.

The ideas behind PostScript make it attractive for digital libraries. However, there are caveats. First, it was not designed for online display. Second, because documents are programs, they do not necessarily produce the same result when run on different interpreters. Third, 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 will encounter mysterious errors when viewing certain documents. Fourth, 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, for Portable Document Format, is a page description language that arose out of PostScript and addresses many of 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. 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 such as compression and encryption are built in—for 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 in 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 is 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.

Inside a PDF file

Figure 4.9d is a PDF file that produces an exact replica of Figure 4.9a. 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 4.9d. 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 4.9d defines the graph structure in Figure 4.9e. 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 4.9d 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 4.9d 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 4.9e. 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 very 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 4.9d 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 4.9d) 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 UTF-16 text, although non-Unicode extensions are also supported.

Features of PDF

The PDF object network supports a variety of different browsing features. Figure 4.12 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

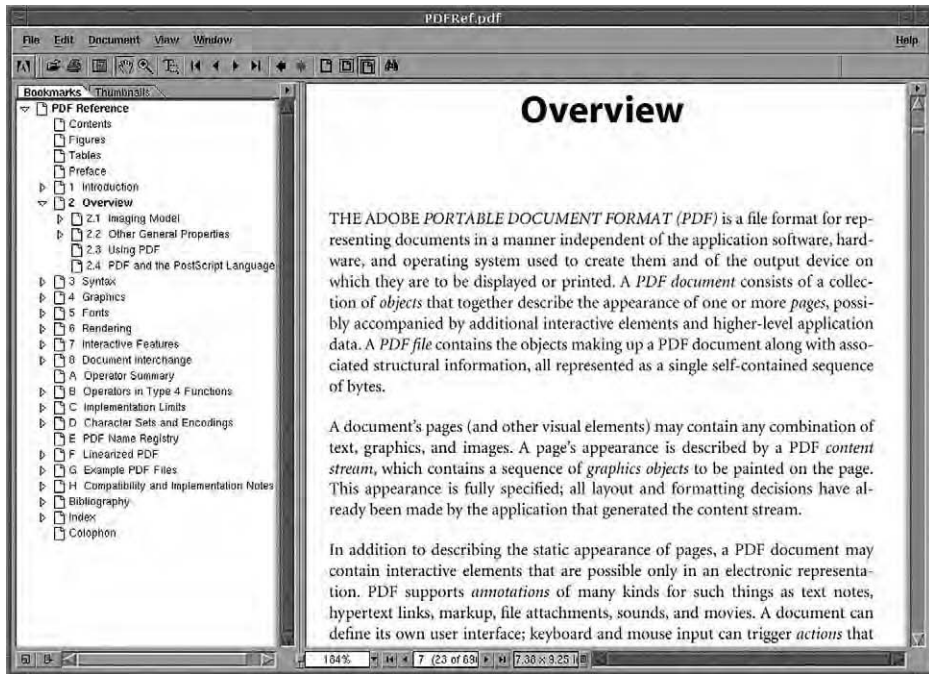


Figure 4.12 Reading a bookmark-enabled PDF document with Acrobat.

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 4.9e, 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 appropriately. 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 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. 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 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.

PDF and PostScript

PDF is a sophisticated document description language that was developed by Adobe, the same company that developed PostScript, as a successor to it. PDF addresses various serious deficiencies that had arisen with PostScript, principally lack of portability. While PostScript is really 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).

PDF incorporates additional features that support online display. Its design draws on expertise that ranges from traditional printing right through to hypertext and structured document display. PDF 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. PDF is not a full programming language, however, 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 lack software to generate PDF. However, the world is changing: the Apple Macintosh computer now displays all online graphics as PDF. (Recall the parallel, mentioned earlier, of a 1980s operating system that controlled the display entirely from PostScript.)

From a digital library perspective, collections 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.

4.4 Word-processor documents

When word processors store documents, they store them in ways that are specifically designed to allow the documents to be edited. There are numerous different formats: we will take Microsoft Word—currently a leading product—as an illustrative example. This has two different styles of document format: Rich Text Format (RTF), a widely published specification dating from 1987, and a proprietary internal format that we call simply *native Word*. As an example of a completely different style of document description language, we end this section by describing 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 rather difficult to learn and is 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 PowerPoint (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 one of those Chinese stone tablets. PDF, as we have seen, supports limited forms of editing—adding annotations or page numbers, for example—but is not designed to have anything like the flexibility of RTF.

Many online documents are in native Word format. Because it is a binary format, it is far more compact than RTF—which translates to 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 present a seamless interface. But there are disadvantages. Non-Microsoft communities may be locked out of digital libraries unless other formats are offered. Although documents can be converted to forms such as HTML using scriptable utilities, Word's proprietary nature makes this a challenging task—and it is hard to keep up to date. Even Microsoft products sometimes can't read Word documents properly. Native Word is really a family of formats rather than a single one and has nasty legacy problems.

Rich Text Format

Figure 4.9f recasts the *Welcome* example 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 page 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*.

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 a formatting 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 4.9f 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, it 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 4.9f’s opening line, the numeric parameter of `\rtf1` gives the version number, 1. The format has grown rapidly because, as well as keeping up with developments such as 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 basic 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 `\def0` in Figure 4.9f 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 which 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 4.9f 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 revealing 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 current 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 4.13 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 (mandatory). Both appear in the *Welcome* example, although the font table has many more capabilities, including the ability to embed fonts.

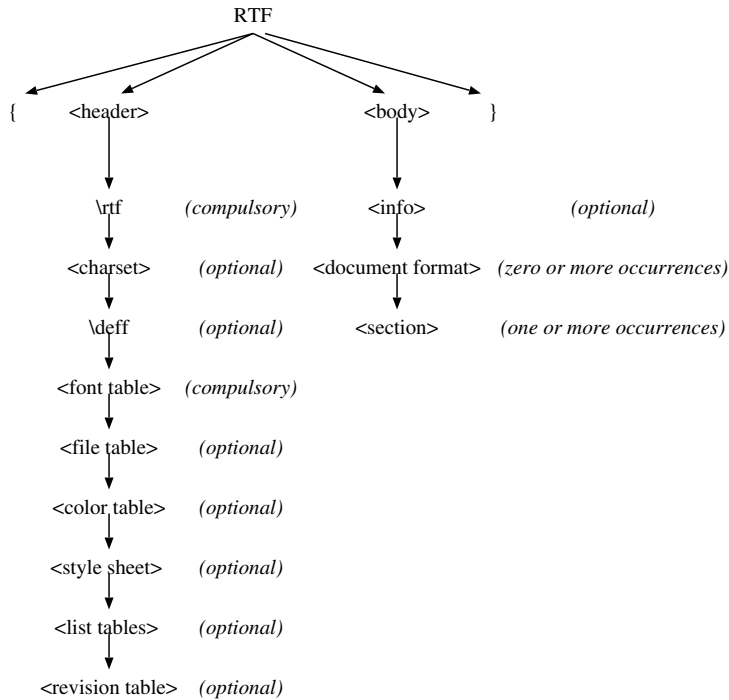


Figure 4.13 Structure of an RTF file.

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 commands such as `\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 4.13: 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, in the last part of the body the document text is specified. 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* here correspond to section breaks inserted by an author using, for instance, Microsoft Word. Sections and paragraphs can both begin with formatting instructions. For sections, these 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, they 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 described how RTF describes typographic text, based around the structure of sections and paragraphs. There are many other features. Different sampled image formats are supported, including open standards such as JPEG and PNG (described in Section 4.5) and proprietary formats such as 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 the latter case care must be taken when transferring the file between operating systems (recall the discussion of FTP's new-line handling in Section 4.2).

Many word processors incorporate a built-in tool that draws lines, boxes, arcs, splines, filled-in shapes, text, and other vector graphic primitives. To support this RTF contains over 100 commands to draw, color, fill, group, and transform such shapes. These resemble the graphical shapes that can be described in PostScript and PDF—not surprisingly, considering that all artwork will ultimately be converted to one of these forms for printing or viewing.

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 the current date, current page, 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. Using fields, metadata such as title and author can be associated with a

document, and this information is stored in the RTF file in the form of an `\info` command. RTF also supports an index of entries and a table of contents, which are also implemented using the field mechanism.

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) which 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—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.

Use in digital libraries

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. For other platforms or for speedier access, you might consider offering documents in a different format. For example, software is available to convert an RTF document to an approximate equivalent in HTML.

Native Word formats

The native Microsoft Word format is proprietary and its details are shrouded in mystery. Although Microsoft has published “as is” their internal technical manual for the Word 97 version, the format continues to evolve. Primarily a binary format, the abstract structures deployed reflect those of RTF. Documents include summary information, font information, style sheets, sections, paragraphs, headers, footers, bookmarks, 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.”

Use in digital libraries

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 need a more widely supported option.

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 in the final version. The solution is simple: switch this option off and resave all documents (using scripting).

LaTeX format

LaTeX—pronounced *la-tech* or *lay-tech*—takes a different approach to document representation. Word processors present users with a “what you see is what you get” interface that is dedicated specifically to hiding 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 details of internal representation. 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 and 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 adopted enthusiastically by members of the academic scientific and technical community. It is a nonproprietary system, and excellent implementations are freely available.

Figure 4.14 shows a simple example that we use for illustration. Commands in the LaTeX source (Figure 4.14a) are prefixed by the backslash character, \. All LaTeX documents have the same overall structure. They open with `\documentclass`, which specifies the document's principal characteristics—whether an article, report, book, and so forth—and gives options such as paper size, base font size, whether single-sided or back-to-back. Then follows a preamble which gives

```
\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)

Welcome example

I. H. Witten and D. Bainbridge

10 August 2001

1 Introduction

Welcome, Haere mai, Willkommen, Bienvenue, Akw'aba

2 Syntax

Latex syntax is little bit like RTF. It uses the \ character for special formatting commands: what you see as the end result is certainly *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:

- bullet point list
- enumerated list
- tables and figures
- drawn graphics
- ...

In particular Latex has a powerful maths mode capable of expressing complex equations. A rudimentary example is:

$$x \geq \sum_{i=0}^{\infty} \frac{1}{i^2 \pi}$$

(b)

Figure 4.14 (a) LaTeX source document; (b) printed result.

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 artwork, 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 standard Internet sites so that others can download them and access them through `\usepackage`.

When writing a document 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 4.14 contains examples. Structural commands include `\section`, which makes an automatically numbered section heading (`\section*` omits the numbering, and `\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 4.14a specifies a mathematical expression. The `\begin{displaymath}` and `\end{displaymath}` commands switch to a mode that is tuned to represent formulas, which enables new commands suited to this purpose. LaTeX contains many shortcuts—for example, math mode can alternatively be entered by using dollar signs as delimiters.

LaTeX and digital libraries

LaTeX is a prime source format for collections of mathematical and scientific documents. Of course these documents can be converted to PostScript or PDF form 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 such 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 lay hands on 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 what 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 LaTeX features.

4.5 Representing images

An image displayed on the screen or printed to paper is formed by a regular matrix of tightly packed dots, black-and-white or colored, called picture elements or *pixels*. Picture quality is determined by the number of pixels per linear unit, usually expressed in dots per inch (dpi), and the number of bits used to represent each pixel's color—typically 1 bit per pixel for black-and-white, 8 bits for grayscale, and anywhere from 8 to 32 bits for full color. Table 2.4 in Chapter 2 shows the typical resolution of an assortment of devices.

Represented in a computer file in the obvious way, images can occupy an inordinate amount of space. When a standard $8\frac{1}{2} \times 11$ inch page is scanned with 1 bit per pixel at a resolution of 300 dpi, a file of just over 1 Mb is produced. Using 8-bit grayscale yields a 9-Mb file; a 24-bit full-color image occupies 28 Mb. Higher spatial resolution greatly amplifies the problem: a 600-dpi scanner will quadruple file size to over 100 Mb in the full-color case. Even a 100-Gb disk can only hold 1,000 such images.

Images are not normally stored this way on disk. Usually they are compressed, and an important component of an image file format is the particular compression method used. Compression does not necessarily imply degradation in image quality. There are two broad classes: *lossless* and *lossy*. Lossless compression ensures that the decompressed file is exactly the same—bit for bit—as the original. For text compression this technique is always used because even a minor change (such as adding a zero at the end of a sum of money) can have a marked effect on the meaning. Lossy compression does not guarantee that the decompressed file will be exactly the same as the original, but tries to ensure that any errors are hard to discern. Its use is confined to applications where small errors are permissible. Often they are completely imperceptible.

Both kinds of compression are used for images. Scanned images may contain *digitization noise*, artifacts caused by the process of digitization itself. If a compression scheme can eliminate these, it will achieve economy while simultaneously *increasing* image quality. Lossy 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 com-

pressed from its original representation (one to four bytes) to well under a single bit, with a barely perceptible loss of quality. However, there are many situations in which exact representation is deemed essential, and lossless compression must be used. For example, some image data must be certified as an exact copy for medical or legal reasons. Archival storage of historical documents needs to be lossless because the requirements of future scholars cannot be anticipated.

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 spatial characteristics of the data. For example, most images contain large areas whose color and texture is constant or slowly varying, interrupted by regions of abrupt change such as lines or edges. Sophisticated image compression algorithms identify these regions and exploit them for compression.

There are countless image-compression formats, so we restrict attention to a few that are in widespread use today in Web applications and digitization projects. We begin by describing the GIF and PNG formats for lossless images, and then proceed to review JPEG, a comprehensive international standard for high-performance lossy compression of photographic images. Next we discuss an important technique called *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.

Lossless image compression: GIF and PNG

Two lossless image compression formats are in widespread use today: the pervasive (but proprietary) Graphics Interchange Format or GIF (usually pronounced *jiff*) and an open standard called Portable Network Graphics or PNG (pronounced *ping*).

GIF: Graphics Interchange Format

Originally specified in 1987, GIF was for many years the most widely used lossless image compression format. It was intended as an exchange medium for graphic images that could be displayed on a variety of graphics hardware platforms and was adopted by CompuServe in order to minimize the time required to download pictures over modem links.

GIF 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 table called a *color map* or *color lookup table* that represents a palette of colors for the image. The color map, which is stored with the image, comprises up to 256 different colors (corresponding to 8-bit pixel codes) and contains a

full 24-bit color specification for each—eight bits for each of 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 which lets the background on which the image appears show through.

The GIF format allows the color map to be tailored specifically for each individual image and given along with it as a prefix to the image file. Alternatively a group of images can share the same color map, or the map 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 a general-purpose scheme called LZW (for Lempel Ziv Welch, the names of its inventors). It was designed for text compression, not images, and exploits the fact that in text many short “phrases” (a few letters, or a few words, long) tend to repeat. However, it does not use a fixed dictionary of phrases like some compression methods. Instead it accumulates a list of phrases as they are encountered in the text file being processed. This strategy of adapting to the particular input file makes it 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. A feature is included to make it easy to skip through the sequence without having to decompress 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 that lies at its core. This caused widespread dismay because GIF was at the time the primary means of storing images on the World Wide Web. Unisys has since refined its position. Now the vendor of any software product capable of generating GIF files must negotiate a license; users of the product are then free to create all the files they like. As a result of the uncertainty instilled by the original announcements, many developers remain wary of using GIF images. Furthermore, the same patent issue also applies to other formats that incorporate LZW compression, including TIFF, PDF, and PostScript (Level 2 and above).

PNG: Portable Network Graphics

Unisys’s surprise announcement catalyzed the development of a new lossless image format, PNG, intended specifically 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 an earlier scheme called LZ77 (after the same Lempel and Ziv cited previously). Gzip performs better than LZW, and PNG usually achieves greater compression than GIF.

A more important factor in compression, however, is the fact that PNG 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 a small number of “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.⁵ 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 these 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 what is achieved by using gzip instead of 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 alternatively include up to 16 bits of grayscale, 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.

One restriction is that while GIF supports animated images, PNG does not—that is left to a different standard, Multiple-image Network Graphics or MNG (pronounced *ming*). Another caveat is that although all modern Web browsers can display PNG images, ancient ones cannot.

Lossy image compression: JPEG

JPEG, named after the Joint Photographic Experts Group that designed it, is a comprehensive standard intended for representing compressed continuous-tone images. It is general purpose and underlies a gamut of image communication services and image applications—desktop publishing, graphic arts, color facsimile, newspaper wirephoto transmission, and medical imaging—as well as hardware devices such as 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.

-
5. 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.

JPEG's compression algorithm works well enough to produce excellent image quality at around 1 bit per 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 per pixel. No lossless image 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. Consequently JPEG does not reconstruct the original image exactly (although the standard incorporates an alternative coding method, called JPEG-Lossless, that does). 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.

The JPEG compression method

Baseline JPEG 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 4.15 depicts the encoding and decoding processes. Images are first divided into 8×8 pixel blocks. Each block is subjected to a signal-processing technique known as the *discrete cosine transform*, which maps the 64 pixel values into 64 numbers called *spatial frequency coefficients*. The transform is reversible: these 64 coefficients characterize the input 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 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 4.16 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

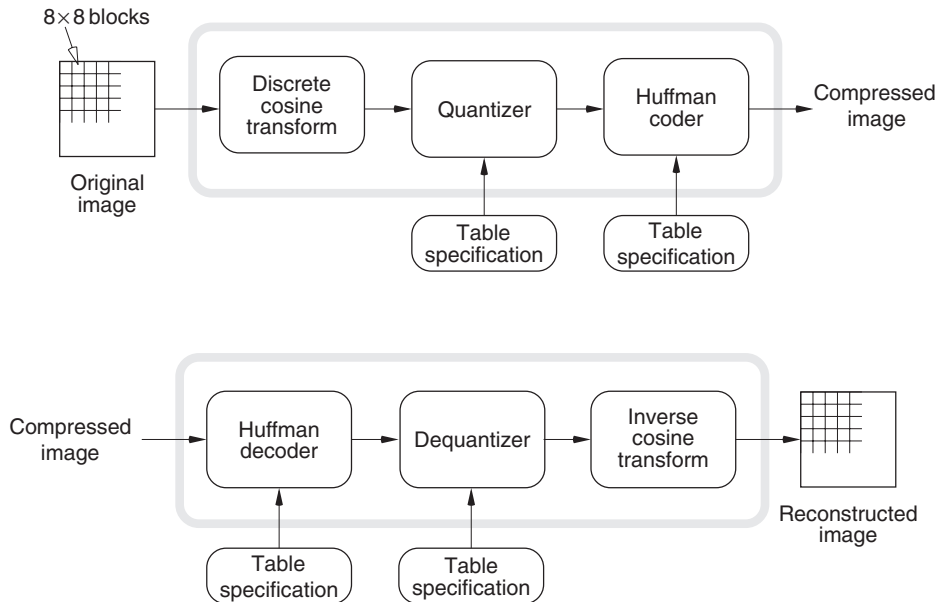
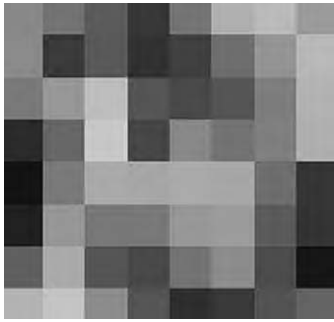


Figure 4.15 Encoding and decoding processes in baseline JPEG.

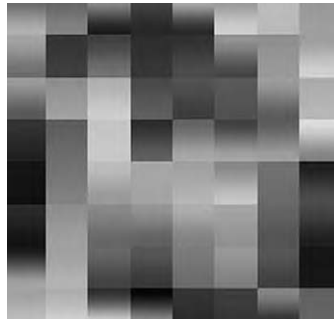
ones to demonstrate the effect. The first version of the image is made from each block's zero-frequency coefficient. This represents the overall grayness of the block, so at this stage each block is a uniform shade of gray. After inclusion of the second coefficient for each block, horizontal cross-sections 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.

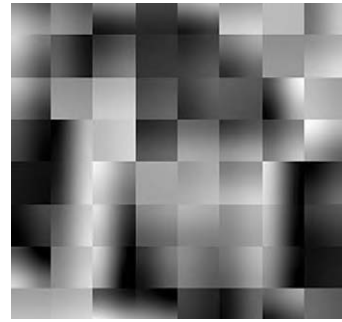
The zero-frequency coefficient is highly correlated from block to block because it is basically the average intensity over the block. Hence it is differentially encoded—in other words, it is represented by the difference between the



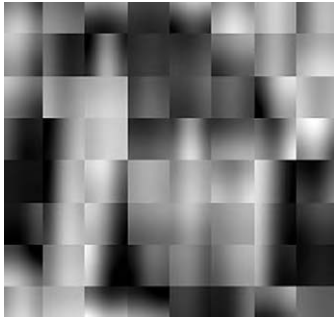
1 coefficient



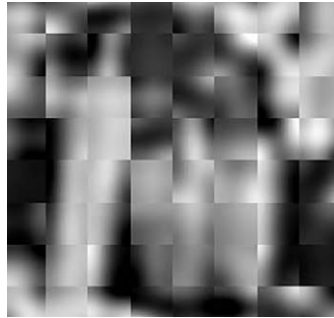
2 coefficients



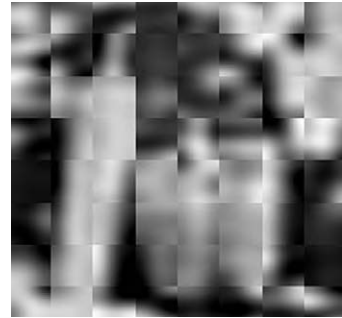
3 coefficients



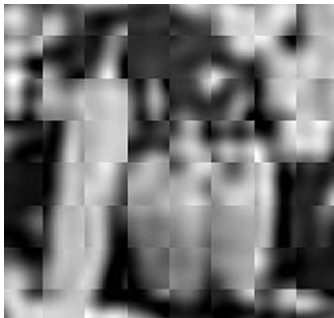
4 coefficients



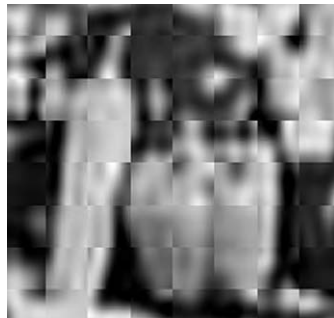
6 coefficients



8 coefficients



10 coefficients



12 coefficients



16 coefficients



20 coefficients



32 coefficients



original (4,096 coefficients)

Figure 4.16 Transform-coded images reconstructed from a few coefficients. University of Southern California Image Processing Institute (USC-IPI) image database.