

CD-ROM Features  
Valuable Source Code  
Plus CGI Programming  
Tools & Utilities



for the  
**working**  
programmer

# **FOUNDATIONS<sup>TM</sup>** *of*

## World Wide Web Programming

## **WITH HTML & CGI**



**Get Complete Coverage of Forms-Handling CGI**

**Learn How to Write a Database Front End**

**Find Special Coverage of WWW Programming  
Languages: Perl, Java, Python, C, and UNIX Shells**





CD-ROM Features  
Valuable Source Code  
Plus CGI Programming  
Tools & Utilities



for the  
**working**  
programmer

# **FOUNDATIONS<sup>™</sup>** *of*

## **World Wide Web Programming**

## **WITH HTML & CGI**



**Get Complete Coverage of Forms-Handling CGI**

**Learn How to Write a Database Front End**

**Find Special Coverage of WWW Programming  
Languages: Perl, Java, Python, C, and UNIX Shells**



**ED TITTEL, MARK GAITHER,  
SEBASTIAN HASSINGER, & MIKE ERWIN**



**FOUNDATIONS™**  
*of*

World Wide Web  
Programming  
**WITH HTML  
& CGI**





## Foundations of World Wide Web Programming with HTML and CGI

Published by

**IDG Books Worldwide, Inc.**

An International Data Group Company  
919 East Hillsdale Boulevard, Suite 400  
Foster City, CA 94404

### Copyright

Copyright © 1995 by IDG Books Worldwide. All rights reserved. No part of this book (including interior design, cover design, and illustrations) may be reproduced or transmitted in any form, by any means, (electronic, photocopying, recording, or otherwise) without the prior written permission of the publisher. For authorization to photocopy items for internal corporate use, personal use, or for educational and/or classroom use, please contact: Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923 USA, Fax 508-750-4470.

Library of Congress Catalog Card No.: 95-077530

ISBN 1-56884-703-3

Printed in the United States of America

Second Printing, November, 1995

10 9 8 7 6 5 4 3 2

Distributed in the United States by IDG Books Worldwide, Inc.

**Limit of Liability/Disclaimer of Warranty:** The authors and publisher of this book have used their best efforts in preparing this book. IDG Books Worldwide, Inc., International Data Group, Inc., and the authors make no representation or warranties with respect to the accuracy or completeness of the contents of this book or the material on the CD-ROM included with this book, and specifically disclaim any implied warranties of merchantability or fitness for any particular purpose, and shall in no event be liable for any loss of profit or any other commercial damage, including but not limited to special, incidental, consequential or other damages.

**Trademarks:** All brand names and product names used in this book are trademarks, registered trademarks, or trade names of their respective holders. IDG Books Worldwide, Inc., is not associated with any product or vendor mentioned in this book.



*Published in the United States of America*

# Table of Contents

<b>INTRODUCTION .....</b>	<b>XXXVII</b>
---------------------------	---------------

<b>PART I FOUNDATIONS OF HTML AND CGI.....</b>	<b>1</b>
--	----------

<b>Chapter 1 Basics of the World Wide Web .....</b>	<b>3</b>
The Internet: An Amorphous Blob .....	4
The HyperText Transfer Protocol (HTTP) .....	5
How HTTP Works .....	6
HTTP Character Sets .....	9
The Common Gateway Interface .....	9
The HyperText Markup Language (HTML) .....	10
How to Read an HTML Document .....	11
Descriptive Markup .....	12
Hierarchical Document Structure Plus Interconnections .....	13
The Formal Specification .....	14
Human- and Computer-Readable Representation .....	14
Summary .....	15

<b>Chapter 2 An Introduction to SGML .....</b>	<b>17</b>
What Is SGML? .....	17
The SGML Philosophy .....	19
SGML's Primary Characteristics .....	19
Descriptive Markup .....	20
Hierarchical Plus Interconnections .....	23

Flexible Tags .....	24
Formal Document Specification .....	26
Human and Computer Readable Representations .....	26
SGML Document Structure .....	27
The SGML Declaration .....	28
The DTD .....	29
Elements .....	33
Element Attributes .....	36
Entities .....	40
Marked Sections .....	43
General SGML Rules .....	45
SGML and HTML .....	46
Summary .....	46
A Brief SGML Bibliography .....	47
 <b>Chapter 3 The HTML DTDs .....</b>	<b>49</b>
History of HTML .....	49
Workings of the IETF HTML Working Group .....	51
IETF Level 1 DTD .....	53
Terms .....	54
The Level 1 DTD .....	55
IETF Level 2 DTD .....	63
DTD Feature Test Entities .....	64
New Elements of HTML 2.0 .....	66
Enhanced Elements of HTML 2.0 .....	72
Deprecation in HTML 2.0 .....	72
IETF Level 3 DTD .....	74
Netscape Extensions .....	76
Migrating from HTML 2.0 to 3.0 .....	76
Summary .....	78
 <b>Chapter 4 HTML and Validation .....</b>	<b>79</b>
What Is HTML, and How Should You Use It? .....	79
What Is Valid HTML? .....	80
Doing the “DTD Thing” .....	81
Testing for Compliance .....	81



HTML 2.0: The Reigning DTD .....	82
HTML 3.0: The Coming DTD .....	83
It's No Accident that Mozilla Is Named after a Monster! .....	83
Choosing an HTML DTD .....	83
Adding Value to Your Web Site .....	86
Advantages of Validation .....	86
Disadvantages of Validation .....	89
Validation Tools .....	90
Ignorance Isn't Always Bliss! .....	90
Get the Right Tool for the Job .....	91
The Not-So-Hidden Beauties of Compliance .....	91
HaLSOFT HTML Validation Service .....	92
HaLSOFT HTML Check Toolkit .....	93
Weblint .....	94
htmlchek .....	95
HoTMetaL .....	96
sgmls and sp .....	96
The DTD is Prolog to the Future .....	97
DTD Reference Methods .....	99
Validation Alone Is Not Enough .....	99
What's in a Process? .....	100
Summary .....	101
 <b>Chapter 5 The Future of HTML .....</b>	<b>103</b>
A Closer Look at HTML 3.0 .....	103
New HTML 3.0 Elements .....	104
Tables .....	104
Figures .....	116
<MATH> .....	122
HTML Style Sheets.....	123
Summary .....	124
 <b>Chapter 6 The Common Gateway Interface .....</b>	<b>125</b>
The Hidden Life of HTML .....	125
What Is the Common Gateway Interface? .....	126
The CGI Specification: Extending HTTP .....	128

A Cast of Environment Variables .....	128
Accessing CGI variables .....	129
The CGI Command Line .....	141
CGI Data Input .....	142
CGI Data Output .....	143
Summary .....	144
<b>Chapter 7 CGI Programming Languages .....</b>	<b>145</b>
Choosing a CGI Programming Language .....	146
The Five Primary Considerations .....	146
Public Source Code Repositories .....	147
Language Support .....	148
Your Level of Knowledge and Understanding .....	149
Desirable Data Throughput .....	149
Desirable “Ilities” .....	150
Compiled CGI Programming Languages .....	150
C .....	151
C++ .....	151
Interpreted CGI Programming Languages .....	153
Perl .....	153
UNIX Shell Scripting Languages .....	155
Tcl .....	156
Python .....	158
Compiled/Interpreted CGI Programming Languages .....	159
Java .....	159
Summary .....	160
<b>Chapter 8 CGI Input and Output .....</b>	<b>161</b>
A Different Sort of Input Method .....	161
GET It While You Can .....	162
Using the GET method .....	163
They’re at the POST! .....	168
The Ins and Outs of <FORM> Data .....	170
<FORM> Tags and Attributes .....	173
An Example Form .....	174
The Inside Scoop on CGI Output .....	177
Summary .....	181



<b>Chapter 9 Designing CGI Applications .....</b>	<b>183</b>
The Requirements Analyzed .....	183
A Language for All Seasons .....	183
Design of the Registration Form .....	185
Mapping Fields to Variables .....	190
Variable Error Checking .....	191
Alternate Methods of Data Capture .....	192
Designing the Output File Format .....	193
Designing a Report Writer .....	194
Building in Counters and Status .....	195
Overall Design Thoughts .....	195
Reusability .....	195
Portability .....	196
Expandability .....	196
Libraryizing .....	197
Summary .....	197
 <b>PART II THE BRIDGE TO CGI IMPLEMENTATION .....</b>	 <b>199</b>
 <b>Chapter 10 Building and Installing CGI Applications .....</b>	 <b>201</b>
html4dum.register.pl Anatomy of a Perl Script .....	202
Setup and Initialization .....	202
Processing User Input .....	205
Closing Arguments .....	211
Debugging Is a State of Mind .....	212
The Raw Data File .....	213
Registration Statistics Summarizer .....	214
The Final Output .....	221
Final Touches .....	222
What's Next? .....	222
Summary .....	223
 <b>Chapter 11 Testing and Installing CGI Applications .....</b>	 <b>225</b>
It's a Dirty Job, but... ..	226
Setting Up a Test Environment .....	226

The Test Web Is the Best Web! .....	227
Installing a Test Server .....	228
The Root of All Testing .....	230
An Alias by Any Other Name .....	231
The Act of Creation Is Best Done in Private .....	233
The Good, the Bad, and the Boundary .....	234
A Simple Test Strategy .....	236
Test Plan for register.pl CGI Application .....	237
Installing Your CGI Application .....	238
Summary .....	239
 <b>Chapter 12 Our Foundations Development Environment .....</b>	<b>241</b>
Basic Assumptions .....	241
For Better or Worse, Our Platform Is UNIX .....	241
Our Language of Choice Is Perl4 .....	243
We'll Work with Either CERN or NCSA httpd .....	244
We Take Good Tools Wherever We Can Get Them! .....	245
Summary .....	246
 <b>Chapter 13 Locating CGI Resources .....</b>	<b>247</b>
Going Straight to the Internet Sources .....	248
Internet Resource Types .....	249
Focused Newsgroups .....	249
Focused Mailing Lists .....	250
Information Collections from "Interested Parties" .....	250
Information from Special Interest Groups .....	250
Other Sources Worth Investigating .....	251
For Every Resource Type, There's a Search Method .....	251
CGI-Related Newsgroups .....	252
CGI-Related Mailing Lists .....	254
Parties Interested in CGI .....	255
CGI-Focused Groups and Organizations .....	256
CGI-Related Publications and Off-Line Resources .....	257
Searching for Satisfaction .....	259
Summary .....	260



<b>Chapter 14 CGI Return Page Templates .....</b>	<b>263</b>
What's in a Page? .....	264
Static Versus Dynamic Content .....	264
Give Me Some Static! .....	265
The Happy Medium: Dynamic but Constant .....	266
The Truly Dynamic .....	267
Building the HTTP Header .....	268
Building the HTML Header .....	270
The HTML Document Body .....	270
The Beginning of the Body .....	270
Dealing with Recurrent Elements .....	271
Incorporating Preformatted Text .....	272
Encoding HTML Markup .....	273
Handling HTML Document Footers .....	274
Other Template Tricks .....	274
Summary .....	275
 <b>Chapter 15 The Major CGI Libraries .....</b>	 <b>277</b>
Looking for CGI Nirvana .....	278
Round Up the Usual Suspects .....	278
Ask an Expert! .....	279
Do Some Reading .....	279
Some Select CGI Sites .....	280
http://hoohoo.ncsa.uiuc.edu/cgi/interface.html .....	280
http://www-genome.wi.mit.edu/ftp/pub	
/software/WWW/cgi_docs.html .....	281
http://www.cosy.sbg.ac.at/www-doku/tools	
/bjscripts.html .....	283
http://www.oac.uci.edu/indiv/ehood/perlWWW/ .....	284
http://WWW.Stars.com/Vlib/Providers/CGI.html .....	285
http://www.w3.org/hypertext/WWW/	
Daemon/User/CGI/Overview.html .....	286
http://www.webedge.com/web.dev.html .....	286
http://www.yahoo.com/Computers/World_Wide_Web/	
CGI_Common_Gateway_Interface/ .....	287

Using CGI Programs Effectively .....	288
Publish and/or Perish! .....	289
Summary .....	290
<b>Chapter 16 &lt;FORM&gt; Alternatives.....</b>	<b>291</b>
From <FORM> to Text?.....	292
From HTML Back to Text.....	292
Registration Form, Part 1.....	297
Registration Form, Part 2.....	298
Registration Form, Part 3.....	300
Registration Form, Part 4.....	301
Registration Form, Part 5.....	302
Shipping the Text-Only Form File .....	304
Summary .....	305
<b>Chapter 17 Gotchas, Warnings, and No-Nos .....</b>	<b>307</b>
HTML's Barest Necessities .....	307
CGI Programming Gotchas .....	309
Miscellaneous .....	312
Summary .....	314
 <b>PART III BASIC CGI PROGRAMMING ELEMENTS</b>	
<b>AND TECHNIQUES .....</b>	<b>315</b>
 <b>Chapter 18 Mastering MIME .....</b>	<b>317</b>
Extending Internet and Electronic Mail .....	317
The Marriage of MIME and HTTP .....	319
Divergence of HTTP from MIME .....	321
MIME and CGI Applications .....	323
The MIME .mailcap File .....	330
Global and Personal .mailcap Files .....	331
Summary .....	332
 <b>Chapter 19 Using WAIS with CGIs .....</b>	<b>333</b>
Welcome to the Wide World of WAIS .....	333
WAIS Architecture Unmasked .....	334



WAIS Protocols .....	336
WAIS as a Client and a Server .....	337
WAIS Databases Revealed .....	337
How Is the Index Used? .....	338
Do I Need a Gateway? What for? .....	340
Finding WAIS Gateways .....	340
wais.pl .....	340
son-of-wais.pl .....	341
kidofwais.pl .....	341
SFgate .....	343
fwais.pl .....	345
wwwwais.c, version 2.5 .....	345
WAISGate (WEB-to-WAIS Gateway) .....	346
Exploring WAIS as a User .....	346
What's the Difference Between waisq and waissearch? .....	350
Other Topics .....	351
SWISH .....	351
Indexing Hierarchies of HTML Documents .....	351
WAIS and HTTP Integration .....	352
Usenet News Archives .....	352
Mailing Lists .....	352
Futures and Cool Engines .....	353
Summary .....	353
 <b>Chapter 20 Indexing Your Web Site .....</b>	 <b>355</b>
Lost in the Funhouse .....	355
What to Include .....	356
Installing and Configuring freeWAIS .....	359
Using CGIs to Format the Output .....	361
Using SWISH to INDEX Your Site .....	365
Customizing Search Results .....	368
Other OS Options .....	369
Apple Macintosh .....	369
DOS/Windows/Windows NT .....	369
Summary .....	370

<b>Chapter 21</b>	<b>Getting Hyper: Principles of Hypertext Design .....</b>	<b>371</b>
	Stringing Pages Together, the Old-Fashioned Way .....	372
	Hierarchies Are Easy to Model in HTML .....	373
	Multiple Tracks for Multiple Audiences .....	374
	The RFC Library .....	376
	The Principles of Hypertext .....	380
	Annotate All External References .....	380
	Indicate the Scope and Size of Your Materials .....	380
	Limit the Complexity of Your Index .....	381
	Match Your Structure to Your Content .....	381
	Present Distinct Topics as Distinct Documents .....	382
	Provide Appropriate Access Methods .....	382
	Supply Easy, Consistent Navigation Tools .....	383
	Use Links as References .....	383
	Summary .....	384
<b>Chapter 22</b>	<b>Monitoring Web Server Activity .....</b>	<b>385</b>
	Who's That Knocking at My Door? .....	385
	Talking with the Log Lady: HTTP Log Analysis, Parsing, and Summaries .....	386
	What's a WAMP? .....	387
	Swiss Army Statistics from getstats .....	389
	Bean Counter's Wet Dream: WWWstat .....	391
	Mac See, Mac Do: WebStat .....	392
	Figure Painting: Making the Leap from ASCII to GIF .....	393
	getgraph.pl .....	394
	gwstat .....	395
	wusage .....	396
	VB-stat .....	397
	For Those of You Who Can't Get Enough Data .....	398
	Professional Help .....	398
	Case Studies .....	399
	Future Logging Models .....	399
<b>Chapter 23</b>	<b>Robots, Spiders, and WebCrawlers .....</b>	<b>401</b>
	A Robot Taxonomy .....	402
	Is the World Ready For Yet Another Robot? .....	403



A Rationale for Robot Creation .....	404
Identify Yourself and Your Robot .....	404
Spread the Word .....	405
Stay in Touch .....	405
Notify the Authorities .....	405
Don't Be a Resource Hog .....	406
Stay in Charge .....	408
Share Your Robot's Wealth .....	409
A Standard for Robot Exclusion .....	410
More Robot Hangouts .....	412
Summary .....	413
 <b>Chapter 24 Webifying Documents .....</b>	<b>415</b>
The Zen of Webification .....	416
Conversion .....	417
Transformation .....	418
Translation .....	418
Filtering .....	420
Tools and Environments .....	420
The Integrated Chameleon Architecture .....	422
Rainbow .....	424
Other Publicly Available Tools .....	424
Life After Translation .....	426
Chunking .....	426
Integration .....	427
Summary .....	429
 <b>Chapter 25 Image Maps .....</b>	<b>431</b>
The Art of GIF-Giving .....	431
Aids to Navigation .....	431
Great Frames for Interactive Input .....	432
Other Exotic Applications .....	433
Making Image Maps Work .....	434
Creating Hot Spot Definitions .....	435
WebMap for Macintosh .....	435
Image Map Editors on the Web: MapMaker .....	436

mapedit: Creating Image Maps in Windows .....	437
Imagemap CGIs .....	438
NCSA imagemap .....	438
CERN htimage .....	441
Image Maps on Macintosh Web Servers .....	442
Oddities and Cool Stuff .....	444
Glorgox .....	444
Cyberview3D Document Generator .....	445
Caveats and Tips to Click .....	446
Small Is Beautiful .....	446
Present Alternatives .....	446
Keep It Simple .....	446
Keep It Professional .....	447
Use Your Imagination! .....	447
One More Very Important Point .....	447
 <b>Chapter 26 HTML Document Creation On-the-Fly .....</b>	<b>449</b>
Browsing Mail .....	450
mail2html .....	450
Webified Man Pages .....	454
man2html .....	454
FTP Browsing .....	457
ftpb .....	457
Oddments, Helpers, and Horizons .....	460
Ghostscript: GIFs-on-the-Fly .....	460
netcloak .....	462
Summary .....	462
 <b>Chapter 27 Getting the Best Buy For Your Bucks .....</b>	<b>465</b>
Where to Begin .....	469
Getting Connected .....	470
Throughputs and Costs .....	470
Line Speeds and Latency .....	471
Housing a Server at Your Site .....	471
Housing Your Server at the ISP .....	473
What Do I Need to Watch For? .....	473



Why Not Lease Space on a Server? .....	474
What Do I Need to Watch For? .....	474
Comparing UNIX Servers .....	475
Other Platforms .....	475
Choosing a Mac Server .....	475
Choosing a DOS/Windows PC Server .....	476
Choosing a Windows NT Server .....	476
Web Server Miscellany .....	477
Proxies .....	477
Load Balancing Strategies .....	477
Summary .....	478

## **PART IV    ADVANCED CGI PROGRAMMING TOOLS               AND TECHNIQUES ..... 479**

<b>Chapter 28    File Access Through the Web .....</b>	<b>481</b>
The Software and How to Get It .....	481
All Browsers Are Not Created Equal .....	482
Through a Glass, Darkly .....	483
The Whole FTP Boatload .....	484
CGIs: the Benedictine Monks of the Web .....	484
Mirror Decisions and Issues .....	486
FTP Versus HTTP as a File Retrieval Protocol .....	487
Less Automation, More Sweat .....	488
More Automation, No Sweat .....	489
Lister, You Lazy Sod! .....	490
File Index Searching .....	492
Archie .....	492
SHASE .....	493
Local Indices .....	494
Individual File Retrieval and Whatnots .....	495
Attack of the Killer httpd Server .....	495
 <b>Chapter 29    Database Management Tools .....</b>	 <b>497</b>
What Makes One Database Unlike Another? .....	498
Relational Databases Overview .....	498

Why Connect a DBMS to the Web? .....	500
Composing an Interface .....	501
Create Your Own CGI .....	501
Consult the Oracle .....	506
Oracle World Wide Web Interface Kit .....	507
What Is OraPerl? .....	508
Sybase .....	508
sybperl .....	508
Web/Genera .....	509
GSQL .....	509
Summary .....	510
 <b>Chapter 30 Audio-Video Capture and Real-Time Pages .....</b>	<b>511</b>
Video .....	511
Refreshed Images at a Click and a Glance .....	512
Real-Time Traffic Watch .....	518
Constantly Refreshed Pictures .....	518
Real-Time Video .....	520
Some Implementation Examples .....	521
The Future of Net Video .....	522
Audio .....	523
Interactive Sound Sites .....	523
External Apps .....	524
AV: Together at Last .....	526
CU-SeeMe .....	526
Summary .....	527
 <b>Chapter 31 Electronic Mail Gateways .....</b>	<b>529</b>
The Proto-Web .....	529
Have Your CGI Mail My CGI .....	530
Proprietary Tagging .....	532
Building a Better Mail Handler .....	533
Too Complex? E-Mail Gets Simpler .....	535
Next Verse, Same as the First (on MacOS) .....	537
Mailing List Interface .....	538



Search and Retrieval .....	540
List Administration Tools .....	542
Summary .....	543
 <b>Chapter 32 WWW Server and CGI Security .....</b>	 <b>545</b>
An Open Door Is Not Always Good... ..	545
Security Is Like Mowing the Lawn... ..	546
A Typical UNIX httpd Configuration .....	547
srm.conf .....	550
UNIX Permissions and Ownerships .....	553
Limiting Access to Internal Servers .....	553
access.conf .....	554
.htaccess Gives You Power .....	555
What Does .htaccess Contain? .....	556
How Would You Employ the .htaccess File? .....	557
To UNIX or Not to UNIX? .....	558
The Benefits of Secure Transit .....	559
Secure-HTTP .....	559
Other WWW Security Mechanisms .....	560
The S-HTTP Specification and Design Goals .....	560
What Can We Do with S-HTTP? .....	561
Some Places to Go... ..	561
Summary .....	562
 <b>Chapter 33 User Interaction Through HTML .....</b>	 <b>563</b>
Everybody's Doing It .....	563
Basic Interaction Through HTML .....	564
addlink .....	564
guestbook .....	565
The Current State Of the Interactive Web .....	567
hypernews .....	568
WebChat .....	570
IRC W3 Gateway and Talker .....	572
Storybase Annotation .....	573
The Future: Shared Space .....	574
Commercial Products Crossover .....	574
Summary .....	576

<b>Chapter 34 Foundations of WWW Programming with HTML and CGI .....</b>	<b>577</b>
Go Home... ..	577
Summary .....	579
 <b>Glossary .....</b>	 <b>581</b>
<b>Contact List.....</b>	<b>609</b>
<b>License Agreement .....</b>	<b>621</b>
<b>Index .....</b>	<b>623</b>
<b>CD Errata.....</b>	<b>649</b>



# The Common Gateway Interface

## The Hidden Life of HTML

Online WWW documents lead a completely different life than their conventional, print-bound counterparts. They can change with a single keystroke and the effects are immediately visible to the whole world. Since server programs can generate custom-built Web documents on-the-fly in response to interaction with end users, this makes online WWW publishing much more responsive and open-ended than traditional publishing.

Quick response to change and dynamic behavior are the underpinnings of the power behind the WWW. But unless you implement the right approach to building and maintaining Web documents, your information can just lie stagnant and be virtually “dead to the world.”

Dynamic behavior is a vital aspect of the Web that is sometimes hard to identify or appreciate. This is particularly true when documents are created on-the-fly, usually in response to some event on the WWW. Although what appears to a user in response to a query may look like “just another Web page,” it might actually be an evanescent document created for one-time use in direct response to that query.

The same is often true for the results of a Web search, or even when clicking a link or a button on a particular page. The result is a custom-built response, tailored around previous interaction with the user (or from information provided by users, even if they’re blissfully unaware of the underlying communications involved).

What makes all this possible is a special mechanism that supports the dynamic creation of HTML documents. This mechanism is based on



the invocation of external applications that run under the auspices of a WWW server, called by the browser in the form of an ordinary-looking URL on the Web document currently in use. Each of these external applications can be tailor-made to deliver customized, on-demand HTML documents.

Today, many commercial institutions realize the wealth and value of information that resides in their customer and product databases. Many of them want to provide this ancillary information for public consumption. Some key questions they're grappling with are: "How can we make this information accessible?" and "What's the best way to present it for easy consumption and use?" and "How can we make money on this thing?"

The WWW provides the infrastructure and mechanisms for seamless integration of this kind of information, along with typical sales documents like spec sheets, brochures, testimonials, technical documents, and whatnot (which we'll refer to rather loosely as *collaterals*). A key technology underlying the WWW permits static information to be queried and imported on demand and also supports custom construction of Web documents on-the-fly. This technology is called the Common Gateway Interface (CGI), the subject of this chapter.

## What Is the Common Gateway Interface?

The Common Gateway Interface (CGI) supplies the middleware among WWW servers, external databases, and information sources. CGI scripts or programs may be considered to be extensions to the core functionality of a WWW server, like that supplied in the CERN or NCSA *httpd* packages. CGI applications perform specific information-processing, retrieval, and formatting tasks on behalf of their WWW servers. The CGI interface defines a method for the Web server to accommodate additional programs and services that may be used to access external applications from within the context of any active Web document.

The term *gateway* describes the relationship between the WWW server and the external applications that handle data access and manipulation chores on its behalf. This gateway incorporates enough built-in



intelligence to guarantee successful communication between the server and the external application. Normally, a gateway handles information requests in some orderly fashion and then returns an appropriate response — for example, an HTML document generated on-the-fly that includes the results of a query applied against an external database.

In other words, CGI allows a WWW server to provide information to WWW clients that would not otherwise be available to those clients in a readable form. This could, for example, allow a WWW client to issue a query to an Informix database and receive an appropriate response in the form of a custom-built Web document. That's why we believe that CGI's main purpose is to support communications with information and services outside the normal purview of WWW, ultimately to produce HTTP objects from non-HTTP objects that a WWW client can render. In short, CGI opens up a door to the whole world of information and services that would otherwise be inaccessible to Web clients and servers alike.

Gateways can be designed and deployed for a variety of purposes, but the most common is the handling of <ISINDEX> and <FORM> HTTP requests. Some common uses of CGI include:

- Gathering user feedback about your product line or your WWW server through an HTML form.
- Dynamic conversion of your system's documentation (for example, *man* pages) into HTML, for easy Web-based access.
- Querying Archie or WAIS databases and rendering results as HTML documents.

Working in tandem with the HTTP server application (*httpd*), CGI applications service requests made by WWW clients by accepting requests for services at the server's behest, handling those requests, and sending appropriate responses back to the client. A client HTTP request consists of the following elements:

- a Universal Resource Identifier (URI)
- a request method
- other important information about the request provided by the transport mechanism of the HTTP

In the sections that follow, you'll have a chance to understand all of these elements in detail as you become familiar with the structure and function of a client HTTP request.

## The CGI Specification: Extending HTTP

The CGI specification was created and documented by the main HTTP server authors: Tony Sanders, Ari Luotonen, George Phillips, and John Franks. These folks discovered that they didn't want to keep adding functionality to their HTTP servers every day, in keeping with the needs of one particular Web activity or another. They decided to build a clearly defined core of WWW server functionality and to provide a way to extend services and capabilities from there.

They needed an application programming interface (API) available to any Perl, C, or shell hacker willing to learn the appropriate interface details. Hence, the creation of CGI as a formal, rigorous specification that includes some nasty notation and grammar.

For more information about the CGI specification, please consult the following URL:

<http://hoohoo.ncsa.uiuc.edu/cgi/interface.html>

In the subsections that follow, you'll have a chance to cover the elements of the specification, as we prepare you to build CGI programs of your own.

## A CAST OF ENVIRONMENT VARIABLES

Environment variables are entities that exist within a particular user's computer environment. Many of these variables attain their values whenever a user logs into a computer, or when the computer is booted for a single-tasking operating system like DOS. Environment variables



are pervasive within UNIX, where they are used to pass information to applications from the runtime environment.

Because the environment persists even as execution threads come and go, environment variables sometimes function as placeholders, to pass data from one application to another within the same user session. In the case of CGI, environment variables known to the server and CGI are used to pass data about an HTTP request from a server to the CGI application. These variables are accessible to both the server and any CGI application it may invoke.

The WWW server may also use command-line arguments, as well as environment variables, to pass data about an information request from a WWW client. This allows specific control over program parameters or execution to originate with the server, as well as providing a mechanism for clients to pass short input sequences to the CGI program involved. Likewise, these variables may also be set (or assigned their values) when the server actually executes a CGI application.

## ACCESSING CGI VARIABLES

The requirements of a specific WWW server control how CGI applications access variables. If a variable does not have a value, this indicates a zero-length value (NULL in the UNIX realm). If the variable is not defined in the server's system (omitted or missing), it is assumed to have a zero-length value and hence, is also assumed to be NULL. As with variable access, the system requirements for the WWW server dictate the means of value representation for CGI variables.

Likewise, the names for environment variables are system specific. For historic reasons, we will discuss UNIX environment variables in detail. Please notice that these names are case sensitive, so they must be reproduced exactly in order to reference the correct information. Table 6-1 is a quick reference to the standard environment variables that are specific to each request handled by a CGI application.

**Table 6-1***Standard Environment Variables Quick Reference*

<i>Environment Variable</i>	<i>Description</i>
AUTH_TYPE	access authentication type
CONTENT_LENGTH	size in decimal number of octets of any attached entity
CONTENT_TYPE	the MIME type of an attached entity
GATEWAY_INTERFACE *	server's CGI spec version
HTTP_(string)	client header data
PATH_INFO	path to be interpreted by CGI app
PATH_TRANSLATED	virtual to physical mapping of file in system
QUERY_STRING	URL-encoded search string
REMOTE_ADDR	IP address of agent making request
REMOTE_HOST	fully qualified domain name of requesting agent
REMOTE_IDENT	identity data reported about agent connection to server
REMOTE_USER	user ID sent by client
REQUEST_METHOD	request method by client
SCRIPT_NAME	URI path identifying a CGI app
SERVER_NAME *	server name; host part of URI; DNS alias
SERVER_PORT	server port where request was received
SERVER_PROTOCOL	name and revision of request protocol
SERVER_SOFTWARE *	name and version of server software

\* not request-specific (set for all requests).

## AUTH\_TYPE

This variable is used to provide various levels of server access security. If your server supports user authentication, this environment variable value indicates the protocol-specific authentication method used to validate a user.

HTTP provides a simple challenge–response authorization mechanism. This mechanism can be used by a server to challenge a client's request or by a client to provide authentication information to a server. The variable's value is deciphered from the HTTP *auth-scheme* token in the client's request HTTP header. If no access authorization is required, the value is set to NULL.

For example, a server may require a client to provide a user ID and password to access a specific portion of its Web. Facebook, Inc. Ex. 1007



happen, the client must set the `AUTH_TYPE` environment variable to a value that matches the supported protocol of the server. For HTTP applications, the most common value is:

```
AUTH_TYPE = Basic
```

This provides a basic authentication scheme where the client must authenticate itself with a user ID and password pair. For more information on this (and other) environment variables, please consult section 10.1 of the HTTP 1.0 specification at the following URL:

```
http://www.w3.org/hypertext/WWW/Protocols/HTTP1.0/HTTP1.0/draft-ietf-http-spec.html#BasicAA
```

## CONTENT\_LENGTH

This variable's value equals the decimal number of octets for the attached entity, if any. If there is no attached entity, it is set to `NULL`. For example:

```
CONTENT_LENGTH = 9
```

## CONTENT\_TYPE

The value of this variable indicates the media type of the attached entity, if any. If there is no attached entity, it is also set to `NULL`. HTTP 1.0 uses MIME Content-Types, which provide an extensible and open mechanism for data typing and for data type negotiation.

With HTTP, you can represent different media like text, images, audio, and video. Within such media, there often exist various encoding methods. For instance, images can be encoded as jpeg, gif, ief, or tiff. The value of the `CONTENT_TYPE` variable for an attached gif image file could therefore be:

```
CONTENT_TYPE = image/gif
```

For HTTP requests, a client may provide a list of acceptable media types as part of its request header. This allows the client more autonomy in its use of unregistered media types. Although data providers are strongly encouraged to register their unique media types and subtypes with the Internet Assigned Numbers Authority (IANA), this does provide additional flexibility. But since both client and server must be able to

handle unregistered items, media types registered with the IANA are preferred over unregistered extensions.

See Table 6-2 for the seven standard predefined MIME Content-Types. (The primary subtype is presented in *italics*.)

**Table 6-2**

*MIME Registered Content-Types Used by HTTP*

<i>Type</i>	<i>Subtypes</i>	<i>Description</i>
application	<i>octet-stream</i>	transmit application data
text	<i>plain</i>	textual information
multipart	<i>mixed</i> , alternative, digest, parallel	multiple parts of independent data types
message	<i>rfc822</i> , partial, external-body	an encapsulated message
image	gif, jpeg	image data
audio	<i>basic</i>	sound data
video	<i>mpeg</i>	video data

### ***application***

This MIME Content-Type describes transmitted application data. This is typically uninterpreted binary data. The primary subtype is 'octet-stream' intended to be used in the case of uninterpreted binary data. An additional subtype, 'PostScript', is defined for transporting PostScript documents within MIME bodies.

### **Subtype Specifications**

Whenever an environment variable supports subtyping, note that some subtype specification is *mandatory* for that variable. That is, there are no default subtypes.

All type, subtype, and parameter names are case insensitive. For example, 'IMAGE',

'Image', and 'image' are all equivalent. The only constraint on subtype names is that their uses must not conflict. It would be undesirable to have two different uses of 'Content-Type: application/grunge'. The name provides a simple mechanism for publicizing its uses, not constraining them.



The simplest and most common action when a client requests 'application' MIME Content-Type data is to offer to write the information to a file. For example, the environment variable:

```
CONTENT_TYPE = application/octet-stream
```

would result in the client asking the user if he or she would like to save the binary data to a local file on his or her hard drive.

Other expected uses for 'application' could include things like spreadsheet data, data for mail-based scheduling systems, compressed binary data such as *gzip*, and languages for “active” e-mail.

### ***text***

This MIME Content-Type describes textual data. The primary subtype is 'plain'. This unformatted text requires no special software to render the text, although the indicated character set must be supported by the client. Text subtypes are forms of enriched text where software may enhance the presentation of the text. The most common non-registered text subtype used on the Web is 'html (text/html). This software must not be required in order to grasp the general idea of the text's content. One such possible subtype is 'richtext' which was introduced by RFC 1341.

An example of this environment variable is:

```
CONTENT_TYPE = text/plain; charset=us-ascii
```



For more information about 'richtext' and RFC 1341 see this URL:

<http://www.cis.ohio-state.edu/htbin/rfc/rfc1341.html>

Please note that since its publication, RFC 1341 has been superseded by RFC 1521, which incorporates some new material not mentioned in 1341. For current implementation details, please consult the most current version of any RFC. The full collection, which includes pointers from obsolete to current versions, can be accessed through the following URL:

<http://www.cis.ohio-state.edu/hypertext/information/rfc.html>

### ***multipart***

This MIME Content-Type describes data consisting of multiple parts of independent data types. There are four common subtypes.

1. 'mixed' is the primary subtype.
2. 'alternative' represents the same data type in multiple formats.
3. 'parallel' for parts intended to be viewed at the same time.
4. 'digest' for parts of type 'message'.

### ***message***

This MIME Content-Type subtype is an encapsulated message. A MIME body of Content-Type 'message' is itself all or part of a fully formatted message. This message must comply with the Internet Mail Header protocol described in RFC 822. This RFC 822-compliant message may contain its own completely different MIME Content-Type header field.

There are three subtypes:

1. 'rfc822' is the primary subtype.
2. 'partial' is defined for partial messages that allow fragmented transmission of MIME bodies deemed too large for normal mail transport facilities.
3. 'external-body' is for specifying large MIME bodies by reference to an external data source such as an image repository.

Here's an example of a standard e-mail text message:

```
CONTENT_TYPE = message/rfc822
```

### ***image***

This MIME Content-Type describes image data. The MIME Content-Type image requires a display device such as a graphical display, a FAX machine, or a printer to view the image data. There are two subtypes that are defined for two widely used image formats, jpeg and gif.

For example, data of a gif encoded image would be:

```
CONTENT_TYPE = image/gif
```



**audio**

This MIME Content-Type describes audio or sound data. The primary subtype is 'basic'. Audio requires an audio output device such as a speaker to display the contents.

An example of a MIDI-attached MIME body would be:

```
CONTENT_TYPE = audio/x-midi
```

NOTE: the “x” prefix declares this MIME Content-Type subtype to be an extended data type.

**video**

This MIME Content-Type describes video data. Video requires the capability to display moving images. This is typically accomplished with specialized hardware and software. The primary subtype is 'mpeg'.

An example of a video data entity would be:

```
CONTENT_TYPE = video/mpeg
```

For more information about the IANA, see this URL:

```
http://ds.internic.net/rfc/rfc1590.txt
```

**GATEWAY\_INTERFACE**

This environment variable represents the version of the CGI specification to which the server that supplies this value complies. This variable is not request-specific and is set for all HTTP requests. An example value would be:

```
GATEWAY_INTERFACE = CGI/1.1
```

**http\_(string)**

CGI environment variables beginning with the string 'HTTP\_' contain HTTP header information supplied by the client. These are the MIME Content-Types that the client will accept, as supplied in HTTP request

headers. Each data type in this list should be separated by a comma as required in the HTTP 1.0 specification.

The HTTP header name is converted to all uppercase characters. All “-” (hyphen) characters are replaced with “\_” (underscore). Finally, the 'HTTP\_' string is prepended, resulting in the set of HTTP\_string environment variables.

The HTTP request header data sent by the client may be preserved, or it may be transformed, but it will not change its meaning. An example of this behavior occurs when a server may strip comment data from an HTTP request header. If necessary, the receiving server must transform the HTTP request header data to be consistent with the legal syntax for a CGI environment variable.

In the course of its operation, the WWW server handles many HTTP request headers. It is not required to create CGI environment variables for all received header data. In particular, the server should remove any headers containing authentication data such as 'Authorization'. This is valuable information that should not be available to other CGI applications. The following example identifies the MIME Content-Types that a server supports and services:

```
HTTP_ACCEPT = application/zip, audio/basic, audio/x-midi, application/x-rtf,  
video/msvideo, video/quicktime, video/mpeg, image/targa, image/x-win-bmp,  
image/jpeg, image/gif, application/postscript, audio/wav, text/plain,  
text/html, audio/x-aiff, audio/basic, application/x-airmosaic-patch, applica-  
tion/binary, application/http, www/mime
```

## PATH\_INFO

This variable represents extra path information, provided by a requesting client. It describes a resource to be returned by a CGI application once it completes successfully. Clients can access CGI applications using virtual pathnames, followed by extra information appended to this path. Because URLs use a special encoding method for character data, this path information will be decoded by the server if it comes from a URL before it is passed to a CGI application.

The value of the PATH\_INFO variable can be one of the following:

1. It may be contained in some trailing part of the requesting client's URI.



2. It may be some string provided by the client to the server.
3. It may be the entire client's URI.

A CGI application cannot decipher how `PATH_INFO` was chosen or constructed based solely on its value. As a CGI programmer, you must determine the context in which this variable is to be used in your CGI application. An example of a valid `PATH_INFO` variable is:

```
PATH_INFO = /raising/arizona/
```

## PATH\_TRANSLATED

This variable represents the operating system path to a file on the system that a server would attempt to access for a client requesting an absolute URI containing the `PATH_INFO` data. The server provides a translated version of `PATH_INFO`, which takes the path data and does any virtual-to-physical file system mapping.

For security reasons, some servers do not support this variable and will set it to `NULL`. This variable need not be supported by a server. The algorithm a server uses to decode `PATH_TRANSLATED` is invariably operating-system dependent. Therefore, CGI applications utilizing `PATH_TRANSLATED` may have limited portability. Here's an example of an NCSA server's 'DocumentRoot' equal to '/u/Web' that builds on the preceding example:

```
PATH_TRANSLATED = /u/Web/raising/arizona/
```

## QUERY\_STRING

This variable represents a URL-encoded search string. The value of this variable follows the "?" character in the URL that referenced a CGI search application. The value of this variable is not decoded by the server, but is passed onto the CGI application untouched.

The information after the "?" in a URL can be added by one of the following:

- an HTML `<ISINDEX>` document
- an HTML `<FORM>` CGI application (using the GET method)

- manually appended by the HTML document's author in which the reference occurs, in an <A> (anchor statement)

The value of QUERY\_STRING is encoded in the standard URL format of replacing a space with a "+" and encoding nonprintable characters with the hexadecimal encoding scheme of '%dd' where "d" is a digit. You will need to decode this hexadecimal information before you use it. (You will learn how to do this in Chapter 8.)

This variable is always set when there is query information requested by a client, regardless of any command-line decoding by the CGI application. An example of a client's query URL for a CGI search application ('places.pl' ) to locate a place named "Sunset Crater" could look like this:

```
http://www.flagstaff.az.us/cgi-bin/places.pl?Sunset_Crater
```

The resulting QUERY\_STRING value for this URL would be:

```
QUERY_STRING = Sunset_Crater
```

## REMOTE\_ADDR

This CGI environment variable represents the Internet Protocol (IP) address for a requesting agent. This is not necessarily the address of the client, but could be the address of the host for that client, depending on the type of connection and operating system the client is using. Here's an example:

```
REMOTE_ADDR = 199.1.78.25
```

## REMOTE\_HOST

This variable represents the fully qualified domain name for a requesting agent. If the server cannot decipher this information, REMOTE\_ADDR is set to NULL. Domain names are case insensitive. Here's an example:

```
REMOTE_HOST = ppl1.aus.sig.net
```

This is the address of the machine of one of the author's Internet service providers, where the client actually resides on his machine at



home. It communicates to the Internet through the host via the Point to Point Protocol (PPP) — in this case, using connection 1 (which is why the first term in the domain name is “pppP1”). Thus, the agent is not necessarily the client.

## REMOTE\_IDENT

This variable represents the remote user's name retrieved from the server. This RFC 931 (Authentication Server) identification must be supported by the HTTP server. It is highly recommended that use of this CGI environment variable be limited to logging actions. The current RFC 931 documentation implies that the HTTP server must attempt to retrieve the identification data from the requesting agent if it supports this feature. Finally, the value of this variable is not appropriate for user authentication purposes. An example would be:

```
REMOTE_IDENT = inmate.state.prison.az.us
```

## REMOTE\_USER

The value of this environment variable is the authenticated user's name. Two conditions must apply for this variable's value to be set:

1. The server must support user authentication.
2. The CGI application must be protected from unauthorized access.

If the AUTH\_TYPE variable's value is 'Basic', then the value for REMOTE\_USER will be the user's identification sent by the requesting client:

```
REMOTE_USER = gail_snokes
```

## REQUEST\_METHOD

The value of this CGI environment variable represents the method with which the client request was made. For HTTP 1.0, this is GET, HEAD, POST, PUT, DELETE, LINK, and UNLINK. The method name is case sensitive. An example for a client posting HTML form data to a server is:

```
REQUEST_METHOD = POST
```

## SCRIPT\_NAME

This variable's value is the URI path that identifies a CGI application. It is the virtual path to a CGI application being executed by a server. In this example, the client's query URL of a CGI application that calculates an engine's maximum horsepower is:

```
http://www.muscle.cars.org/cgi-bin/calc_max_hp.pl
```

and the corresponding SCRIPT\_NAME variable's value is:

```
SCRIPT_NAME = /cgi-bin/calc_max_hp.pl
```

## SERVER\_NAME

This variable represents the server's hostname, DNS alias, or IP address as it would appear in URLs. This variable is not request-specific and is set for all requests. An example is:

```
SERVER_NAME = www.hal.com
```

## SERVER\_PORT

This variable is the port on which a client request is received. This is deciphered from the appropriate part of a URL. Most HTTP server implementations use port 80 as the default port number. URLs that do not explicitly specify a port can be accessed via port 80. For example, the following URL specifies that port 8119 is the designated HTTP server port:

```
http://www.chevelle.org:8119/index.html
```

## SERVER\_PROTOCOL

This variable's value represents the name and revision of the information protocol that the requesting client utilizes. The protocol is similar to the URL scheme used by a requesting client, and it is case insensitive. An example is:

```
SERVER_PROTOCOL = HTTP/1.0
```



## SERVER\_SOFTWARE

This variable represents the name and version of the information server software. This variable is not request-specific and is set for all requests:

```
SERVER_SOFTWARE = NCSA/1.4
```

## THE CGI COMMAND LINE

Some operating systems support methods for providing data to CGI applications via the command line. This technique is used only in the case of an HTML <ISINDEX> query.

A server identifies this type of request by the GET method, accompanied by a URI search string that does not contain any *unencoded* “=” characters. This method trusts the clients to encode the “=” character in an <ISINDEX> query. This practice was considered safe at the time of the design of the CGI specification and requires that you, as a CGI programmer, be careful to observe this restriction.

A server parses this string into words using the rules in the CGI specification, as follows:

1. The server decodes the query information by first splitting the string on the “+” characters in the URL (which map to spaces, following URL encoding techniques).
2. The server then performs any additional decoding before placing each split string into an array named 'argv'.
  - Typically, this consists of separating out individual variable strings by splitting the input stream at each ampersand (“&”).
  - Then, the resulting substrings must be split again (at the equal sign “=” character) to establish name and value pairs, where the left-hand-side value provides the name of the variable, and the right-hand-side value supplies its corresponding value.

This approach follows the UNIX method for passing command-line information to applications. 'argv' is an array of pointers to strings,

where the length of the 'argv' array is stored in the environment variable 'argc'. (Love that UNIX!)

If the server finds that it cannot send the 'argv' array to the CGI application, it will include *no* command-line information. Instead, it will provide the non-decoded query information within the environment variable QUERY\_STRING. This failure may be attributed to internal limitations of the server like those imposed by *exec()* or */bin/sh* command-line restrictions. Another example of such a failure occurs when the number of arguments sent by the server to the CGI application may exceed the operating system's or server's limitations. It may also occur when a string in the 'argv' array is not a valid argument for the CGI application that it enters.

For demonstrated examples of command-line usage see this URL:

<http://hoohoo.ncsa.uiuc.edu/cgi/examples.html>

Make sure when you run these examples that you pay close attention to the values for the command-line variables 'argc' and 'argv'. We'd argue strongly that the first integrity check on accepting CGI input in a program is to make sure that 'argv' is non-zero, and that the number matches the expected number of arguments to your program!

It's also noteworthy that this command-line data-passing is seldom used for handling HTML forms, which can contain arbitrarily large amounts of data. HTML forms normally utilize environment variables to pass data using the POST method, because this method bypasses command-line length restrictions and the occasional data-passing problem that context-switching can sometimes cause in the UNIX runtime environment.

## CGI DATA INPUT

For requests with data attached following the HTTP request header, like HTTP POST or PUT, the data is delivered to the CGI application using a standard input file descriptor. All CGI applications follow this method of reading data. For UNIX operating systems, this uses the so-called standard input device, commonly referred to as *stdin*.

The server will send a number of bytes of information, at least as long as the value specified by the variable CONTENT\_LENGTH, using the *stdin* file descriptor to pass the data to the CGI application. The CGI



application is not required to read all of this data, but neither must it attempt to read more than `CONTENT_LENGTH` bytes.

The server also provides the `CONTENT_TYPE` for data passed to the CGI application. This permits the application to use this information to decide how to interpret the data it receives. At the end of the data stream, the server is not required to transmit an end-of-file marker; both server and application assume that reading will cease immediately after the CGI application reads the number of bytes specified by the value of `CONTENT_LENGTH`.

For example, let's assume you are working with an HTML form that provides its data using the `<FORM> METHOD="POST"`. Let's say this form's results are 15 bytes encoded, and look like `"one=bob&two=cat"` (in decoded form).

In this example, the server will set `CONTENT_LENGTH` to `'15'` and `CONTENT_TYPE` to `'application/x-www-form-urlencoded'`. The first byte on the script's standard input file descriptor will be the character `"O"`, followed by the rest of the encoded string.

## CGI DATA OUTPUT

A CGI application always returns *something* to the client that invokes it, or to the server. The CGI application sends its output to a standard output file descriptor. UNIX calls this *stdout*. The CGI application output might be a document generated by the application (typically HTML, destined for delivery to the client) or it might be data retrieval instructions to the server (e.g., to transfer a file to the client).

Output is usually returned in one of two ways. The first form is non-parsed header output. Using this form, a CGI application must return a complete HTTP response message.

The second form of returned data is parsed header output. A server is only required to support this form. In this case, the CGI application returns a CGI response message. This response consists of headers and a body, separated by a blank line. These headers are either CGI headers that will be interpreted by a server or they are HTTP headers to be included within the response message. Here, the body is optional; but if it is supplied, the body must be prefaced by a MIME Content-Type header.

If the body is excluded, the CGI application must transmit either a Location or Status header. The Location header is used to indicate to the server that the CGI application is returning a reference to a document rather than the document's bits and pieces. The Status header is utilized to provide information to the server about the status code the server should use in its response message to the client. For information about the syntax of these CGI headers, please consult the CGI specification (or at least Chapter 8, which covers CGI input and output in considerable detail).

## Summary

In this chapter we presented the middleware needed to bridge the abyss between Web and non-Web information repositories. In the next chapter, we introduce you to some old and proven programming languages and also to some new and exciting programming languages that you can use to implement CGI applications.