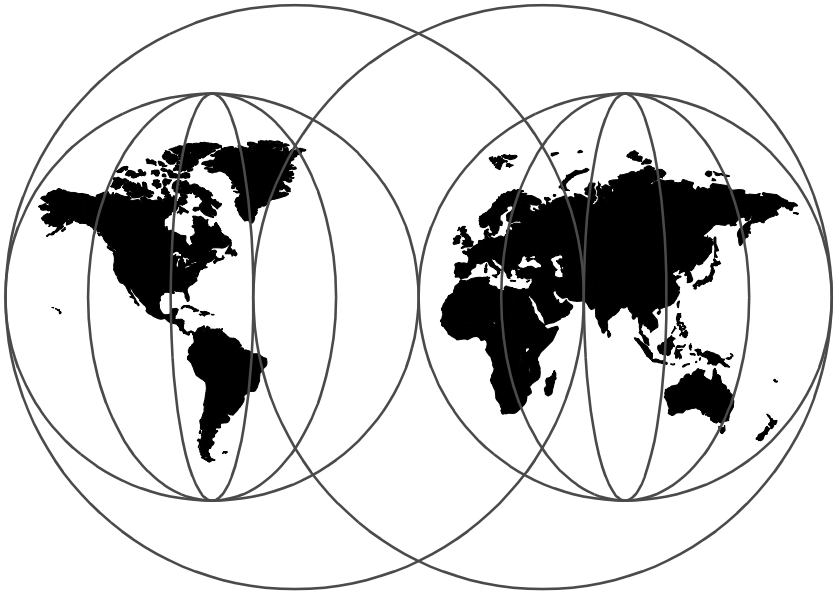
IBM HTTP Server Powered by Apache on RS/6000

Heinz Johner, Jouni Auer, Vitolis Bendinskas, Ng Chang Chyn, Shane Owenby, SunJong Park



International Technical Support Organization

http://www.redbooks.ibm.com

SG24-5132-00

SG24-5132-00

International Technical Support Organization

IBM HTTP Server Powered by Apache on RS/6000

March 1999

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix A, “Special Notices” on page 225.

First Edition (March 1999)

This edition applies to the IBM HTTP Server powered by Apache, Version 1.3.3, as part of the IBM WebSphere Application Server V2.0, Standard Edition and Advanced Edition, Program Numbers 39L9724 and 39L9063 for use on IBM RS/6000.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JN9B, Building 003 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493

Contents

Figures ix

Tables xi

Preface xiii
The Team That Wrote This Redbook xiii
Comments Welcome xv

Chapter 1. The History of the Apache Server 1
1.1 How It All Began 1
1.2 The Success of the Apache Server 2
1.2.1 Who Uses the Apache Server? 4
1.3 IBM's Choice of Web Server 5
1.3.1 The IBM HTTP Server 6
1.3.2 IBM's Participation in the Apache Project 7
1.4 The Next Step 8

Chapter 2. The Building Blocks 11
2.1 Features of the Apache Server 11
2.2 The Apache Server Model 14
2.2.1 The DSO Concept 14
2.2.2 Implementation of DSO in the Apache Server 15
2.3 Standard Modules in the Apache Server 18
2.3.1 Security Modules 20
2.3.2 Translation Modules 21
2.3.3 Server-Management Modules 24
2.3.4 Real-Time-Handling Modules 26
2.3.5 Environment-Changing Modules 28
2.3.6 Protocol-Enhanced Modules 29
2.3.7 Dynamic-Linking Modules 30
2.4 WebSphere and Apache 30

Chapter 3. Installation and Initial Setup 33
3.1 Product Contents 33
3.2 Updates to the IBM HTTP Server 36
3.3 Installation Prerequisites and Considerations 36
3.4 Default File and Directory Structure 37
3.5 Installing the IBM HTTP Server 39
3.5.1 Pre-Installation Setup 39
3.5.2 Installing Using SMIT 40
3.6 Initial Setup 42

3.7 Server Process Structure 43
3.8 Running the IBM HTTP Server for the First Time 44
3.8.1 The apachectl Utility 45
3.9 Uninstalling the IBM HTTP Server 47

Chapter 4. Basic Configuration 49
4.1 Recommended Directory Structure 49
4.2 Starting and Stopping the HTTP Server 51
4.2.1 Automatic Startup 52
4.2.2 Automatic Shutdown 53
4.2.3 Restarting the HTTP Server 54
4.3 Customizing the Configuration File 54
4.4 Enabling DSO Modules 56
4.5 Online Documentation 57
4.6 The Configuration File 57
4.7 Distributed Configuration 58
4.7.1 .htaccess and Performance 59
4.7.2 Restricting the Directives within .htaccess Files 60
4.8 Sections 61
4.8.1 <Directory> 62
4.8.2 <Files> 63
4.8.3 <Location> 63
4.8.4 Sections Processing Rules 64
4.8.5 Recommendations on Sections Usage 66
4.9 Request Mapping 66
4.10 Options 67
4.10.1 Syntax 68

Chapter 5. Advanced Configuration 71
5.1 Virtual Hosts 71
5.1.1 Concepts 72
5.1.2 IP-Based and Name-Based Virtual Hosts 72
5.1.3 Setting It Up 74
5.1.4 Testing 77
5.1.5 Logging 78
5.1.6 Compatibility with Older Browsers 78
5.2 Automatic Directory Indexing 80
5.2.1 Simple and Fancy Indexes 80
5.2.2 Adding Text to an Index 82
5.2.3 Excluding Files from an Index 83
5.2.4 Additional Customization 83
5.2.5 Security Considerations 85
5.3 User Directories 85

5.4 Multiple Language Support 87
5.4.1 Server Configuration 87
5.4.2 Browser Configuration 90
5.5 Customized Error Messages 91
5.5.1 Customizing Error Messages 92
5.5.2 Multilingual Error Messages 94
5.6 File Uploading 95
5.7 Logging 97
5.7.1 Common Log Format 97
5.7.2 Error Log 100
5.7.3 Customizing the Log Format 101
5.7.4 Rotating the Server Logs 103
5.8 Auditing 104
5.9 Other Features 106
5.9.1 Fixing Typos in URLs 106
5.9.2 Caching Proxy Function 106

Chapter 6. Deploying Security 111
6.1 Basic Elements of Security 111
6.1.1 Physical Security 112
6.1.2 Logical Security 113
6.1.3 Authentication Schemes Supported by the IBM HTTP Server . . 118
6.2 Basic Authentication 118
6.2.1 Setting Up Basic Authentication 119
6.2.2 Using the .htaccess File 123
6.2.3 Authentication Files and Databases 124
6.2.4 Performance Impact of Basic Authentication 127
6.3 The HTTP/1.1 Digest Authentication 127
6.4 Secure Sockets Layer, SSL 129
6.4.1 Principles of SSL 130
6.4.2 Establishing the SSL Connection 131
6.4.3 Cipher Specifications Supported by the IBM HTTP Server 133
6.4.4 The Alphabet Soup 134
6.4.5 Creating a Self-Signed Certificate 136
6.4.6 Using Certificates Signed by a Well-Known Trusted CA 139
6.4.7 Requesting a Certificate from an Unknown CA 142
6.4.8 Configuring the HTTP Server to Use SSL 143
6.4.9 SSL and Virtual Hosts 145
6.5 SSL Client Authentication 146
6.5.1 Client Certificates and the IBM HTTP Server 148
6.6 Security Considerations in the Server Configuration File 149

Chapter 7. Performance and Scalability 153
7.1 Basic Performance Consideration 153
7.1.1 Link Bandwidth 154
7.1.2 Hardware and Operating System 154
7.1.3 The Web Server 158
7.2 Performance Monitoring 165
7.2.1 Hardware and Operating System 165
7.2.2 Web Server 169
7.3 Scalability for the IBM HTTP Server 171
7.3.1 Load Balancing 171
7.3.2 File Sharing 174

Chapter 8. Building HTTP Server Modules 177
8.1 The Programmer's View of DSOs 177
8.1.1 Hooks for the Config Phase 179
8.1.2 Hooks for the Request Phase 180
8.1.3 DSO Reference Lists 183
8.2 The Apache Information Module (mod_info) 185
8.2.1 Building the Apache Information Module (mod_info) 186

Chapter 9. Migration Considerations 191
9.1 IBM ICSS and Lotus Domino Go Webserver 191
9.1.1 Installation 191
9.1.2 Directory Structures 192
9.1.3 Basic Configuration 192
9.1.4 Request Mapping 193
9.1.5 Virtual Hosts 194
9.1.6 Authentication and Access Control 195
9.1.7 Logging and Reporting 196
9.1.8 Web Applications 197
9.2 Netscape FastTrack and Enterprise Server 198
9.2.1 Installation 199
9.2.2 Directory Structures 199
9.2.3 Basic Configuration 200
9.2.4 Request Mapping 202
9.2.5 Virtual Hosts 204
9.2.6 Authentication and Access Control 207
9.2.7 Logging and Reporting 209
9.2.8 Web Applications 210

Chapter 10. Web Applications 213
10.1 Concepts 213
10.2 CGI Programs 214
10.2.1 Server Configuration 215

10.2.2 Environment Variables 216
10.2.3 CGI Performance Considerations 217
10.2.4 CGI Security 218
10.3 Modules 218
10.4 Server-Side Includes 219
10.4.1 Server Configuration 219
10.4.2 Security Considerations 220
10.5 Image Maps 220
10.6 Web Applications Development Languages 222
10.6.1 C 222
10.6.2 Shell Script 222
10.6.3 Perl 222
10.6.4 Java 223
10.6.5 PHP 223
10.6.6 REXX 223
10.7 WebSphere Application Server 223

Appendix A. Special Notices 225

Appendix B. Related Publications 227
B.1 International Technical Support Organization Publications 227
B.2 Redbooks on CD-ROMs 227
B.3 Other Publications and Links 228

How to Get ITSO Redbooks 229
IBM Redbook Fax Order Form 230

List of Abbreviations 231

Index 233

ITSO Redbook Evaluation 241

Figures

1. The Apache Server Model 17
2. Basic WebSphere Architecture 31
3. SMIT Screen after Successful Installation 41
4. IBM HTTP Server Welcome Screen 45
5. IP-Based Virtual Hosts 73
6. Name-Based Virtual Hosts 74
7. Error Page for Compatibility on Old Browsers 79
8. Example of a Default Automatic Directory Index 81
9. Example of a Fancy Automatic Directory Index 81
10. Directory Index with Additional Text 82
11. Type Map Variants List 88
12. MultiViews Variants List 89
13. Netscape Communicator Language Setup 90
14. Microsoft Internet Explorer Language Setup 91
15. Standard Error Message 92
16. Error Message with Server Signature 93
17. Example of Custom Error Message 94
18. Netscape Communicator Proxy Setup 109
19. Logical Structure of a Web Site (Example) 116
20. Authentication Pop-Up Dialog 120
21. SSL in Relationship with Other Protocols 131
22. SSL Handshake 132
23. Creating New Key Database with IKEYMAN 137
24. Creating a Self-Signed Certificate 138
25. Filling in the Certificate Information 139
26. Selecting the Personal Certificate Requests Setting 141
27. Receive Certificate from File 142
28. Selecting Signer Certificates in IKEYMAN 143
29. Marking/Checking a CA’s Root Certificate as Trusted 149
30. Basic Internet Connection 153
31. Factors Affecting Performance of a Web Server 155
32. Principle of Round-Robin DNS 172
33. eNetwork Dispatcher Operation Flow 173
34. Hook and Handler Relation 178
35. Client Request Parsing Process 181
36. Sample Output from mod_info 186
37. HTML Form Example 214
38. CGI Environment Variables 217

Tables

1. Web Server Market Penetration (Netcraft Ltd., February 1999) 3
2. Some Large Sites Running the Apache Web Server 4
3. Advantages of Dynamic Linking over Static Linking 15
4. Tested Platform Supporting DSO 16
5. Standard Modules in the Apache Server 19
6. Standard Modules in the IBM HTTP Server Filesets 35
7. Fileset Installation Matrix 40
8. Default and Recommended Directory Structure 50
9. Commonly Seen Status Codes in HTTP Requests 98
10. Custom Log Format Arguments 102
11. Operation Arguments for dbmmanage 126
12. Cipher Specifications Supported by the IBM HTTP Server 133
13. Attributes Included in the Client's X.509 Certificate 147
14. Module Matrix, Config Phase 183
15. Module Matrix, Request Phase 184
16. Directory Structure (Comparison) 192
17. Basic Directives (Comparison) 192
18. Directory Structure (Comparison) 200
19. Basic Directives (Comparison) 200

Preface

The reliable operation of Web servers is very important for organizations that are present on the World Wide Web. The offering of Internet services as a new business opportunity is largely dependent on Web servers and has grown dramatically in the past few years.

To the surprise of many software vendors that offer their own products, a non-commercial, freely available software has become the de-facto standard for Web servers. Named Apache, this Web server has quickly gained more than a 50% share in the market and it has proven to be a very versatile and reliable Web server software. Due to this occurrence, IBM has chosen to support the development of Apache and make it available to customers with valuable security enhancements as part of its e-business product suites with full IBM support.

This redbook gives a broad understanding of the architecture of the new *IBM HTTP Server powered by Apache* on RS/6000, and it will help you install, tailor and configure that software successfully.

The reader of this redbook is assumed to have some experience in running a Web server.

The redbook introduces the general concepts of Apache and then describes its installation and basic configuration. Advanced configuration options are then described, followed by security implementation and setup information. Performance and management issues are covered, as well as the development of new modules that extend the functions of a Web server. This redbook ends with a discussion of the migration from other Web server software and of running applications on a Web server.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.

Heinz Johner is an Advisory Systems Engineer at the International Technical Support Organization, Austin Center. He writes extensively on the Distributed Computing Environment (DCE), security-related areas and eNetwork deployment. Before joining the ITSO, he worked in the Professional Services organization of IBM Switzerland and was responsible for DCE and Systems Management in medium and large customer projects.

Jouni Auer is an I/T Specialist working at the Professional Services in Global Services in IBM Finland. For the last few years, he has been working on customer projects involving Internet servers, Net.Commerce software, SET-payment and other e-business projects. Jouni has focused extensively on UNIX-security and server management issues. He has participated in a project of deployment of the Tivoli management in large scale environment and improving the Internet hosting services in IBM, Finland.

Vitolis Bendinskas is a System Programmer at Sigma Americom, an IBM Business Partner in Lithuania. In the last few years, he has been working with IBM AIX, OS/2, DB2 and Universal Database, Lotus Notes and Domino, Novell NetWare and Microsoft Windows. His areas of expertise include systems management, systems programming, technical support and consulting, as well as systems integration. Vitolis holds a degree in mathematics from Vilnius University in Lithuania.

Ng Chang Chyn is a System Engineer at Asirius Pte. Ltd., an IBM Business Partner in Singapore. He holds a degree in computer engineering from Nanyang Technological University in Singapore and has been working in IBM Business Partners in the IBM RS/6000 AIX field. He also assists with the installation and administration of Web servers such as Netscape FastTrack Server and Microsoft Internet Information Server, as well as developing Web applications for his customers.

Shane Owenby is a Software Engineer working on the Apache/IBM HTTP Server project in Research Triangle Park, North Carolina, USA. He holds a degree in computer science with minors in business, electronics, and mathematics. His areas of expertise include Apache, Solaris, and Linux. Shane has provided technical reviews of technical books from O'Reilly and Associates and book proposals from Addison Wesley-Longman.

SunJong Park is a Senior I/T Specialist working at Sales & Technical Support Center, IBM, Korea. He has many years of experience in RS/6000 and related fields. He was responsible for System Implementation and Management in the customer project before participating in this Apache residency.

Thanks to the following people for their invaluable contributions to this project:

Tara Morrell, IBM Raleigh
Richard Nesbitt, IBM Raleigh
Kevin Vaughan, IBM Raleigh

Special thanks go to the editors for their help in finalizing the text and publishing the book:

Marcus Brewer
Tara Campbell

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in “ITSO Redbook Evaluation” on page 241 to the fax number shown on the form.
- Use the online evaluation form found at `http://www.redbooks.ibm.com`
- Send your comments in an Internet note to `redbook@us.ibm.com`

Chapter 1. The History of the Apache Server

With a great telecommunication infrastructure established by our predecessors, it is no wonder that the evolution of the Internet has conquered the computer world. The Internet quickly became a good channel to provide world-wide broadcasting and dissemination of information. This means improved communication and interaction between people regardless of their geographical location on the globe, which draws the world even closer. Based on definitions of the *hypertext* and Internet protocols, millions were even amazed by the shear capability of the Web servers, that is, simply providing static information.

With the growing contributions to the World Wide Web, Web servers today have advanced in their capabilities to provide information to the public. They no longer only provide static information, but also provide dynamic information based on the requests submitted from clients. Furthermore, data processing was made available on Web servers, too, which allowed applications to process input provided by users. Electronic commerce is now possible over the World Wide Web and users can purchase or sell items with just a click. Inevitably, the concern for security and reliability of a Web server becomes an important issue.

The rate at which the Web community grows comes as no surprise, and one of the reasons is probably the ease of use of Web *browsers*; the “user interface” for the World Wide Web. The technical knowledge of the Web servers, on the other hand, requires more in-depth discussion for someone who wishes to set up a secure and reliable Web server for his or her environment, especially when commercial business is involved. This book discusses the issues involved in setting up the IBM HTTP Server powered by Apache (in the following simply referred to as the *IBM HTTP Server*) on IBM RS/6000.

This first chapter gives you an overview of how the Apache server was created, its features and IBM’s participation in the Apache project.

1.1 How It All Began

The fundamental ideas behind and the basic design of the World Wide Web evolved from work being done at the European Laboratory for Particle Physics (CERN) in Geneva, Switzerland. In its roots, the Apache server was developed at the National Center for Supercomputing Application (NCSA), and it was based on the NCSA HTTP daemon (NCSA HTTPd 1.3). The NCSA Web server at that time was adopted and used by a large number of

webmasters in the market. In mid 1994, however, the development for this Web server stalled and left many webmasters finding their own solutions to problems encountered in their environments. Some of them developed their own extensions and problem fixes, which could be applicable to other webmasters searching for the same solution.

In February 1995, a group of webmasters volunteered to consolidate all information related to the server and placed it in a publicly accessible domain for all the webmasters to access. The *Apache Group* was then formed from people who had made substantial contributions to the Apache server. NCSA later revived the suspended development of their NCSA Web server and two members from that development team joined the Apache Group so that ideas and contributions could be exchanged among both projects.

The Apache Group reviewed some of the enhancements and bug fixes and added them to their own server for testing purposes. It was in April 1995 that the Apache server made its first public release at Version 0.6.2. This name was given since it is the “patched” version (A PAtCHy server) of the NCSA HTTPd 1.3 Web server.

In May-June 1995, some general overhaul and redesign was made to fine-tune the Apache server, along with the introduction of some new features in the Version 0.7.x. The next release of the Apache server at 0.8.8 in August brought about a change in the architecture of the server with the modular structure and API features. At the time of writing this book, the latest level available for the Apache server is at Version 1.3.3, while development is in progress for Versions 1.3.4 and 2.0.

1.2 The Success of the Apache Server

Ever since the introduction of the World Wide Web, Web servers have been busy serving requests from Web users all over the world. With the availability of many variations of Web server products in the market, consumers have the freedom to choose.

How does the Apache server compare to other Web servers?

As an open source HTTP server (the term *HTTP server* is commonly used as a synonym to *Web server* throughout the literature), the Apache server acquired a remarkable result in capturing a substantial part of the Internet market. According to a Web server survey conducted by Netcraft Ltd. on a monthly basis, the Apache server has already achieved more than 50% of the market share (see Table 1, for the Netcraft survey results as of February

1999). For further analysis of the Web server market, please refer to Netcraft's Web site at `http://www.netcraft.com`.

Table 1. Web Server Market Penetration (Netcraft Ltd., February 1999)

Server	February 1999	Percent
Apache	2350748	54.7
Microsoft IIS	1008566	23.5
Netscape Web Servers (FastTrack and Enterprise)	178027	4.1
Other Web Servers	approx. 760000	17.7

Thanks to the Apache Group and their goal to develop and support open software, webmasters from all over the globe have the opportunity to obtain the Apache server for free, including all the latest *modules* and fixes. Despite the fact that it is free, the Apache server runs on the most widely used platforms with great performance and extensible features. It also progresses with the advance development of the HTTP standards, such as the latest HTTP/1.1 protocol, and remains compatible with HTTP/1.0. An impressive graphical representation of the survey conducted by Netcraft Ltd. on the growth of Web servers usage over the past few years can be found at: `http://www.netcraft.com/survey`.

As the most popular Web server, there is no doubt about the quality and features that Apache can offer. Since it was developed with platform-independence in mind, the Apache server runs traditionally on all major variants of UNIX, Windows 95 and NT (only after Apache 1.3b3) and OS/2 Warp. In addition, it also supports the HTTP/1.1 protocol and utilizes the APIs specific to the operating systems, such as the Internet Server Application Programming Interface (ISAPI) used on Windows NT. The modular design (to be covered in 2.2, "The Apache Server Model" on page 14) of the Apache server enables flexibility in customizing the Web server to any specific environment. Apart from the standard modules that come with the Apache server, new modules are also developed and contributed by individuals and organizations to help other webmasters function effectively in their environment. Besides adding more beneficial modules to further improve the functionality of the server, any problem discovered can be reported to the Apache Group and actions will be taken so that the corresponding fix is readily available in a minimum amount of time. As of today, this avalanche process of development on the Apache server, known as the Apache Project, has already created a robust and reliable server as the user base expands throughout the globe. The Apache Group has set forth very high standards for code contributions, which results in a high level of

quality and robustness of the Apache server. In fact, some special design principles have been introduced just to increase the robustness of the Apache server and to overcome some common problems in operating systems, such as memory leaks, that could negatively interfere with the Web server.

What are Modules?

The architecture for the entire Apache server (to be discussed in 2.2, "The Apache Server Model" on page 14) can be imagined as a server constructed with pieces of building blocks, called *modules*. Each of these modules performs specific tasks for the main core block that does all the coordination work. The standard modules come packaged with the Apache server when downloaded from the Apache Group's Web site (`http://www.apache.org`). Developers around the world make customized modules available from `modules.apache.org` or from the respective developer's Web site.

1.2.1 Who Uses the Apache Server?

The Apache server is the Web server behind thousands of Web sites on the Internet today. Table 2 contains just a few of the larger companies and their respective URLs that are currently using Apache as their corporate Web servers.

Table 2. Some Large Sites Running the Apache Web Server

Organization Name	URL
Webcrowler	`http://www.webcrowler.com`
Financial Times	`http://www.ft.com`
The British Monarchy	`http://www.royal.gov.uk`
W3C Consortium	`http://www.w3.org`
SIPB at MIT	`http://www.mit.edu`
Hotmail	`http://www.hotmail.com`
GeoCities	`http://www.geocities.com`
The Internet Movie Database Ltd.	`http://www.imdb.com`
JavaSoft	`http://java.sun.com`
Organic Online, Inc.	`http://www.organic.com`

Of course, the information listed in Table 2 is only a snapshot, valid at the writing time of this book.

1.3 IBM's Choice of Web Server

On June 22, 1998, IBM made the following announcement:

IBM will ship the Apache HTTP server with the IBM WebSphere Application Server, helping current Apache users to evolve to e-business solutions. As part of the WebSphere Application Server package, IBM will provide commercial, enterprise-level support for the Apache HTTP Server. In addition, IBM will be a full participant in the Apache HTTP Server Project, a collaborative development effort, and will make contributions to enhance the capabilities of the Apache HTTP Server.

What are the benefits for this partnership? With the recognition of the quality and power of the open source development of the Apache Group, IBM can assure the delivery of an HTTP server that is exactly what customers want. As the fastest growing server in the Internet world, the Apache server allows IBM to follow the HTTP server market with the help of its leading momentum. In addition to the large installed base and webmasters who are already familiar with the Apache server, IBM has also a good starting point in creating awareness of the IBM WebSphere Application Server in the market (see note below). An overview of how the Apache server fits into the IBM WebSphere Application Server package is given in 2.4, “WebSphere and Apache” on page 30.

Conversely, IBM helps the Apache Group to boost the status of a freeware product into a larger enterprise market. Large corporations which hesitated to adopt the use of a widely available product with no brand name are now more assured as IBM joins in the support in the development of the Apache server. With the additional force of engineers and developers from IBM, the Apache Group has a stronger development team focussed on providing open-source software. IBM customers can benefit from having their common support structure for the IBM HTTP Server (which is IBM's name for the Apache server) while non-IBM customers can benefit from quality and function enhancements brought to the Apache server by IBM and its customers.

About IBM WebSphere

The IBM WebSphere Application Server V2.0 package consists of a comprehensive set of Web-based application enablers for electronic commerce application development. While the application server programs (servlets) that run on top of an HTTP server deal with linking client applications with data, the HTTP server deals with the communication between the Web users via the Web browsers and the database software.

1.3.1 The IBM HTTP Server

IBM named the Apache server to be shipped with the IBM WebSphere Application Server V2.0 the *IBM HTTP Server*. Besides AIX, versions running on Windows NT and Solaris are also available from IBM. At the time of writing, IBM has announced that a version will be provided for AS/400, too. The IBM HTTP Server is basically a Web server based on the Apache server V1.3.3, with some additional adjustments that are discussed in this section. As the development of the Apache server in the Apache Group progresses, IBM will incorporate new technology and fixes in the IBM HTTP Server. It should be mentioned that IBM is not developing its own version of Apache, but rather ships a compiled and ready-to-use version that strictly follows the open software concept of Apache.

Installation for the IBM HTTP Server has been made easier by catering for their specific platform. System administrators of operating systems such as AIX simply need to install the “installp” version through SMIT (System Management Interface Tool) or using the “install shield” for NT environment. Unlike the IBM HTTP Server, the downloaded Apache server, although available in binary versions for most operating systems, requires more steps before the relevant files are extracted for installation and, if a customized version is desired, the code is compiled and linked.

The IBM HTTP Server comes as a compiled and tested version of the Apache server for the specific platforms. Users have the convenience of using the modular structure of the IBM HTTP Server and including or excluding modules to suit their needs. All configurations pertaining to the IBM HTTP Server are consolidated into one configuration file (httpd.conf) provided with the product, as compared to the downloaded Apache server, which provides three files (httpd.conf, srm.conf and access.conf) to comply with previous versions which may require more maintenance. Furthermore, as the name implies, no compilation is needed for the pre-compiled IBM HTTP Server. Installation of the IBM HTTP Server on RS/6000 is explained in Chapter 3, “Installation and Initial Setup” on page 33.

To prevent any security breaches, the most important enhancement in the IBM HTTP Server over the Apache server is the addition of the Secure Sockets Layer (SSL) function, which supports up to SSL V3 Triple-DES SHA. A more detailed description and how to set up SSL can be found in 6.4, “Secure Sockets Layer, SSL” on page 129.

What is Secure Sockets Layer?

Secure Sockets Layer (SSL), developed by Netscape Communications Corporation, is a widely accepted protocol that uses the RSA public key cryptography for communication authentication and encryption between clients and servers.

Due to U.S. export regulation restrictions, IBM cannot provide the source code for the IBM HTTP Server because it would expose interfaces to SSL libraries that are controlled by these regulations. However, webmasters can still add modules dynamically to the IBM HTTP Server, as we will discuss in 2.2, “The Apache Server Model” on page 14 and Chapter 8, “Building HTTP Server Modules” on page 177.

1.3.2 IBM’s Participation in the Apache Project

IBM has a group of people dedicated to the support of Apache and the IBM HTTP Server. A test team is involved in testing new developments on the server. IBM has two representatives as members in the core team of the Apache Group. The core team deals with the server core development; there is another team specialized in developing value-added modules. The entire team has already made a major contribution in implementing the *Fast Response Cache Accelerator* (FRCA) feature on the Apache Server that runs on Windows NT. This feature uses the kernel space cache that dramatically increases the performance of the Apache server. Due to IBM’s involvement, it can be assumed that the next platform to have the FRCA technology implemented will be AIX.

Developments made within IBM will go through an internal review before they are submitted to the Apache Group for voting. Thus the compatibility between the IBM HTTP Server and the Apache server is maintained. Similarly, any problems reported by customers pertaining to the core server will be, perhaps along with some suggested corrections, fed back to the Apache Group for their necessary actions. However, bugs pertaining to the additional functions that IBM offered will be dealt with by IBM developers.

For IBM customers, rather than posting the questions and doubts to the Apache Group or searching the answers through the massive documentation

provided on the Web or news group, they can get support on the IBM HTTP Server through established IBM support channels just like with any other IBM product. IBM customers may also visit the IBM newsgroup ibm.software.websphere.http-servers at `news.software.ibm.com`. This newsgroup is maintained by the IBM development and support teams.

1.4 The Next Step

The Apache Group will continue to offer the Apache server as a free open source server for the public, and keep in pace with the new developments of the Web as well as the HTTP protocol. They will also continue to respond to users by reviewing suggestions, terms of fixes, and improvements. IBM is following the Apache Group towards the development of an even more enhanced Web server that can meet the future needs of advancement in the Web environment. Apart from contributing to the Apache Project, the Web server development team at IBM is looking into the upgrade in functionality and performance for the base HTTP Server in future releases.

At present, the Apache Group is looking into these areas of enhancement for the next release of the Apache server, Version 2.0:

- Multi-Threading — Multi-threaded support involves the use of only one HTTP daemon handling requests via inter-process communication for the coordination of operations such as content retrieval. This method, which is already implemented on the Apache server for the Windows NT, may provide better overall performance on those platforms where threads are well implemented. The current version running on UNIX spawns multiple processes of the HTTP daemon to handle the requests, which might consume more resources (but provides the advantage that a single process can abort without affecting the others).
- Better system configuration — At present, the file *Configure,* which contains hard-coded definitions for some particular platforms is being reviewed for better compatibility with other platforms that are not supported now.
- Stabilized API model — The developers are deriving a more stabilized API model to prevent disruption in modules development and also to ease the process of developing new modules. Documentation on the use of the modules should also be informative for anyone who wants to write or use the modules more efficiently in their environment.
- Configuration Syntax — The syntax used in languages for the configuration files are also reviewed to remove any limitations and ambiguities from the webmasters.

- Source code language — The Apache release 2.0 may be written in C++.
- Graphical user interface (GUI) — The current version of the Apache server requires the webmaster to be conversant with the AIX environment, or at least UNIX commands. Though a command-based interface is enough to do wonders, a graphical user interface (GUI) would ease administration for less experienced webmasters, as well as the experts who wished to eliminate complication in advanced server configurations. As of the writing of this book, there is already some development under the Apache GUI-Dev Project that is aimed to accomplish this task in the near future.

Since IBM is working together with the Apache Group, customers can expect to see the above enhancements to also be present in the IBM HTTP Server.

Chapter 2. The Building Blocks

There are millions of people accessing the World Wide Web every day. This is mostly because it requires only basic skills to use a Web browser to surf the Web. Webmasters, on the other hand, need to know a little more. They could, of course, simply follow guided instructions and have a Web server up and running without the need of an in-depth understanding of the Web server they are using. For those webmasters who want or need more than just step-by-step checklists, this chapter lists and describes the working mechanism of the Apache Web server. Since the IBM HTTP Server is basically identical to the Apache server (with the exception of the added SSL support), the original Apache server model is selected as a base for the discussion in this chapter.

In the section that follows, some general features of the Apache server are explained. In the second section of this chapter, some specialities about the architecture are discussed. The sections that follow explain all the modules that come with the Apache server and how the IBM HTTP Server fits in the IBM WebSphere architecture.

2.1 Features of the Apache Server

As a Web server, the Apache core server fulfills the basic necessity of any Web server. In addition, due to the flexibility of modular design, the server's functionality can be greatly configured and enhanced to suit any environment easily. The following sections list and describe the most important features of the Apache server, which are available in the downloadable version of the Apache server (`http://www.apache.org/dist/binaries`). The IBM HTTP Server is identical to the Apache server (except for IBM's addition of the SSL security) so these features apply to both the Apache server and the IBM HTTP Server.

Dynamic Shared Objects (DSO) – The most important feature of the Apache server is probably the concept of modules. The core Apache server provides basic Web server functionality while requiring very few resources. As requirements grow, functionality can easily be added to the Apache server through modules. There are a lot of modules available for the Apache server. These modules can either be built into the core Web server or loaded dynamically. Due to the U.S. government restrictions on the export of SSL technology, the IBM HTTP Server cannot be shipped in a state to allow modules to be statically built into the Web sever core. Due to these restrictions, modules can only be added to the IBM HTTP Server as dynamic shared objects (DSOs).

DSOs are mentioned and explained at various places in this book, specifically in 2.2.1, “The DSO Concept” on page 14.

HTTP Compliance – The Apache Group is always on the cutting edge in standards activities. The HTTP protocol implementation in Apache was chiefly architected by one of the HTTP/1.1 authors. Also Apache is being used as one of the two reference implementations for the HTTP/1.1 proposal to move to draft standard. This ensures that Apache (and therefore the IBM HTTP Server) will be compatible with the greatest percentage of the Web clients and agents in use on the Internet today. For more information on Internet Protocols visit `http://www.w3.org/Protocols/`.

Persistent Connections – This feature allows inline objects to be downloaded from the server using a single connection. Web pages today have evolved from a static text document to the inclusion of multi-media presentations containing sounds or animations. Each of these objects require separate TCP connections for downloading, which causes a great deal of unneeded TCP overhead, including opening the new connection, and closing down the new connection. With the introduction of persistent connection in the HTTP/1.1 specification, the server keeps the connection opened and assumes subsequent requests on that same connection by default. Since the same TCP connection can be used, the server saves the overhead of the new TCP connection. In order for persistent connections to be used, both the server and client must support HTTP 1.1.

Byte Ranges – With HTTP/1.1, browsers can request a specific range of bytes for the document to be downloaded in a single request. This is useful when the requested document is lengthy and only a specific portion of it is required. In addition, this feature can also be used to resume the process of downloading after a transfer of a document has been interrupted in a previous attempt.

Virtual Hosts – One of the key features of the Apache server is the support of virtual hosts (to be further discussed in 5.1, “Virtual Hosts” on page 71). A single server can serve requests sent to different server names or even IP addresses. Thus, a single server can appear and behave exactly as if there were more than one server involved. This feature is very useful for running a single Web server machine to serve multiple domains.

Error Logging and Processing – By default, as governed by the HTTP protocol, errors encountered by the Web server when attending to client requests are returned to the browsers with an error number and some standard messages. The Apache server has two log files, *access_log* and *error_log*. The former is used to log information about who requested which

information with the resulting status, the latter contains information about failed requests and the probable cause. Webmasters can customize error messages, to be covered in 5.5, “Customized Error Messages” on page 91. Besides the messages, the server can be configured to execute scripts that analyze the problem or perform some remedial actions for the error conditions before returning informative responses to the clients.

Content Negotiation – Another key feature of the Apache server is its ability to perform content negotiation. This feature allows the server to negotiate with the client before retrieving the most appropriate information based on the client's settings. In simpler terms, the browser tells the server how capable it is to accept different kinds of content and the server determines the most appropriate type of content to be returned to the browser. This is extremely useful for serving selective contents such as language-specific documents that are retrieved based on the settings on the browsers. Another common area of use are requests for information concerning a particular media-type, such as graphic images (for example GIF or JPEG), where the browser can inform the server of its preferences. The configurations required on the server and the browser to support content negotiation is dealt with in 5.4, “Multiple Language Support” on page 87.

Automatic Index File Selection and Index Creation – With this feature, webmasters can specify default directory index files (for example index.html) or the format of automatically created directory indexes to be returned to the client when an index was requested for a specific directory, for example by specifying a “/” character as the last character in an URL. The IBM HTTP Server (Apache) supports several options for customizing automatically created directory indexes, including HTML file headers, text files or even involving CGI programs. To learn how to configure automatic directory indexing, refer to 5.2, “Automatic Directory Indexing” on page 80.

User Authentication – There are several methods provided by the Apache server for implementing user authentication. Webmasters can make use of flat files to create a database containing the list of users allowed to retrieve information from the server. User authentication for large groups of users can be also done using DBM files supported by the Apache server to speed up the process of checking user names against the list. More information about user authentication and security can be found in Chapter 6, “Deploying Security” on page 111.

Stability – Although this word sounds like it is from a sales brochure, it should be mentioned that the development principles of the Apache server involve a number of special precautions that guarantee a high level of robustness. To mention two of them: The Apache Group sets high standards

on how the source code of any enhancements or bug fixes must be written in order to be evaluated and accepted. Another precaution is that the Apache server consists of multiple processes within the operating system and that each only processes a definitive number of requests before it terminates intentionally. This avoids certain problems that can arise with long-living processes (see also 3.7, “Server Process Structure” on page 43).

Popularity – Since every new software introduced also introduces parasite costs for training and education of the involved personnel, it is beneficial when that software is well known on the market. Apache, as was shown in 1.2, “The Success of the Apache Server” on page 2, is the most successful Web server on the market, and thus, most Web administrators already have a good understanding of at least the fundamentals of this Web server.

2.2 The Apache Server Model

Before we start exploring each component of the Apache server, there is one important concept that needs to be mentioned and understood. It is the concept of *Dynamic Shared Objects* (DSO). It differentiates the Apache Web server from most, if not all, other Web servers.

2.2.1 The DSO Concept

Traditionally, applications used a method known as static linking to resolve procedure calls in the executable programs. The linking is done when the executable program is compiled and linked. Dynamic linking, on the other hand, allows external symbols (procedures) that are referenced in the application code, but are defined in a separate shared library. They are resolved only during run-time execution. Dynamic shared objects (DSOs) in the Apache server are fragments of application code (shared resources) that may be loaded and executed only at run-time when necessary. There are basically two implementations of dynamic linking making use of DSO.

- The first method is to make use of a dynamic or shared library containing the DSOs which are normally located in a system directory. When the program to be executed is built, the linker hard codes the path of the dynamic library into the program. With these references to the location of the objects (library), the UNIX loader then resolves the unresolved symbols in the program at start-time.
- The other method focuses on individual object files usage. These object files are usually located at a program-specific directory, and they are loaded only when required by the application code.

One of the advantages of using dynamic libraries is the efficient use of disk space since the libraries only need to be stored once for all the programs sharing them. Similarly, memory resources are used more efficiently because only those libraries are loaded that are actually necessary for the application to function. Also, the latter method allows for more efficient testing of single object files without the need to load the entire library. On the whole, dynamic linking contributes to many different areas in application design, as shown in Table 3.

Table 3. Advantages of Dynamic Linking over Static Linking

Scope	Description
Scalability	There is virtually no limit to the amount of compiled code that can be (dynamically) added.
Development Time	The development and test cycle of a program can be drastically reduced as it gets larger in size. Even small changes can take an extraordinary amount of time to re-link (and test) a whole executable if it has to be updated.
Memory	Memory consumption is normally reduced because not all possible dynamic modules need to be loaded at all times.
Disk Space	Dynamic linking saves disk space since common code is not included in each executable program.
Down Time	The server needs not be shut down in order to add new dynamic modules.
Start Time	Start time is reduced because the shared library code does not need to be loaded at start time, but only when required.
Performance	Performance may be improved because fewer page faults will be generated when the shared library code is already in memory.

Note: It should be mentioned at this point that Apache also offers the option of either DSOs or statically linking modules to the base server, which requires not only a configuration change, but also a compile and link operation. However, due to the addition of the SSL security code, which is subject to U.S. export restrictions, the IBM HTTP Server cannot be recompiled. So, the only option for modules with the IBM HTTP Server is through the DSO support.

2.2.2 Implementation of DSO in the Apache Server

As of the 1.3 version of the Apache server, both methods of using modules mentioned above are supported. Thus, the HTTP main process can be seen

as the executing program, while the object files can be seen as the Apache modules that build up much of the functionality of the Apache server.

Using the second method, individual modules, when built as DSOs, can be easily added by using the *LoadModule* and *AddModule* directives in the server configuration file. Also, a new program called *apxs* (originated from “APache eXtenSion”) is available to simplify the creation of DSO files for the Apache modules. The use of DSOs is increasing and Apache already supports dynamic modules on the operating systems listed in Table 4.

Table 4. Tested Platform Supporting DSO

Operating System	Version
FreeBSD	2.1.5, 2.2.5, 2.2.6
OpenBSD	2.x
NetBSD	1.3.1
Linux	Debian/1.3.1, RedHat/4.2
Solaris	2.4, 2.5.1, 2.6
SunOS	4.1.3
Digital UNIX	4.0
IRIX	6.2
HP/UX	10.20
UnixWare	2.01, 2.1.2
SCO	5.0.4
AIX	3.2, 4.1.5, 4.2, 4.3
ReliantUNIX/SINIX	5.43
SVR4	-

The concept of DSOs is one of the key feature of the Apache server. With this dynamic capability, as briefly mentioned in Table 3 on page 15, modules can be loaded into the server process space at run-time, thus resulting in a significant overall reduction in memory usage. In addition, the development process for extending the functionality of the server is greatly enhanced without the need for re-compilation. For further information about implementing modules, see Chapter 8, “Building HTTP Server Modules” on page 177, or read the documentation at `http://www.apache.org/docs/dso.html`.

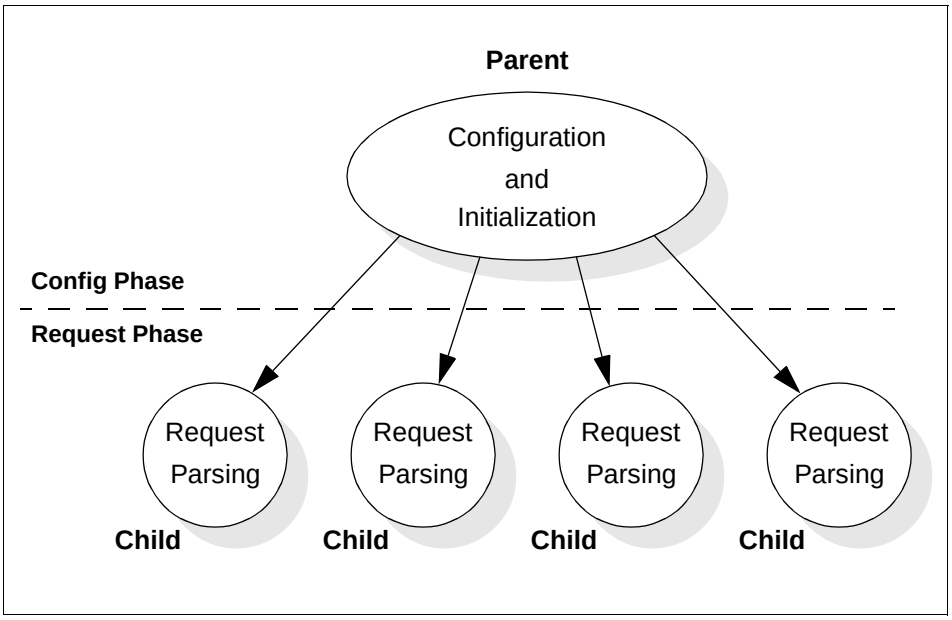


Figure 1. The Apache Server Model

The block diagram shown in Figure 1 shows a simplified model of the Apache server. The parent at the top depicts the Apache main server process setting up its own environment before multiple child processes are spawned based on the specific configuration parameters. Basically, the entire operation can be divided into the processes as shown, where the parent reads the configuration files and performs module initialization. Thereafter, the server deals with initialization of the child processes to handle requests from the clients. As mentioned earlier, the HTTP processes can be viewed as the core executable programs in the DSO concept discussion, while the modules are the dynamic modules loaded at run-time. Most modules support their specific set of directives. Directives are used widely in the configuration file(s) to denote a certain behavior or instructions to be performed or enforced on resources, such as files or directories.

The directives within the server's configuration file(s) are the webmaster's controls for the modules. Some modules require little configuration, while others support a fairly larger number of directives that control their operation.

2.3 Standard Modules in the Apache Server

As explained earlier, a module is simply a piece of C code that is compiled into the Apache server statically or added dynamically at runtime. The set of standard modules that come along with the downloaded package is listed in Table 5 on page 19. Since the IBM HTTP Server is based on Apache, it has the ability to either build modules statically into the core or load them dynamically. Due to U.S. export regulations, the IBM HTTP Server is not legally allowed to ship complete source code due to the inclusion of the SSL technology. Because of this, modules can only be added dynamically to the IBM HTTP Server. Due to security concerns, the IBM HTTP Server does not provide the mod_info module, which provides a list of information about the configuration of the server and the modules that are loaded. However, if webmasters choose to use this module, they can follow the instructions outlined in Chapter 8, “Building HTTP Server Modules” on page 177. The modules shipped with the Apache server are listed in

Table 5 on page 19, and the modules included with the IBM HTTP Server are listed in Table 6 on page 35. All of the modules are briefly discussed below.

Table 5. Standard Modules in the Apache Server

Name	Description	Classification
mod_access	Host-based access control	Security
mod_actions	Filetype/method-based script execution	Real-time
mod_alias	Aliases and redirects	Translation
mod_asis	The “asis” file handler	Protocol enh.
mod_auth	User authentication using text files	Security
mod_auth_anon	Anonymous user authentication, FTP-style	Security
mod_auth_db	User authentication using Berkeley DB files	Security
mod_auth_dbm	User authentication using DBM files	Security
mod_autoindex	Automatic directory listings	Real-time
mod_cern_meta	Support for HTTP header metafiles	Protocol
mod_cgi	Invoking CGI scripts	Real-time
mod_digest	MD5 authentication	Security
mod_dir	Basic directory handling	Translation
mod_env	Passing of environments to CGI scripts	Management
mod_expires	Apply Expires: headers to resources	Protocol enh.
mod_headers	Add arbitrary HTTP headers to resources	Protocol enh.
mod_imap	The imagemap file handler	Real-time
mod_include	Server-parsed documents	Real-time
mod_info	Server configuration information	Management
mod_log_agent	Logging of User Agents	Management
mod_log_config	User-configurable logging replacement for mod_log_common	Management
mod_log_referer	Logging of document references	Management
mod_mime	Determining document types using file extensions	Translation
mod_mime_magic	Determining document types using "magic numbers"	Translation
mod_negotiation	Content negotiation	Translation
mod_rewrite	Powerful URI-to-filename mapping using regular expressions	Translation
mod_setenvif	Set environment variables based on client information	Env.-changing
mod_so	Support for loading modules at runtime	Dynamic-linking Real-time
mod_speling	Automatically correct minor typos in URLs	Management
mod_status	Server status display	Management
mod_unique_id	Generate unique request identifier for every request	Translation
mod_userdir	User home directories	Management
mod_usertrack	User tracking using Cookies (replacement for mod_cookies.c)	

2.3.1 Security Modules

As the name implies, modules under this category deal with security enforcement measures on the server. The Apache server provides modules to suit different authentication schemes. These modules generally handle the three phases of user authentication and authorization: Verify User ID, Verify User Access and Check Access.

mod_access
This module deals with basic security checks based purely on hostname or IP address at the early stages of request parsing.

Directives:

- order – 'allow,deny', 'deny,allow', or 'mutual-failure'
- allow – 'from' followed by hostnames or IP-address wildcards, or ’env=’
- deny – 'from' followed by hostnames or IP-address wildcards or ’env=’

mod_auth
This module deals with user security checks based text files after the Check Access stage.

Directives:

- AuthUserFile – Text file containing usernames and passwords
- AuthGroupFile – Text file containing group names and member usernames
- AuthAuthoritative – Set to 'off' to allow access control to be passed along to other modules if the user ID is not known to this module

mod_auth_anon
This module offers the use of anonymous log in using the username as “anonymous” and password as the e-mail address of the requesting user, similar to the FTP-implementation.

Directives:

- Anonymous – A list of user IDs separated by spaces
- Anonymous_MustGiveEmail – Controls the need for an e-mail address
- Anonymous_NoUserId – If set ’on’, no user ID is required
- Anonymous_VerifyEmail – If set ’on’, e-mail address is verified
- Anonymous_LogEmail – If set ’on’, e-mail address is logged
- Anonymous_Authoritative – If set 'on', user ID must fulfill one of the users specified in the list under the Anonymous directive

mod_auth_db
This module offers user security checking using Berkeley DB files in DB format that are used to hold information pertaining to a large group of users more efficiently than just plain text files.

Directives:

- AuthDBUserFile – DB file containing usernames and passwords
- AuthDBGroupFile – DB file containing group names and member usernames
- AuthDBAuthoritative – Set to 'off' to allow access control to be passed along to other modules if the user ID is not known to this module

mod_auth_dbm
This module offers user security checking using DBM files which are more efficient and convenient to manage when the number of users is large. A DBM file contains a key (normally the username) used for fast retrieval through indexing, and a value (the encrypted password).

Directives:

- AuthDBMUserFile – DBM file containing usernames and passwords
- AuthDBMGroupFile – DBM file containing group names and member user names
- AuthDBMAuthoritative – Set to 'off' to allow access control to be passed along to other modules if the user ID is not known to this module

mod_digest
This module offers user security checking using MD5 Digest Authentication. Digest Authentication involves the client browser encrypting the user's password before sending it to the server for decrypting. With this module, the server is capable of handling this kind of security measures provided the client's browser has the capability to do the encryption.

Directives:

- AuthDigestFile – digest file containing user IDs and passwords

2.3.2 Translation Modules

Modules in this category deal mainly with the translation of Web addresses.

mod_alias
This module translates Web addresses to filesystem locations in the document tree.

Directives:

- Alias – Maps a virtualname to a realname
- ScriptAlias – Maps a virtualname to a realname and executes the script
- Redirect – Provides an optional status, then redirects old URL to new URL
- AliasMatch – Uses a regular expression and maps it to a filename
- ScriptAliasMatch – Uses a regular expression and maps it to a script name
- RedirectMatch – Provides an optional status, then redirect URL of certain regular expression to destination URL
- RedirectTemp – Reports a temporary status then redirects to destination URL
- RedirectPermanent – Reports a permanent status, then redirects to destination URL

mod_dir

This module tells the server the name of the file to return as the index of the directory being accessed. It comes into play when someone connects to a Web site with a URL that ends in a slash or a directory name, not a file name, as for example in `http://www.CompanyA.com/`.

Directives:

- DirectoryIndex – Identifies the index file that Apache should look for before creating a dynamic directory index

mod_mime

This module informs the server of the type of files based on the file extensions.

Directives:

- AddType – A mime type followed by one or more file extensions for determination of content type
- AddEncoding – An encoding (such as gzip), followed by one or more file extensions for determination of encoding type
- AddLanguage – A language (such as fr), followed by one or more file extensions for determination of language type
- AddHandler – A handler name followed by one or more file extensions for assignment of handler to react to these files
- ForceType – Forced a media type

- RemoveHandler – A handler name followed by one or more file extensions for removal of handler
- SetHandler – A handler name to parse all matching files
- TypesConfig – Sets location of the MIME types config file

mod_mime_magic
This module determines the MIME type of a file based on magic numbers and some bytes in the contents of the file.

Directives:

- MimeMagicFile – Name of the MIME Magic file

mod_negotiation
Based on the client's capability to accept the requested document, it selects the most appropriate one and returns it to the client. There are two ways of handling such situations, namely by using file extensions that map to the standard language tag (application/x-type-map) or by using a variants file that categorizes all the documents along with their representation types (type-map), according to the common resources they represent.

Directives:

- CacheNegotiatedDocs – No arguments (either present or absent), but provides caching of documents on proxy servers
- LanguagePriority – A list of MIME language abbreviations separated by space for the language selection priority when no preference is stated from client's browser

mod_rewrite
This module is not compiled by default, but it is a powerful tool to rewrite URLs on the fly. It supports unlimited sets of rules that operate many variables like the server variable, environment variable, HTTP headers, timestamps, and so forth, as well as their respective conditions. It operates on full URLs in both the server context and the directory context.

Directives:

- RewriteEngine – On or Off to enable or disable (default) the whole rewriting engine
- RewriteOptions – List of option strings to set
- RewriteBase –The base URL of the per-directory context for rewrites
- RewriteCond – An input string and a to-be-applied regular expression pattern for definition of a rule condition

- RewriteRule – A URL-applied regular expression pattern and a substitution URL for rewriting
- RewriteMap – A mapname and a filename for substitution of strings of a rewriting rule
- RewriteLock – The filename of a lockfile used for inter-process synchronization
- RewriteLog – The filename of the rewriting logfile
- RewriteLogLevel –The level of the rewriting logfile verbosity (0=none, 1=std, .., 9=max)

mod_userdir
This module deals with translation of URLs to a user's home directory.

Directives:

- UserDir – The public subdirectory in users' home directories, or 'disabled', or 'disabled username username...', or 'enabled username username...' to govern the resources

2.3.3 Server-Management Modules

Modules in this category are good tools for providing information needed for managing of the server.

mod_env
This module passes environment variables to scripts such as CGI or SSI. The variables can be set or unset unconditionally or exported from the server's environment to the document's environment for use.

Directives:

- PassEnv – A list of environment variables to pass to CGI
- SetEnv – An environment variable name and a value to pass to CGI
- UnsetEnv – A list of variables to remove from the CGI environment

mod_info
This module provides a great deal of information about the server settings and its environment. This module is not found in the IBM HTTP Server, but the procedure to include it is found in 8.2, "The Apache Information Module (mod_info)" on page 185.

Directives:

- AddModuleInfo – A module name and additional information on that module to be displayed

mod_log_agent
This module logs the UserAgent header of the client. This enables the server to know what software (mainly the browser) the client is using to send the request.

Directives:

- AgentLog – The filename of the agent log containing the UserAgent header

mod_log_config
This module is much more flexible than the module above. Basically it enables webmasters to log anything, anywhere. This implies any number of log files can be used to keep track of any information pertaining to a specific virtual host or the entire server.

Directives:

- CustomLog – A file name and a custom log format string or format name where log records are written
- TransferLog – The filename of the access log based on definitions under the LogFormat directive
- LogFormat – A log format string and an optional format name that customizes the format of the default log file
- CookieLog – The filename of the cookie log for the logging of cookies

mod_log_referer
This module keeps track of the external URLs that are linked to the pages in our server. This means that we are able to know where the client “jumped” from and to which page in our server it is referred to.

Directives:

- RefererLog – The filename of the referer log containing the source referer header
- RefererIgnore – Referer hostnames to be ignored in the referer log file

mod_status
This module provides statistics on the “health” of the server. Unlike mod_info that provides information about the server configuration, this module shows the current activities of the server.

Directives:

- ExtendedStatus – “On” to enable extended status information,”Off” to disable

mod_unique_id
This module allocates a unique magic number per request, which is assigned to the environment variable *UNIQUE_ID.* The use of this magic number is similar to the use of the process ID in the UNIX operating system for server management purposes.

Directives: none

mod_usertrack
This module tracks the client’s movement or traversal in the document tree of the Web server by using cookies.

Directives:

- CookieExpires – An expiry date code for the cookie to expire
- CookieTracking – Determines whether to enable cookies

2.3.4 Real-Time-Handling Modules

Modules in this category focus their attention on responding to the client’s requests.

mod_actions
This module informs the server which CGI script to execute based on the MIME type specified in the request.

Directives:

- Action – A media type followed by a script name to be used for execution
- Script – A method followed by a script name to be used for execution

mod_autoindex
This module provides a directory listings for the users. This directory listing which contains information such as the directory and file sizes, is generated either by an index file created or by the server itself.

Directives:

- AddIcon – An icon URL followed by one or more filenames
- AddIconByType – An icon URL followed by one or more MIME types

- AddIconByEncoding – An icon URL followed by one or more content encodings
- AddAlt – Alternate descriptive text followed by one or more filenames
- AddAltByType – Alternate descriptive text followed by one or more MIME types
- AddAltByEncoding – Alternate descriptive text followed by one or more content encodings
- IndexOptions – One or more index options to set the indexing behavior
- IndexIgnore – One or more file extensions to hide when listing directory
- AddDescription – Descriptive text followed by one or more filenames
- HeaderName – A filename to be listed on top of the index listing
- ReadmeName – A filename to be appended to the end of the index listing
- FancyIndexing – Limited to 'on' or 'off' (superseded by IndexOptions FancyIndexing)
- DefaultIcon – An icon URL to be displayed when no other icon is known

mod_cgi

This module deals with the creation of the environment variables that contain information about the server and the clients to be passed on to the CGI scripts for their necessary execution. Files with MIME type application/x-httpd-cgi or handler cgi-script are executed as CGI scripts by the server, which returns the corresponding results to the clients. There are also other types of scripts such as perl, PHP and fastCGI that are supported by the Apache server. Though the modules supporting them such as mod_perl, mod_php and mod_fastcgi are not standard modules, the implementation procedures are covered in Chapter 8, “Building HTTP Server Modules” on page 177.

Directives:

- ScriptLog – The name of a log for script debugging info
- ScriptLogLength – The maximum length (in bytes) of the script debug log
- ScriptLogBuffer – The maximum size (in bytes) to record of a POST request

mod_imap

This module deals with image mapping of graphical image maps to Web page locations. Besides supporting the traditional use of a CGI-program to do the coordinates to document mapping, the Apache server offers this module to

perform the same task more efficiently without the need for a separate process. In addition, the module also supports the use of client-side processing where the relevant mapping information are returned to the browser that is capable of highlighting the image regions containing links. Thus, with this module, both server-side and client-side imagemaps, when configured, can optimize the whole process of image mapping.

Directives:

- ImapMenu – The type of menu generated: none, formatted, semiformatted, unformatted
- ImapDefault – The action taken if no match: error, nocontent, referer, menu, URL
- ImapBase – The base for all URLs: map, referer, URL (or start of)

mod_include
This module offers documents to be included within documents and handles the parsing of server-parse documents with the server-parse handlers.

Directives:

- XBitHack – Set Off, On, or Full to control the parsing of HTML documents

mod_speling
This module is not compiled by default, but it is a useful tool for the server to perform spell check and corrections on the URL requested. Basically, it compares all the document names in the requested directory against the name of the requested document, fore-going case sensitivity and even allowing up to one misspelling in the word, before returning the “giving-up” error message back to the client.

Directives:

- CheckSpelling – Determines whether to fix miscapitalized/misspelled requests

2.3.5 Environment-Changing Modules

Throughout the client request parsing phases, modules communicate either by changing the values of request itself or changing environment variable values. The modules here are dedicated to the latter.

mod_setenvif
This module is used to change the environment variables such that other parts of the server can decide how to react with what actions.

Directives:

- SetEnvIf – A header-name, regular expression and a list of variables based on attributes of the request
- SetEnvIfNoCase – A header-name, regular expression and a list of variables which are not case-sensitive
- BrowserMatch – A browser regular expression and a list of variables based on UserAgent HTTP request header field
- BrowserMatchNoCase – A browser regular expression and a list of variables that are not case-sensitive

2.3.6 Protocol-Enhanced Modules

Modules in this category help the Apache server to extend its capability in supporting optional features of the HTTP/1.1.

mod_asis
This module allows file types to be sent without using the HTTP headers, but, rather, using their definition. This implies that all kinds of data can be sent from the server without using CGI scripts.

Directives: none

mod_cern_meta
This module is incorporated to support the CERN Web server metafiles which are HTTP headers other than the default HTTP headers output to the files.

Directives:

- MetaFiles – Limited to on or off for Meta file processing
- MetaDir – The name of the directory containing meta files
- MetaSuffix – The filename suffix for meta files

mod_expires
This module is used to set the expiration time for the web document the client requested using the Expires HTTP header, so that the client can assume the validity of the document fetched from their local cache before the time expires. This header can be set in two ways, either by the last-modified-time or by the time of client's access.

Directives:

- ExpiresActive – Limited to on or off for generation of Expires header
- ExpiresBytype – A MIME type followed by an expiry date code

- ExpiresDefault – An expiry date code generation for all defined documents

mod_headers
This module provides modification of the HTTP response headers before returning to the client. With this function, headers can basically be added, removed or replaced.

Directives:

- Header – An action, header and value for modifying, adding or removal

2.3.7 Dynamic-Linking Modules

Modules in this category help to build the server dynamically with the modules.

mod_so
This module is used to load modules into the server at runtime.

Directives:

- LoadModule – A module name and the name of a shared object file to load it from
- LoadFile – Shared object file or library to be loaded into the server at runtime

2.4 WebSphere and Apache

When the use of Web browsers and servers became known to the general public, many were overwhelmed by this new era of gathering information through electronic means. Most Web servers began with housing HTML pages, which was, at that time, sufficient to serve the enthusiastic Web user. As the users began to grow, there was a need to provide information to specific demands requested by the users. Webmasters were able to fulfill the demand with the use and development of CGI programs with some limitations. The on-going development and contribution in areas such as the protocol and languages have made the Web a possible means for commercial operations in today's world. IBM offers the IBM WebSphere Application Server as a tool for the commercial world to perform their operations more efficiently in this the new era of Web technology. Figure 2 on page 31 shows a basic model of the IBM WebSphere Application Server to illustrate how the Apache Server fits into the entire package.

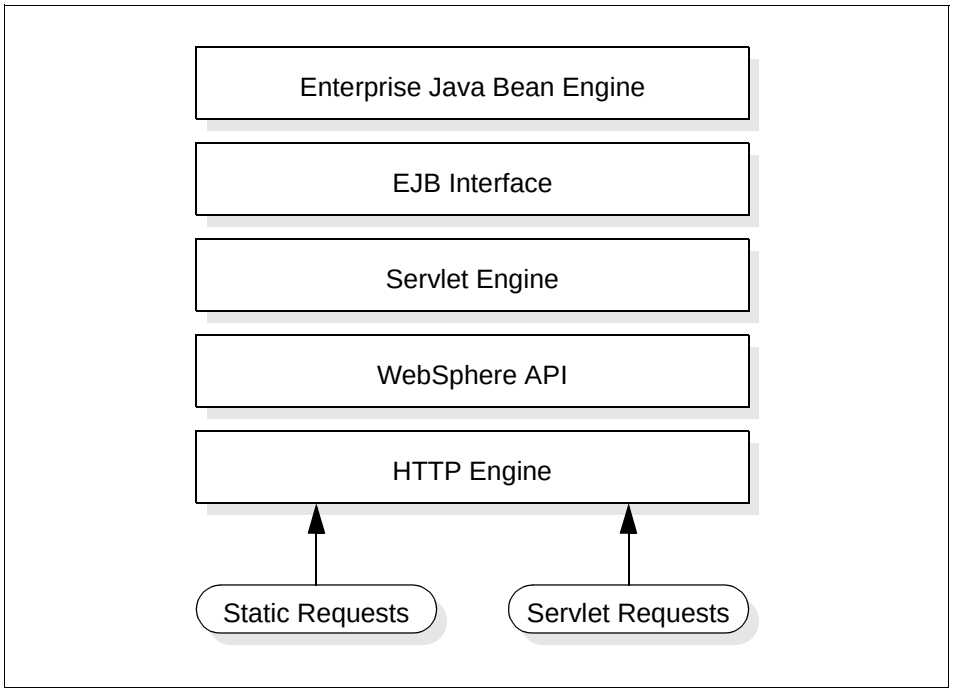


Figure 2. Basic WebSphere Architecture

The open standards adopted by the IBM WebSphere Application Server include CORBA (Common Object Request Broker Architecture), JDBC (Java Database Connectivity) and Java to allow developers to make full use of the Java servlets, which replaced, if not improved, the performance of those written as CGI scripts.

Basically, there are three main engines that deal with the entire operation. They are the HTTP Engine, the Servlet Engine and the Enterprise Java Bean Engine. The HTTP Engine sits on top of the operating system (with Java capability) and is the layer that deals directly with the client's requests from the browsers. Most of the requests, such as HTML documents, CGI scripts, GIF images, and so forth, are under the category Static Requests and are dealt with by the HTTP engine. The WebSphere Application Server provides an Apache module which lets the Apache server exploit the services provided by the WebSphere Server. Since WebSphere can be used in a Netscape Web server via NSAPI, or with Internet Information Server via ISAPI, the migration path from other servers to Apache can be very smooth. Requests pertaining to Java Server Pages (JSP) and servlets are also handled by the Servlet Engine. Last, but not least, stands the Enterprise Java Beans (EJB) engine which is dedicated to

ensuring the integrity of the transaction activities and the business logic of the application. It communicates with the servlet engine via the EJB interface layer.

IBM is shipping the IBM HTTP Server with WebSphere Application Server V2.0. WebSphere will also work with the original Apache Web server (among other Web servers), but requires a different module than the IBM HTTP Server.

Chapter 3. Installation and Initial Setup

This chapter covers the installation of the IBM HTTP Server on an RS/6000 machine and the initial setup necessary to start it for the first time. First, the contents of the IBM HTTP Server product file packages are listed, followed by the hardware and software prerequisites that are required in order to run this server. The installation, as explained later in this chapter, is an easy step and it uses standard methods provided by the AIX operating system. The chapter then describes some minimal setup that might be necessary to run the server, and some hints are provided in the case the server does not start successfully after installation.

3.1 Product Contents

The IBM HTTP Server is bundled with the IBM WebSphere Application Server V2.0 and the install images can be found on the product CD-ROM. The IBM HTTP Server can also be downloaded from the following URL: `http://www.software.ibm.com/webservers/appserv/download.html`.

IBM employees can also download the IBM HTTP Server from the internal URL `http://w3.software.ibm.com/webservers/html/downloads.html`.

The IBM HTTP Server comes in several file packages which contain the IBM HTTP Server and SSL filesets as follows:

- Base package, without SSL security:
  - **http_server.base** – Contains the IBM HTTP Server base and source filesets
- SSL module and SSL library packages (required for SSL):
  - **http_server.modules** – Contains the IBM HTTP Server SSL module fileset
  - **gskrf301** – Contains the gskrf301.base fileset, which are the base SSL libraries for use in France and as a prerequisite for the other SSL filesets
  - **gskre301** – Contains the gskre301.base fileset, which contains additional SSL libraries for export outside U.S. and Canada (excluding France)
  - **gskru301** – Contains the gskru301.base fileset, which contains the additional, export-controlled SSL libraries for use in the Unites States and Canada

Note that the gskrf301.base (for France) is an install prerequisite for either of the other two SSL library filesets. Depending on product packaging available in your country, you might not have all of the packages as listed above available on the install media.

Note on Packaging

At the time of writing, packaging of the IBM HTTP Server was still subject to change. You should check WebSphere installation media and/or the IBM Web server Web site for latest information about packaging and availability at `http://www.software.ibm.com/webservers`.

The IBM HTTP Server includes most of the Apache standard modules (see 2.3, “Standard Modules in the Apache Server” on page 18). Table 6 shows the modules that are included with the IBM HTTP Server.

Table 6. Standard Modules in the IBM HTTP Server Filesets

Fileset	Included Modules (File Names)
http_server.base	mod_access.so mod_actions.so mod_alias.so mod_asis.so mod_auth.so mod_auth_anon.so mod_auth_dbm.so mod_autoindex.so mod_cern_meta.so mod_cgi.so mod_digest.so mod_dir.so mod_env.so mod_expires.so mod_headers.so mod_imap.so mod_include.so mod_log_agent.so mod_log_config.so mod_log_referer.so mod_mime.so mod_mime_magic.so mod_negotiation.so mod_rewrite.so mod_setenvif.so mod_speling.so mod_status.so mod_unique_id.so mod_userdir.so mod_usertrack.so
http_server.module	mode_ibm_ssl.so

The standard Apache modules that are not listed in Table 6 (and thus not included as loadable modules with the IBM HTTP Server) are:

- **mod_auth_db** – This module performs basic authentication using Berkley-type DB authentication files. The IBM HTTP Server supports DBM files through the mod_auth_dbm module.

- **mod_so** – This module is necessary for dynamic loading of other modules. This module is already linked into the core httpd executable and thus not available as a separate module.
- **mod_example** – This module is not available as a compiled module, but it is included in source code as a sample for custom module development.
- **mod_info** – This module is not included because it may create a security exposure in a Web server.
- **mod_proxy** – This module is not included because it does not (yet) support the HTTP/1.1 protocol and, at the time of product release, did not meet the high quality level of Apache in terms of stability.

The file name extension used for compiled modules is .so, which stands for *shared object*. More information on using and customizing some of these modules can be found in Chapter 5, “Advanced Configuration” on page 71.

3.2 Updates to the IBM HTTP Server

As was mentioned earlier, the IBM HTTP Server is built from open source code available and supported by the Apache Group. Although IBM is not primarily responsible for this code, customers using the IBM HTTP Server should report any problems using the established IBM support channels. The IBM support team will then forward any problem reports (along with any suggested fixes) to the Apache Group to have them implemented in the core code. For the latest information on support, product refreshes, or upgrades to the IBM HTTP Server, customers should visit the following Web site:

`http://www.software.ibm.com/webservers`

Alternatively, customers may contact their IBM representatives or look at the newsgroup ibm.software.websphere.http-servers at `news.software.ibm.com`.

3.3 Installation Prerequisites and Considerations

There are a few checks that should be done before the IBM HTTP Server is installed on an RS/6000 machine.

Note: This description only applies to the installation of the IBM HTTP Server. If you install it together with the IBM WebSphere Application Server V2.0, please also check the prerequisites for that product.

Hardware and Software Requirements

The IBM HTTP Server running on RS/6000 requires the following hardware and software for successful installation and operation:

- Any IBM RS/6000 capable of running IBM AIX Version 4.2.1 or higher
- A configured TCP/IP stack is required for running the IBM HTTP Server
- Although not required, a graphic console with mouse is recommended for easy testing and troubleshooting

The requirements listed above are minimal. In fact, the IBM HTTP Server will run in a very limited environment. However, if you plan to run a high-performance, large-scale Web server, you will certainly have to have more CPU, memory and disk resources available. Additional information on performance and scalability can be found in Chapter 7, “Performance and Scalability” on page 153.

DIsk Space Considerations

A full installation of the IBM HTTP Server on RS/6000 requires less than 25 MB in the /usr filesystem. Please bear in mind that this covers the base installation only and does not include any user files. In a practical environment, disk space requirements will almost certainly grow as HTML files, image graphics, and applications are added. Also, log files on a production Web server with heavy access can grow quickly, requiring additional disk space.

3.4 Default File and Directory Structure

The IBM HTTP Server is installed in a default directory structure that slightly deviates from the original Apache distribution code to meet some minimal standards in the AIX directory file tree. It should be noted at this point that the original Apache Web server is strictly platform-independent and therefore requires some adaptation to fit individual operating systems’ standards.

All files of the IBM HTTP Server are installed underneath the /usr/lpp/HTTPServer directory. This includes the executable binaries, log files, online documentation, and others. There is a Readme.httpserver file in /usr/lpp/HTTPServer that you should consult after installation of the IBM HTTP Server or any updates to it.

Following is a list with a short description of the directories beneath /usr/lpp/HTTPServer:

- **/usr/lpp/HTTPServer/apachesrc** – Contains a single file, *apache_tar.gz*, which is a tar-ed and gzipped collection of the original source files from

the Apache Group from which the IBM HTTP Server was built. Note that this is not identical with the actual source code from which the IBM HTTP Server (with the SSL additions) is built.

- **/usr/lpp/HTTPServer/bin** – Holds the three executables: dbmmanage, htdigest, and htpasswd. See 6.2.3, “Authentication Files and Databases” on page 124 for more information.
- **/usr/lpp/HTTPServer/sbin** – Stores the main httpd executable, the control program apachectl, along with some other executables.
- **/usr/lpp/HTTPServer/example_module** – Contains the source code and related files for the sample module mod_example, as shipped with the original Apache distribution. See Chapter 8, “Building HTTP Server Modules” on page 177 for more information on how modules can be compiled and included in the IBM HTTP Server.
- **/usr/lpp/HTTPServer/include** – Contains the header files for the IBM HTTP Server. These files are available for your reference, or when new modules need to be compiled (see Chapter 8, “Building HTTP Server Modules” on page 177).
- **/usr/lpp/HTTPServer/etc** – Contains the main configuration file (*httpd.conf* and *httpd.conf.default*), the mime-type definition file (*mime.types* and *mime.types.default*), and the definition file for mod_mime_magic (*magic.default*). Depending on your installation, a sample configuration file for SSL (*httpd.conf.sample.ssl*) may be included as well.
- **/usr/lpp/HTTPServer/share** – Contains the following subdirectories:
  - **cgi-bin** – Is the default directory for CGI programs, containing two sample CGI scripts.
  - **htdocs** – Contains the online HTML documentation.
  - **icons** – Contains a collection of public domain GIF images that the IBM HTTP Server uses for various purposes. The README file in this directory explains the suggested use of some of these images.
- **/usr/lpp/HTTPServer/man** – Contains the Apache man pages for the executables. To be able to read them with the UNIX man command, enter the command:

  ```
  export MANPATH=$MANPATH:/usr/lpp/HTTPServer/man
  ```

  You might find them handy as references for the commands. However, you might also find and even prefer similar information in the online or public Apache documentation.
- **/usr/lpp/HTTPServer/var –** Contains the following subdirectories:

- **log** – For the log files, which are the *error_log* and *access_log* files
- **proxy** – Not used by the IBM HTTP Server
- **run** – For some run-time control files

- **/usr/lpp/HTTPServer/libexec** – Contains the loadable modules as listed in Table 6 on page 35.
- **/usr/lpp/HTTPServer/ssl** – Contains the IKEYMAN application for certificate creation and handling, if installed (see 6.4.5, “Creating a Self-Signed Certificate” on page 136 for more information).

In 4.1, “Recommended Directory Structure” on page 49, a slightly modified directory structure will be introduced that might be better suited towards a production-oriented installation.

3.5 Installing the IBM HTTP Server

This section describes the installation of the IBM HTTP Server on an IBM RS/6000 machine. In contrast to the standard Apache distribution, the IBM HTTP Server is shipped as an AIX installp image that can easily be installed using SMIT (System Management Interface Tool).

3.5.1 Pre-Installation Setup

Before starting with the IBM HTTP Server installation process, you should ensure that the following is in place:

- You should have the install packages available. See 3.1, “Product Contents” on page 33 for more information on which packages you need. Depending on your source, these can be available as separate files or on a product CD-ROM.
- You must have root authority on the system on which the IBM HTTP Server is to be installed.
- You should have the minimum hardware and software requirements available as listed in 3.3, “Installation Prerequisites and Considerations” on page 36.
- You should determine which filesets you need to install. The following matrix (Table 7) provides you with the names and descriptions of the respective filesets. Select the column on the right that applies to your environment and then select the fileset(s) applicable for that installation listed on the left. The fileset description in the first column of Table 7 are the ones that SMIT lists when installing the product. Note that the filesets

are contained in file packages as listed in 3.1, “Product Contents” on page 33.

Table 7. Fileset Installation Matrix

Fileset Description (Fileset Name)	Base HTTP Server	Base Server & SSL Security			Source Code
		France	Export	U.S. & Canada	
IBM HTTP Server (http_server.base.core)	✔	✔	✔	✔	
IBM HTTP Server source files (http_server.base.source)					✔
IBM HTTP Server SSL Module (http_server.modules.ssl)		✔	✔	✔	
gskrf301 for AIX (gskrf301.base)		✔	✔	✔	
gskre301 for AIX (gskre301.base)			✔		
gskru301 for AIX (gskru301.base)				✔	

The installation is fairly easy and straight forward, but if you are unfamiliar with the standard product installation process or with the SMIT tool on AIX, you might want to refer to the *AIX Installation Guide*, SC23-4112, before you start with the installation.

3.5.2 Installing Using SMIT

The following description guides you through the installation of the IBM HTTP Server using the SMIT tool:

1. Log on as root on the system where the IBM HTTP Server is to be installed.
2. Ensure that you have the IBM HTTP Server file packages accessible from your system. Depending on your environment, they might be available on a CD-ROM, on a remotely mounted directory, or as separate files.
3. Run SMIT with the install_latest fastpath:

   ```
   # smit install_latest
   ```

   (If you do not want the graphical version of SMIT, run `smitty` instead. The following description and figure refer to this non-graphical version.)

4. Enter the directory name or the device name where the package files are available (or select the available devices from the list shown with PF4) and press **Enter**.
5. From the Install and Update LATEST Available Software menu, press **PF4** for SOFTWARE to install in order to show the list of available filesets in that directory (or on the selected media).
6. From the SOFTWARE to install list, select the filesets that you want to install (with PF7) and press **Enter** when you have finished the selection. Refer to Table 7 on page 40 for selecting the correct fileset(s).
7. Review the other installation options on that SMIT panel and adjust as required. Then, press **Enter** to start the installation.
8. After successful installation, an `OK` indication will be displayed in the top left corner as shown in Figure 3 on page 41.

```
                                    COMMAND STATUS

Command: OK            stdout: yes           stderr: no

Before command completion, additional instructions may appear below.

[TOP]
installp -acgNQqwX -d /inst.images/apache -f File 2>&1

File:
    gskrf301.base                 3.0.1.30
    gskre301.base                 3.0.1.30
    http_server.base.core         1.3.3.0
    http_server.base.source       1.3.3.0
    http_server.modules.ssl       1.3.3.0

+-----------------------------------------------------------------------------+
                    Pre-installation Verification...
+-----------------------------------------------------------------------------+
Verifying selections...done
Verifying requisites...done
[MORE...66]

F1=Help             F2=Refresh          F3=Cancel           F6=Command
F8=Image            F9=Shell            F10=Exit            /=Find
n=Find Next
```

Figure 3. SMIT Screen after Successful Installation

This completes the basic installation of the IBM HTTP Server. You can find the default directory structure and files as explained in 3.4, “Default File and Directory Structure” on page 37 in place.

The installed filesets can be displayed and verified with the `lslpp` command from the command line:

```
# lslpp -l http_server.\*
  Fileset                      Level  State      Description
----------------------------------------------------------------------------
Path: /usr/lib/objrepos
  http_server.base.core     1.3.3.0  COMMITTED  IBM HTTP Server
  http_server.base.source   1.3.3.0  COMMITTED  IBM HTTP Server source files
  http_server.modules.ssl   1.3.3.0  COMMITTED  IBM HTTP Server SSL Module

# lslpp -l gskr\*
  Fileset                      Level  State      Description
----------------------------------------------------------------------------
Path: /usr/lib/objrepos
  gskrf301.base           3.0.1.30  COMMITTED  gskrf301 for AIX
  gskre301.base           3.0.1.30  COMMITTED  gskre301 for AIX
```

Note: The version numbers and fileset descriptions shown above were correct at the time this book was written. They may be different at a later date.

3.6 Initial Setup

After the IBM HTTP Server has been installed successfully, there are a few steps to be done (or at least considered) before the server can be run. These are:

- Creating a user and group under which the IBM HTTP Server will run
- Adapting the server configuration file to your environment

First of all, for security reasons, a Web server should never be run with root authority; it should only be run as a user application under a user that has limited privileges in the operating system. To make things a bit more confusing, however, a Web server *must* start with root authority in order to have sufficient privileges to open port 80 (the default HTTP protocol port).

The solution to this is that the IBM HTTP Server runs a main process under root, which then in turn spawns child processes that change their user identity to whatever user and group are configured in the server's configuration file. (There are actually more reasons for spawning multiple processes; this is only one of them.) By default, these child processes run under the user and group *nobody*. Since nobody is an anonymous user, not only used for a Web server, you may consider creating a separate user and group for the IBM HTTP Server to run under. This way, you can keep better

separation of your Web server from other tasks running on the same machine. This user and group can be called whatever you decide; for example, *www* as user and group names.

As a further security precaution, this user does not need to have a login password and login should be disabled, thus preventing the user account from being misused, intentionally or unintentionally.

After you have decided on and created a separate user and group, you must edit the server configuration file to reflect these changes. The default configuration file that you need to edit for the IBM HTTP Server is /usr/lpp/HTTPServer/etc/httpd.conf. Locate the directives User and Group and change their default settings from nobody to the user and group name that you have created.

There could, of course, be a number of changes done to that configuration file at this time, but none of them are actually required before the IBM HTTP Server can be fired up for the first time. The remainder of this book explains this configuration file and possible changes in more details.

3.7 Server Process Structure

Before moving on and starting the IBM HTTP Server for the first time, it might be worthwhile to look at some particulars about this server from a system point of view.

The IBM HTTP Server runs, as mentioned in the previous section, as a main process and one or more child processes. The main process runs with root authority, while the child processes run under a different user authority. These child processes, not the main process, do the actual Web serving work. This architecture allows for better performance and parallelization of client requests. The number of child processes is automatically adapted to the server's load; the minimum and maximum number can be configured in the configuration file (along with some other numbers). By default, there is a minimum of five child processes and a maximum of 150.

Once the server is running, these processes can be seen by running the `ps` command:

```
# ps -ef | grep httpd
   www  27024 27454   0 13:50:54    -  0:00 /usr/lpp/HTTPServer/sbin/httpd
   www  27160 27454   0 13:50:56    -  0:00 /usr/lpp/HTTPServer/sbin/httpd
   root 27454     1   0 13:50:52    -  0:00 /usr/lpp/HTTPServer/sbin/httpd
   www  27658 27454   0 13:50:53    -  0:00 /usr/lpp/HTTPServer/sbin/httpd
   www  28132 27454   0 13:50:53    -  0:00 /usr/lpp/HTTPServer/sbin/httpd
```

```
www  28388 27454   0 13:50:55    -  0:00 /usr/lpp/HTTPServer/sbin/httpd
www  29002 27454   0 13:50:55    -  0:00 /usr/lpp/HTTPServer/sbin/httpd
```

The main httpd process supervises the child processes. If, for any reason, a child process terminates, the main process evaluates to start a new one, depending on certain criteria, like server load and configuration parameters. Each child process has a counter for the client requests it has served. After reaching a certain maximum (the default is 10000000), it terminates to allow a new process to be started. This is a designed, yet elegant way to circumvent any problems that usually exist in long-living processes, such as memory leaks. At the time of writing, there was some work in progress to on a module that dynamically adapts the maximum number of requests per child process as opposed to the currently fixed number.

3.8 Running the IBM HTTP Server for the First Time

After you have completed the minimal server customization as described in 3.6, “Initial Setup” on page 42, the server is ready to be started. Although this is certainly not the final configuration in which the server will eventually run, it is a good idea at this time to check whether the basic server installation and the other involved operating system components work well.

To start the IBM HTTP Server, follow these steps:

1. Login as root.
2. On a command line, change to the /usr/lpp/HTTPServer/sbin directory.
3. Run the following command (and observe the output):

   ```
   # ./apachectl start
   ./apachectl start: httpd started
   ```

   (More about the `apachectl` command can be found in the following section 3.8.1, “The apachectl Utility” on page 45.)

You will most likely receive the positive output as shown in the last step above, indicating that the httpd process (actually the IBM HTTP Server) was successfully started. You may also check this with the `ps` command as shown in the previous Section 3.7, “Server Process Structure” on page 43.

If there was a problem, the chance is that there is some additional error message returned along with the message “httpd could not be started”. Another, even better place to check for errors is the error log file at /usr/lpp/HTTPServer/var/log/error_log. The most common errors that cause starting the IBM HTTP Server to fail are either a lack of permissions (not being root) or some sort of resource problem, such as a full filesystem.

After the IBM HTTP Server has successfully been started, you should be able to connect to it from a Web browser. Just enter the URL of the newly installed Web server and you should get the IBM HTTP Server's welcome page as shown in Figure 4 on page 45.

Figure 4. IBM HTTP Server Welcome Screen

Note that the default configuration of the IBM HTTP Server is such that this welcome screen is shown when no other specific page is requested in the URL (in the example shown in Figure 4: `http://my_test.itso.austin.ibm.com/`).

3.8.1 The apachectl Utility

As was already mentioned in the previous section, apachectl is a utility to start and stop the IBM HTTP Server. The apachectl utility offers more functions that are briefly described here.

The syntax for `apachectl` is:

```
apachectl [start|stop|restart|graceful|configtest|status|fullstatus|help]
```

The command line options are:

`start`	Starts the IBM HTTP Server. If it was already running, an error is returned.
`stop`	Stops the IBM HTTP Server. An error message will be shown if the server was not running upon invocation.
`restart`	Immediately restarts (or starts if not already running) the IBM HTTP Server. The server's child processes are stopped immediately and new processes are created, causing open connections to break.
`graceful`	Gracefully restarts (or starts if not already running) the IBM HTTP Server. This is usually run after a configuration change has been made in order for the server to reread the configuration file. Open requests will be processed first before the processes get restarted.
`configtest`	Reads the server configuration file and indicates whether or not there is a syntax error. This is helpful as a first step after changing the configuration file.
`status`	Displays a current short status of the server. In order to work, mod_status must be configured and lynx must be installed (see note below).
`fullstatus`	Displays a comprehensive current status of the server. In order to work, mod_status must be configured and lynx must be installed (see note below).
`help`	Displays a brief help message about the `apachectl` command.

mod_status and lynx?

mod_status is a module that comes with the IBM HTTP Server. It can be enabled through the configuration file (by default it is disabled) and allows you to query the server about its configuration and status using a Web browser through a normal URL.

lynx is a text-based Web browser that is not available with AIX or the IBM HTTP Server. It can be downloaded, however, from various Web sites, such as `http://www-frec.bull.com/`.

Because mod_status exposes internal information about a Web server, most webmasters prefer not to configure and use it.

You should note that the apachectl is a shell script that assumes the default directory and file structure. If you change these defaults, for example, to adapt to the directory structure explained in 4.1, “Recommended Directory Structure” on page 49, you might have to adapt this shell script in order for it to work properly.

New Times, New Names

At the time this book was written, there were some discussions going on in the Apache Group about renaming the httpd daemon and the apachectl utility. Because of this, you should not be surprised to see new names in a future release.

3.9 Uninstalling the IBM HTTP Server

Should you later on decide to uninstall the IBM HTTP Server, the AIX software maintenance tools come in handy. Type:

```
# smit remove
```

(or `smitty remove` for the text-based version of SMIT) on the command line to get to the SMIT Remove Installed Software panel. Use the PF4 key to display a list of installed software, from which you should select (PF7) the filesets (http_server.* and, if applicable, gsk*) to remove.

Bear in mind that this only removes the standard files that were installed with the IBM HTTP Server. If you changed the directory structure or added links to the executables (for example as explained in 4.1, “Recommended Directory

Structure” on page 49), such changes cannot be removed by this process and must be removed manually.

A Word of Caution

As always, when removing installed software from a system, exercise extreme care to select the correct filesets that you want to remove. Selecting the wrong filesets permanently removes those components from your system.

Chapter 4. Basic Configuration

After you have successfully installed the IBM HTTP Server on your system in a way that it starts with a minimum setup, there are some additional basic configuration steps that can or need to be done to further meet your needs. Configuration of the IBM HTTP Server is done by means of a configuration file, introduced in the previous chapter, that the server reads when it is being started or restarted. This chapter lists and explains the most common directives used in the configuration file that will be used in most IBM HTTP Server installations. But, before the configuration file is examined, a recommended directory structure is introduced in the first section that you might find more suitable than the default as explained in 3.4, “Default File and Directory Structure” on page 37. Furthermore, a more comfortable way of automatically starting and stopping is described in this chapter.

4.1 Recommended Directory Structure

While the IBM HTTP Server will certainly run perfectly in the default directory structure as explained in the last section, there are two major considerations that you should take into account when setting up a production Web server. These are:

- There should not be any variable-size files, such as log files, in the /usr filesystem. The /usr filesystem should normally only contain static program and configuration files.

  Thus, the main recommendation is therefore be to move the log (and similar) files from /usr/lpp/HTTPServer/var to a filesystem that is commonly used for this purpose, such as /var.

- The Web documents that a Web server serves, such as HTML documents, graphic images and CGI scripts, are normally stored in a separate file system other than the /usr filesystem.

The configuration file shipped with the IBM HTTP Server contains directives that specify the location of a number of files and directories. (Note that in an actual installation there can be more than just one configuration file.) Because of this, whenever such a file or directory is moved to another location, the respective directive in the configuration file(s) must be changed as well and the server must be restarted in order to recognize those changes.

Table 8 on page 50 lists and explains the directives in the server configuration file(s) that relate to other files and directories. Also listed are their default and recommended settings. As you can see, the main purpose of this

recommendation is to move the files contained in /usr/lpp/HTTPServer/var to the /var filesystem.

Table 8. Default and Recommended Directory Structure

Directive	Description, Settings
ServerRoot	Used as a directory prefix for all other directory or file directives that are not specified with their full path names. This directive is normally not important as most Web administrators choose to specify full directory names for all other files to avoid any confusion. Default Setting: /usr/lpp/HTTPServer Recommended: /usr/lpp/HTTPServer (no change)
ErrorLog	The server's error log file. Default Setting: /usr/lpp/HTTPServer/var/log/error_log Recommended: /var/httpd/log/error_log
CustomLog	The server's access log file. Default Setting: /usr/lpp/HTTPServer/var/log/access_log Recommended: /var/httpd/log/access_log
PidFile	A file that contains the process ID (PID) of the main server process. Please read the note below concerning the PidFile directive. Default Setting: /usr/lpp/HTTPServer/var/run/httpd.pid Recommended: /var/httpd/httpd.pid
ScoreBoardFile	File for temporary storage of server-internal data. This file is not used by the IBM HTTP Server on AIX.
LockFile	Lock file for server-internal use. It is important that this file is not on a shared filesystem, such as in an NFS-mounted directory. Default Setting: /usr/lpp/HTTPServer/var/run/httpd.lock Recommended: /var/httpd/httpd.lock
DocumentRoot	Specifies the root directory for files served by this server. Default Setting: /usr/lpp/HTTPServer/share/htdocs Recommended: (see notes below)

Notes:

- **General** – The /var filesystem was chosen for the above recommendations because it is the commonly used place for log files. Be careful, however, since log files can grow and may even use all available space in /var, preventing other services from functioning properly (or vice-versa). You might instead choose to create a separate filesystem for the log files to completely uncouple them from other services running on the same machine.
- **Shared File Systems** – It is recommended to store the log, lock, and error files on a local filesystem rather than on a remote filesystem.
- **PidFile** – The apachectl utility (see 3.8.1, "The apachectl Utility" on page 45) is a UNIX shell script that uses the PidFile. You must also adapt (edit)

a single line in that script when you change the location of the pid file, otherwise the apachectl utility does not work correctly any more.

- **Internal Defaults** – The IBM HTTP Server uses default settings for all files mentioned in Table 8 on page 50, even if they are not specified or commented out in the server configuration file (such as the LockFile is). You can see the internal settings by running the command `httpd -v`. Notice that these internal settings are relative to ServerRoot.
- **Multiple HTTP Servers** – Bear in mind that there may be multiple error log, access log, PID, and lock files, depending on the server's configuration. You might then want to further separate them in separate subdirectories, for example /var/httpd/<server name>/log.
- **DocumentRoot** – This is potentially a large storage area for all Web pages, images and programs that your Web server serves. It is therefore recommended that you create at least a separate filesystem, for example /www/htdocs, for this purpose. If you have multiple HTTP servers, you may choose to further separate them, for example, by using directories like /www/<server name>/htdocs.
- **Binaries** – Unlike other applications, the IBM HTTP Server does not copy or link its executables to a directory that is included in the default $PATH environment variable, such as /bin or /sbin. You may consider adding links like the following:

```
# ln -s /usr/lpp/HTTPServer/bin/dbmmanage /usr/bin/dbmmanage
# ln -s /usr/lpp/HTTPServer/bin/htdigest /usr/bin/htdigest
# ln -s /usr/lpp/HTTPServer/bin/htpasswd /usr/bin/htpasswd
# ln -s /usr/lpp/HTTPServer/sbin/ab /usr/sbin/ab
# ln -s /usr/lpp/HTTPServer/sbin/apachectl /usr/sbin/apachectl
# ln -s /usr/lpp/HTTPServer/sbin/apxs /usr/sbin/apxs
# ln -s /usr/lpp/HTTPServer/sbin/httpd /usr/sbin/httpd
# ln -s /usr/lpp/HTTPServer/sbin/logresolve /usr/sbin/logresolve
# ln -s /usr/lpp/HTTPServer/sbin/rotatelogs /usr/sbin/rotatelogs
# ln -s /usr/lpp/HTTPServer/sbin/sidd /usr/sbin/sidd
```

4.2 Starting and Stopping the HTTP Server

In Section 3.8, "Running the IBM HTTP Server for the First Time" on page 44, the manual way of starting and stopping the HTTP Server using the `apachectl` command was described. In practice, most administrators like to have a process in place that automatically starts the HTTP Server whenever the system boots up. Also, provisions should be implemented for a clean shutdown of the HTTP Server when a system is being shut down.

The following two sections describe methods for implementing such automatic startup and shutdown processes.

4.2.1 Automatic Startup

In AIX, automatic startup of services is done by the *init* process. At boot time, the init process reads the */etc/inittab* file and executes the commands it contains. The /etc/inittab file is not a shell script; it contains a series of commands (with optional comments) in a textual form that follow a specific syntax required for the init process. Care must be taken when editing this file with a text editor. If the init process comes across an error in /etc/inittab, the boot process might not complete successfully. If the init process encounters errors in some critical commands in /etc/inittab, you might have to boot the system in maintenance mode in order to correct the file. For this reason, our example uses the `mkitab` command to add a command to the /etc/inittab file, which lessens the chance for errors.

Follow these steps to make the HTTP Server start up automatically when the system boots up:

1. Login as root user to the system where the HTTP Server will be started automatically.
2. Create a simple startup shell script /etc/rc.httpd for starting the IBM HTTP Server, such as:

```
#!/usr/bin/ksh
# Configures the Automatic Startup of the IBM HTTP Server
# This file should be owned by root:system and have permissions 0774

BINPATH=/usr/lpp/HTTPServer/sbin

if [[ ‘/bin/id -u‘ != 0 ]]; then
   echo "Error: you must be root to execute this command" >&2
   exit 1
fi

echo "Starting IBM HTTP Server..."
$BINPATH/apachectl start
```

   Note: The name for this file (/etc/rc.httpd) is not important and can be any valid file name, but this example follows common rules.

3. Change the ownership and permissions for the file:

```
# chown root.system /etc/rc.httpd
# chmod 0774 /etc/rc.httpd
```

This changes the ownership to user *root* and group *system*, and makes the file executable for either the owner or any member of the system group.

4. Create an entry in /etc/inittab for automatic execution of the startup file:

   ```
   # mkitab "httpd:2:once:/etc/rc.httpd >/dev/console 2>&1 # IBM HTTP
   Server startup"
   ```

   (Note that this is a single line.)

The last step adds the `/etc/rc.httpd` command as the very last entry to the /etc/inittab file. In other words, the HTTP Server will only be started when all other services have already been started. Note that the `mkitab` command has options that allow you to add the `/etc/rc.httpd` command at any specific place in /etc/inittab, should you choose to have the HTTP Server started before certain other services are started.

If the Web server does not start successfully, the logfiles are a good place to start looking for the cause. One common problem with SSL that hinders the automatic startup of the Web server is when the password for the key database is not stored (stashed) in a file. In this case, the server startup hangs when it tries to ask for the key database password (see 6.4, “Secure Sockets Layer, SSL” on page 129 for more information).

4.2.2 Automatic Shutdown

Technically speaking (neglecting any operational and application-related issues), stopping a Web server is not a crucial issue. A Web server is basically just a program that delivers files from the disk to the network. Unless complex applications are involved, it does not have online databases that might end up in an inconsistent state when the system is shut down without properly stopping the IBM HTTP Server. However, it is a good idea to explicitly stop the IBM HTTP Server when shutting down the system.

Like other UNIX operating systems, AIX provides an option to run a shell script when the system is being shut down. When AIX is being shut down, it checks for the existence of the file /etc/rc.shutdown (note that the correct file name is relevant, and it must be executable). If such a file is found, it will be executed early in the shutdown process. If your system does not already have an /etc/rc.shutdown file, you should create it. Otherwise, append the relevant statements to it. Below is an example of an /etc/rc.shutdown file that shuts the IBM HTTP Server down.

```
#!/usr/bin/ksh
BINPATH=/usr/lpp/HTTPServer/sbin
```

```
echo "Shutting down the IBM HTTP Server..."
$BINPATH/apachectl stop
exit 0
```

Note

If /etc/rc.shutdown does not terminate successfully, that is if it returns a non-zero return code, the system shutdown process stops. For this reason, the /etc/rc.shutdown script should always exit with return code zero (exit 0), unless there is a strong reason for not doing so.

Note that the `apachectl` command, called in the example above, is written in a way that it will return even if the IBM HTTP Server is not running or, even worse, not responding to any commands. In the shutdown script you should not run commands that might hang, or, for example, wait for some input from a user. More information on the `apachectl` command can be found in 3.8.1, “The apachectl Utility” on page 45.

4.2.3 Restarting the HTTP Server

The IBM HTTP Server allows an administrator to change the configuration file(s) and apply those changes to the running server without the requirement to shut the server down and then restart it again. This does not cause any noticeable interruptions for a user. To restart the HTTP Server, or, more correctly, to reread the configuration file, use the `apachectl` command:

```
# /usr/lpp/HTTPServer/sbin/apachectl graceful
```

More information about the `apachectl` command can be found in 3.8.1, “The apachectl Utility” on page 45.

4.3 Customizing the Configuration File

This section gives you more guidance on configuring a standalone IBM HTTP Server after it has been installed according to the previous chapter.

Standalone or inetd?

The IBM HTTP Server can be run in two modes: standalone and inetd. In the inetd mode, the server only gets started when an HTTP request is received. The inetd mode is not recommended and, therefore, not further explained in this book.

Customizing the IBM HTTP Server is done by editing its configuration file(s) with a text editor. There can be more than one such configuration file, but by default there is only one, the httpd.conf, located in /usr/lpp/HTTPServer/etc. The IBM HTTP Server reads its configuration parameters from this configuration file when it starts up or when it is being restarted.

The configuration file (http.conf) contains the related directives such as how the server runs, the user and group ID definition that the server used to run as, where the log files are to be written to, the port it listens to, the location of other files, and so on.

The IBM HTTP Server supports a large number of directives in its configuration file and each module that is added to the server may add a few or many additional directives to it. It is beyond the scope of this book to list and explain them all (since the online documentation contains a description of them all). What follows is an explanation of the most often used directives that are likely to be required for a basic server setup. More directives will be discussed in subsequent chapters.

Port specifies the port number that server listens on. The default port number is *80* for Web servers. Note: If the port number is 1023 or below, the IBM HTTP Server must be started as root. If any other port than 80 is used, it must be specified in the URL for that server. For example, if port 8080 is to be used instead, the corresponding lines in httpd.conf are:

```
Port 8080
Listen 8080
```

An example of an URL for that non-standard port at www.CompanyA.com could be `http://www.CompanyA.com:8080/home.html`.

ServerType specifies whether the server runs as a standalone daemon or whether it will be started by inetd on demand. Running the IBM HTTP Server as an inetd process is not recommended, thus, the directive in httpd.conf should look like:

```
ServerType standalone
```

User defines the user ID (or UID) under which the server will run. Although the server will be started as root in most cases, the actual HTTP request servers run under a different user ID for security and other reasons. The user *nobody* is the default value, but it is recommended to create a special user for the server (see 3.6, “Initial Setup” on page 42). The corresponding directive in httpd.conf for a user ID *www* would look like:

```
User www
```

If the server is started by someone other than root, this parameter is ignored.

Group is similar to the User directive explained above. It specifies the group the HTTP server processes should run under. An example is:

```
Group www
```

ServerAdmin specifies an e-mail address that may be used in error messages (see 5.5.1, “Customizing Error Messages” on page 92). For example:

```
ServerAdmin webadmin@CompanA.com
```

ServerRoot specifies the absolute directory that serves as a root directory for other files specified by their respective directives that do not contain an absolute filename, such as the error and access log files. However, it is recommended to use fully qualified filenames in all other directives that involve files because relative filenames may easily be confusing. The default is:

```
ServerRoot /usr/lpp/HTTPServer
```

ErrorLog specifies the filename for the error log file. The default is:

```
ErrorLog /usr/lpp/HTTPServer/var/log/error_log
```

You might want to change this according to 4.1, “Recommended Directory Structure” on page 49.

ServerName is how the IBM HTTP Server identifies itself in error messages that are sent to client after an error occurred. It should specify the server’s hostname that can be resolved to an IP address. Example:

```
ServerName www.CompanyA.com
```

DocumentRoot is used to translate the document portion of a URL into the actual directory and file tree. All references given in a URL are relative to this directory on the server. For example, if all your Web pages are beneath /www/html, the directive would be:

```
DocumentRoot /www/html
```

The URL `http://www.CompanyA.com/index.html` would then request the file /www/html/index.html.

4.4 Enabling DSO Modules

At the beginning of the default configuration file, there is a list of LoadModule directives, a ClearModuleList and another list of AddModule directives. The

LoadModule directives add dynamic shared object (DSO) modules to the server that enhance the functionality of the core server (see also 2.2.1, “The DSO Concept” on page 14). Each DSO module that is to be used must be enabled with a LoadModule directive. An example would be:

```
LoadModule    auth_module          libexec/mod_auth.so
```

The first parameter of the LoadModule directive specifies the module’s internal name as it was specified by the programmer of the module. The second parameter specifies the filename of the module, either absolute, or relative to the directory specified with the ServerRoot directive.

The ClearModuleList directive that follows clears the list of modules that is already compiled into the server by default (as with most configuration options, the server comes with a compiled-in default module list that applies if no other definitions are done in the configuration file).

The list of AddModule directives that follows enables the modules. Bear in mind that the order might be important for a certain function to work properly. You should consult the documentation of the respective module for more information on the order.

4.5 Online Documentation

Documentation about the IBM HTTP Server is available online. After you have installed and started the Web server, the default home page (pointing to /usr/lpp/HTTPServer/share/htdocs) contains links to the respective documentation pages.

The original Apache Web server documentation is also available in /usr/lpp/HTTPServer/share/htdocs/manual or at `http://www.apache.org/docs`.

4.6 The Configuration File

The IBM HTTP Server configuration file /usr/lpp/HTTPServer/etc/httpd.conf is a plain text file. There is no standard graphical configuration tool available with the current version of the server, but you can find interesting work to develop such tools at `http://gui.apache.org`. A graphical management tool can be expected for the IBM HTTP Server in a future version.

For historical reasons (as descendant of the NCSA httpd server), Apache (and thus the IBM HTTP Server) supports three configuration files: httpd.conf, access.conf and srm.conf. The NCSA httpd server used each of these files for different configuration directives. In the current release of Apache (IBM

HTTP Server) all configuration is merged into one file, the httpd.conf configuration file. Other files, though still supported, should only be used when specific reasons require this.

Each line in the configuration file is either a comment or a configuration directive. Empty lines are ignored. Comment lines start with the pound symbol (#). They are disregarded, just like empty lines.

Every directive line starts with the directive's name, followed by optional or mandatory parameters for that directive. Each directive must end on the same line; it is not allowed to continue directive across multiple lines. A parameter must be separated from the directive and other parameters by at least a space or tabulation symbol.

For readability reasons it is recomended to indent directives inside a section (see also 4.8, "Sections" on page 61). The following example shows the use of such indentation:

```
<Location /confidential>
   AuthName "Authorized Staff Only"
   AuthType Basic
   AuthUserFile /www/security/users.pwd
   <Limit GET>
      Order deny,allow
      Deny from all
      Allow from 1.2.3
      require valid-user
      Satisfy all
   </Limit>
</Location>
```

Directives are not case sensitive. You can mix uppercase and lowercase letters in them. Some parameters, however, are case sensitive, particularly file and directory names. Case sensitivity of URLs can be removed by using the spell checking feature of the IBM HTTP Server as explained in 5.9.1, "Fixing Typos in URLs" on page 106.

4.7 Distributed Configuration

A single configuration file (httpd.conf) is the preferred method for better centralized system management and security concerns. In some situations, however, distributed configuration may be required. The IBM HTTP Server allows the user to solve this situation by using .htaccess files for local configuration settings in some directories.

File Name of .htaccess

You may be thinking that the purpose of .htaccess files is access control as file name implies. That is only partially true. They were invented for that purpose and still can be used for that. They can actually contain most of configuration directives that are used in the httpd.conf file, so they are more of a distributed configuration tool.

The file name .htaccess is only a default configuration setting. It can be changed with the AccessFileName directive. For example, it can be set to wwwacl with the following directive:

```
AccessFileName wwwacl
```

A .htaccess file placed in a particular directory applies to that directory and all its subdirectories. It is equivalent to a <Directory> section (see 4.8, “Sections” on page 61) in the httpd.conf file.

For example, assuming there is a .htaccess file in the /www/html directory with the following contents:

```
Order Deny,Allow
Deny from All
Allow from 1.2.3
```

That would be equivalent to the following section in the httpd.conf file:

```
<Directory /www/html>
   Order Deny,Allow
   Deny from All
   Allow from 1.2.3
</Directory>
```

4.7.1 .htaccess and Performance

The use of .htaccess files has its own advantages and disadvantages. They are read and processed on each client request to a directory, so any configuration change applies immediately. The server does not need to be restarted as is the case after a change of the httpd.conf file. The drawback of using the .htaccess file is its negative impact on server performance because of the increased disk I/O operations. When the use of the .htaccess file is enabled (see directive AllowOverride in 4.7.2, “Restricting the Directives within .htaccess Files” on page 60), the Web server also checks all parent directories for any .htaccess files and merges their configuration.

For example, assuming the httpd.conf file contains the following section:

```
<Directory /www/html/public/support>
   AllowOverride All
   ...
</Directory
```

Whenever a document from within that directory is requested, the Web server looks for the following files:

```
/.htaccess
/www/.htaccess
/www/html/.htaccess
/www/html/public/.htaccess
/www/html/public/support/.htaccess
```

If any of these files exist, the Web server reads and applies their contents to the currently effective configuration, which affects server performance. Thus, for performance reasons, it is recomended to use .htaccess files only when really required. More about configuration processing can be found in 4.8.4, “Sections Processing Rules” on page 64.

The server’s default for AllowOverride is *All* for all directories if it is not specified at all. The IBM HTTP Server therefore contains the following directive in its default configuration file to prevent the server from searching for such files:

```
<Directory />
   AllowOverride None
...
</Directory>
```

4.7.2 Restricting the Directives within .htaccess Files

The use of .htaccess files may raise some security concerns. In environments where not only (trusted) webmasters administer the contents of the document directories, it is desirable to disable the use of some directives in the .htaccess file. If disabled, such directives cannot override corresponding directives in higher directories. This can be done with the AllowOverride directive with one or more appropriate parameters in the main configuration file (httpd.conf).

The AllowOverride directive supports the following parameters:

All	Every configuration directive that can be used in .htaccess files is allowed to be used.
AuthConfig	Only authentication control directives (such as AuthType and AuthUserFile) are processed in .htaccess files.

FileInfo	Only document controlling directives (such as ErrorDocument and AddHandler) are processed.
Indexes	Only directory index controlling directives (such as FancyIndexing and AddDescription) are processed.
Limit	Only access control directives (allow, deny and order) are processed.
Options	Option directives are processed (see also 4.10, “Options” on page 67). This is the most dangerous parameter because it allows the user to enable Server Side Includes, CGI programs, automatic directory indexes and the MultiView feature. Every of these features may expose some security risks.
None	The use of .htaccess files is disabled entirely. The Web server does not look for any .htaccess files at all.

For best performance (and security) it is recomended to use:

```
AllowOverride None
```

If you need to use the .htaccess file, it is recommended to specify the following in the main configuration file:

```
AllowOverride AuthConfig Limit
```

Note: The AllowOverride directive cannot be used within .htaccess files.

The .htaccess files can contain confidential information (for example the path and filenames of authentication files). It is possible to restrict access to these files from Web clients using the following section in the main configuration file or .htaccess files themselves:

```
<Files .htaccess>
   Order allow,deny
   Deny from all
</Files>
```

4.8 Sections

The IBM HTTP Server has very flexible features to define configuration parameters for individual URLs, directories and even single files. This can be done by placing configuration directives into special *sections* (sometimes also called *containers* or *scopes*) of the configuration file.

Sections can contain most of the supported configuration directives, including some other sections. If in doubt, consult the online documentation for a particular directive to find out whether or not it can be included in a section.

This section explains the three basic section types: <Directory>, <Files>, and <Location>. A configuration file can also contain other sections, such as <DirectoryMatch>, <LocationMatch> and <FilesMatch>, which are used for regular expression matching. The sections <IfModule> and <IfDefine> are used for conditional processing of directives. More about the <VirtualHost> section can be found in 5.1, “Virtual Hosts” on page 71 and about the <Limit> section in 5.6, “File Uploading” on page 95.

Configuration options are accumulated through sections. The ones in the most precise scope overwrite the previously defined ones.

4.8.1 <Directory>

The <Directory> section is the most commonly used sort of section. It contains configuration directives that apply to a specific directory and its subdirectories. The directory can be specified by an absolute path, by any string with wild-card characters (“?” for single character and “*” for any sequence of characters) or by a regular expression. For security reasons and for simplicity, it is recomended to always use absolute path names.

For example, the following section found in the default IBM HTTP Server configuration file defines the configuration for the whole AIX filesystem (although not all of it will be accessible by clients):

```
<Directory />
   Options FollowSymLinks
   AllowOverride None
</Directory>
```

It enables symbolic links (see 4.9, “Request Mapping” on page 66), but disables automatic directory indexes (see 5.2, “Automatic Directory Indexing” on page 80), server side includes (see 10.4, “Server-Side Includes” on page 219) and MultiViews (see 5.4, “Multiple Language Support” on page 87). The above example also disables the use of .htaccess files (see 4.7.1, “.htaccess and Performance” on page 59).

The following example can also be found in the default configuration file shipped with the IBM HTTP Server:

```
<Directory /usr/lpp/HTTPServer/share/htdocs>
   Options Indexes FollowSymLinks
   AllowOverride None
```

```
   Order allow,deny
   Allow from All
</Directory>
```

It overwrites the previous configuration for the root directory (see previous example) for the /usr/lpp/HTTPServer/share/htdocs directory and its subdirectories.

The <Directory> section cannot be used within .htaccess files.

4.8.2 <Files>

The <Files> section in the server configuration file is very similar to the <Directory> section. The difference is that the <Files> section settings apply to files according to file name match. You can also use wild-cards (like “?” for single character and “*” for any sequence of characters) or regular expressions in the file names.

<Files> sections can be included in <Directory> sections. In this case, it applies only to files in that particular directory and its subdirectories.

For example, the following configuration file fragment forbids access to all .htaccess files beneath /www/html:

```
<Directory /www/html>
   AllowOverride All
   <Files .htaccess>
      Order Allow,Deny
      Deny from All
   </Files>
</Directory>
```

Other than the <Directory> and <Location> directives, the <Files> section can also be used in .htaccess files.

4.8.3 <Location>

The <Location> section does not apply to directories and/or files, but to requested URLs. For example, the following section allows access to URLs /internal (and below) only from the IP network 1.2.3.*:

```
<Location /internal>
   Order Deny,Allow
   Deny from All
   Allow from 1.2.3
</Location>
```

Although similar, the <Location> section has nothing in common with the directory structure. The <Location> directive has no effect if the same directory structure can be accessed through another URL. The following example shows such case:

```
DocumentRoot /www/html

<Location /internal>
   Order Deny,Allow
   Deny from All
   Allow from 1.2.3
</Location>

Alias /int /www/html/internal
```

Files in the /www/html/internal directory can be accessed from any IP address through URL /int. More about the Alias directive can be found in 4.9, “Request Mapping” on page 66.

The <Location> section cannot be used within .htaccess files.

4.8.4 Sections Processing Rules

Configuration directives can appear in the main part of the configuration file, within sections and within .htaccess files. The question arises regarding how that all works together. Here, we define some rules that helps you understand main principles.

Priority – There is a defined priority of each place where configuration directives can appear. Directives with higher priority overwrite ones with lower priority.

Directives found in the main section of the configuration file httpd.conf have the lowest priority. They are overwritten by directives found in <Directory> sections, followed by those in .htaccess files. Directives within <Files> sections and finally within <Location> sections have the highest priority.

In the following example, the directives within the <Location> section overwrite the directives within the <Directory> section:

```
DocumentRoot /www/html

<Directory /www/html/internal>
   Order Deny,Allow
   Deny from All
   Allow from 1.2.3
</Directory>
```

```
<Location /internal>
   Order Deny,Allow
   Allow from All
</Location
```

Inheritance – All configuration parameters are inherited by descendants. That applies to both subdirectories and sub-URLs. For example:

- The section <Directory /www/html> applies also to all subdirectories of /www/html.
- The directives in /www/html/.htaccess apply also to all subdirectories of directory /www/html.
- Directives in the <Location /internal> section apply to all URLs that begin with string /internal after the host name.

Level – Directives in lower subdirectories overwrite the same directives in higher level subdirectories. For example:

- The directives in section <Directory /www/html/internal> overwrite the same directives in section <Directory /www/html>.
- The directives in /www/html/internal/.htaccess overwrite the same directives in file /www/html/.htaccess.

Order – Sections are processed according to their order in the configuration file. If multiple directives apply to the same object on the same level, the directives in the last section overwrite the same directives in previous sections.

For example, in the following configuration, the second section overwrites the first one:

```
<Directory /www/html/internal>
   Order Deny,Allow
   Deny from All
   Allow from 1.2.3
</Directory>

<Directory /www/html/int*>
   Order Deny,Allow
   Allow from All
</Directory>
```

Inclusion – Any directives within a <Files> section that is inside a <Directory> section will be processed only if the <Directory> section matches the request. The same is applicable to sections inside of <VirtualHost> section.

4.8.5 Recommendations on Sections Usage

Using nested sections and additional configuration directives can create some confusion on how they are actually processed by the server. For this reason, we suggest a few simple rules to keep your configuration more clear (at least for yourself).

- Do not use <Location> sections at all (unless truly necessary). Almost everything can be done with <Directory> sections.
- Use <Files> sections only when you really need them. In most cases, a solution can be found by putting all these files into a separate directory and using a <Directory> section.
- Use .htaccess files only when you need to have distributed administration and configuration (see 4.7, “Distributed Configuration” on page 58). Avoid using several .htaccess files in one directory path (for example, /www/.htaccess and /www/html/.htaccess).
- Keep the number of sections in the configuration file at minimum. Sometimes rearrangement of the directory structure helps.

Additional information on this topic can be found at `http://www.apache.org/docs/sections.html`.

4.9 Request Mapping

The base task of a Web server is to respond with HTML documents (or some other type of file) to clients’ requests. This section explains the basic directives that apply to the mapping between URLs and the physical filesystem of the Web server. More about dynamic response generation can be found in Chapter 10, “Web Applications” on page 213.

The basic directive that maps a URL to the filesystem of the Web sever is the DocumentRoot directive. For example, the directive

```
DocumentRoot /www/html
```

This instructs the Web server to insert the directory /www/html in front of all requested resources in URLs. To continue with this example, when the Web server www.CompanyA.com gets a request with the URL `http://www.CompanyA.com/images/products.gif`, it looks for the file /www/html/images/products.gif. This requires that all files are stored underneath one directory (except, of course, when using symbolic links within the filesystem).

Sometimes it is necessary to access files from other directory trees. There are two common ways to do that. First, it is possible to create a symbolic link to another directory. Since this operation might raise some security concerns, the IBM HTTP Server supports the Options FollowSymLinks directive that controls the use of symbolic links. More about the Options directive can be found in 4.10, “Options” on page 67.

Another way to use the other directory tree is the Alias directive. This directive is implemented by the standard module mod_alias. For example, the following directive maps all requests that begin with /download to the directory tree /ftp/pub/download:

```
Alias /download /ftp/pub/download
```

The IBM HTTP Server also has a possibility to tell a client to look for the requested resource in another location. This feature is useful when a document has been moved to a different location or server. The Redirect directive that is used for such cases is also implemented by the mod_alias module.

Here is an example. A configuration file contains the following directive:

```
Redirect /support http://support.CompanyA.com
```

Assuming a client requests `http://www.CompanyA.com/support/index.html`, it will be told to request the document with another URL, which in this case would be `http://support.CompanyA.com/index.html`.

More information about the Alias and Redirect directives can be found in the online documentation.

4.10 Options

The Options directive is used to enable some advanced features of the IBM HTTP Server. Many of these features are explained in other sections in this book in the context or their respective meaning. Here we briefly overview each option and provide references to more detailed descriptions. Then, the syntax of this directive and its usage in configuration files is explained.

The Options directive can have the following parameters:

None	No additional features are enabled. Best case in terms of security.

Indexes	Automatic directory indexing is enabled. See 5.2, “Automatic Directory Indexing” on page 80 for more details.
Includes	Server side includes (SSI) are enabled. See 10.4, “Server-Side Includes” on page 219 for more details.
IncludesNOEXEC	SSIs are enabled, but program execution inside SSIs is disabled. See 10.4, “Server-Side Includes” on page 219 for more details.
MultiViews	The automatic multiple variants feature is enabled. See 5.4, “Multiple Language Support” on page 87 for more details.
ExecCGI	CGI program execution is enabled. See 10.2, “CGI Programs” on page 214 for more details.
FollowSymLinks	AIX file system symbolic link support is enabled; the server follows symbolic links. See 4.9, “Request Mapping” on page 66 for more details.
SymLinksIfOwnerMatch	Acts similar to the previous parameter, but with additional restrictions. See online documentation for more details.
All	This option is equivalent to: `directive Options Indexes Includes ExecCGI FollowSymLinks` Please notice that MultiViews not included.

4.10.1 Syntax

The parameters after the Options directive can have the prefixes + or -. These prefixes indicate that the result should be accumulated with other Options directives. That applies to directives within single sections (such as <Directory>) and directives from other sections or .htaccess files. For more about directives processing, see Section 4.8.4, “Sections Processing Rules” on page 64.

In the following example, the effective options in the directory /www/html/demo are FollowSymLinks and IncludesNOEXEC.

```
<Directory /www/html>
   Options FollowSymLinks Indexes
</Directory>

<Directory /www/html/demo>
```

```
   Options -Indexes +IncludesNOEXEC
</Directory>
```

Any option without a + or - prefix resets all previously set options. For example, the following sequence of directives

```
Options +Includes
Options Indexes
Options +MultiViews
```

is equivalent to the single directive

```
Options Indexes MultiViews
```

because the second line (Option Indexes) without any prefix also resets the previous Option Includes.

Chapter 5. Advanced Configuration

Chapter 4, “Basic Configuration” on page 49 covered some basic features of the IBM HTTP Server that every Web server manager (webmaster) should be familiar with. The IBM HTTP Server has a number of other attractive functions that can be useful in most Web environments.

This chapter explains the most commonly used, advanced features of the IBM HTTP Server and provides some examples on how to use and customize them. It would be beyond the scope of this book to describe all features of the IBM HTTP Server since some of them are seldom used and the list of available functions is almost endless, given the number of generally available modules. For additional information on advanced features, including seldom-used functions that are not described here, we refer you to the online IBM HTTP Server documentation, which is shipped with the product, or to the Apache documentation at `http:www.apache.org`.

IBM HTTP Server versus Apache

It ought to be mentioned once again that the IBM HTTP Server is actually identical to the corresponding release of the Apache server as far as configuration is concerned. The only major differences are the installation process (adapted to IBM AIX) and IBM’s addition of the SSL protocol, that cannot be found in the original Apache server.

The IBM HTTP Server ships with most of the commonly available modules. Should you require to add other modules that are not shipped with the product, you should also read Chapter 8, “Building HTTP Server Modules” on page 177.

5.1 Virtual Hosts

A virtual host appears to be a host of its own, although it is not actually a separate physical machine. The term virtual host in the context of a Web server means that a single physical machine may appear to a client as if there were multiple Web servers.

The following is a typical use: Some companies provide content hosting services for other companies, which are their customers. Such customers would like to appear on the Internet as if they had their own servers for their Web site, thus appearing with their own, unique URL. This is possible using the virtual hosts feature of the IBM HTTP Server.

5.1.1 Concepts

Let's look at an example first: Two companies, CompanyA and CompanyB, would like to establish presence on the Internet with their own URLs: `http://www.CompanyA.com` and `http://www.CompanyB.com`. Both companies, for some reason or another, do not want to build up and maintain their own Web servers. Thus, both companies may ask an Internet Service Provider (ISP) to host and publish their Web pages on their behalf. ISPs usually have Web servers that are powerful enough to handle multiple customers' needs, but they need to set them up such that it looks like several different servers. Each of these servers is then called a *virtual host* because they are actually running on the same physical computer.

What is an Internet Service Provider?

An Internet Service Provider (ISP) is a company that has a permanent connection to the Internet and sells Internet related services for their customers: Internet access, e-mail, Web hosting and other.

5.1.2 IP-Based and Name-Based Virtual Hosts

There are two different kinds of virtual hosts: IP-based virtual hosts and name-based virtual hosts. The difference is in the usage of IP addresses and DNS names. Name-based virtual hosts are not supported by HTTP protocol version 1.0 and therefore require both server and client (browser) to support the HTTP/1.1 protocol. However, some clients that do not support the HTTP/1.1 standard actually do support this virtual host feature.

Each IP-based virtual host must have its own IP address (see Figure 5 on page 73). Usually the computer running the virtual hosts has multiple IP addresses assigned to a single network interface adapter if the operating system supports this (or it might optionally have multiple adapters). IP-based virtual hosts provide a transparent solution to any browser; the function does not rely on any specific browser functionality and therefore tends to be the preferred method for many sites that implement virtual hosts.

From the browser's view, there is no difference between a virtual host and a real host; both have their own hostname (as included in the URL) and associated IP address.

Host versus Web Server

The terms *server*, *Web server*, and *host* are often used interchangeably for the same meaning; a physical machine with adequate software that responds to HTTP-type requests. While *host* is normally only used for a physical machine, it is used in the context of this section with a broader meaning, describing a (virtual) machine including all software that makes up a full Web (HTTP) server.

The key to handling IP-based virtual hosts is the server's ability to handle multiple IP addresses, be it on one single network interface or on multiple interfaces. The Web server software, on the other hand, needs to be able to distinguish and handle separate requests accordingly.

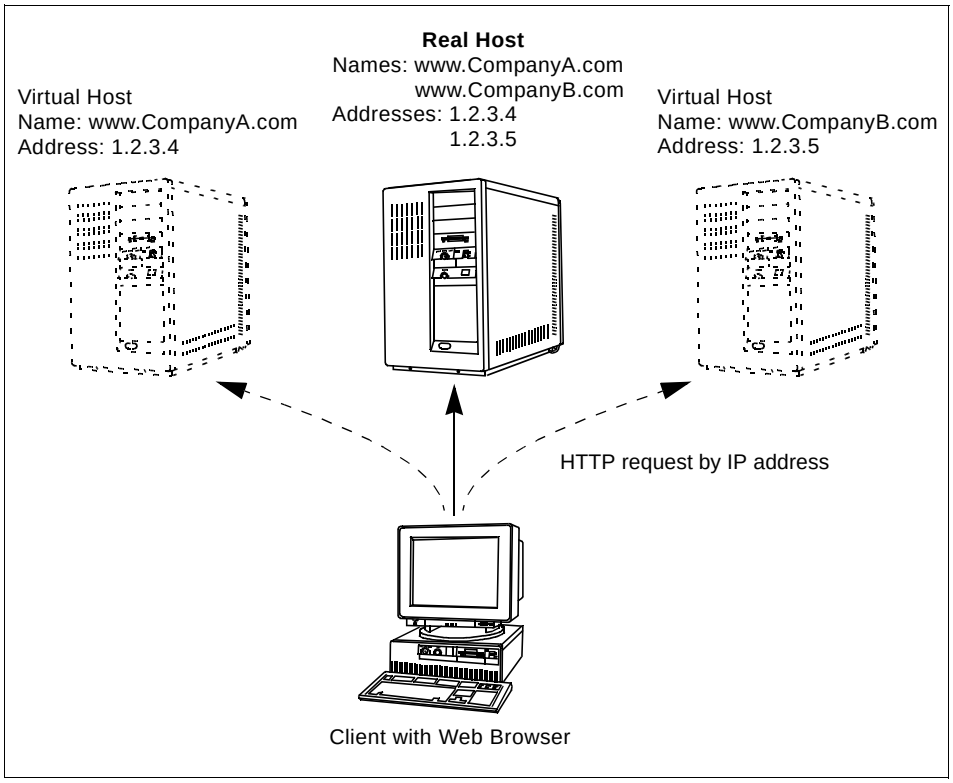


Figure 5. IP-Based Virtual Hosts

Since IP addresses are becoming a short Internet resource, name-based virtual hosts were introduced. All name-based virtual hosts on a single

physical Web server have the same IP address, but are distinguished by the server name in the HTTP protocol header that the browsers sends to the server (see Figure 6). Name-based virtual hosts rely on the HTTP Version 1.1 protocol implementation which may not be supported by older browsers.

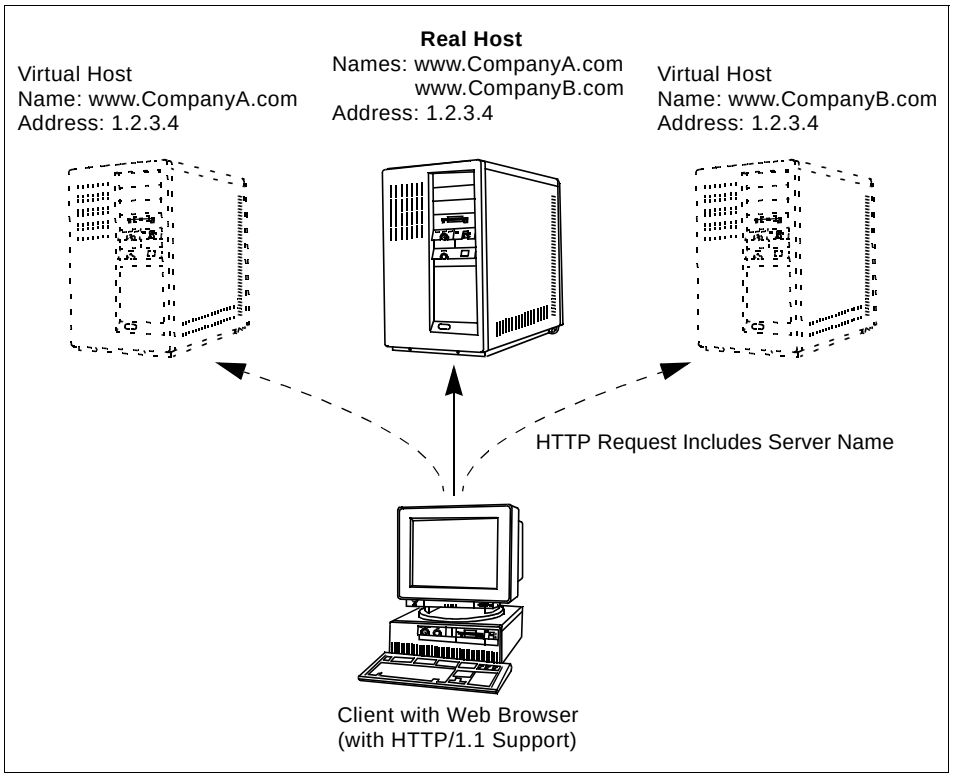


Figure 6. Name-Based Virtual Hosts

The key in name-based virtual hosts is that the clients' requests, which are being routed to the same physical interface with the same IP address, carry the hostnames in their headers such that the Web server software can distinguish them. This feature was only introduced with HTTP Version 1.1 protocol, which must be supported by both the server and the browser.

The IBM HTTP Server supports both IP-based and name-based virtual hosts on the same computer. The next sections explain how these can be set up.

5.1.3 Setting It Up

Setup for IP-based and name-based virtual hosts have much in common, so we do not separate them, but explain the differences in each step.

Setting up virtual hosts on the IBM HTTP Server involves the following tasks:

- TCP/IP and DNS setup
- IBM HTTP Server configuration
- Testing the new environment

The sections that follow discuss each of these tasks.

5.1.3.1 TCP/IP and DNS Setup

Setting up basic operations of TCP/IP and DNS on IBM RS/6000 machines is beyond the scope of this book; we will only cover additional configuration steps required for setting up virtual hosts. You should refer to standard operating system manuals and other publications listed in the Appendix B, “Related Publications” on page 227 if you need more information on this.

As we have seen, each IP-based virtual host must have its own IP address. Multiple IP addresses can be added to a system by either installing multiple network adapters or by assigning multiple IP addresses to a single network interface. To assign an additional IP address to an existing and configured network interface on AIX, use the command `ifconfig` with the `alias` option. The following example shows how to assign an additional IP address (1.2.3.4 with network mask 255.255.255.240) to the Ethernet interface en0:

```
# ifconfig en0  1.2.3.4  netmask 255.255.255.240  alias
```

For a more detailed description of network interfaces and the `ifconfig` command, refer to the AIX documentation.

For each name-based virtual host, you must have a DNS record (alias, also known as canonical name, CNAME) pointing to the same IP address. Such configuration is usually done by a DNS administrator on a different computer that serves as a DNS server, and the details depends on the individual DNS setup.

Either way, for IP-based and name-based virtual hosts, you should check that the newly introduced hostnames correctly resolve into the assigned IP addresses. This can easily be done, for example, with the `host <hostname>`, `ping <hostname>`, or `nslookup <hostname>` commands on AIX.

5.1.3.2 IBM HTTP Server Configuration

The IBM HTTP Server has virtual hosts functionality built into its core, so no additional modules are required. Each virtual host must have its own <VirtualHost> section in the configuration file. The syntax of this section is:

```
<VirtualHost Name_or_Address[:Port] ...>
```

```
   ...
</VirtualHost>
```

Name_or_Address is the fully qualified DNS name or IP address of the virtual host being defined. It is preferable to specify the IP address with this directive for performance and availability reasons because the server does not need to do DNS lookups to find its associated IP address. Port is an optional field; it only needs to be included if a non-standard port (which is 80) is to be used for this virtual host.

You can use most of the IBM HTTP Server directives inside the <VirtualHost> section, but some of them are highly recommended:

- DocumentRoot – This directive should be here because that is most likely the reason for using a virtual host to serve a separate document tree under another server name.
- ServerName – This directive is useful for performance and availability reasons because, when used, the server does not need to do a DNS lookup to find its name. It is not needed, however, when the server's name is specified in the <VirtualHost> directive (see above).
- TransferLog and ErrorLog – These directives specify separate log files for such a virtual server. For details, see 5.7, "Logging" on page 97.
- ServerAdmin – Specify different e-mail address for each virtual host. This e-mail address can be automatically appended to error messages in order to give users some help in case of errors. See 5.5.1, "Customizing Error Messages" on page 92 for more information on using e-mail addresses in error messages.

A recommended minimal <VirtualHost> section of an IP-based virtual host could look like:

```
<VirtualHost 1.2.3.4>
   ServerName www.CompanyA.com
   ServerAdmin webmaster@CompanyA.com
   DocumentRoot /www/html/CompanyA
   ErrorLog /www/logs/CompanyA/error_log
   TransferLog /www/logs/CompanyA/access_log
</VirtualHost>
```

If you decide to use name-based virtual hosts, you must use the NameVirtualHost directive (outside of the <VirtualHost> section). This directive specifies the IP address which will be used for name-based virtual hosts. In this case, a corresponding section of a configuration file could look like:

```
NameVirtualHost 1.2.3.4
...
<VirtualHost 1.2.3.4>
   ServerName www.CompanyA.com
   DocumentRoot /www/html/CompanyA
   ...
</VirtualHost>
...
<VirtualHost 1.2.3.4>
   ServerName www.CompanyB.com
   DocumentRoot /www/html/CompanyB
   ...
</VirtualHost>
```

Do not forget to grant appropriate access rights for document directories with appropriate <Directory> directives. More about directory access rights and related directives can be found in 6.2, “Basic Authentication” on page 118.

Name-based virtual hosts have some additional configuration options. If it is necessary to access the same virtual host by more than one name, additional names can be listed with the ServerAlias directive. An example might be that you would like to access the name-based virtual host support.CompanyA.com also with the short name supp. In this case, you can define an additional `<VirtualHost>` section or add a ServerAlias directive to a `<VirtualHost>` section as shown in the following example:

```
NameVirtualHost 1.2.3.4
...
<VirtualHost 1.2.3.4>
   ServerName support.CompanyA.com
   ServerAlias supp
   DocumentRoot /www/html/CompanyA
   ...
</VirtualHost>
```

It should be pointed out regarding the example above that it is required that the server aliases only work as long as the hostname supp resolves into the IP address 1.2.3.4.

5.1.4 Testing

After changing the server configuration file as shown, it is recommended that you verify the configuration file syntax with the `apachectl configtest` or the `httpd -t` command. If there are errors, the commands will notify you immediately. If the commands reports “Syntax OK”, you can restart the server

with command `apachectl graceful` in order to read and apply the new configuration to the running server.

After successful restart of the server, try to access the new virtual servers from a browser. Make sure that the Web server provides the correct pages for each virtual host. If problems arise with name-based virtual hosts, it may be caused by the browser if it does not fully support the HTTP/1.1 protocol standard. If you suspect such a problem, try a newer version or another product. Most current Web browsers support HTTP/1.1, but not all.

5.1.5 Logging

If you have virtual hosts set up, you would possibly like to have separate log files for each of them. The IBM HTTP Server allows you do this by using the TransferLog and ErrorLog directives in the <VirtualHost> section. If you do not specify these directives, the server writes all log records into a single log file specified in the main section of the configuration file. (However, there are tools available on the Internet that allow you to split single log file into several log files, along with other functionality to analyze log files.)

More about logging can be found in 5.7, “Logging” on page 97.

5.1.6 Compatibility with Older Browsers

There is some compatibility workaround for name-based virtual hosts support by Web browsers not supporting HTTP/1.1. It uses the IBM HTTP Server directive ServerPath and module mod_rewrite.

Note on mod_rewrite

The module mod_rewrite is actually very powerful and supports functions far beyond the single use that will be explained. For more information about this module, refer to 2.3.2, “Translation Modules” on page 21 or the original Apache documentation.

When a client request does not include a server name, the IBM HTTP Server uses the first <VirtualHost> section that matches the IP address. It is therefore recommended to create a special <VirtualHost> section as the first that always returns a list of all available virtual hosts with references to the separate subdirectories of each virtual host (see example shown in Figure 7, where an old version of Microsoft Internet Explorer was used that did not support the HTTP/1.1 protocol). It is then the user’s choice to select the correct page.

Figure 7. Error Page for Compatibility on Old Browsers

The following example shows how to set up a configuration file for compatibility with old Web browsers:

```
...
NameVirtualHost 1.2.3.4

<VirtualHost 1.2.3.4>
   DocumentRoot /www/html
   RewriteEngine On
   RewriteRule ^/.* /www/html/index.html
</VirtualHost>

<VirtualHost 1.2.3.4>
   DocumentRoot /www/html/CompanyA
   ServerName www.CompanyA.com
   ServerPath /com_a
   ...
</VirtualHost>

<VirtualHost 1.2.3.4>
   DocumentRoot /www/html/CompanyB
   ServerName www.CompanyB.com
   ServerPath /com_b
   ...
```

```
</VirtualHost>
```

Important Note

In order to make everything work, use either relative references between documents belonging to the same virtual host (for example ../img/logo.gif), or include the full corresponding subdirectories (for example /com_a/img/logo.gif). Do not use absolute references (like /img/logo.gif), because in that case the server will return the virtual hosts list again.

5.2 Automatic Directory Indexing

It is quite common in HTML requests not to request a single file but rather a directory. Most Web servers in such case search for some default files in the specified directory. The IBM HTTP Server, by default, looks for and returns the file index.html (which can be changed with the DirectoryIndex directive) if it is present in that directory. If that default file cannot be found, the IBM HTTP Server can built a directory index automatically. A directory index is just a single formatted list of files in that directory, annotated with some links and icons (if configured). This feature is implemented with the standard module mod_autoindex.

The main advantage of an automatic directory index is that it requires almost no effort to keep it up to date. On the other hand, an automatically created directory index may not be as flexible as an index file and not as intuitive to user. Automatic directory indexes are useful for big or frequently changing directories, or, for example, simple collections of files, such as picture images. They are also widely used for downloadable files listings.

5.2.1 Simple and Fancy Indexes

Automatic directory indexing can be enabled using the directive Options Indexes. By default, it returns a simple file and directory listing, as shown in Figure 8 on page 81.

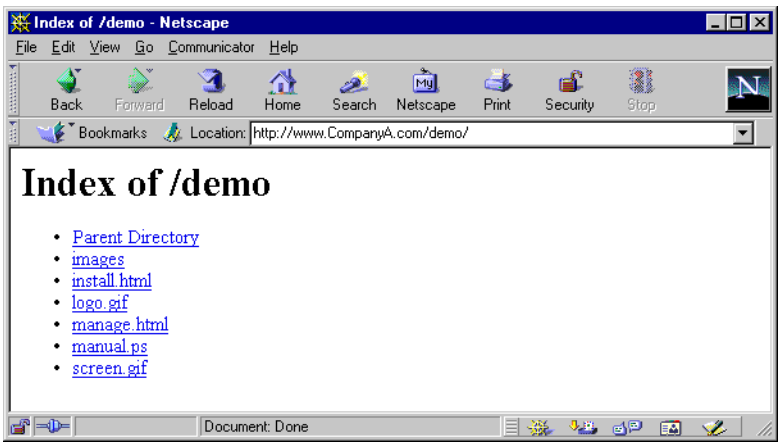


Figure 8. Example of a Default Automatic Directory Index

The directive IndexOptions FancyIndexing can improve the look of an automatically created directory index by adding more information about file types and directories, as shown in Figure 9.

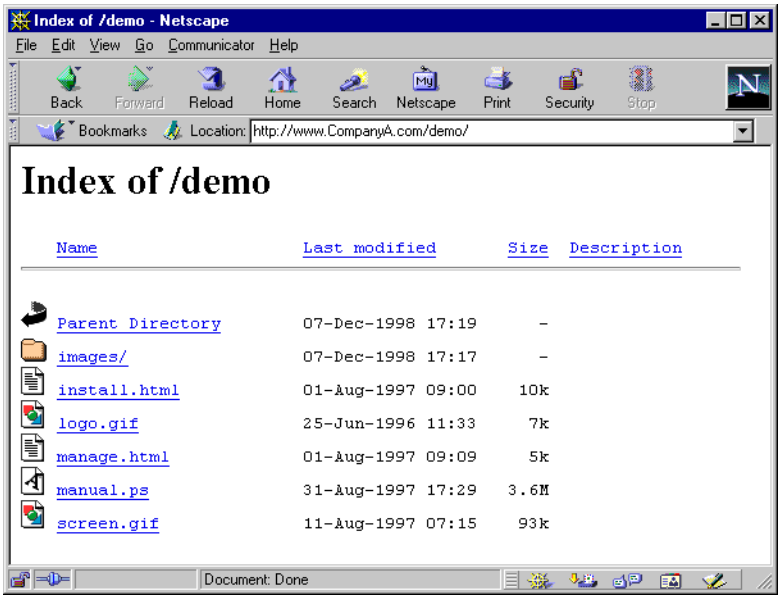


Figure 9. Example of a Fancy Automatic Directory Index

Here is an excerpt of a possible configuration file that specifies fancy indexing:

```
<Directory /www/html/demo>
   Options Indexes
   IndexOptions FancyIndexing
   ...
</Directory>
```

5.2.2 Adding Text to an Index

The module mod_autoindex provides many ways to customize the appearance of an index. One of the most powerful features is the ability to add some information in front of the index listing and after it. This information can clarify the contents of the directory being listed (see Figure 10).

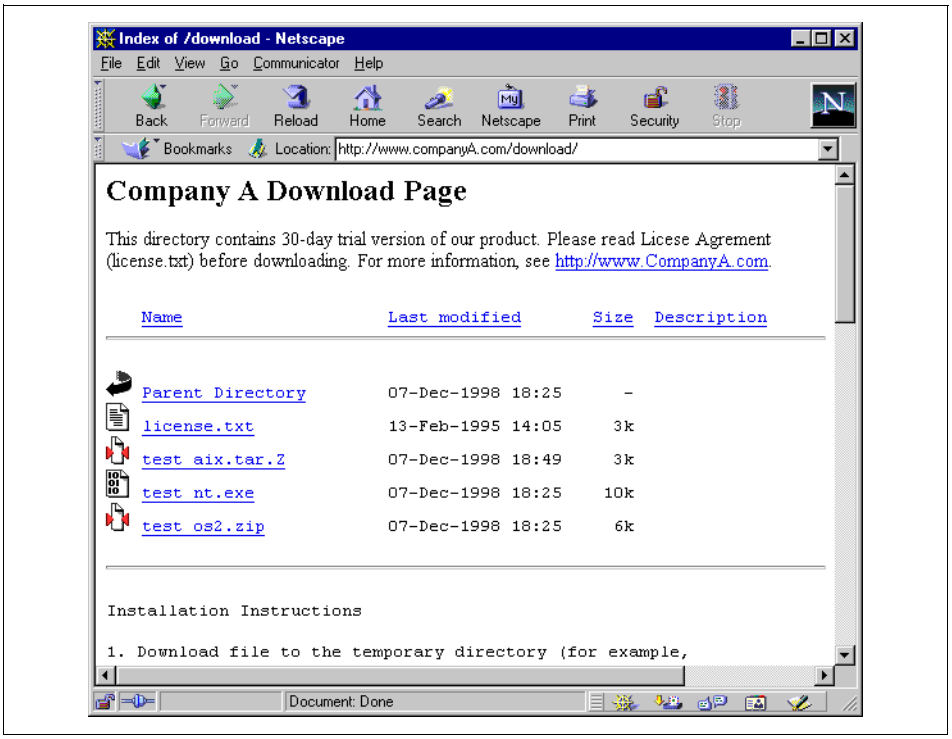


Figure 10. Directory Index with Additional Text

Adding text to an index can be done by using the directives HeaderName and ReadmeName. They specify names of files that are to be included into the generated HTML document representing the index. These files can be plain text files or HTML fragments. For example, the directive ReadmeName

README instructs the IBM HTTP Server to append the contents of a file named REAME.html or README to the end of the directory index. Usually files specified by HeaderName and ReadmeName directives cannot have <HTML>, <HEAD> and <BODY> tags since they are only part of an HTML document. However, the directive IndexOptions SuppressHTMLPreamble allows the use of HTML headers in the file specified by the directive HeaderName. In this case, the server does not automatically generate an HTML header.

5.2.3 Excluding Files from an Index

Another useful directive, IndexIgnore, specifies files that should be excluded from an index. First candidates in that list are HEADER and README files mentioned in the previous section, and .htaccess files used for access control in that directory. Wildcard expressions can also be used. For example, the following directive excludes the parent directory (..) and GIF files from the index listing of directory /www/html/samples:

```
<Directory /www/html/samples>
   IndexIgnore .. *.gif
   ...
</Directory>
```

Right after installation, the IBM HTTP Server's configuration file includes the following directive in its main section:

```
IndexIgnore .??* *~ *# HEADER* README* RCS
```

This default directive excludes the following files from all indexes:

- Files whose names begins with dot and are longer than two characters (that means that the parent directory is not excluded)
- Files whose names end with either the tilde (~) or the pound (#) character
- Files whose names begin with HEADER and README (usually appended to directory index)
- File with the name RCS (Revision Control System)

5.2.4 Additional Customization

Additional directory index customization can be done using the IndexOptions directive. For example, the following directive allows only icons and file names in the directory listing (thus excluding modification date, size and description):

```
IndexOptions SuppressLastModified SuppressSize SuppressDescription
```

Syntax of IndexOptions

The options in the IndexOptions directive can have the prefixes + or -. These prefixes indicate that the result should be accumulated with previous IndexOptions directives. That applies to options within single section (like <Directory>) and are inherited from higher-level sections. For more about directives processing in sections, see 4.8.4, “Sections Processing Rules” on page 64.

In the following example, the effective options in the /www/html/demo/images directory are SuppressLastModified and SuppressDescription.

```
<Directory /www/html/demo>
   IndexOptions SuppressLastModified SuppressSize
</Directory>

<Directory /www/html/demo/images>
   IndexOptions -SuppressSize +SuppressDescription
</Directory>
```

Any option without prefix + or - resets all previously set options. For example, the following sequence of directives:

```
IndexOptions +SuppressDescription
IndexOptions SuppressSize
IndexOptions +SuppressLastModified
```

is equivalent to the single directive:

```
IndexOptions SuppressSize SuppressLastModified
```

because option SuppressSize without prefix also resets the previous option SuppressDescription.

The directive IndexOptions ScanHTMLTitles instructs the server to display HTML document title information in the description field. Note that this can seriously impact server performance because all HTML documents in a directory need to be scanned on each request. A much more lightweight method for adding descriptions to files is through the use of the directive AddDescription. The following example shows how to use this directive:

```
AddDescription "Network diagram" /www/images/diag.gif
```

It can be used in the main configuration file, but a much more convenient place to use it is the local .htaccess file in that directory.

The following example shows how to combine automatic directory indexing directives:

```
<Directory /www/html/download>
  Options Indexes
  IndexOptions FancyIndexing
  IndexOptions +ScanHTMLTitles
  IndexIgnore .??* HEADER* README*
  HeaderName HEADER
  ReadmeName README
  AddDescription "Compressed file" *.zip
</Directory>
```

5.2.5 Security Considerations

Enabling automatic directory indexing has some security aspects. They can expose directory structure and file names that you otherwise might want to hide. If automatic directory indexing is enabled for some directories, file listings will be shown for other directories in case the index file is deleted or renamed accidentally. It is therefore recommended to disable automatic directory listings for all directories except for those you really want to show as file listing. To disable directory indexes for all directories, add the directive Options -Indexes to the root directory section (<Directory />) in the server configuration file.

5.3 User Directories

Some organizations let their employees (or students in case of universities) publish their own Web pages through their main Web server. Users simply put their files into some predefined subdirectory in their home directories.

The IBM HTTP Server supports this feature with the module mod_userdir. This module is by default included in the appropriate LoadModule and AddModule section of the httpd.conf file, thus no action is required to enable them. The directive UserDir defines where the Web server must look for user files. A user directory is specified by using the ~ (tilde) prefix in front of a user's name in an URL (for example: `http://www.CompanyA.com/~joe/`).

For example, a server's configuration file contains the directive UserDir public_html. Assuming that Joe's home directory is /home/joe, the URL `http://www.CompanyA.com/~joe/images/foto.gif` would then return the file /home/joe/public_html/images/foto.gif.

The use of the user directory feature has also some security aspects to consider. For example, if you specify UserDir ./, the whole file system could

become accessible through the URL /~root (assuming root's home directory is /). That is why the UserDir directive also supports the keywords *enabled* and *disabled*. Using these, access to root's home directory can explicitly be disabled using by the following:

```
UserDir disabled root
```

Another security hole can be exposed by using erroneous CGI programs and server side includes in a user's directory. The IBM HTTP Server allows you to disable CGI programs and server side includes by using the directive Options -ExecCGI -Includes in the server configuration file. Another option, the Options -IncludesNOEXEC, allows server-side includes, but the #exec command and #include of CGI scripts are disabled.

To read more about CGI programs and server side include, see Chapter 10, "Web Applications" on page 213. More about security can be found in Chapter 6, "Deploying Security" on page 111.

The IBM HTTP Server accesses users' home directories use the AIX operating system APIs. This makes the location of the directories and files transparent to the server and it does not care where these directories are physically located. If the users' home directories are located on some sort of a distributed file system (such as NFS, AFS or DCE/DFS), they can be on separate machine.

Distributed File Systems

NFS (Network File System), AFS (Andrew File System) and DCE/DFS (Distributed Computing Environment/Distributed File System) are variants of distributed file systems that allow secure and efficient file sharing across different locations and operating systems in local or wide area networks.

Using a distributed file system adds more flexibility, but may rise at the same time some performance and availability concerns arise. The main advantage of such a configuration is the ability for users to publish their files without the necessity for any special file transfer to the Web server machine. HTML file editing can be done locally on each user's workstation and, after saving a file to the specified directory (which appears to be local on the user's workstation), it is instantaneously available on the Web. On the other hand, a Web server cannot access and serve these files if some network problems arise or if a machine on which the files are physically located will experiences a problem.

5.4 Multiple Language Support

In many countries, more than one common language is used. Companies also want to address the international market through the Internet. For this reason, it is desirable to provide information in several languages. The IBM HTTP Server supports multiple language configurations by providing means for transparent language negotiation between server and browser.

Multiple language support is a part of the broader IBM HTTP Server feature called *Content Negotiation* which allows a server and browser to negotiate document language, encoding and media type (for example, a client can specify which video format it prefers: MPEG, MOV, or AVI). But this feature is not widely used because most browsers do not fully implement the content negotiation feature.

This section covers only language negotiation because it is most useful and most widely supported. Encoding and media type negotiation works similar and the following discussion on language negotiation applies to that as well.

5.4.1 Server Configuration

The IBM HTTP Server supports the content negotiation feature using the mod_negotiation and mod_mime modules. They are both enabled by default in httpd.conf by their respective LoadModule and AddModule directives, so you do not need to do anything special (unless they have been disabled, of course).

When the same resource (text, image) is available in several formats (encoding, language), they are called *variants*. A server must select the most appropriate variant according to client preferences.

There are two different ways to configure variants on the server:

- Type maps, which are special files that explicitly list variants with related information
- MultiViews, which are variants lists automatically generated by the server according to file extensions

5.4.1.1 Type Maps

Type maps are special files processed by the type-map handler. To enable the handler and associate it with, for example, the file extension .var, the IBM HTTP Server configuration file must include the following line:

```
AddHandler type-map var
```

Type map file lists files and associated content information separated by empty lines.

For example, CompanyA has some warranty information in English (file warranty.html) and German (file garantie.html). The contents of the type map file, named info.var, would then be the following:

```
URI: warranty.html
Content-type: text/html
Content-language: en

URI: garantie.html
Content-type: text/html
Content-language: de
```

In that case, a request for `http:/www.CompanyA.com/info.var` will return information in the appropriate language according to the browser's configuration (see next section). If no appropriate content can be found (for example if the browser accepts only French documents), a list of available documents will be displayed, as can be seen in Figure 11.

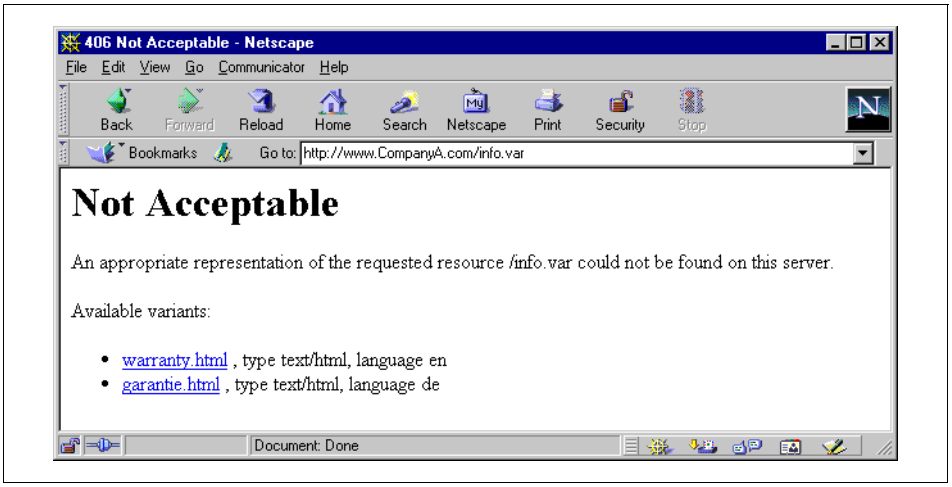


Figure 11. Type Map Variants List

This gives the user the choice of selecting the appropriate document (which should certainly be avoided by proper configuration of browsers and servers).

5.4.1.2 MultiViews

The MultiViews feature allows you to build variant lists automatically according to file extensions. The MultiViews feature can be enabled by the

directive Options MultiViews in the configuration file or by means of the .htaccess file. Note that Options All does not set MultiViews. The configuration file should also contain the file extension to language mapping directives AddLanguage. It is also recommended to use the language priority directive LanguagePriority for those cases when a browser does not specify any language priority. The following fragment of the configuration file illustrates how to setup MultiViews:

```
Options MultiViews
AddLanguage en .en
AddLanguage de .de
LanguagePriority en de
```

These directives associate file extension ".en" with the English language and ".de" with the German language. They also assign the English language documents a higher priority when clients do not specify any language priority.

Now, if there are files info.html.en and info.html.de in the document root directory, the request `http://www.CompanyA.com/info.html` will return the appropriate document according to the browser's configuration. If it is not possible to find an appropriate document (for example, if the browser accepts only French documents), the IBM HTTP Server returns a list of available variants, as shown in Figure 12.

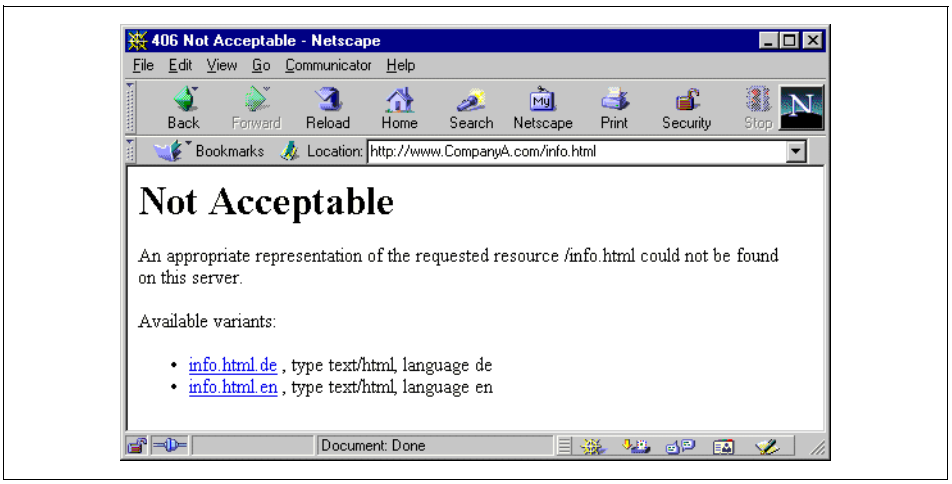


Figure 12. MultiViews Variants List

While the screen shown in Figure 16 is certainly not desirable for a user, it shows you the basic operation of MultiViews in an non-working configuration.

5.4.2 Browser Configuration

Users who want to use the language negotiation function of the HTTP protocol must setup their Web browsers language preferences. Most popular browsers support this feature. The following examples show how to set up multiple language support in Netscape Navigator, Netscape Communicator and Microsoft Internet Explorer.

Netscape Navigator (Version 4.X) and Netscape Communicator users can access the language preferences from: **Edit** -> **Preferences...** -> **Navigator** -> **Languages**. Here they can add or remove languages and change their priority by clicking on the appropriate function buttons (see Figure 13).

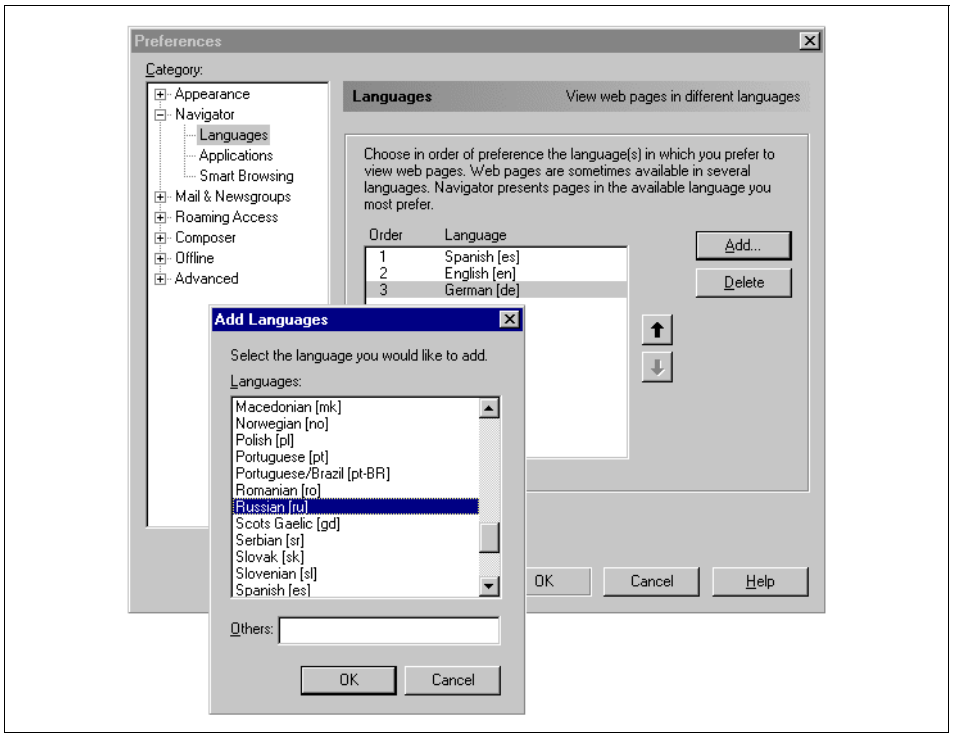


Figure 13. Netscape Communicator Language Setup

In Microsoft Internet Explorer (Version 4), users can access the language preferences from: **View** -> **Internet Options...** -> **General** -> **Languages...**. An example is shown in Figure 14 on page 91.

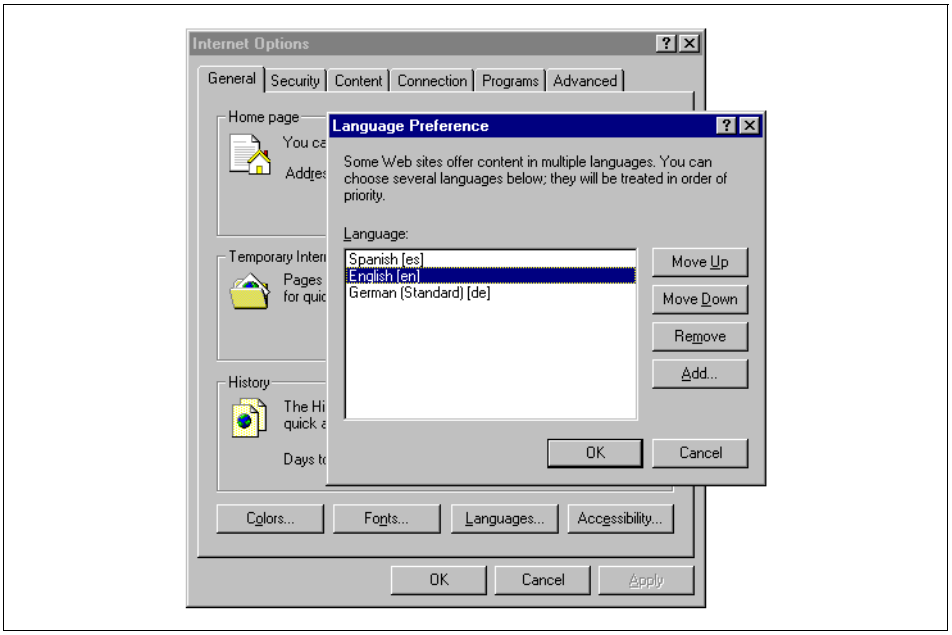


Figure 14. Microsoft Internet Explorer Language Setup

Although the negotiation feature was supported in the HTTP protocol prior to Version 1.1, many browsers did not support it in earlier versions. The HTTP protocol also provides the option of assigning weights to each variant to further tune the negotiation process between server and browser (which is not detailed here any further).

5.5 Customized Error Messages

When the IBM HTTP Server encounters an error when serving a client's request (for example, a requested file does not exist or the access to it is restricted), it returns a default error message to the client (a well known error message for most of us, but if you have not seen it yet, see Figure 15 on page 92). These default error messages are very simple and give no clues to the user as to what can be done next. You probably would like to customize these messages to better suite your environment. For example, in case a document cannot be found, you can give a user further advice about where to look for it, point to a search engine, or simply link back to the home page.

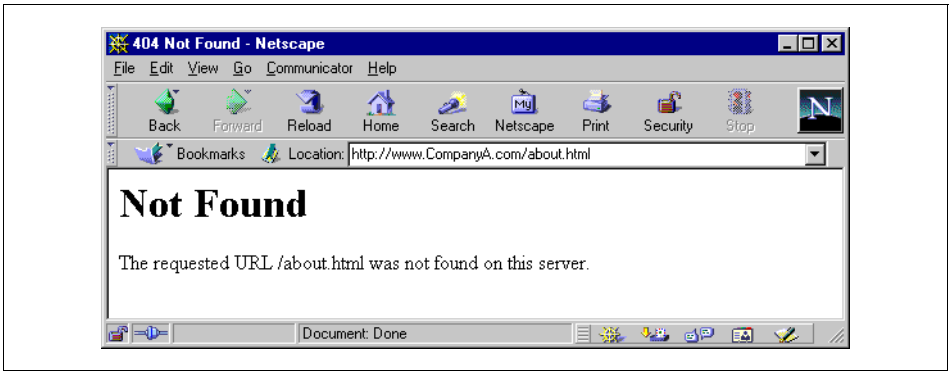


Figure 15. Standard Error Message

There are a number of situations when the IBM HTTP Server returns an error message. Each of these errors has an assigned number, a so-called error code. A full list of error codes can be found in the HTTP protocol specification (RFC 1945 for Version 1.0 and RFC 2068 for Version 1.1; see B.3, “Other Publications and Links” on page 228 for information on how to access RFCs).

The most common error codes are:

- 401 – Authorization Required. This error means that in order to access the resource, the user must prove his identity (usually with a password). In case of this error, typical Web browsers will prompt for a user name and password. If you skip that prompt (by pressing **Cancel** or **ESC**), you will typically get this error message in your browser window.
- 403 – Forbidden. This error means that the access to the requested resource is prohibited. That could be as a result of a deny directive (for example, restrictions by client IP address) or AIX file access permissions.
- 404 – Not found. This error means that the requested resource, such as a file, directory, or a script, does not exist at the specified location (see also Figure 15). This error is very common for most users because of typos in URLs or removed documents. It is highly recommended to customize this error message to give a user some additional help.

5.5.1 Customizing Error Messages

The IBM HTTP Server supports custom error messages in its core. You can instruct the server what to do in case of an error:

- Return the default message
- Return a customized message

- Redirect the request to another local or remote URL (including CGI programs)

The first option is the default. It takes place if no other option is specified. You can, however, specify that the e-mail address specified with the ServerAdmin directive be included automatically as a hotlink in the default error message. For example, we assume that the following two lines are included in the server's configuration file:

```
...
ServerAdmin webmaster@CompanyA.com
...
ServerSignature email
...
```

This causes the IBM HTTP Server to include a hotlink in its default error messages that allows a user to click on and send an e-mail to the specified address at webmaster@CompanyA.com (see Figure 16, notice the underlined hotlink in the footer of the error page).

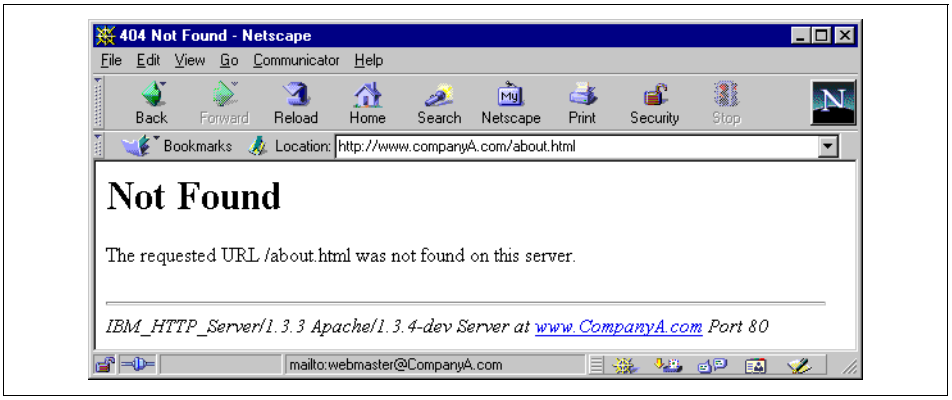


Figure 16. Error Message with Server Signature

The following example shows how to define a custom error message using a ErrorDocument directive in the server configuration file:

```
ErrorDocument 404 "The requested resource cannot be found on this server.
Please return to our <A HREF="http://www.CompanyA.com">home page</A>.
```

(Note that the above example is a single line that has been split for representation.)

Syntax of the ErrorDocument Directive

The ErrorDocument directive has a syntax that sometimes misleads webmasters. A single quotation mark (") after the error code is required, which indicates the beginning of the message and it does not require a closing quotation mark. In fact, any other quotation mark is treated as part of the message that would be shown along with the error message.

Figure 17 shows the result of the custom error message defined above.

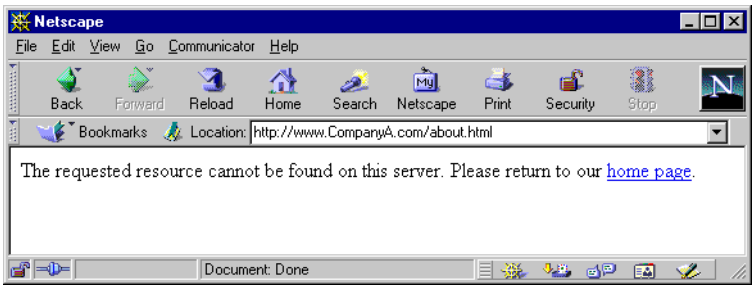


Figure 17. Example of Custom Error Message

As can be seen in the example above, HTML tags can be used in customized error message. However, some browsers may not interpret them correctly because that message does not have a standard HTML header.

The third option can be configured by placing a local or remote URL after the respective error code behind the ErrorDocument directive. That URL can point to either a static HTML file or to a script that processes errors. Local URLs begin with a slash (/) and remote URLs include protocol and host name. Here are some examples of error redirections:

```
ErrorDocument 401 http://www.CompanyA.com/security.html
ErrorDocument 403 /forbidden_pages.html
ErrorDocument 404 /cgi-bin/search.pl
```

You can use different error handling directives for each virtual host, directory or location. You can also use them in the .htaccess files for individual directories.

5.5.2 Multilingual Error Messages

There are some tricks that allow you to have error messages in different languages appropriate to the browser's configuration. These tricks use the

ErrorDocument directive, the content negotiation feature (see also 5.4, “Multiple Language Support” on page 87) and eXtended Server Side Includes (XSSI, see 10.4, “Server-Side Includes” on page 219).

Server side includes allow you to have a common page layout for all languages and the content negotiation feature selects the corresponding language.

A good place for further information on this tricky setup can be found at `http://www.apache.org/docs/misc/custom_errordocs.html`.

5.6 File Uploading

Normally, HTML documents and relates files are sent to the Web server using FTP protocol. That requires the user to run and manage an FTP server, which consumes administrator time and system resources (and poses a certain security threat). The IBM HTTP Server provides an opportunity to upload files using the PUT and POST methods of the HTTP protocol. That integrates all functions under one server, but improperly configured, it can create a serious security breach.

While the HTTP protocol includes ways to upload files from a Web browser to a Web server using either the PUT or POST mechanism, there is no standard way of handling such requests on the server side. Thus, the IBM HTTP Server does not include any handler that can process such uploads. You can, however, find them on the Internet or write a small program yourself. PUT and POST requests can be processed by either CGI programs or Web server modules. The configuration is different in each case, but their common purpose is to receive a file from the client, while at the same time tightening security as much as possible.

In case of a CGI program, it is highly recommended to put it into a separate directory and protect that directory using at least basic authentication (see 6.2, “Basic Authentication” on page 118) and, if appropriate, other restrictions, such as IP addresses. With such methods, only authorized users can use the upload function. An even higher level of security can be achieved using SSL client authentication (see 6.5, “SSL Client Authentication” on page 146).

The following example shows how to configure the IBM HTTP Server to use a Perl script called /www/put-cgi/put.pl for file uploading to the directory tree /www/html using basic authentication and restriction by IP address:

```
<Directory /www/html>
```

```
   Script PUT /put-cgi/put.pl
</Directory>

<Directory /www/put-cgi>
   AuthName "Web Publishing"
   AuthType Basic
   AuthUserFile /www/passwords/put.pwd
   Order allow,deny
   Deny from All
   Allow from 1.2.3.6
   Require valid-user
   Satisfy All
</Directory>

ScriptAlias /put-cgi /www/put-cgi
```

The PUT method can also be handled by server modules, rather than a CGI program. In that case, the configuration is dependent on that module's capabilities, but the security considerations and configuration basically remain the same. You can find such a module by searching the Web site `http://modules.apache.org` for keyword "put".

More about using the PUT method can be found in the *Apache Week* article "Publishing Pages with PUT" at `http://www.apacheweek.com/features/put`.

Another method of file uploading is the POST method. You also need at least a CGI program or a module to handle this HTTP method. One of the most widely used examples of such a file uploading method is through Microsoft's FrontPage Server Extensions.

FrontPage Server Extensions

Microsoft FrontPage is a popular Web authoring tool. It includes server extensions as server side components that, among other things, provide file uploading capabilities. The IBM HTTP Server is supported by Microsoft FrontPage Server Extensions as Apache 1.3.3.

More information about Microsoft FrontPage Extensions can be found at these URLs:

- `http://www.microsoft.com/frontpage/wpp/`
- `http://www.rtr.com/fpsupport/`

WebDAV (World Wide Web Distributed Authoring and Versioning) is a new Internet protocol for file uploading being developed by the Internet

Engineering Task Force (IETF). This protocol is not widely used (yet) but it can be assumed that it will become the file uploading standard for the Web. There is already a module mod_dav available which implements a limited version of the WebDAV protocol. For more information on WebDAV, see RFC 2291 and point your browser at `http://www.ics.uci.edu/pub/ietf/webdav/`.

5.7 Logging

The IBM HTTP Server provides good logging options to track user behavior, server usage and find problems that are related to Web pages. The figures of Web server usage are often needed for marketing purposes. There are lots of free applications on the Internet to parse the log files to fit into presentation formats. In this section we will concentrate on showing how you can log every useful detail in the Web servers to log files and how to read them. Many of the problems and configuration errors are easily solved if the logging is defined properly. In the IBM HTTP Server, the log files are ASCII text files that can be viewed with basic UNIX tools.

5.7.1 Common Log Format

The log format is basically similar among all the Web servers. This log format is also known as CLF, Common Log Format. Below is an example of a transaction log file (TransferLog) entry:

```
1.2.3.4 - ihsadm [11/Dec/1998:16:14:48 -0600] "GET /acme.html HTTP/1.0" 200 238
```

The first field shows the IP address of the client, accessing the server. It can also be an IP address of a firewall or a proxy server the client uses to access the Internet. In this example, the Web server was not configured to resolve the hostname and domain of the client, which is advisable for performance reasons. The more efficient way to resolve the hostnames is to use the logresolve program that is shipped with the IBM HTTP Server. The logresolve program is introduced in Section 7.1.3.4, “Resolution and Mapping” on page 162. If the server had looked up the DNS hostnames, the IP address would have been replaced with a hostname (if defined in DNS). The DNS lookups can be defined on with HostnameLookups (on | off | double) directive. The value double should not be used unless it is unavoidable. When double DNS lookups are defined, the server does an additional lookup for the IP address of the hostname it has resolved using the client’s IP address. Double DNS lookups are done regardless of the value of the HostnameLookups directive when the resources in a protected area are requested. If double DNS resolution fails, that is, the IP address of the connecting client and the IP address resulting from a double look up do not match, the access is denied.

The second field of a transaction log entry is usually just a hyphen (-). The hyphen in the log files represents that the information is not available. The field is for *Ident* information. Ident is a protocol (RFC 931) based on the presumption that the Ident daemon is running on the client machine. The Ident daemon is usually not implemented (or configured) on client machines, so the benefit are not worth the overhead it causes in the Web server. The checking of the client's Ident information can be controlled with the directive IdentityCheck (on | off). The IdentityCheck directive defaults to off.

The third field displays the user name of the client, if the client user has authenticated to the Web server using the Web authentication methods. Remember that this file easily reveals the authorized Web users of the system. If the file is not protected properly, anyone can produce a list of possible user IDs with a command such as (<log file> is the server's access log file):

```
# cat <log file> | cut -d' ' -f3,9 | grep -v 401 | grep -v '-'
```

The date, time and timezone are enclosed into brackets. The date representation consists of day number, three letter abbreviation for the month and the four digit number for the year. The language of the abbreviation for the month depends on the language environment used on the server machine.The time is presented in 24 hour format: hours, minutes and seconds, separated by colons. The timezone is presented with a +/- sign and four digits without a separator (+/-hhmm).

Within the double quotes is the HTTP request that the client sent. Usually it begins with GET, which is the basic method to request a file from the server. The file name in the log file is written as the browser sees it. That is, the location is not yet parsed to match the actual file name.

The following three-digit-number field (the second last), represents the status code of the operation, the most commonly used status codes are as follows:

Table 9. Commonly Seen Status Codes in HTTP Requests

Status Code	Description
200	OK - The request was completed successfully.
302	Moved Temporarily, Redirection - The request is redirected to some other location by the server.
304	Not Modified - A Web server responds with this status code to a client's conditional GET request, where the client asked whether it can use the cached copy of the requested resource.
400	Bad Request - Client has performed a request with a malformed syntax.

Status Code	Description
401	Unauthorized - The Web server denies access because it has some access limitations defined to requested resource (See 6.2, “Basic Authentication”).
403	Forbidden - The server configuration denies the access to this requested resource.
404	Not Found - The server was not able to find the requested resource.
405	Method Not Allowed - The Web server configuration denies the use of this method.
406	Not Acceptable - The client requested a resource with content characteristics that are not acceptable (See 5.4, “Multiple Language Support”)
500	Internal Server Error - The Web server encountered an internal error, which prevented it from fulfilling the request.
503	Service Unavailable - The Web server can give that status code, if the defined limit of users is exceeded.

A more complete list of status codes can be found in the RFC document for HTTP Version 1.1 (`http://www.w3.org/Protocols/rfc2068/rfc2068`).

The last field in the transfer log entry is the count of transferred bytes. This does not include the header parts in the response.

Robot Exclusion

Sometimes you might see a log file entry like this:

```
204.123.9.20 - - [11/Nov/1998:15:24:25 -0600] "GET /robots.txt HTTP/1.0"
404 336
```

A robot is a program that traverses Web sites on the Internet and indexes the keywords it finds into it's own database. Robots are typically used to update page indexes of search engines. The machine in the example above is called scooter and it is one of Altavista's Web robots.

robots.txt is a file that you can add to your document root and define some restrictions for robots; what they should not index and what they should. You might want to deny robots of indexing some frequently updating pages.

Here is an example of a robots.txt file:

```
# The rules below applies to all robots
User-agent: *
Disallow: /pics/ # Nothing to index here
Disallow: /Docs/Serv/hot/ # Content due to change frequently
```

The file robots.txt must be located directly under document root and the access rights should permit anyone to read it.

More information about robot exclusion can be found, for example, at `http://info.webcrawler.com/mak/projects/robots/robots.html`.

5.7.2 Error Log

The error log does not only contain error messages, but also informational messages, debug messages and user tracking information. For more information about the user tracking with client certificates, see 6.5.1, "Client Certificates and the IBM HTTP Server" on page 148. The logging level of the IBM HTTP Server can be adjusted with the LogLevel directive. The possible values for LogLevel are *debug*, *info*, *notice*, *warn*, *error*, *crit*, *alert* and *emerg*. By default, the logging level is set to warn. That means that all the events, except debug, info and notice information are written to the error log file.

The structure of error log entries differs from CLF (Common Log Format). Below is an excerpt from an example error log:

```
[Fri Nov  6 14:24:48 1998] [error] [client 1.2.3.4] File does not exist:
/usr/docs/status
```

```
[Fri Nov  6 14:24:57 1998] [error] [client 1.2.3.4] File does not exist:
/usr/lpp/HTTPServer/share/htdocs/status
[Fri Nov  6 14:25:05 1998] [error] [client 1.2.3.4] File does not exist:
/usr/lpp/HTTPServer/share/htdocs/server-status
[Fri Nov  6 14:26:35 1998] [notice] SIGHUP received.  Attempting to restart
[Fri Nov  6 14:26:37 1998] [notice] IBM_HTTP_Server/1.3.3 Apache/1.3.4-dev
(Unix) configured -- resuming normal operations
[Fri Nov  6 14:26:41 1998] [error] [client 1.2.3.4] File does not exist:
/usr/lpp/HTTPServer/share/htdocs/server-status
```

In each error log entry, the date and time are enclosed into brackets first, and the following field describes the severity of the event. If a client was involved in the event, its IP address (or hostname) is contained in the next field. The explanation part begins with the name of the module where the event occurred. If the event occurred in the server core, the field begins with “httpd” or just the message text.

ErrorLog messages can also be directed to the syslog daemon that handles all the system error massages on AIX. This opens up a possibility to combine the IBM HTTP Server with some system management software like Tivoli TME. By default, the directive uses syslog facility local7, but the facility can be overridden by adding a colon and the facility name to the directive. For more information about the syslog daemon, see the online manual page for syslogd or the AIX manuals. The syslog logging can be turned on by defining the directive ErrorLog as shown below:

```
ErrorLog syslog or
ErrorLog syslog:local7
```

Error log files can also be rotated using the rotatelogs command. The use of rotatelogs is described later in 5.7.4, “Rotating the Server Logs” on page 103.

5.7.3 Customizing the Log Format

The IBM HTTP Server provides powerful tools to customize the log format to suit your purposes. You can even log events to multiple log files. If you are not afraid of the performance of your server, this feature can be used to build, for example, troubleshooting logs and security logs.

While the TransferLog directive accepts only a file name for logging transactions, the CustomLog directive also accepts formatting information or a nickname for predefined formats as shown below:

```
TransferLog   /var/log/http/access_log
CustomLog   /var/log/http/custom_log "%h %l %u %t \"%r\" %s %b"
```

These two directives would create two log files that have the same content. You can also define log formats by using the LogFormat directive. Let's go a little further with log files:

```
LogFormat "%h %l %u %t \"%r\" %s %b" CLF
LogFormat "%h %l %u %t \"%r\" %s %b \"%{Referer}i \"%{User-agent}i\"" ECLF
LogFormat "%t \"%404{Referer}i\" \"%r\" 404_requests
LogFormat "%!200,304,302h %u %t \"%r\" %s" Failed_requests

CustomLog   /var/log/http/access_log CLF
CustomLog   /var/log/http/404_log 404_requests
CustomLog   /var/log/http/problems_log Failed_requests
```

We have defined a couple of log formats that are used in the CustomLog directives. We have replaced the TransferLog directive with the CustomLog directive that was shown earlier. We made a special log file for logging all the 404 (Not Found) errors. The log format can be defined so that it will produce an event only for requests that end up with a certain status value. This condition is defined in the 404_requests LogFormat by setting the status code just after the percentage character in the %{Referer}i argument.

The example below is a clip of the 404_log file defined in the previous example. That reveals that on the HTML page new_prod.html is a broken link. The second row is probably the same client who has now tried to reload the missing document, since it has not the referrer field.

```
[14/Dec/1998:16:50:39 -0600] "http://www.CompanyA.com/new_prod.html" "GET
/Docs/Serv/hot/33823_rel_notes.html" HTTP/1.0"
[14/Dec/1998:16:50:42 -0600] "-" "GET /Docs/Serv/hot/33823_rel_notes.html"
HTTP/1.0"
```

The last log definition example shown above logs all the events that have not ended successfully. This is done by excluding the successful status codes in %h argument. The exclusion is done by adding a exclamation mark (!) before the status code or the list of status codes. The status codes can be added as a condition statement into any % argument in the syntax. Here are the arguments that can be used in log format definitions:

Table 10. Custom Log Format Arguments

Argument	Definition
b	The size of the content data in response
f	Requested filename
{ENV}e	Value of environment variable ENV
h	The IP address of the client (or hostname, if DNS lookups are used)

Argument	Definition
a	The IP address of the client
{REQ_HEAD}i	The value of content of header line REQ_HEAD in the request
l	Remote logname (from ident)
{FOOBAR}n	The contents of note "Foobar" from another module
{REPL_HEAD}o	The value of content of header line REPL_HEAD in the response
p	The port number that was used to make a connection to the server
P	The process ID of the server process that served the request
r	The request
s	Status code of the request
t	Time in common log format time format
{TIMEFMT}t	Time in customized time format. Time format must follow the rules that are specified for `strftime` system command.
T	The time that it took to serve the request
u	Remote user name that the client has given the server
U	The URL from the request
v	Server name that was used to connect the server

There is a number of free log analyzing tools available on the Internet that you might find helpful, for example *http-analyze* (`http://freshmeat.net`).

5.7.4 Rotating the Server Logs

The IBM HTTP Server does not change or archive the log files; it uses the same log file as long as the filesystem has free space in it. In order to copy the log files to another storage media, you have to use a special application for switching the log files without shutting the server down. The log files are opened for append when the Web server is started.

Because the IBM HTTP Server provides the possibility to redirect the logging to a UNIX pipe file, it is possible to use an external application to do the log file change. The *rotatelogs* application that ships with the IBM HTTP Server can be found in the directory /usr/lpp/HTTPServer/sbin. It takes two arguments; a log file name and a timer value that counts the time in seconds after the log file switch should be done. The rotatelogs creates the log file and adds a system time when the server is started as an extension to the log file name. It creates

a new log file for the next entry that occurs after the expiry of the timer, such that there are no empty log files created during times when there is no activity in the Web server.

The rotatelogs program can be used with the directives ErrorLog, TransferLog and CustomLog:

```
CustomLog "|rotatelogs /path/to/logs/access_log 86400" ECLF
```

The timer value represents seconds, so the 86400 value represent 24 hours.

The piping of the log messages to a program opens up a lot of possibilities to implement auditing and accounting systems. Bear in mind, however, that such a program is executed with the user ID that is used when starting the main server, which in most cases is the root user. All the programs that are executed with root user ID should be designed, programmed, checked and double-checked with extreme caution in order to not introduce security holes into your system.

In case you need to change the server logs manually, for instance, because you are running out of free disk space in a file system, you can do this by copying the log file to another location (or to tape archive) and then emptying the file without recreating it, as shown below:

```
# cp /usr/lpp/HTTPServer/var/log/access_log /archive/access_log.121298
# cat /dev/null > /usr/lpp/HTTPServer/var/log/access_log
```

Note: Although this procedure frees the allocated space on disk for the particular log file, the `ls` command may still report the file to be large. To avoid such confusion, this procedure should not normally be used in daily operations.

5.8 Auditing

Logs are quite useless, unless they are looked after every now and then or saved for future examination, in case it becomes necessary. The Web server logs reveal, for instance, disfunctionalities in Web sites, such as broken links and certificate expiries. Logs can be used to work around the problem with the lack of control in Web server authentication. In Web authentication there is usually no possibility to protect the system from brute force password cracking. Since the HTTP communication is stateless, it is difficult to detect that two failed authentication failures have anything in common with each other.

When it comes to security, attacks against authentication can be easily noticed, but attacks against CGI programs and other Web applications are very difficult to notice. Keep track on the executable CGI-programs that are in your system and use only programs that are verified to handle all the parameters correctly and are considered to be secure. Security of CGI programs is discussed in more detail in Section 10.2.4, “CGI Security” on page 218.

In the log file, unauthenticated requests to protected documents produce an event, where the status code is 401 and the user name is empty. If the user does not pass the authentication, the log file entry includes the user name he has tried. The failed “logins” can be found from the log file by searching for entries that have status code 401 and whose username field is not empty.

To provide some level of security, you could define a maximum number of failed access attempts per user within a certain time frame. For example, a relatively simple shell script program that searches the failed access attempts for each user could disable such users that exceed these limits.

Let’s say that you tolerate only two failed access attempts per day and you have collected all the 401 events to a log file. The log file could look something like this:

```
[14/Dec/1998:08:47:16 -0600] 1.2.3.6 -
[14/Dec/1998:08:47:17 -0600] 1.2.3.6 goofy
[14/Dec/1998:09:01:13 -0600] 123.3.3.12 -
[14/Dec/1998:09:01:19 -0600] 123.3.3.12 dale
[14/Dec/1998:09:30:03 -0600] 1.2.3.6 -
[14/Dec/1998:09:44:16 -0600] 1.2.3.6 -
[14/Dec/1998:11:02:32 -0600] 1.2.3.6 goofy
[14/Dec/1998:13:40:22 -0600] 127.0.0.1 -
[14/Dec/1998:13:47:16 -0600] 127.0.0.1 -
[14/Dec/1998:13:47:24 -0600] 127.0.0.1 goody
[14/Dec/1998:15:47:44 -0600] 1.2.3.6 -
[14/Dec/1998:15:47:49 -0600] 1.2.3.6 ihsadm
[14/Dec/1998:17:34:44 -0600] 1.2.3.6 -
[14/Dec/1998:22:34:52 -0600] 209.3.244.2 -
[14/Dec/1998:22:34:55 -0600] 209.3.244.2 ihsadm
[14/Dec/1998:22:34:59 -0600] 209.3.244.2 ihsadm
[14/Dec/1998:22:35:07 -0600] 209.3.244.2 ihsadm
[14/Dec/1998:22:35:10 -0600] 209.3.244.2 ihsadm
[14/Dec/1998:22:35:14 -0600] 209.3.244.2 ihsadm
```

From this kind of log, you could easily notice that someone has tried to guess the password of `ihsadm`. A script could detect that the failed authentication count is exceeded for this user.

5.9 Other Features

The extensibility of the IBM HTTP Server through modules is almost endless. Many modules have been developed besides the ones that are available from the Apache Group. This section mentions just a few that might be useful in certain environments.

5.9.1 Fixing Typos in URLs

The module mod_speling, included with the IBM HTTP Server, allows you to automatically correct minor errors in URLs, such as single character omissions or insertions, transposition or single wrong characters. It also enables case insensitive URLs. The latter feature is especially useful when HTML documents are prepared on case insensitive operating systems like Windows.

Note: The author(s) of *mod_speling* actually pronounced the need for such a module right by misspelling its name; the module is correctly spelled mod_speling, not mod_spelling.

For example, assume there is a file support.html. All the following requests would cause the server to correctly access and return that file:

- suport.html (one character is missing)
- suppoprt.html (one character is added)
- suppotr.html (order of two adjacent characters are changed)
- sypport.html (one character is wrong)
- SupporT.HtmL (wrong case)

The main drawback of this URLs fixing feature is its impact on the server's performance because the file system needs to be scanned on each request. If two or more similar files are found, the server returns a list of these files by asking the user for a selection. That can expose unwanted file names and can be treated as security flaw.

The URL correction feature can be enabled using the directive CheckSpelling in the server configuration file. The required module (mod_speling) is enabled by default with a corresponding LoadModule and AddModule directive.

5.9.2 Caching Proxy Function

The caching proxy feature is not included in the current distribution of the IBM HTTP Server. Should the caching proxy functions be necessary in your environment, the module mod_proxy can be acquired from the standard

Apache server distribution (http://www.apache.org) and compiled for the IBM HTTP Server. For information about how to add modules to the IBM HTTP Server, refer to Chapter 8, “Building HTTP Server Modules” on page 177.

The caching proxy module mod_proxy performs two basic functions:

- It allows internal users to access the Internet and at the same time as restricting access from Internet to internal network (firewall function).
- It improves performance and reduces network load by caching retrieved information on local disk.

The proxy function is defined in the HTTP protocol specification. A proxy Web server acts as server and client at the same time. It receives proxy requests from a client (browser) and forwards these requests to a destination server as if it was a client itself. The answer received is then returned back to the client. Since all information passes through the proxy server, it can do logging and caching.

The required mod_proxy is not included with the IBM HTTP Server (see 3.1, “Product Contents” on page 33). Please read Chapter 8, “Building HTTP Server Modules” on page 177 to learn how such modules can be included.

The caching proxy configuration consists of a server configuration part (proxy function and cache parameters) and a client configuration part (proxy server address and port).

After you have compiled mod_proxy as a DSO library module, copy it to the /usr/lpp/HTTPServer/libexec directory and add the following directives to corresponding parts of the configuration file:

```
LoadModule proxy_module libexec/mod_proxy.so
...
AddModule mod_proxy.c
...
ProxyRequests On
```

The caching function can be enabled by the following directives:

```
CacheRoot /www/cache
CacheSize 10000
CacheGcInterval 1.5
```

CacheRoot defines a directory in which retrieved information will be cached and CacheSize specifies the desired total cache size in kilobytes. Though actual cache size can exceed this value, a garbage collection task tries to keep it below the defined limit. The directive CacheGcInterval specifies the

frequency of the garbage collection process in hours (for example, 1.5 means 90 minutes).

The proxy server access can be protected by the same configuration as any other Web server resource (see 6.2, “Basic Authentication” on page 118). In that case, users must provide authentication information (usually name and password) to get access to the proxy function.

For further information about server configuration refer to original mod_proxy documentation at: `http://www.apache.org/docs/mod/mod_proxy.html`.

For a proxy client configuration example, we use Netscape Communicator. Select the following sequence of menus to get to the configuration dialog as shown in Figure 18 on page 109: **Edit** -> **Preferences...** -> **Advanced** -> **Proxies** -> **Manual proxy configuration** -> **View...**. On this dialog, fill in the HTTP proxy server name and address fields according to the server’s configuration.

A Replacement for a Proxy Server?

While the mod_proxy module basically represents the functions of a proxy server, it is not intended to be used as a full-fledged proxy server. Specialized products, such as the IBM Web Traffic Express, are certainly better suited for large-scale proxy server exploitation.

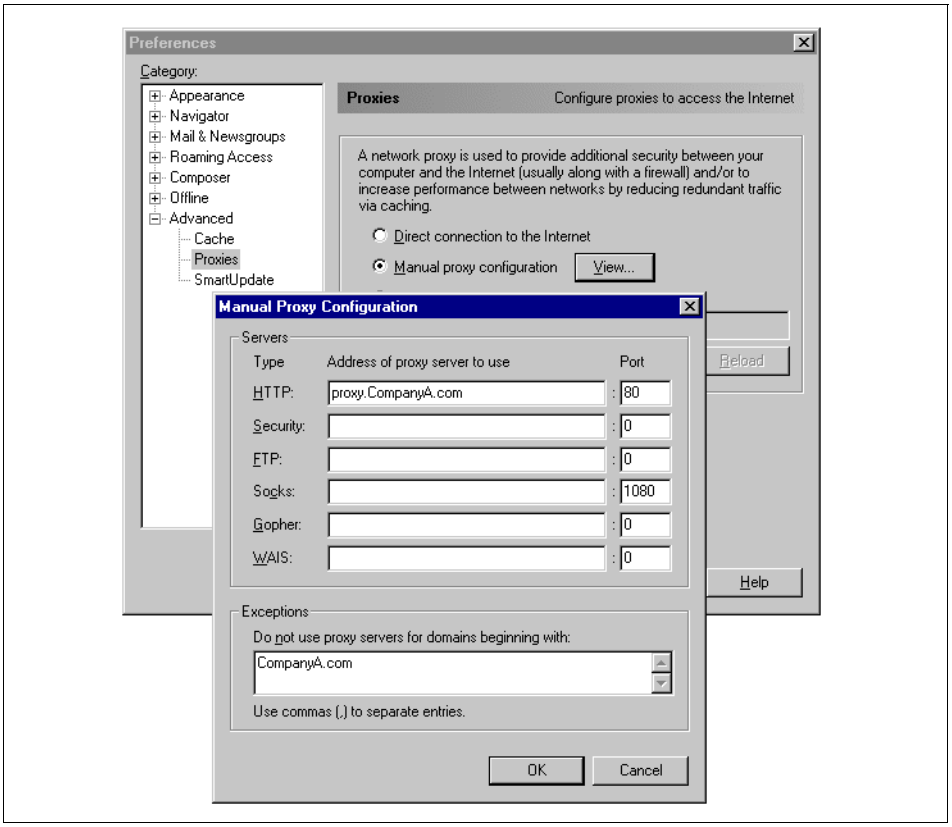


Figure 18. Netscape Communicator Proxy Setup

If a browser is set up this way (see Figure 18), it sends its requests to the proxy server at the specified address and port, rather than to the Web server that might be running on the same machine and behind the same port. The proxy server will then forward that request to other Web servers, including the one that might be running on the same machine and port. Note that, as shown in Figure 18, browsers also allow you to exclude certain domains from using a proxy.

Chapter 6. Deploying Security

Because of the rapid growth of the Internet, security has become one of the most essential issues in network communication. Commercial transactions have taken place on the Internet and organizations' internal intranets. Electronic commerce evolves so rapidly that it is easy to give reasons for paying attention to security; lapses in security have a price-tag.

The common believe of the Internet as a paradise for scoundrels and villains who try to swindle your money might be a little overstated, but a network has its pitfalls. When the Internet started to expand, security was not one of the main concerns. Everyone was so enthusiastic to invent new ways to utilize this exciting new, nonprofit network, that the concern about security was left aside. Quite often, only simple ways of hiding information from unauthorized users were chosen.

The Internet protocol is founded on trust that every party obeys some common rules. It is easy for a hacker, not behaving according to these rules, to use that trust against the principles of the Internet itself. Today the Internet and the many intranets cover all the aspects of community, from personal communication to government processes, and from research to business. It is unfortunate that this kind of opportunities also attracts a few individuals with something other than honesty in mind.

After introducing some basic elements of security, this chapter will introduce the important concepts of the security-related features and the configuration of the security functions of the IBM HTTP Server.

6.1 Basic Elements of Security

Security is an abstract issue that goes hand-in-hand with risk. Minimizing the risk goes along with costs in terms of time and money. One of the most important things to do for security is to define the level of risk you are prepared to take. Finding the balance between the risk and security is often difficult—no one knows what kind of threats there are. Security is not a one-time thing; keeping the security standard that was once defined and introduced requires regular checking of log files and continuous tracking of new challenges, hacking methods and threats.

The ISO 7498-2, the International Organization for Standardization (ISO), defined the common security services found in modern IT systems. The list consisted of authentication, access control, data integrity, data confidentiality and non-repudiation. The list was updated later in the ISO 10181 with

availability, security audit and key management. Some of these important terms are described briefly below:

Authentication	Authentication is the process of verifying the validity of a claimed identity. The ideal authentication would be identification.
Access Control	Assurance that each user or computer that uses the service is permitted to do what he or she asks for. The term *authorization* is often used as a synonym for access control, but it also includes granting the access or right to perform some actions based on access rights.
Data Confidentiality	Sensitive information must not be revealed to parties that it was not meant for. Data confidentiality is often also referred to as privacy.
Data Integrity	Data integrity assures that the data is not altered or destroyed in an unauthorized manner.
Non-Repudiation	Assurance that a sender cannot deny being the source of a message, and that the recipient cannot deny the receipt of a message.
Availability	The availability (over time or in terms of response time) of a system. While this is normally not a security-related term, it may become so when dealing with denial of service attacks that may intentionally slow down or completely paralyze a system.
Security Audit	A process where an independent party checks the security level of an organization or computer system.
Key Management	Key management deals with the secure generation, distribution, and storage of keys used in cryptography.

As far as Web servers are concerned, all of these security areas may be involved.

6.1.1 Physical Security

Physical security is the base for any viable security implementation. If someone wants to steal your information, it could be easy to take a whole machine and dig the information out of it on a kitchen table. The physical access to servers should therefore be controlled and logged. As another precaution, it is a good habit to always log off from of a computer's console when it is not used. UNIX passwords protect the running system fairly well.

Network connection to the Internet should be provided by a trusted Internet Service Provider (ISP). Web servers that are accessible from the Internet should be protected against unlimited access. This is usually done by protecting the connection between the ISP and the Web server with a firewall or a screening router. A screening router is basically a router that has some screening rules defined for the purpose of protecting the systems that are behind it. Almost every modern router has the option of defining some rules, for example denying all traffic that originates from the untrusted side and try to make connection to unspecified ports on the secure side. A firewall has basically the same function, but compared to a screening router, a firewall has better logging options for traffic.

Depending on the importance of the Web server, some precautions should be taken. It is advisable to turn off any unnecessary services and keep the system as simple as possible in terms of additional applications. It is easier to keep track of the current users, file permissions and machine behavior if you have dedicated your system to serve just as a Web server. One of the basic steps in implementing security is the classification of information and users. It is advisable to have some information about every account in the system and a clear view of what those accounts are allowed to do and when they do it. In systems that are used by many people, it is crucial to not have multiple users who use the same user accounts. The user accounts should be reviewed periodically against expired accounts and password rules should exist and be enforced that do not allow trivial passwords. In public domain software there is a good selection of different kinds of password cracking tools that can serve positively as applications for security maintenance and audit purposes.

Physical security must always be part of a complete security system that incorporates all the preconditions and procedures that are involved. This should be documented in a security policy paper. Such a document should also contain clear instructions about how to handle situations when security is compromised.

6.1.2 Logical Security

Logical security consists of authentication, authorization and encryption. Logical security only makes sense when a certain level of physical security is implemented. For example, biometric authentication, such as a fingerprint reader, can easily be spoofed if the sensor is not physically secured and its output is not encrypted. Passwords can be eavesdropped in a company's intranet or somebody may have left an authenticated communication session open and left the desk for a cup of coffee. Since the issue also concerns end-users, it is important to design security procedures in a user-friendly way.

6.1.2.1 Authentication and Identification

Authentication is usually done by challenging something you know, something you have, or something you are. Asking for a user ID and password pair is a typical example of the first category. Some physical device, such as smartcards, count for the second category. Also, certificates fit into this category. The third, something you are, is considered to be the most accurate way to authenticate a person, but it is also the most difficult to implement. There are devices which can measure human biometrics, such as fingerprints or eye retinas.

What About Non-Human Authentication?

It has been said in the paragraph above that biometrics are the best authentication methods. This is only true as long as the biometric sensors are reliable enough and the “object” to be authenticated is a human being.

If a computer, a client, or an application needs to be authenticated, then there is obviously no way to do biometric authentication. In those cases, the second method as described above, using certificates, is the most common way used for reliable authentication.

The simplest (and most common) way to authenticate a user is to ask for a user ID and a password. This method is good as long as the passwords are not too easy and are either transferred over the network securely or not at all. The simple passwords are easily broken by using *brute-force method*. In most operating systems it is possible to define some rules for passwords. If such rules for password are too strict, users may have problems remembering them, creating another exposure since users have to write them down.

Using brute-force methods means that the password is cracked by guessing and testing possible passwords, or using dictionaries of most often used passwords. Using the brute-force attack against the operating system might not be possible because most operating systems can limit the number of invalid logins. User authentication in Web servers usually do not have limitations of this kind.

In Web servers, authentication can be based on the client’s IP address, subnet or the domain name. However, the IP address and domain name are not fully trustworthy. A deterministic hacker can quite easily spoof his IP address to be something else. It is possible to overcome these problems by defining secure interfaces and implement anti-spoofing in firewalls and routers. However, checking the IP addresses or domain names might not keep hackers out of your system, but it can keep the honest person honest.

The IBM HTTP Server provides an option to double-check IP addresses using DNS service (HostnameLookups Double), but turning on this option may have an effect on Web server performance.

Except when SSL connections are being used (see 6.4, “Secure Sockets Layer, SSL” on page 129), the HTTP authentication passwords are transferred over the network without decent protection. The Base64-encoded user ID and password can be decoded in no time.

Today there are also more sophisticated methods to authenticate a user. One of them is the unforgeable SSL client certificates. Client certificates are described later in this chapter in 6.5, “SSL Client Authentication” on page 146.

Digital certificates are a good way to prove the identity of a person or another object. Certificates are digital documents that contain the information about the person or object that owns the certificate. A certificate also contains a public key that can be used for encrypting or decrypting messages. The basic idea behind a certificate is that a trusted authority, called a Certificate Authority (CA), has once identified you and proves that you are who you claim to be. The public key contained in such a certificate can be used to encrypt information that is only intended for the certificate owner because the certificate owner is in possession of the secret private-key that is required to decrypt the messages encrypted with his public-key. The certificate and the public key are freely-distributed information. The certificate also contains the digital signature of the signer Certificate Authority, which protects the certificate from tampering.

Because the HTTP is a stateless protocol, it makes authentication schemes difficult. Neither the client nor the server keep track of the status of a current connection. Because of this, the protocol does not support a status like “logged on” or “logged off”. If required, some additional logic must be added, for example by using server-side programs that keep track of such connections and states. In addition, cookies in Web browsers can be used to maintain a certain status.

6.1.2.2 Authorization

Access control takes place after authentication. Authorization grants or denies the user accesses or rights to perform some operation based on the user’s identity. In UNIX systems, for example, the file access control is based on the user ID and access control attributes attached to each directory and file.

In a Web server environment, the authorization is the Web server’s responsibility. The underlying file system restrictions should normally not

restrict the access to files within the *scope* of the server. When the Web server cannot access a file that is in its scope, it returns an error message (such as 403 Forbidden). The Web server serves files from its scope, according to scoping rules set forth in the server's configuration file, normally the httpd.conf file. It is highly recommended that a Web administrator has a logical sketch of that scope, such as the example shown in Figure 19. With such a drawing, it is easy to define access permissions on directory and subdirectory levels that must then be defined with corresponding directives in the server's configuration file. The drawing does not need to be too detailed. The scope of a Web server should be limited such that it contains only files that clients are allowed to request. Security or configuration related files should never be stored within a Web server's scope. Trusting that no one will ever come across some files in a filesystem is foolish (that's called security by obscurity).

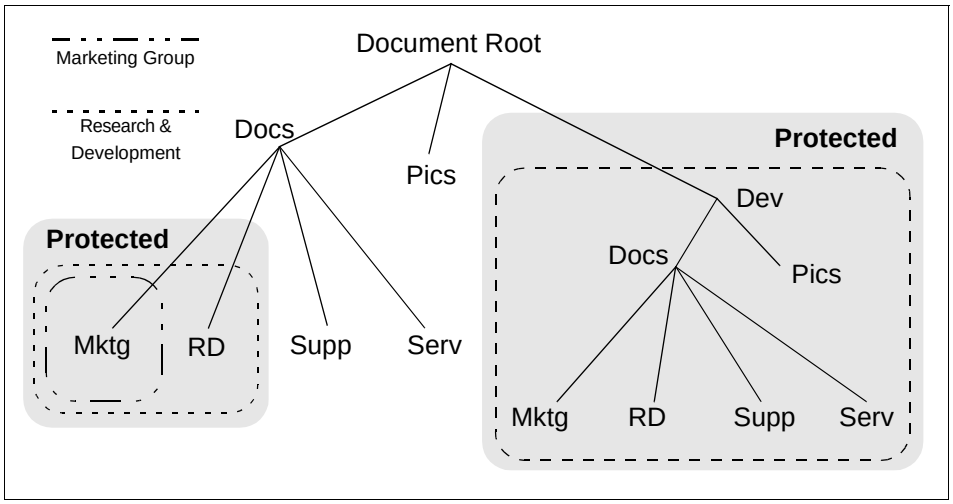


Figure 19. Logical Structure of a Web Site (Example)

You should be careful with symbolic links, too, because those can unintentionally enlarge the Web server's scope. In the server configuration file you can define whether the Web server should follow symbolic links or not. It is a good idea to not allow this. The Directory options to look for are `FollowSymLinks` and `SymLinksIfOwnerMatch`.

The use of the `UserDir` directive can also be a security flaw. If users are allowed to publish their own Web pages off their home directories by setting the `UserDir` to something like "./", accessing `http://www.CompanyA.com/~root/` could expose unwanted files to the users.

6.1.2.3 Encryption and Data Integrity

Cryptography provides mechanisms for transforming plain text to ciphertext by encryption and, vice versa, ciphertext to plain text by decryption. The modern key-based cryptography uses keys as a seed for the algorithm. The key is a binary string. There are two kinds of key-based cryptography; symmetric and asymmetric cryptography. The latter is also known as *public-key* cryptography. The main difference is that in symmetric cryptography the encryption and decryption are done with the same key, while public-key cryptography uses a key pair for encryption and decryption. A message encrypted with one of the two keys can only be decrypted with the other key. The following example clarifies the difference, using two well-known characters often referred to in the security literature, Alice and Bob.

Alice would like to send confidential e-mail to her colleague Bob. If they used symmetric cryptography, they would have to agree on a common key they would use for encryption and decryption. A problem that arises is that the key cannot be simply exchanged in e-mail messages because of the lack of security in e-mail. Because of this, Alice and Bob would have to find a way to securely exchange their key, which might not be easy. Encryption and decryption would then be done with that single key used on both ends of the conversation. If either Alice or Bob wanted to securely communicate with other parties as well, a separate secret key would have to be maintained and exchanged for each communication partner.

If Alice and Bob used public-key cryptography instead, Alice could have asked Bob to send his public key to her first. In fact, because it is not a secret key, Bob could also furnish his public key by any other means, such as through his Web site. Alice could then encrypt her message using Bob's public key and send the encrypted message to him. Only Bob would be able to decrypt the message because only he is in possession of the private key that must be used to decrypt that message. Not even Alice would be able to decrypt the message that she encrypted.

Now a new problem arises: How does Alice know for sure that the public key she is using *is* Bob's public key and she is indeed talking to Bob? One way to verify the integrity of data in such occasions would be that Bob runs the key through some hash function, and then includes the resulting hash (also known as fingerprint, see below) with the message. Alice can then run the same hash function and compare the fingerprints. If the fingerprints match, the key is not corrupt. But this still does not guarantee that the sender was Bob; it could have been anybody pretending to be Bob, using a fake private-public key pair. The way to solve this is by involving a trusted third

party that certifies Bob's identity by issuing a certificate to Bob. Bob can then use this (his) certificate and send it to Alice, along with a message and a fingerprint. Only the use of such a certificate allows Alice to be assured that Bob was the sender of the message. (Certificates actually involve some more steps that are not discussed here.)

Hash functions, as mentioned above, are complex mathematical functions that reduce the length of any arbitrary length message to a fixed length fingerprint. The fingerprint is very unlikely to be the same for different or altered messages and it is considered impossible to construct another message that yields the same fingerprint. Also, it is impossible to reverse engineer the original message from a fingerprint. The most often used hash functions are MD5 and SHA-1.

The above was a very brief and simplified description of how public-key cryptography works and what certificates are. There will be more details in the sections that follow.

The IBM HTTP Server implements these methods in the SSL module and in the authentication methods, as explained later in this chapter.

6.1.3 Authentication Schemes Supported by the IBM HTTP Server

The authentication schemes that are available with the IBM HTTP Server are *basic authentication* and *digest authentication*. Because of the modular structure of the IBM HTTP Server (as well as the Apache Web server) and the authentication API, there is a variety of authentication modules available for different use. In the two sections that follow, basic authentication and digest authentication are discussed, along with some issues related to them.

The IBM HTTP Server also supports the Secure Sockets Layer (SSL) protocol used for secure connections between browsers and servers. This should not be confused with authentication; SSL only assures a secure connection that prevents any attacker from spoofing the network. The SSL module and the SSL client certificates will be discussed later in 6.4, “Secure Sockets Layer, SSL” on page 129.

6.2 Basic Authentication

The basic authentication is defined in the HTTP/1.0 protocol and is implemented in most (if not all) of the widely used Web browsers and servers. Basic authentication implies that the user is authenticated according to his identity in the network (IP address, domain name, subnet of IP address) or the user ID and password he or she submits. The authentication can also be

based on a combination of these elements. However, as mentioned earlier, basic authentication based on network parameters such as IP addresses is not very secure since it is relatively easy to falsify.

The basic authentication configuration can be defined statically in the server configuration file(s) that the server reads when it is started, or, more dynamically, in the *.htaccess* file(s). If the .htaccess file is used, the server reads it every time it accesses the directory, and thus, changes will automatically be picked up the next time that directory is accessed. The configuration elements are the same on both methods (see 6.2.2, “Using the .htaccess File” on page 123 for more details about the .htaccess file). User IDs and passwords, if used, can be stored in either a text file very similar to the /etc/passwd file on UNIX systems, or in an indexed database file. The use of database files, rather than textual flat files, positively affects server performance as the number of user accounts increases. More about authentication files can be found in 6.2.3, “Authentication Files and Databases” on page 124.

6.2.1 Setting Up Basic Authentication

Basic authentication is implemented in the *mod_auth* module which is configured in the configuration file by default. It provides basic user authentication using simple textual user ID/password and group files.

Security-related directives in the server configuration file(s) may occur in the outermost level or within any <VirtualHost>, <Directory> or <Location> containers. The outermost definitions apply to all the resources in the server that are not specifically redefined by further container definitions.

The following example shows a `<Directory>` container with a minimal set of security-related directives for the directory structure that was described earlier in the Figure 19 on page 116:

```
<Directory /usr/lpp/HTTPServer/share/htdocs/Docs/Mktg>
   AuthType Basic
   AuthName "Protected Material"
   AuthUserFile /usr/lpp/HTTPServer/security/users
   AuthGroupFile /usr/lpp/HTTPServer/security/groups
   Require group mktg_grp rd_group
   ...
</Directory>
```

In the configuration file clip above, the location /Docs/Mktg is protected using basic authentication such that it can be accessed only by people who are members of the mktg_grp or rd_group groups. The members of these groups

are listed in the /usr/lpp/HTTPServer/security/groups file, as specified by the AuthGroupFile directive. The file specified with the AuthUserFile directive contains the authentication data (the user IDs and the encrypted passwords) for the users. These authentication files are described in 6.2.3, “Authentication Files and Databases” on page 124. The AuthType directive tells the server what authentication protocol is to be used for that resource. In order to have the Require directives working, you have to define at least the AuthName (*realm*) for the protected region within the Web server’s scope. The text specified in the AuthName directive will be presented to the user on the logon pop-up dialog upon first access to that directory, as shown in Figure 20.

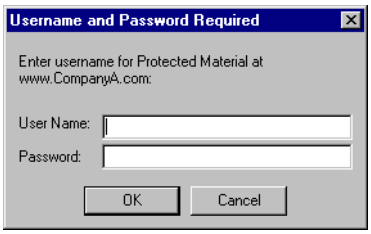


Figure 20. Authentication Pop-Up Dialog

The Require directive defines which users and groups are to be granted access the area. The Require directive also accepts the value “valid-user” that tells the server to grant access to anyone who is listed in the defined password file.

Alternative Basic Authentication Methods

Besides the mod_auth module, there is another modules that works in a very similar manner. It is the module *mod_auth_dbm*, which provides the possibility to use database-type files as a repository for the user IDs and passwords rather than plain text files as the mod_auth module does. The description given here about the configuration of the mod_auth module equally applies to the mod_auth_dbm module, except that the file name directives are AuthDBMUserFile and AuthDBMGroupFile, rather than AuthUserFile and AuthGroupFile, respectively. The IBM HTTP Server supports the use of multiple authentication modules simultaneously.

The following directory container protects the /Docs/RD directory:

```
<Directory /usr/lpp/HTTPServer/share/htdocs/Docs/RD>
   AuthType Basic
```

```
   AuthName "Protected Material"
   AuthUserFile /usr/lpp/HTTPServer/security/users
   AuthGroupFile /usr/lpp/HTTPServer/security/groups
   Require group rd_group
   Order allow,deny
   Deny from All
   Allow from 1.2.3.4
   Satisfy All
   ...
</Directory>
```

You may have noticed that the realm name is defined to be the same as in the previous example (“Protected Material”). This is done in this example because if the realm names were different and a user from the rd_group wanted to move from the /Docs/Mktg location to /Docs/RD, he/she would have to re-authenticate to the server. As long as a user accesses resources that have the same realm name, he or she is not required to re-enter the user ID and password.

The example above shows more configuration elements: The /Docs/RD directory contains information that must not be accessible by anybody except by members of the rd_group group. Now let’s assume that this Web server is located outside a firewall, accessible by anybody on the Internet and, through the firewall, from anybody on the company’s internal intranet. The “Allow from 1.2.3.4” directive only allows accesses from that single IP address, which, in our example, represents the company’s firewall. This way, although the Web server is located on the external Internet, the /Docs/RD directory is only accessible from members of the rd_group group who access the server through the company’s firewall from within the intranet. This kind of arrangement requires that anti-spoofing is enabled in the Internet router or firewall. Anti-spoofing prevents machines from the Internet from impersonating the firewall machine’s IP address (1.2.3.4 in the example above).

Security Definitions within <Limit> Containers

Security directives can also be specified for specific HTTP methods (such as PUT, GET, POST), although it is not recommended. The definitions for methods is done with the <Limit> container. It is easy to leave some method undefined by accident because there is no wildcard-type method name for all the other methods that are not specifically defined. If some limitations are defined, all methods should be remembered to be defined as well. (Note: The method levels and the Limit container are not covered any further in this book.)

When the Allow and Deny directives are used, the sequence in which they appear in the container is important. In the example above, the Allow directive is processed first, and after that the Deny from all clause defines the access rights to default to no access. If the Allow, Deny and Require directives are used within the same container, the Satisfy directive must be used as well. Satisfy All means that all the requirements must be met, that is the Allow, Deny and the Require. Satisfy Any would grant access to a user if any of the requirements was met. One important value for the Satisfy directive is *mutual-failure*. Mutual-failure authorizes the client's request if the client is listed in the Allow list at least once and it is not listed in the Deny list.

The following example restricts access even more:

```
<Directory /usr/lpp/HTTPServer/share/htdocs/Dev>
   AuthType Basic
   AuthName "Development Site"
   AuthUserFile /usr/lpp/HTTPServer/security/users
   AuthGroupFile /usr/lpp/HTTPServer/security/groups
   Require group rd_group
   Require user ihsadmin
   Order allow,deny
   Deny from all
   Allow from 1.2.3.4
   Satisfy All
   ...
</Directory>
```

The /Dev directory is protected such that only the users in the rd_group group and user ihsadmin are granted access. The realm name is now different ("Development Site"), so users cannot move from the protected directories underneath /Docs to /Dev without logging in again.

In the examples above, absolute path names were used for the AuthUserFile and AuthGroupFile files because using absolute path names lessens the chance of logical configuration errors. If relative path names were used in the Directory containers, they would have been relative to the directory specified by the ServerRoot directive, which represents a security exposure. The path names of security files like in the AuthGroupFile and AuthUserFile directives should therefore be absolute.

It is possible to also protect individual files with the <Files> container, but you should keep in mind to not store any confidential configuration files in the scope of Web server, even though it can be separately protected.

Allow and Deny Directives

The Allow and the Deny directives can have the following values (as described in the mod_access description in the Apache documentation):

ALL All hosts are granted access.

A (partial) domain-name
Hosts whose names match, or end in, this string are allowed access.

A full IP address
An IP address of a host allowed access.

A partial IP address
The first 1 to 3 bytes of an IP address, for subnet restriction.

A network/netmask pair
A network address (a.b.c.d), and a netmask (w.x.y.z), for more fine-grained subnet restriction (example: 10.1.0.0/255.255.0.0).

A network/nnn CIDR specification
Similar to the previous case, except the netmask consists of the nnn high-order bits (for example: 10.1.0.0/16 is the same as 10.1.0.0/255.255.0.0).

It is also possible to Allow or Deny access according to the existence or non-existence of an environment variable. This can be useful for the implementation of separate authentication methods. An example could be:

```
Deny from env=NO_ACCESS
```

6.2.2 Using the .htaccess File

All access control directives described in 6.2.1, “Setting Up Basic Authentication” on page 119 can also be defined in the .htaccess file. If .htaccess files are used, the scope of those directives is the directory where the files reside and the directories below them (unless there is another .htaccess file). The use of .htaccess files adds a dynamically configurable security component into the Web site. Changes done to any .htaccess file do not require the restart of the server or any other administrative action since the .htaccess is read and interpreted every time any resource is fetched from a directory (if configured to do so).

The use of .htaccess file is controlled by the AllowOverride directive (see also 4.7.2, “Restricting the Directives within .htaccess Files” on page 60). If the AllowOverride directive has not been defined within any part of Web server’s scope, it defaults to AllowOverride All, which is not advisable. In the

httpd.conf file that is shipped with the IBM HTTP Server, the AllowOverride None is included in every <Directory> container. If AllowOverride was not defined, the server would unnecessarily check for the .htaccess files. Thus, it is a good idea to define AllowOverride None even when no .htaccess files are used.

The .htaccess file is likely to contain sensitive information like user IDs and server names that are allowed to connect to the Web server. By definition, the .htaccess files are located in the Web server's scope, and may therefore be accessible from browsers. To protect all .htaccess files in the system, add the following <Files> container to the httpd.conf file at the top level.

```
<Files ~ "\.htaccess$">
   order deny,allow
   deny from all
</Files>
```

The use of .htaccess files allows you to define multiple things dynamically; almost any file and directory related directives (except the Directory container itself) can be defined in an .htaccess file. Before using the .htaccess files, be sure that you understand the AllowOverride directive (see also 4.7.2, "Restricting the Directives within .htaccess Files" on page 60).

It is also possible to change the name of the .htaccess file(s) by means of the AccessFileName directive in the httpd.conf file.

6.2.3 Authentication Files and Databases

User and group authentication information is stored in text files when basic authentication is used. The IBM HTTP Server ships with a utility called `htpasswd` that is located in /usr/lpp/HTTPServer/bin. The `htpasswd` utility can be used to maintain the authentication data in the AuthUserFile. The textual password files are often called flat files.

You can create the AuthUserFile authentication password file while adding the first user to it as shown below.

```
# htpasswd -c /usr/lpp/HTTPServer/security/users ihsadmin
Adding password for ihsadmin.
New password:
```

Now you have to define a password for user `ihsadmin`, which you are about to create into the /usr/lpp/HTTPServer/security/users file. After that, the program will ask you to retype the password for verification.

Note

The `-c` argument with the htpasswd program is used only when you create a new password file. If you use the `-c` argument later by accident to, for example, add a new user to your password file, the utility will overwrite your current file.

After adding the first user, the contents of the password file will look something like:

```
ihsadmin:6uRBipvs0Jc22
```

The password file is actually identical to a UNIX password file (and uses the same password encryption algorithm), except that only the first two fields (User ID, encrypted password) are used. Further fields as found in a UNIX password file are ignored. There is a temptation for administrators to simply use the UNIX password file rather than creating and maintaining a separate file for Web user authentication. This should never be done because the UNIX password file contains more information and also sensitive users, such as root.

The group file is a plain text file that can be crated and edited with a text editor. Each line in the group file consists of a group name followed by a colon and a space, and then the user IDs of the group members in a space-separated list. Here is an example that defines two groups:

```
mktg_grp: huey dewey louie
rd_group: dale chip
```

The textual password file is good for occasional use or if there is a small number of users. If the use is frequent or if it contains a large number of users, it might slow down the server. For large systems that use basic authentication extensively, the use of indexed database files as supported by the mod_auth_dbm module, is favorable.

The *dbmmanage* utility, a management tool for the DBM authentication database, is included with the IBM HTTP Server distribution package. It requires Perl to be installed in the /usr/local/bin directory. The dbmmanage utility requires the strict.pm and Fcntl.pm Perl modules, as well as the gzip utility. The dbmmanage utility can be found in the /usr/lpp/HTTPServer/bin directory.

The dbmmanage utility only requires a few changes to the protection containers used in the server configuration file(s): the AuthUserFile directives

need to be changed to AuthDBMUserFile and AuthGroupFile must be named AuthDBMGroupFile.

The usage of `dbmmanage` is best explained with an example. Creating an entry into the DBM file can be done with this command:

```
# dbmmanage /usr/lpp/HTTPServer/security/dbmuserfile.pag adduser ihsadmin
```

The first argument is the name of the DBM database file. If the DBM file does not exist, it will be created. The second argument is the operation that is to be performed and the third argument, in this case, is the user ID that is to be added. The command above inserts the user ID ihsadmin into DBM file /usr/lpp/HTTPServer/security/dbmuserfile.pag. The command will subsequently ask for the user's password. The operations that dbmmanage accepts are listed in Table 11.

Table 11. Operation Arguments for dbmmanage

Operation	Role
add	Adds a user to the DBM file. Requires a fourth argument that is the encrypted password for the user.
adduser	Adds a user to DBM file. Prompts for the password.
check	Asks for a password for the user and compares it to the password that is stored in the DBM file.
delete	Deletes a user from the DBM file.
import	Imports user ID and password pairs from a textual flat file into a DBM file. The user IDs and the encrypted passwords must be separated with a colon (:), as found in UNIX password files.
update	Updates the password of the user and checks that the user exists in the DBM file.
view	Shows the selected user ID and encrypted password that are defined in the DBM file. If no user ID argument is given, it lists all the user IDs in the database. Can be used to import the user IDs and passwords into a flat file.

More information about dbmmanage can be found in the man page that ships with the IBM HTTP Server (3.4, “Default File and Directory Structure” on page 37 for an explanation on how to use the man pages).

User authentication in the IBM HTTP Server can easily be combined with other authentication methods, such as Kerberos or the Distributed Computing Environment (DCE). There are many authentication modules available for the Apache Web server that can be used with the IBM HTTP Server as well.

6.2.4 Performance Impact of Basic Authentication

The exploitation of basic authentication may have an effect on performance. Since HTTP is a stateless protocol - that is, the Web server does not keep track of users and their authentication status, the browser needs to send the user ID and password each times it requests a document from the Web server.

When the browser requests a protected resource, the Web server returns a 401 error code (Unauthorized) and includes the *WWW-Authenticate header*. The WWW-Authenticate header tells the browser which authentication schemes the Web server supports and the name of the realm that contains the protected resource. If the user has not already entered a valid user ID and password for this Web server and realm, the password dialog box is presented (see Figure 20 on page 120). In subsequent requests, the browser can find the information from its internal cache or password file. The browser then requests the same document, but this time with an Authorization header that includes the user's ID and password. Thus, each single request actually creates to requests being sent to the Web server if basic authentication is involved, which also includes any images that might be included in a page.

Because of this, basic authentication has its impact on performance and should, therefore, only be used when necessary.

6.3 The HTTP/1.1 Digest Authentication

Digest authentication is an alternative authentication method to basic authentication. Digest authentication uses the MD5 hash functions to hash the passwords and other data. The one goal of the digest authentication method is to have secure authentication without complex algorithms that may be subject to U.S. export regulations.

Digest authentication is not widely used and, unfortunately, many Web browsers support it. Mosaic, for example, is one of the rare Web browsers that support digest authentication. It is, therefore, generally recommended not to be used unless there are specific reasons.

The basic idea of digest authentication is that the Web server does not store the user's password encrypted in its authentication files, but stores MD5 hashes of strings that contain user ID, password, and the authentication realm name.

When the user requests a protected resource from the Web server, the Web server returns the 401 (Unauthorized) rejection message, which includes the

authentication method digest that the server requires. The server also sends a string that is called *nonce*, which is generally a string that is constructed from the time of day and the IP address of the requester. If the browser does not find the user ID and password in its internal cache memory (or password file) it prompts the user through an authentication dialog box. The browser subsequently does not transmit the user ID and the password as with the basic authentication. Digest authentication works as follows:

1. The browser concatenates the user ID, the authentication realm name and the password, and computes an MD5 fingerprint of that string.
2. The browser concatenates the requested URL and the method for the request, and computes an MD5 fingerprint of that string.
3. In the third phase the browser concatenates the nonce string with the fingerprints from the first two steps. After running this string through the MD5 algorithm, the resulting fingerprint is sent to the Web server.

   This fingerprint is sent to the Web server in lieu of a user ID and password pair.
4. The Web server stores in its authentication file a fingerprint that was created in an equivalent process as explained in the first step above. This can be thought of as a user's password. It concatenates the URL the client requested and the request method to a string and computes the MD5 fingerprint for the string.
5. The Web server now concatenates the last fingerprint with the same nonce string it has sent to the browser and computes the MD5 fingerprint of that string.
6. Access is granted if the fingerprints computed by the browser (step 3) and the server (step 5) match.

Setting up digest authentication is fairly easy. You should make sure that the mod_digest module is configured in the server configuration file(s) with a LoadModule and an AddModule directive. Follow the instructions for basic authentication, but specify "Digest" as the authentication method as in the following example:

```
<Directory /usr/lpp/HTTPServer/share/htdocs/Confidential>
   AuthType Digest
   AuthName "Protected Material"
   AuthDigestFile /usr/lpp/HTTPServer/security/digestusers
   ...
</Directory>
```

The authentication file (`digestusers` in the example above) can be created and maintained using the *htdigest* utility that is shipped with the IBM HTTP Server

(located in /usr/lpp/HTTPServer/bin). The htdigest utility works similar to the htpasswd utility used for creating and maintaining a password file for basic authentication (see 6.2.3, “Authentication Files and Databases” on page 124). It requires an additional parameter which is the `realm` name in the following example:

```
# htdigest -c digestusers "Protected Material" ihsadmin
Adding password for ihsadmin in realm Protected Material.
New password:
```

The `-c` flag indicates that the authentication file is to be created. If it existed with this name, it will be overwritten. The htdigest utility then asks for the user’s password twice (for verification) before it adds the user to the file.

Authentication on Systems Using Virtual Hosts

When using authentication with virtual hosts, you should use separate authentication files and separate realm names, regardless of what authentication method is being used. The directives specific to the authentication method can be defined in the <VirtualHost> container.

6.4 Secure Sockets Layer, SSL

SSL (Secure Sockets Layer) is a widely-used way to implement transport layer security into TCP/IP communication. SSL was developed by Netscape Communications Corp. and it is implemented in most Web browsers. The current version of SSL is 3.0. Microsoft has also introduced a proposal for securing network transport layer. Microsoft’s PCT (Private Communication Technology) is not widely used and the only Web browser that supports it is Microsoft’s Internet Explorer. SSL and PCT are very similar and can even be used at the same time. But neither of these protocols are declared as an Internet standard. The IETF (Internet Engineering Task Force) has made an approach to combine the best aspects of both into a new standard called TLS, which stands for Transport Layer Security.

What is the IETF?

The Internet Engineering Task Force is a large, open international community of network designers, operators, vendors and researchers whose purpose is to coordinate the operation, management and evolution of the Internet and to resolve short- and mid-range protocol and architectural issues. The IETF Secretariat maintains the Internet Drafts and passes them to Internet Architecture Board (IAB) for final approval.

You can find more information about the IETF by visiting their Web site at `http://www.ietf.org`.

This book only covers the SSL protocol and the key management issues that are related to it because SSL has become a de facto standard in today's Internet and intranet applications.

6.4.1 Principles of SSL

SSL is a mixture of public-key and symmetric-key cryptography, as well as hashing. It has been constructed to provide an effective and secure way to implement transport-layer security. The main reason SSL incorporates both public-key and symmetric-key cryptography is performance. While public-key cryptography offers the most desirable functions, it is very processing-intensive. SSL only uses public-key cryptography for session initiation, and for data encryption it uses symmetric cryptography for better performance.

What SSL provides, is:

- Privacy
- Authentication
- Data integrity

The SSL protocol consists of two parts:

- *SSL Handshake Protocol*, which is used for the initial authentication and transfer of the encryption keys between the participants.
- *SSL Record Protocol*, which is used for the transfer of actual data using agreed upon sets of authentication and cipher methods.

SSL uses public-key cryptography in the handshake phase to transfer the servers public-key to the client. When the SSL connection is established, the data is encrypted with symmetric cipher using a session key that is derived from the server's public key by both parties, and some additional data. Data

integrity is validated by comparing the message digests and hash values that are sent along with the data.

SSL adds an additional layer between network protocols and the protocols that are used on the application level. It encapsulates TCP/IP socket so that, in theory, every application using TCP/IP could use SSL to secure the connections (Figure 21).

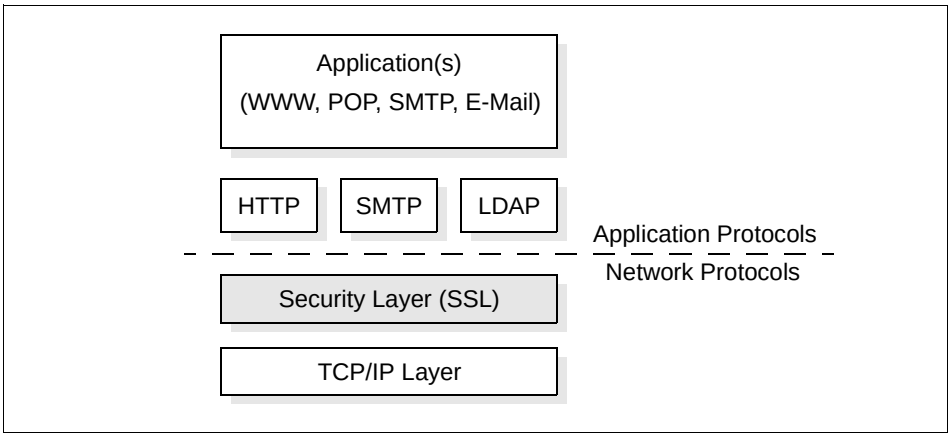


Figure 21. SSL in Relationship with Other Protocols

SSL is usually used for privacy (data encryption) and server authentication. SSL can optionally be used to authenticate a client by using client certificates. The client certificates are discussed later in this chapter in 6.5, “SSL Client Authentication” on page 146.

A Word about Authentication

In the first part of this chapter, the term authentication was used for user authentication only. SSL provides authentication as well, but primarily for servers and clients, rather than for users. If secured user authentication is required, both mechanisms need to be combined; basic authentication for users, using SSL for a secured connection between the client and the server. In addition to basic authentication, however, SSL can ensure a user that he or she is communicating with the server he/she wanted to.

6.4.2 Establishing the SSL Connection

A simplified explanation of the SSL handshake protocol is shown in Figure 22 on page 132. The numbered steps in Figure 22 are explained thereafter.

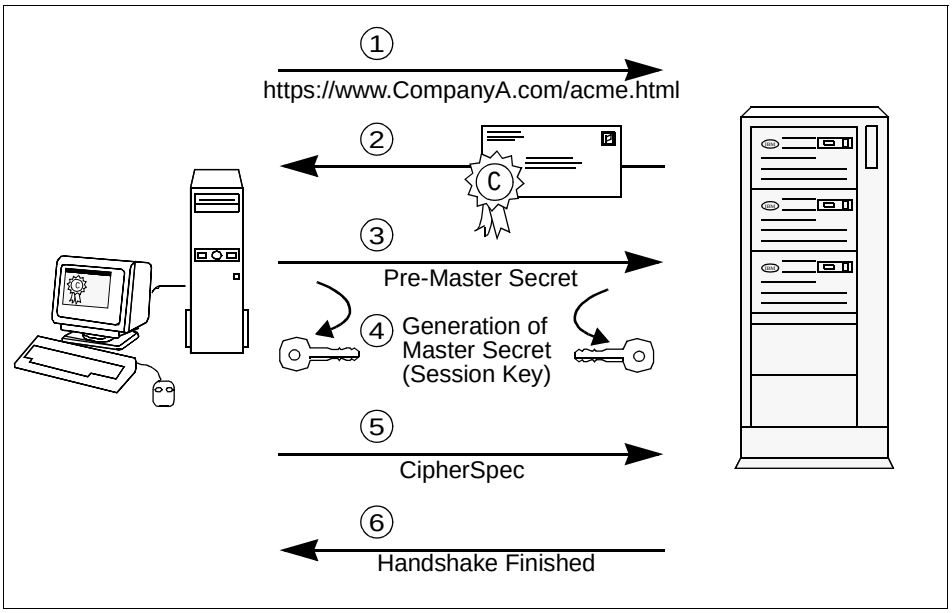


Figure 22. SSL Handshake

1. The client (in this case actually the Web browser running on the machine to the left in Figure 22) makes a request to get an SSL connection from the server www.CompanyA.com. The client includes in its request, among others, a session identifier, list of compression methods and encryption options that the client supports, as well as a random number that will be used later. Notice that the URL in the client's request starts with *https* rather than *http*, which is a request for a secure connection.
2. The server includes in its response the encryption options it supports and its random number. In addition, the server delivers its X.509 certificate to the client. The X.509 certificate includes the public key of the server. In this step, the server can optionally request the client to provide its client certificate for client authentication (see 6.5, "SSL Client Authentication" on page 146).
3. The client encrypts the random number it has sent in step 1 and the random number it has received from the server with the server's public key and sends this message (also called *pre-master secret*) to the server. The server decrypts it with its private key. If the decrypted numbers match the originals, it proves that the client must have received the server's X.509 certificate (with its public key) correctly.

4. Both the server and the client create a hash that will be used as a session key. This is also known as the master secret and is used later in the SSL record protocol as a key for symmetric encryption and decryption of data.
5. The client sends a message to the server to announce what cipher options it is going to use. SSL supports several options that need to be negotiated between the client and the server before actual encryption can take place.
6. The server confirms the successful handshake phase by sending the first message encrypted with the session key. From this time on, the cipher used is symmetrical and contains the message digest.

These steps result in a secured, encrypted connection between the client and server. Server authentication on the Web is done by the Web browser (client); if the server's certificate was not signed by a well-known Certificate Authority, the browser will alert and ask the user whether or not this server should be trusted (subject to the browser's individual configuration). Data integrity is guaranteed by using the *keyed message authentication codes, or MACs*. The keyed message authentication codes are hashes of messages that are calculated and included with every message during the SSL handshake process. MAC's are created with a secure hash functions, like MD5 or SHA-1.

Private key cryptography requires more computation than symmetric-key cryptography and in SSL it is used only in the handshake protocol negotiations. The session key for symmetric data encryption/decryption is calculated and shared between the server and the client.

6.4.3 Cipher Specifications Supported by the IBM HTTP Server

The cipher specifications that are supported by the IBM HTTP Server are listed In Table 12. Some of the ciphers are not available in the export versions of the IBM HTTP Server due to U.S. export regulations. The French version has even more limitations as compared to the common export version. The different versions of SSL-implementations are marked with a one letter identifier (U = U.S., E = Common export, and F = French).

Table 12. Cipher Specifications Supported by the IBM HTTP Server

Cipher Specification Number and Name	Cipher Used	U	E	F
27, SSL_DES_192_EDE3_CBC_WITH_MD5	Triple-DES (168 bit)	x		
21, SSL_RC4_128_WITH_MD5	RC4 (128 bit)	x		
23, SSL_RC2_CBC_128_CBC_WITH_MD5	RC2 (128 bit)	x		
26, SSL_DES_64_CBC_WITH_MD5	DES (56 bit)	x	x	
22, SSL_RC4_128_EXPORT40_WITH_MD5	RC4 (40 bit)	x	x	x

Cipher Specification Number and Name	Cipher Used	U	E	F
24, SSL_RC2_CBC_128_CBC_EXPORT40_WITH_MD5	RC2 (40 bit)	x	x	x
3A, SSL_RSA_WITH_3DES_EDE_CBC_SHA	Triple-DES SHA (168 bit)	x		
35, SSL_RSA_WITH_RC4_128_SHA	RC4 SHA (128 bit)	x		
34, SSL_RSA_WITH_RC4_128_MD5	RC4 MD5 (128 bit)	x		
39, SSL_RSA_WITH_DES_CBC_SHA	DES SHA (56 bit)	x	x	
33, SSL_RSA_EXPORT_WITH_RC4_40_MD5	RC4 MD5 (40 bit)	x	x	x
36, SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5	RC2 MD5 (40 bit)	x	x	x
32, SSL_RSA_WITH_NULL_SHA	-	x	x	x
31, SSL_RSA_WITH_NULL_MD5	-	x	x	x
30, SSL_NULL_WITH_NULL_NULL	-	x	x	x

There is no difference between defining the used cipher specification with the complete name as shown in Table 12 above, or with the corresponding number. When cipher specifications need to be defined in the server configuration file(s) either their specification numbers or their names may be used (see Table 12). If problems occur, you might want to change the LogLevel parameter to debug and search log files for messages about cipher loading failures. Cipher load may fail, for example, when the SSLCipherSpec name is misspelled or is not valid in a particular export version.

What Cipher Specs Are Supported in My Server?

If you have defined log level to be “info” or “debug”, the available cipher specifications are written to the error log when the SSL module is loaded.

The log level can be changed with the LogLevel directive and the valid values for it are debug, info, notice, warn, error, crit, alert and emerg.

6.4.4 The Alphabet Soup

Throughout the last sections and in the sections that follow, there are many terms introduced that you might not be familiar with. Here are some terms that are frequently used in the security literature:

PKCS10 Public Key Cryptography Standard #10 by RSA Data Security, Inc. A standard that describes a syntax for certification requests.

DES	DES (Data Encryption Standard) was developed by IBM in 1975. DES is a symmetric block cipher that uses 56 bit keys.
DES-EDE3-CBC	This cipher is also known as Triple DES. In Triple DES, each block of data is processed three times with at least two different keys. With Triple DES, the possibility to use brute-force to crack the message is minimized.
EDE3	Encryption - decryption - encryption with three keys. The message is processed three times. First encrypted, then decrypted and once more encrypted. Each step is performed with a different key. There is also a EDE method that uses two keys, in which case both encryptions are done with the same key. This method is, for example, used in the export versions of SSL implementations.
CBC	Cipher Block Chaining is used to chain encrypted blocks into a stream. The ciphers like DES and RC2 usually encrypt the messages in fixed size blocks (often 64 bit).
RSA	Named after its creators, Rivest, Shamir and Adleman. RSA was one of the first effective public-key algorithms. RSA is used to encrypt messages and create digital signatures. RSA is an asymmetric cipher. RSA is also often referred to in the context of RSA Data Security Inc., which is the company founded in 1992 by the same people.
RC2	RC2 (Ron's Code 2 or Rivests Cipher 2) was RSA Data Security's proposal to be a successor of the DES cipher. It is faster and provides stronger security. It is also a symmetric block cipher, but the key length can be variable in the range from 1 to 128 bits. It can be used in the same modes as DES, including triple cipher.
RC4	RC4 is a symmetric stream cipher from RSA Data Security, Inc. RC4 can use keys of up to 2048 bits in length. However, the key length can be limited to meet U.S. and other export or governmental regulations.
SHA-1	Secure Hash Algorithm that is adopted as Federal Information Processing Standard (FIPS). SHA-1 generates 160 bit fingerprints.

MD5	Message Digest algorithm. MD5 is a hash algorithm that generates 128 bit fingerprints. It is slightly faster than SHA-1.
BASE64	MIME encoding scheme that converts binary data into 7-bit ASCII. It is used, for example, for transferring binary data in mail.
DER	Distinguished Encoding Rules. Rules that define a presentation format of binary data.
X.509	An International Telecommunication Union (ITU) recommendation for the format of certificates.

6.4.5 Creating a Self-Signed Certificate

Self-signed certificates are certificates that are created without involving an external, trusted Certificate Authority (CA). Self-signed certificates might not be usable with all Internet browsers, and the use of them should be limited to testing purposes and situations where the “official” certificate from the Certificate Authority (CA) is not yet received. Web browsers usually notify users about the non-trusted issuer when accessing a site that uses the self-signed certificate. The user then has the choice of accepting or rejecting the connection to such a server.

IBM provides an application with a graphical user interface for managing the SSL key database(s). A key database is a file which includes root certificates of well-known Certificate Authorities and also the SSL-keys that are issued to the system. The key database is protected with a password that is the key to manage all the key information in the key database. *IKEYMAN* is a Java application that comes with the IBM HTTP Server for handling the key management procedures. Although it is not necessary to run the IKEYMAN utility as root user, it is preferable because the managed keyfiles are owned by the user who starts the IKEYMAN application. Since IKEYMAN is a graphical application, it must be run on a graphical display.

Before you can start IKEYMAN, you have to define the environment variable JAVA_HOME

```
# export JAVA_HOME=/usr/jdk_base
```

The IKEYMAN application is started with the command:

```
# /usr/bin/ikeyman &
```

In order to create a self-signed certificate, you have to first create a key database for it:

1. The key database is created by selecting **Key Database File** -> **New...** as shown in Figure 23.

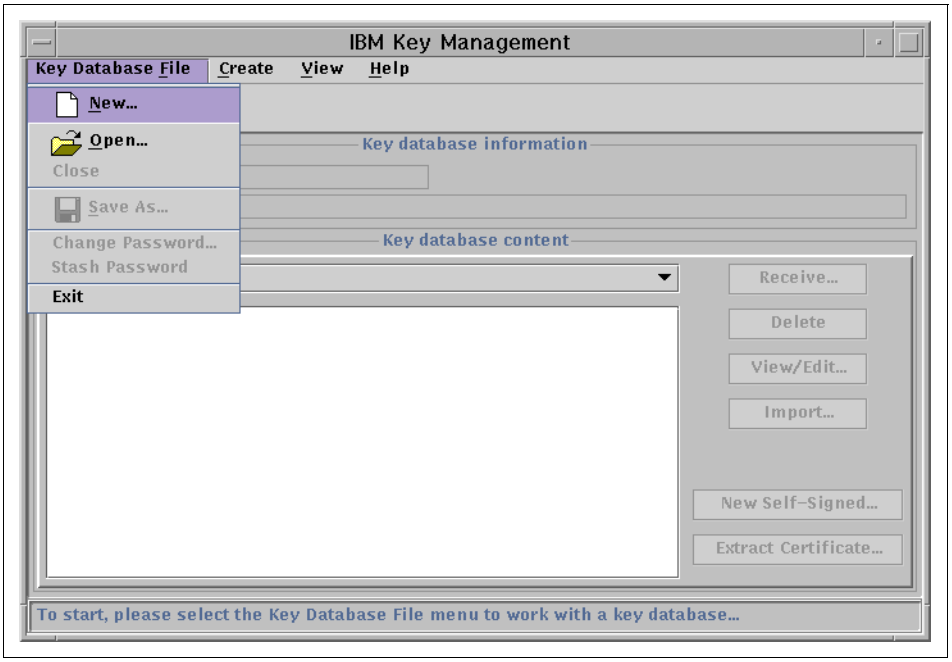


Figure 23. Creating New Key Database with IKEYMAN

2. In the upcoming New dialog, select the **CMS key database file** from the list of key database types and type in the name and location for the file and click **OK**.

   The file extension of the key database file should be .kdb.

   The location for the kdb file can be freely chosen. In the examples shown here, the key database file CompanyA.kdb is located in the directory /usr/lpp/HTTPServer/keys. If the server will be started as root user, you might want to cut down the access permissions to that directory with the commands:

   ```
   # chown root.system /usr/lpp/HTTPServer/keys
   # chmod 0700 /usr/lpp/HTTPServer/keys
   ```

3. The password dialog box opens, and you are asked for a password for this database. You might want to define and record the expiration time for the key database. Click the “Stash the password to a file?” checkbox. If the password is not stashed into a file, the Web server will not be able to start automatically. If you do not stash the password to file, you are asked for it

when you start the server. Click **OK**, and IKEYMAN displays the contents of the key database in the application window. The new key database is not empty. It contains information about some well-known Certificate Authorities already.

The password should be selected carefully, because all the key management operations including deletion of the keys depends on it.

> **Note**
>
> At this point, you have to remember that even if you are creating this key database for testing purposes only, you might someday want to add a production key into it. To ensure adequate security, a good password should be used. Also the password expiration should be defined.

SSL will not work anymore after either the certificate itself expires or the key database password expires. In the latter case, the server's error log contains an error message like "GSK could not initialize, Unrecognized error code returned from GSK."

4. Select **Create** -> **New Self-Signed Certificate...** as shown in Figure 24.

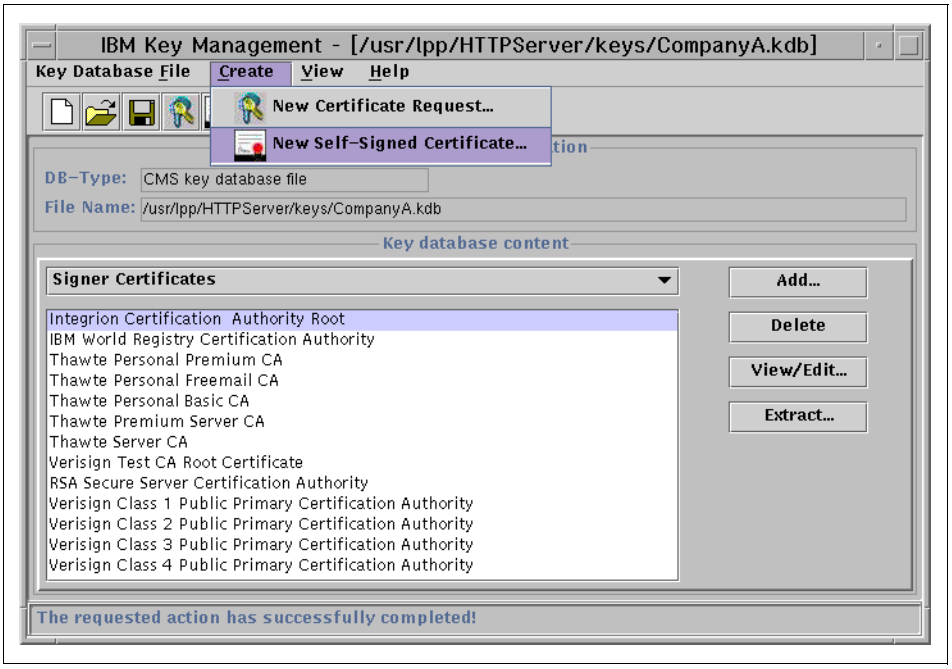


Figure 24. Creating a Self-Signed Certificate

5. Fill in all information about the organization (see Figure 25 on page 139). This information is available to anyone who accesses the site using this certificate and it should therefore contain valid, representative information about your organization. Then click **OK** to create and save the certificate.

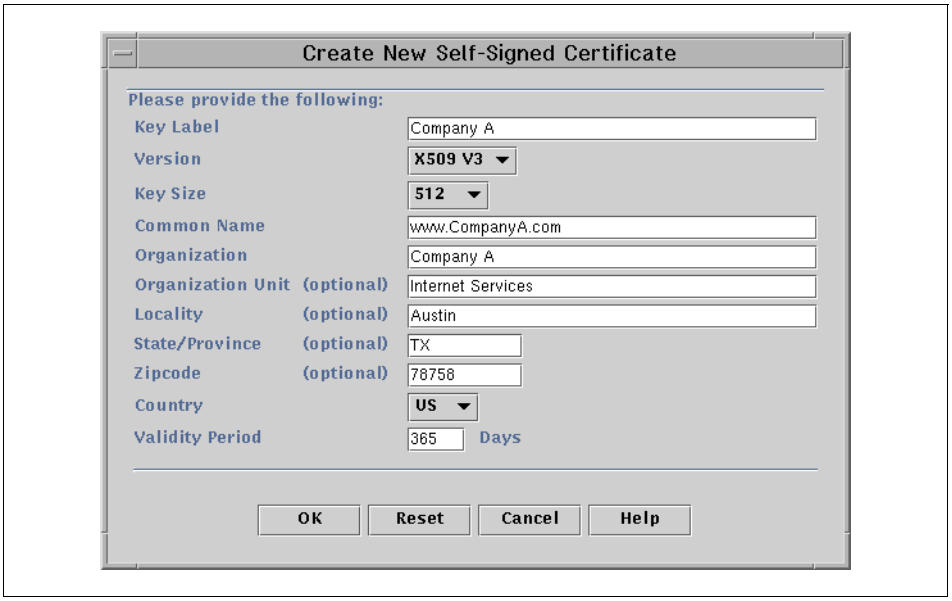


Figure 25. Filling in the Certificate Information

Information about your organization in a certificate is proof of your identity to the user accessing the Web site. However, if you have a self-signed certificate, it's just your word of it.

6.4.6 Using Certificates Signed by a Well-Known Trusted CA

When the key database has been created as described in the previous chapter, it is preconfigured with the root certificates of the most often used certificate authorities. These are:

- Integrion Certification Authority Root
- IBM World Registry Certification Authority
- Thawte Personal Premium CA
- Thawte Personal Freemail CA
- Thawte Personal Basic CA
- Thawte Personal Server CA
- Thawte Server CA
- RSA Secure Server CA (can also be obtained from Verisign)
- VeriSign Class 4 Public Primary CA

- VeriSign Class 3 Public Primary CA
- VeriSign Class 2 Public Primary CA
- VeriSign Class 1 Public Primary CA
- Verisign Test CA

Verisign offers a possibility to use a test certificate which is signed according to their own root certificate. It is intended only for testing purposes. There are no other test certificates preconfigured and if you plan to test some other CA's test certificates, their root certificate have to be inserted into the key database first. This procedure is described in section 6.4.7, Requesting a Certificate from an Unknown CA on page 142.

The following section describes the procedure to request certificate from the Certificate Authorities whose root certificates are predefined in the key database.

6.4.6.1 Requesting a Certificate from a Known, Trusted CA

Requesting a "real" certificate for your server has its price. For example, at the time of writing, such certificates cost several hundred U.S. dollars, depending on whether they are valid in the U.S. only or internationally. The certificate is valid for one year and renewing of the certificate is usually less expensive. The Certificate Authorities have some requirements for a company or organization that is requesting the certificate.

Here are the directions of how to request a certificate from trusted Certificate Authority:

1. Launch the IKEYMAN application and open the key database you have created.
2. Choose **Personal Certificate Requests** from the pull down list in the middle of the application window as shown in Figure 26. Click on the **New...** button in the button list that appears on the right.

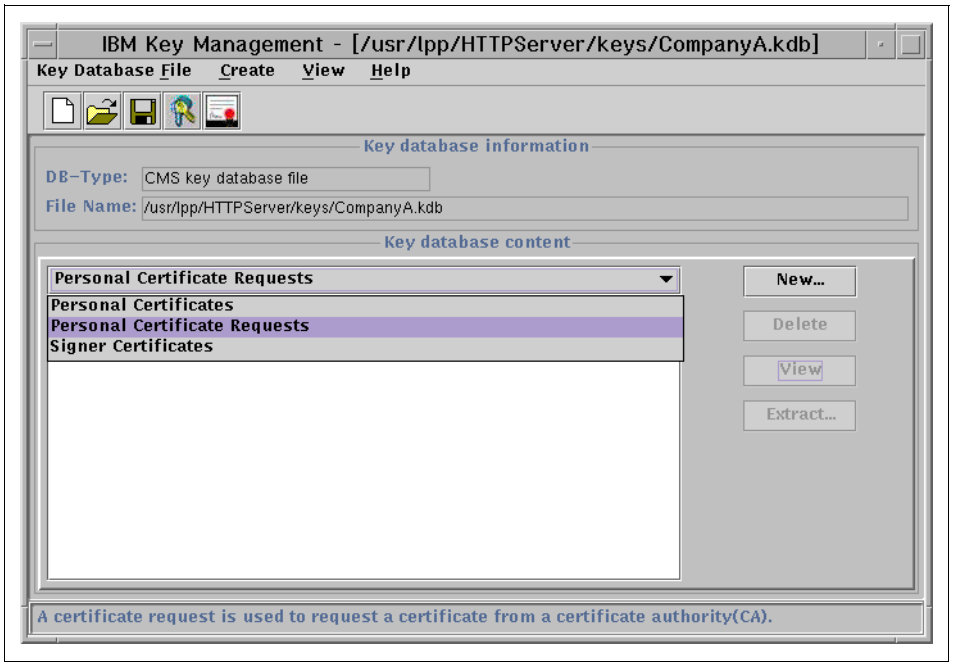


Figure 26. Selecting the Personal Certificate Requests Setting

3. On the dialog that appears, enter the key label for the certificate and fill in the other information about your certificate. You can choose from two key lengths: The 512 bit key size is sufficient for most applications unless maximum security is required. A key of 1024 bit size, on the other hand, requires more processing power, which might be a performance factor.
4. Specify the directory and file name for the certificate request. The file type with extension .arm is PKCS 10 type file in armored 64 bit format. The private key is stored to the directory that contains your key database files. The file name extension of the stored private key database is .rdb. The file locations or file names of these files should not be changed.
5. Follow the Certificate Authority's instructions of how to submit the certificate request to the certifier.
6. When you receive the certificate from the CA, the key has to be imported into your key database. To do so, launch the IKEYMAN application and open the key database.
7. Select **Personal Certificates** from the pull-down menu and click the **Receive...** button on the right. The Receive Certificate from a File dialog appears (see Figure 27 on page 142).

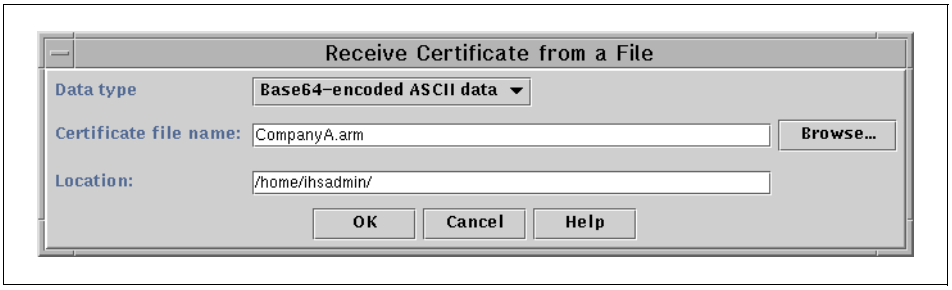


Figure 27. Receive Certificate from File

8. Enter the certificates file name and location and click **OK**.
9. Highlight the new certificate on the list of Personal Certificates and click **View/Edit...** The **Key information dialog** appears.
10. If not selected, select **Set the certificate as the default** checkbox. If you have some other software that uses the same key database, this change may affect them.

The certificate is now ready for use. Do not forget to add a reminder to your calendar when the key database password or certificate is going to expire. Some of the CAs send e-mail about a month before the certificate expires.

6.4.7 Requesting a Certificate from an Unknown CA

When acquiring a certificate from an unknown certification authority, you have to obtain it's root certificate first and store it in your key database. The term *unknown CA* does not mean anything about the reliability or trustworthiness of the CA, but that the CA's root certificate is not preconfigured in the key database.

6.4.7.1 Storing the Root Certificate of the CA

The procedure to request and configure a certificate from an unknown CA is basically the same as with well-known CAs (see 6.4.6, "Using Certificates Signed by a Well-Known Trusted CA" on page 139). Prior to obtaining the certificate, you have to get the root certificate of the CA. The CAs provide the information on how to obtain it. The root certificate must be stored in the key database before the certificate. Make sure that the root certificate is in BASE64 armored ASCII data format.

1. Start the IKEYMAN application and open your key database. Select **Signer Certificates** from the pull-down list as shown in Figure 28 on page 143.

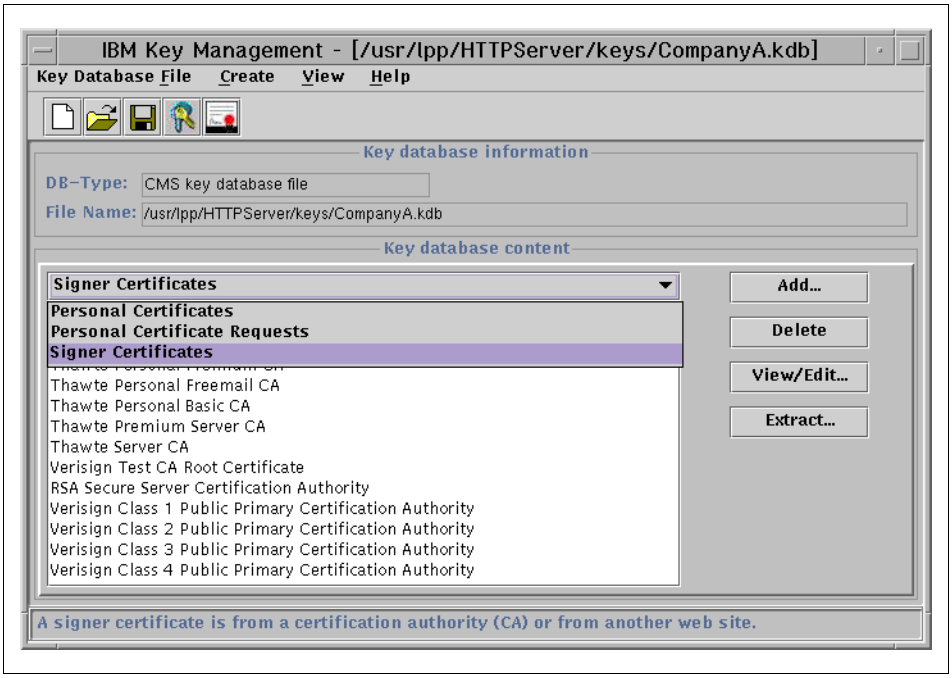


Figure 28. Selecting Signer Certificates in IKEYMAN

2. Click on the **Add...** button to add CA's root certificate from a file.
3. On the pop-up dialog, verify that the file type is **Base64-encoded ASCII data**. Fill in the location and file name of the certificate file.
4. Click **OK** to mark the certificate trusted and to store it.

After completion, the new root certificate will show up in the Signer Certificates list. The root certificate is now available to every certificate you intend to include in this key database.

6.4.7.2 Storing the Certificate that is Signed by an Unknown CA

The root certificate you obtained acts like the root certificate of any known signer. In order to store the signed certificate into your key database, you can follow the instructions in 6.4.6.1, "Requesting a Certificate from a Known, Trusted CA" on page 140.

6.4.8 Configuring the HTTP Server to Use SSL

In the standard server configuration file (httpd.conf) that is shipped with the IBM HTTP Server, there are some directives you should check before starting to configure it. The ServerAdmin directive points to a dummy e-mail address

you@your.address and the ServerName is commented out. It is recommended to use a valid e-mail address for ServerAdmin, and SSL configuration requires that the ServerName directive matches the fully qualified host name of your server, such as www.CompanyA.com.

The minimum configuration changes that need to be done are:

- Definitions for loading the mod_ibm_ssl module
- Port number for the SSL virtual host
- Definition for the SSL virtual host
- Keyfile location
- SSL timeouts

The default port number for SSL is 443. In order to achieve this, defining a virtual host comes in handy. When editing the httpd.conf file, keep in mind that comments within the configuration sections are not allowed.The following actions guide you through these steps:

1. First add the following row into the httpd.conf file as the first item of the LoadModule list:

   ```
   LoadModule ibm_ssl_module      libexec/mod_ibm_ssl.so
   ```

2. Add the following row as the first line to the AddModule list:

   ```
   AddModule mod_ibm_ssl.c
   ```

3. Add the port number for the virtual server just below the “Listen 80” statement. The default port number for SSL is 443.

   ```
   Listen 443
   ```

4. Check that you have defined the ServerName directive:

   ```
   ServerName www.CompanyA.com
   ```

   Add following text-block to the end of the httpd.conf:

   ```
   <VirtualHost :443>
      SSLEnable
      SSLClientAuth none
      DocumentRoot /www/html/CompanyA
      ErrorLog /www/logs/CompanyA/error_log
      TransferLog /www/logs/CompanyA/access_log
   </VirtualHost>
   SSLDisable
   Keyfile /usr/lpp/HTTPServer/keys/CompanyA.kdb
   SSLCacheEnable
   SSLCachePortFilename /usr/lpp/HTTPServer/tmp/siddfile
   ```

```
SSLV2Timeout 100
SSLV3Timeout 1000
```

SSLClientAuth should be set to none in cases where there is no client authentication (see also 6.5, “SSL Client Authentication” on page 146).

5. Save the file and restart the server.

```
# apachectl graceful
```

The SSL timeout parameters are related to caching of the SSL session IDs. SSL session IDs should be cached in order to reduce the expense of repeating SSL handshaking. The IBM HTTP Server uses an internal daemon process *sidd* to cache the SSL session IDs to a file that is accessible by the HTTP server processes. Make sure that the file and the directory, defined in the directive SSLCachePortFilename, is writable by the user the server processes ran under.

6.4.9 SSL and Virtual Hosts

Running multiple virtual hosts that support SSL can be done with the IBM HTTP Server by defining IP-based virtual hosts (see also 5.1, “Virtual Hosts” on page 71). You are limited, however, to only one key database that must include all certificates that are used. The certificate file is specified with the SSLServerCert directive.

The following example defines two Web sites in the same httpd.conf file. This requires four VirtualHost containers to be defined in order to have it working:

```
<VirtualHost 1.2.3.4>
   ServerName www.CompanyA.com
   ServerAdmin webmaster@CompanyA.com
   DocumentRoot /www/html/CompanyA
   ErrorLog /www/logs/CompanyA/error_log
   TransferLog /www/logs/CompanyA/access_log
</VirtualHost>

<VirtualHost 1.2.3.5>
   ServerName www.CompanyB.com
   ServerAdmin webmaster@CompanyB.com
   DocumentRoot /www/html/CompanyB
   ErrorLog /www/logs/CompanyB/error_log
   TransferLog /www/logs/CompanyB/access_log
</VirtualHost>

<VirtualHost 1.2.3.4:443>
   SSLEnable
   SSLClientAuth none
```

```
   SSLServerCert Company A
   ServerName www.CompanyA.com
   ServerAdmin webmaster@CompanyA.com
   DocumentRoot /www/html/CompanyA
   ErrorLog /www/logs/CompanyA/error_log
   TransferLog /www/logs/CompanyA/access_log
</VirtualHost>

<VirtualHost 1.2.3.5:443>
   SSLEnable
   SSLClientAuth none
   SSLServerCert Company B
   ServerName www.CompanyB.com
   ServerAdmin webmaster@CompanyB.com
   DocumentRoot /www/html/CompanyB
   ErrorLog /www/logs/CompanyB/error_log
   TransferLog /www/logs/CompanyB/access_log
</VirtualHost>

SSLDisable
Keyfile /usr/lpp/HTTPServer/keys/Keyfile.kdb
SSLV2Timeout 100
SSLV3Timeout 1000
```

The SSLServerCert specifies the certificate label to be used for a specific virtual host. A requirement for the use of SSL in multiple Web sites in the same machine is that you have to use the IP-based virtual hosts.

Note

The file httpd.conf.sample.ssl that ships with the SSL module of the IBM HTTP Server contains a wealth of information in the the form of comments that further explain the setup of SSL, including client authentication (subject to the next section). This file is located by default in

6.5 SSL Client Authentication

Client authentication is an option that is not normally used unless there is a particular reason. In order to get authenticated during the SSL handshaking process, the client must have a *client certificate*. Most of the well known Certificate Authorities also sign personal client certificates. A client certificate is a binary file which has the information about its owner in an X.509 certificate format.

The Certificate Authority needs to have some information about an individual in order to create a certificate. The certification process is usually quite straight forward. The CA usually require a name, nationality, maybe some postal information, and an e-mail address. Some CAs, like Thawte, require a social security number or a citizen number, depending on the country of residence or citizenship. Common to most (or even all) of the international CAs is that the only identifying information that connects the user to certification process is the e-mail address. A client certificate incurs some fees, but they are relatively cheap, that is, in the area of about 10 US dollars. Client certificates are usually valid for one year. After that, the certificate must be renewed.

Table 13 lists the attributes that are contained in an X.509 certificate.

Table 13. Attributes Included in the Client's X.509 Certificate

Attribute	Shortname	Definition
CommonName	CN	The client's common name
Org	O	The organization of the client
OrgUnit	OU	The organizational unit of the client
Locality	L	The locality in which the client resides
StateOrProvince	ST	The state or province in which the client resides
Country	C	The country in which the client resides
Email	E	The e-mail address of the client
IssuerCommonName	ICN	The CA's common name
IssuerOrg	IO	The organization of the CA
IssuerOrgUnit	IOU	The organizational unit of the CA
IssuerLocality	IL	The locality in which the CA resides
IssuerStateOrProvince	IST	The state or province in which the CA resides
IssuerCountry	IC	The country in which the CA resides
IssuerEmail	IE	The e-mail address of the CA

The need for client authentication and the level of identification depends greatly on the needs of Web site owners. For example, the owner of a Web site that needs to be very certain about the identity of the individuals who have access to their Web page might choose to run their own CA software

and issue the certificates according to their own policies. A financial institute would probably require application by person and hand signature before they issue a certificate to their customer.

The IBM HTTP Server supports client certificates issued by any CA software that is capable of issuing X.509 certificates. If you are going to use certificate revocation lists (CRL), the IBM Vault Registry is suitable to provide that function. The CRL is a database of certificates that are revoked before their expiration date for any reason. You can find more information about the IBM Vault Registry at `http://www.software.ibm.com/commerce/registry`.

6.5.1 Client Certificates and the IBM HTTP Server

The IBM HTTP Server supports three levels of SSL client authentication. The directive that controls the client authentication level is SSLClientAuth and it accepts the values *required*, *optional*, and *none*.

If you choose *required*, only users with valid certificates that are signed by a trusted CA are granted access. The *optional* value causes the server to ask for the client certificate, but it is not necessarily required. This option is often used to allow more specific authentication for certain administrative users.

The use of client authentication adds some complexity to the authentication process and increases the network traffic (and thus server load), and should therefore only be used if specifically required.

There are two types of access control that can be used in conjunction with SSL client authentication. The first, fake basic authentication, uses the client certificate's distinguished name as the user for normal basic authentication. The fake basic authentication functionally is deprecated and should not be used if possible as it does not provide effective authentication. The directive to specify the fake basic authentication is SSLFakeBasicAuth. The better alternative to client authentication is to use the SSLClientAuthRequire directive.

The SSLClientAuthRequire provides the whole variety of X.509 attributes to be used in the Web server and allows complex logical comparisons of the certificates information.

The SSLClientAuthRequire directive supports conditional expressions such as in the following example:

```
SSLClientAuthRequire (CommonName = "Chip Nut" OR CommonName = "Dale Nut")
AND Org = "Company A" AND OrgUnit = "Research and Development"
```

Or, more compactly, using the shortnames:

```
SSLClientAuthRequire (CN = "Chip Nut" || CN = "Dale Nut") && O = "Company A"
&& OU = "Research and Development"
```

The Web server accepts the client certificate only if it is signed by a CA whose root key is marked as a trusted root in the server's key database. The CA's root key is marked trusted when the certificate is listed under the signer certificates and when selected with the **View/Edit...**, the **Set the certificate as a trusted root** checkbox is selected as shown in the following Figure 29.

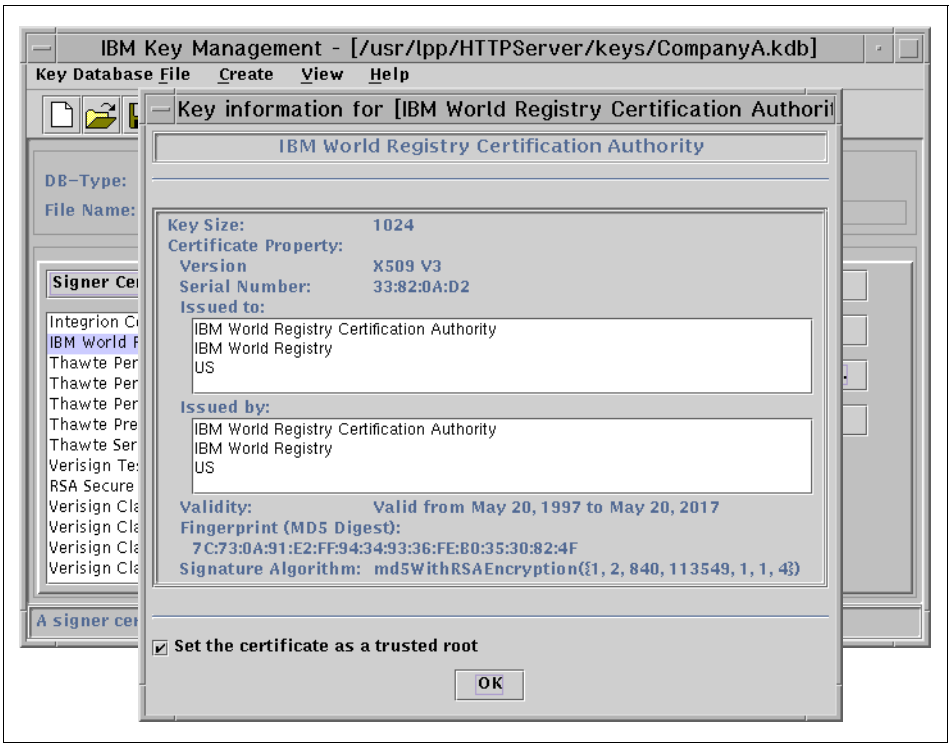


Figure 29. Marking/Checking a CA's Root Certificate as Trusted

6.6 Security Considerations in the Server Configuration File

The IBM HTTP Server is usually started under root authority because UNIX systems limit the access to port numbers that are smaller that 1024 only to the root user. When the server is started, it launches the actual HTTP server processes under the user ID that is defined in the server configuration file (httpd.conf). All file accesses, including execution of CGI programs, are done with the permissions of this user.

The file ownerships and access rights should be strictly controlled in the Web servers scope and in the areas where configuration files and binaries reside. As a rule of thumb, all configuration files and binaries (other than CGI programs) should not be accessible by the user under which the HTTP server runs. Furthermore, the log files and the directories where they reside should not be changeable by that user because the log files are potentially the first thing a possible hacker would want to delete or alter.

There are some security considerations in the httpd.conf file. As has been mentioned earlier in 3.6, “Initial Setup” on page 42, the server should not be run under the user account *nobody* because there are other processes using the same account. The best approach is to define dedicated user and group for running the server processes.

As discussed before, you should beware of symbolic links that can easily and inadvertently enlarge the scope of the server. The Directory options to look for are FollowSymLinks and SymLinksIfOwnerMatch.

Server side includes (SSI, see also 10.4, “Server-Side Includes” on page 219) provides the possibility to execute some local programs via HTML pages. This can be a good feature for some occasions, but you might want to deny the execution by defining:

```
Options IncludesNOEXEC
```

It might also be wise to restrict the execution of CGI scripts to certain directories (see also 10.2, “CGI Programs” on page 214).

The use of the .htaccess files can also be controlled. One way to deny all the .htaccess overrides, includes and accesses is to create a directory container for the root directory of the server:

```
<Directory />
   AllowOverride None
   Options None
   Allow from all
</Directory>
```

You might also consider denying the access to the root directory by changing the “Allow from all” above to:

```
Order deny,allow
Deny from all
```

That kind of access control denies access to all locations except to the ones that are specifically allowed in other directory or location containers. The sequence to parse the containers is Directory, Files and Location. Beware of

such situations where you accidentally override some restriction you have defined in the directory container by adding an improperly defined location container.

As discussed in 5.3, “User Directories” on page 85, the UserDir directive can also cause some security exposures if it applies to the root user. This can be avoided by defining:

```
UserDir disabled root
```

Another issue that ought to be mentioned is to define CoreDumpDirectory directory. A core dump usually is created when an application aborts. The core dump is used for debugging purposes and it contains, among other things, the memory environment of the server. This file may contain, for example, some passwords or other sensitive information. The core dump is written by default to the ServerRoot directory, which creates two risks. First, a core dump file might fill up the file system and disturb the stability of the Web server. Second, since the core file is within the server’s scope, a Web user could download files such a core file and examine its contents. You should therefore direct the core file to a directory outside the server’s scope where disk space is not an issue, or even to /dev/null (if you do not intend to use them for analysis) as in this example:

```
CoreDumpDirectory /dev/null
```

Chapter 7. Performance and Scalability

There are many benchmarking-type performance tests done in the market to persuade and convince consumers that each product stands out from others. In this chapter, no actual performance data is measured or published (it is widely known that Apache is a well-performing Web server). Rather, some reminders are given to webmasters who wish to pay attention to the health of their Web servers, as well as recommendations for configuring the IBM HTTP Server to improve the overall performance of the Web server. Absolute performance figures depend on many factors that would barely meet the needs of any particular environment. Some tools that are available for performance monitoring are also introduced. In addition, a discussion pertaining to the scalability issue is presented for webmasters who wish to expand their Web servers to a bigger audience.

7.1 Basic Performance Consideration

Figure 30 shows computers connected to the Internet on both the provider and the client sides.

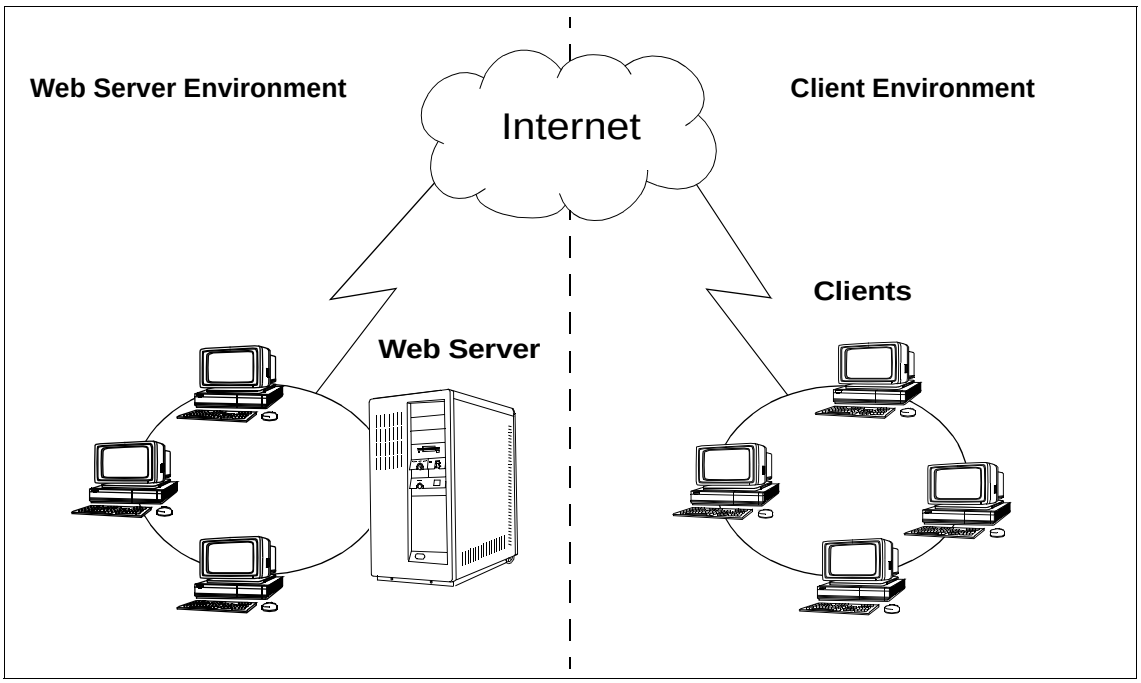


Figure 30. Basic Internet Connection

The three main components necessary for a Web server are the Web server hardware and operating system, the Web server software (the IBM HTTP Server in this case) and the link to the network (Internet). Inevitably, these are also the three main factors that contribute to the overall performance of the Web server, as discussed in the following sections.

7.1.1 Link Bandwidth

The closest component to the network (Internet) from the Web server is the link, as shown in Figure 30 on page 153. At present, there are many options on the market that provide fast connections for Web servers. Typical values used by major networks and ISPs are ISDN lines running at 128 Kbps, T1 lines catering to a speed of 1.5 Mbps, or the T3 lines offering 44.7 Mbps. The choice of the lines used determines the speed of incoming and outgoing traffic, which affects the response time of the Web server at the end user's workstation. A wider throughput generally gives a faster response time from the Web server, assuming that the Web server itself is not the major bottleneck. However, when compared to the speed of the network environment the Web server is connected to, the link is more often the hindering factor.

If the Web server is not connected to the Internet, but only to a corporate-internal intranet, network connections are usually faster since the server has a more direct network connection. It is still wise, however, to estimate and measure the network traffic to and from the Web server because it might be close to the upper limit of a normal Ethernet connection, which is 10 Mbps. Particularly when there are a number of machines connected to the same network segment, the network can easily become saturated. In fact, most modern RS/6000 hardware is easily capable of saturating a 10 Mbps link when simply providing static HTML page serving as a Web server. It is, therefore, certainly worth connecting it directly to a high-speed backbone network, such as an ATM or a 100 Mbps Ethernet link, if high volume traffic is anticipated.

7.1.2 Hardware and Operating System

Basically, these two areas are closely related since the operating system forms the logical hardware abstraction layer between the hardware devices and the Web server, as shown in Figure 31 on page 155. Any requests for files or the execution of programs by the Web server requires the use of the hardware resources. Thus, these areas can be considered with the operating system to enable better performance of the entire server machine.

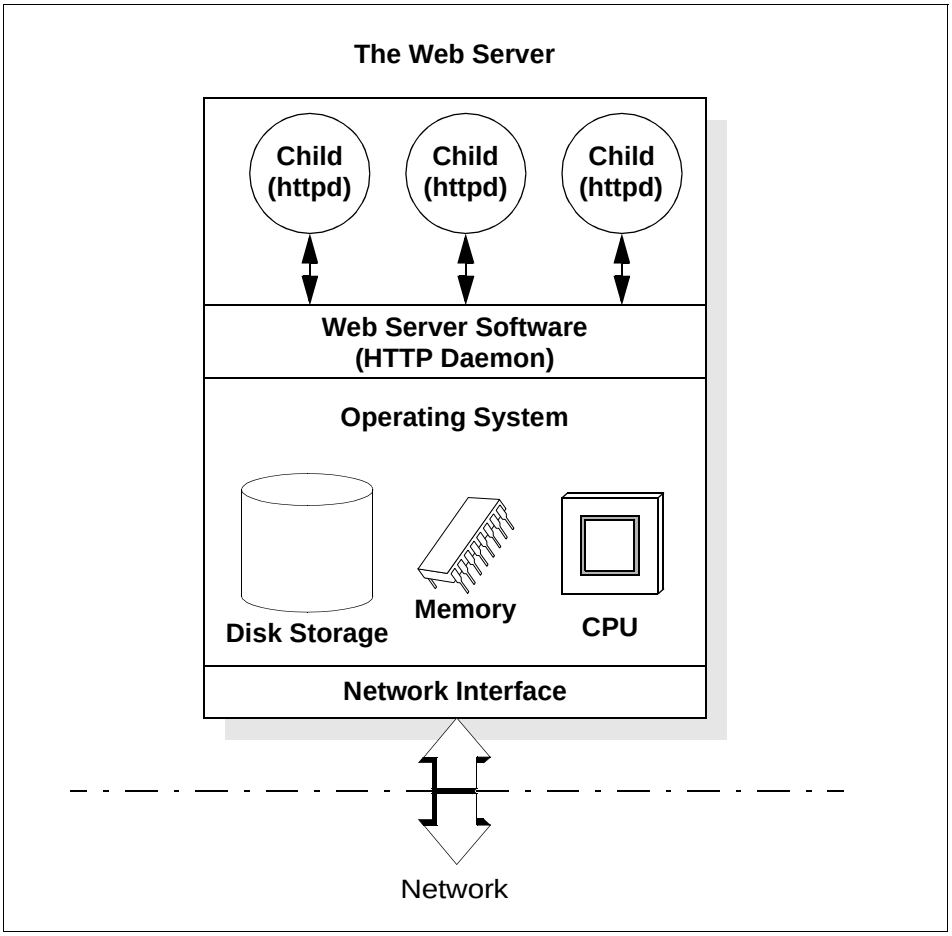


Figure 31. Factors Affecting Performance of a Web Server

A typical Web server usually only requires a system with a single processor and a limited amount of memory and hard disk storage space. However, companies may not purchase a dedicated Web server machine for providing electronic information to their clients, unless they are in the ISP area of business, or have a large number of clients. Thus, in small environments, the Web server may have to share CPU, memory, disk, and network resources with other applications and the operating system running on the same machine.

In the following sections, we will briefly explain the importance of each of these components. A more thorough analysis can be found, for example, in

the redbook *Understanding IBM RS/6000 Performance and Sizing*, SG24-4810, or in the *IBM AIX Performance Tuning Guide*, SC23-2365.

7.1.2.1 CPU

The CPU is mostly used for client request processing and in the rare instance when the IBM HTTP Server parent process spawns new child processes to handle a new request. Client-request processing can be as easy as simply delivering a static HTML page, or it can involve a considerable amount of application code. Depending on the amount of extra processing necessary for client requests, the CPU can be a limiting factor, although in many cases the CPU is not the dominating bottleneck of a Web server machine.

Spawning new child processes can slow down the operation of a Web server if it happens too often. The IBM HTTP Server keeps a certain number of httpd processes running that can process client requests in parallel (see also 3.7, “Server Process Structure” on page 43). The minimum and maximum number of these httpd processes, along with other, related numbers, can be configured (see 7.1.3.1, “Process Handling” on page 158). If the configuration is balanced, there should not be many httpd processes spawned over time in normal operation and, therefore, the Web server performance is not considerably affected by spawning new processes. Process spawning may become more of an issue when CGI programs and application code is involved. Please read 10.2.3, “CGI Performance Considerations” on page 217 for more information.

For machines that are required to handle heavy Web serving functions, as well as other applications, database or programs that are CPU-intensive, a multiprocessor system may be considered. Programs can then run concurrently in these machines and not be held back by Web processes. The IBM HTTP Server will use all CPUs in a multiprocessor system. On the other hand, webmasters are reminded that multiple processors do not always increase the performance tremendously because other common resources, such as memory, disk storage, and network connections are still shared by all processors and programs. Please refer to 7.3, “Scalability for the IBM HTTP Server” on page 171 for further discussions.

7.1.2.2 Memory

Together with the raw performance of the CPU, the amount of available physical memory greatly affects Web server performance. There should be enough physical memory available that the system does not need to start paging (see the next section). The amount of memory that is necessary to fulfill this requirement greatly depends on the following factors:

- Requirements of any other application running at the same time

- Number of httpd processes that are running at any given time (see 7.1.3.1, “Process Handling” on page 158)
- Number and size of DSO modules loaded with the IBM HTTP Server
- Requirements of any Web applications that might be started through the Web server

Note: The amount of additional memory required by DSO modules is not only dependent of the number and size of those DSO modules, but also on the number of httpd server processes that are running at the same time since each httpd server process may instantiate the DSO modules.

7.1.2.3 Paging Space

Paging space should not affect a Web server. In other words, there should be enough physical memory available that the server does not need to page. Paging activities should be avoided since it may degrade online performance of a Web server drastically for two reasons:

1. An otherwise runnable process may have to wait some time for a missing page to be read in from the paging space.
2. The disk I/O performance may be seriously affected by excessive paging.

However, should paging occur, the paging space(s) on the disk(s) should be placed so that the least impact on performance results. The three most important rules for paging space allocation are:

- Have a paging space on every disk in the system (provided they have about the same average access time).
- Only use one paging space per disk. Multiple paging spaces per disk reduce paging performance.
- Check the characteristics of each disk and define the paging space at the best-performing place (usually *center* or *edge*, depending on disk model).

7.1.2.4 DIsk I/O

The amount of disk I/O that takes place in a Web server heavily depends on the specific environment. Some Web servers only serve static HTML requests, while others do extensive application processing. Thus, there is no general rule stating that disk I/O is the predominant bottleneck in a Web server. In many cases, disk I/O is not a critical issue. However, if heavy disk I/O is expected or observed, follow these guidelines to tune this subsystem:

- Having additional memory in the system reduces disk I/O through AIX disk caching for read accesses.

- Distributing the data among several disks reduces the average seek and access time as opposed to having all data on a single disk.
- If SCSI disks are being used, there should be no more than two to three disks per SCSI adapter.
- Use fast disk subsystems, such as SSA (Serial Storage Architecture).
- Follow the pertinent guidelines of any applicable database subsystem.

7.1.2.5 Network I/O

The most commonly used network interface adapters today are of Ethernet standards, with speeds of either 10 or 100 Mbps. In many cases, this theoretical value is considerably more than what the actual link can offer. The actual link speed available to a Web server depends on the capabilities and configuration of the network components and the network topology. The slowest link in a network will determine the overall throughput between two systems. Network interface tuning involves the settings for the number and sizes of interface buffers available to the operating system. AIX has a self-tuning mechanism for network buffers that works perfectly in most cases. However, if network I/O needs to be tuned to its very best, you should consult pertinent manuals, such as the *AIX Performance Tuning Guide,* SC23-2365.

7.1.3 The Web Server

The IBM HTTP Server in Figure 31 on page 155 resembles an inverted version of the Apache server model shown in Figure 1 on page 17, which illustrates the two important phases: the configuration phase and the request parsing phase. The performance issues for the Web server mostly emphasize the configuration settings in the IBM HTTP Server's configuration file, httpd.conf, which affects the nature of the Web server after it is initialized and when it is serving the client requests in the request parsing phase. The following sections group some directives into specific categories that are relevant for the performance of the IBM HTTP Server.

7.1.3.1 Process Handling

The process-handling phase is largely dependent upon the configuration made by the webmasters in the httpd.conf file before starting the IBM HTTP Server. Thus, at this stage, prior considerations should have been made by the webmasters, and these decisions vary with different operation environments. This category contains directives that are mostly related to the httpd processes (see also 3.7, "Server Process Structure" on page 43).

MaxClients — Restricts the maximum number of child httpd daemons created that can process client requests in parallel. If the value is too low,

clients are locked out and placed in a limited queue determined by another directive (see ListenBacklog). If this is the case, the next client request can be served only when any child httpd process has finished processing a current request. Alternatively, a value that is too large may incur memory resource overheads and the Web server begins to swap, thus decreasing performance. A general method of calculation is shown below:

MaxClients = Total memory available to the IBM HTTP Server / memory consumption for each child httpd

The default value is 150.

MaxRequestsPerChild — Restricts the number of requests handled by each child httpd daemon. Once this value is hit, the child process terminates. One of the intentions of this parameter is to limit the lifetime of an httpd client process in order to prevent it from using too much memory resource in case of memory leaks. A zero value results in an infinite lifetime of the child process. This is not advisable for the reasons mentioned above. The number specified can be fairly high (the default is 10.000.000) if stable operation is expected.

MaxSpareServers — Specifies the upper number of idle httpd child processes which are not handling any requests. The parent process that spawned these processes will kill all excess idle processes if the number of existing idle processes are higher than this value. If the machine is dedicated for Web serving purposes, a high value can be set to provide good response times. On the other hand, if the machine provides other functional roles or has relatively low memory resources available, a lower value should be set to keep resource consumption low. The default is 10.

Please read the General Guidelines at the end of this section for more discussion on this topic.

MinSpareServers — Specifies the lower number of idle httpd child processes which are not handling any requests. The parent process that spawned these processes will create more processes at a maximum rate of 1 child per second if the number of existing processes is lower than this value. If the machine is dedicatedly used for Web serving purposes only, a high value can be set to provide good response times since more processes are spawned in advance to quickly handle client requests, especially in situations where the Web server encounters burst loads. If the machine has little memory resources available, this value should be kept low. The default is 5.

Please read the General Guidelines at the end of this section for more discussions.

StartServers — Specifies the number of httpd child processes to be created at startup. There must be at least one child process to handle requests. If the load is heavy, set the values for StartServers, MaxSpareServers and MaxClients close. Otherwise, keep the number small if the MaxSpareServers value is set low, because the spare processes that are spawned will ultimately be killed before they are even utilized. The default is five.

Please read the General Guidelines at the end of this section for more information.

ListenBacklog — Restricts the maximum length of the queue of pending connections from the clients. The value is limited to the operating system that uses the listen() system call, but it may be increased when required. The default value is 511 and generally requires no tuning.

General Guidelines

The three directives MaxSpareServers, MinSpareServers and StartServers are closely related because they control the number of httpd processes running on the Web server. Basically, adjustments should only be applied to very busy sites and considerations should be made in these three areas regarding the operating system, the number of preloaded modules and the machine load. If the machine load is high, increase the MinSpareServers and StartServers directive values, but generally the values should not be set too high for all these directives. A good idea is to set the MinSpareServers and MaxSpareServers directives to similar values, or even the same value. A value for the MaxSpareServers directive close to that of the MaxClients (mentioned above) results in an optimized response time since it minimizes the chance that a new process needs to be spawned prior to client request processing.

7.1.3.2 Connection Issues

The directives discussed in this category focus on the persistent connection (also known as KeepAlive) feature in the HTTP/1.1 specification. This feature is designed to reduce the number of times a client needs to establish a new connection when transferring documents with multiple components, such as multiple images. The IBM HTTP Server supports this feature via the *KeepAlive* directive and thus allows multiple requests to be sent across the same connection. Besides the server configuration, webmasters must also be aware of the browsers' behavior when responding to persistent connections, which, however, is not related to performance of the Web server and thus is not within the scope of this discussion.

MaxKeepAliveRequests — Restricts the number of requests allowed per connection. A zero value grants an unlimited number of requests, which is not advisable since the process serving this client will be held up while keeping the connection open. According to the type of documents to be served, webmasters should consider an appropriate value to ease the incoming request traffic and also the system's resources utilization. However, a relatively high value is recommended for maximum server performance. The default is 100.

KeepAliveTimeout — Sets the amount of time the IBM HTTP Server holds the connection for a subsequent request before closing it on acknowledgment. Webmasters should consider the trade-off between network bandwidth and server resources when changing this value (the default is 15 seconds). However, it is not advisable to raise this value to more than 60 seconds. The timeout value after the request has been received is governed by the Timeout directive mentioned below. A large value is not advisable because it blocks the system resources if no requests are submitted.

Timeout — Sets the amount of time the IBM HTTP Server waits for these three events:

- Time taken to receive a GET request
- Time taken between receipt of TCP packets on a POST or PUT request
- Time taken between acknowledgments on transmissions of TCP packets in responses

The default is 300 seconds.

KeepAlive — Enable or disable persistent connections. Ultimately, the webmaster can decide not to support the persistent connections feature by turning it off (the default is on).

7.1.3.3 Resource Usage

In this category, the directives determine values related to the operating system and apply for resource usage. The limitation configured by each directive restricts the resource usage of the httpd child processes.

Note: The directives explained in this section are not included in the default configuration file since they are normally not used, because they can (or even should) be specified on the operating system level, if required. These directives should only be used when the values need to be set lower that what the operating system permits.

RLimitCPU — Sets the soft and hard limits for CPU utilization. This limits the amount of time the CPU should be utilized by a process, and is expressed in seconds per process. If the machine is solely dedicated to performing as a Web server, a high value does not limit client request processing. On the other hand, taking too much CPU time in a multipurpose server may result in a degrading of performance for other areas.

RLimitMEM — Sets the soft and hard limits for the memory usage (in bytes) per httpd child process. The webmaster should consider this value when deciding on the corresponding value in the MaxClients directive in 7.1.3.1, “Process Handling” on page 158.

RLimitNPROC — Sets the soft and hard limits for the number of processes per user. For the case of CGI processes running under the Web server’s UID (which is the normal case), the limitation set with this directive restricts the number of processes the server itself can create by forking. Thus, it might limit the server’s ability to create new httpd processes.

SendBufferSize — Specifies the TCP buffer size in bytes.

7.1.3.4 Resolution and Mapping

This category includes the directives that affect the httpd processes specifically in the client-parsing phase in the runtime environment.

HostnameLookups — Enables or disables DNS lookups to be performed such that host names (rather than IP addresses) can be logged. Such a translation, though useful to CGI or SSI scripts and logging, incur additional latency. Furthermore, webmasters should recognize the trade-offs when using the values allow from domain or deny from domain (refer to 2.3.1, “Security Modules” on page 20 for more information). These add even more latency by performing a double (reverse and forward) lookup for security purposes. To increase performance in any case, webmasters should set the HostnameLookups directive to off. However, this does not completely seal the fate for the need of a hostname translation. The following are three recommended methods to obtain the hostname:

1. Scoping the use of HostnameLookups directive limits the scope and performs host name translation only when necessary. For instance, enable host name translation for certain file extensions only, as shown below:

```
HostnameLookups off
<Files ~ "\.(html|cgi)$>
   HostnameLookups on
</Files>
```

2. Consider using the *gethostbyname* call in CGI programs to obtain the host name.
3. The utility `logresolve`, provided in the /usr/lpp/HTTPServer/sbin directory, can also be used to look up host names from logged IP addresses offline, or even on another system. The syntax for the use of this program is:

   ```
   logresolve [-s statfile] [-c] < input > output
   ```

Options — Grants certain server features in the specified directory. The server features are enabled by stating the corresponding values. There are two parameters that affect the behavior in which directories are mapped when requested, as shown below:

- FollowSymLinks — The server follows symbolic links in the specified directory.
- SymLinksIfOwnerMatch — The server follows symbolic links only if the target file or directory and the link is owned by the same user ID.

If FollowSymLinks is not specified in the URL-space, the server needs to do additional system calls to check for symbolic links, which might incur some performance burdens. The example below shows a configuration for the server's root directory (for example, `http://<server>/index.html`). This definition causes extra execution of the stat() system call on /www, /www/htdocs, and the document /www/htdocs/index.html.

```
DocumentRoot /www/htdocs
<Directory />
   Options SymLinksIfOwnerMatch
</Directory>
```

These results are not cached within the server, which causes the checking procedure to occur for each client request. The same applies for the use of SymLinksIfOwnerMatch, which is used for security purposes. To optimize the use of the Options directive with these two parameters, here is what the webmaster can do:

```
DocumentRoot /www/htdocs
<Directory />
   Options FollowSymLinks
</Directory>
<Directory /www/htdocs>
   Options -FollowSymLinks +SymLinksIfOwnerMatch
</Directory>
```

In the example above, FollowSymLinks is enabled for the entire root directory, such that extra checks are avoided from the DocumentRoot path. It

is then disabled in the specific directory /www/htdocs, but additional checks for security purposes are configured for that directory through the use of SymLinksIfOwnerMatch. For ultimate performance, and if security is not the main concern, webmasters can consider avoiding protection of symlinks completely by specifying FollowSymLinks for the whole directory tree.

AllowOverride — Enables the use of directives that overrule the server configuration to directories that contain such directives in the corresponding .htaccess files. The use of this directive with a parameter ALL causes the IBM HTTP Server to search for and open.htacess files in each directory in the path of a requested file. For instance, a request for the URL /index.html (given the configuration shown below) results in the IBM HTTP Server checking for and opening the files /.htacess, /www/.htaccess and /www/htdocs/.htaccess.

```
DocumentRoot /www/htdocs
<Directory / >
   AllowOverride all
</Directory>
```

The effects are similar to that of the FollowSymLinks parameter mentioned above. Likewise, a better implementation is somewhat similar to the solution provided for the FollowSymLinks parameter as illustrated below:

```
DocumentRoot /www/htdocs
<Directory />
   AllowOverride None
</Directory>
<Directory /www/htdocs>
   AllowOverride all
</Directory>
```

Here again, if performance is the key issue for the Web server, webmasters can just specify AllowOverride None for all the directories.

DirectoryIndex — Specifies a list of resources to be returned to a request for a directory specified in an URL. This is the case when the last portion of the URL specifies a directory (rather than a file), or when a / is used at the end of the URL. For example, a request for /www/htdocs/ results in the server mapping it to a document /www/htdocs/index.html given the configuration below:

```
DirectoryIndex index.html
```

When it comes to performance, using a wildcard for the directive would cause the server to depend on content negotiation (introduced in 2.1, “Features of the Apache Server” on page 11) to find out what the client’s browser is

capable of before delivering a suitable document. It is, therefore, not recommended to use wildcards for the `DirectoryIndex` as shown below:

```
DirectoryIndex index.*
```

It is, however, recommended to specify a specific list so that the server does not spend time and resources trying to get the best match. The following is an example of such an explicit list:

```
DirectoryIndex index.cgi index.pl index.shtml index.html
```

7.1.3.5 Logging and Auditing

Log files in any Web servers are a major source of information for webmasters to keep track of the Web server's status and for security checks, too. Webmasters can control the amount of information to be logged or even whether to enable or disable logging at all. Since logging creates some overhead that degrades Web server performance, webmasters should review the settings for logging and keep them at a minimum. More about logging and auditing can be found in 5.7, "Logging" on page 97 and 5.8, "Auditing" on page 104.

7.2 Performance Monitoring

In the first part of this chapter, the basic guidelines for performance issues were explained to remind and caution webmasters when they consider the hardware and the configurations that may affect the performance of the IBM HTTP Server. After setting up the Web server, tuning the configuration parameters is the fastest and the cheapest way to improve the performance. From then on, webmasters need to find out if changing some parameters actually improved the performance at all. Others may wish to do some data collection of the behavior of the Web server before they consider which parameters or changes are most suitable for their specific environment. In this section, some techniques of performance monitoring are presented to facilitate the webmasters in their work.

7.2.1 Hardware and Operating System

The IBM HTTP Server can just be considered an application that runs under AIX on an IBM RS/6000. Any performance-tuning guidelines and tools that apply to other applications also apply to the IBM HTTP Server. AIX comes with a set of utilities that allow a system administrator to monitor the utilization of the system resources in order to identify potential bottlenecks.

The following shows some useful utilities on AIX and a simple example of usage. Most of these utilities have other command-line options that let you

select from a variety of information they can provide. For more detailed information, please refer to the *AIX Commands Reference,* SBOF-1877, or to the *AIX Performance Tuning Guide,* SC23-2365.

vmstat

This very basic, yet powerful utility, displays some statistics about system resource usage, such as CPU utilization (percentage of user, system, idle, and waiting times), memory usage, paging activities (page lists, pageins, pageouts), and number of runnable and blocked processes. The statistics can be shown once (since system start), or periodically. The following shows an example where four records are to be displayed with a five second interval:

```
# vmstat 5 4
kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 0  0 18631   131   0   0   0   0    0   0 263  427 150  4  1 95  0
 0  0 18631   131   0   0   0   0    0   0 183  515 141  5  2 93  0
 0  0 18632   130   0   0   0   0    0   0 171  864 187  9  3 88  0
 0  0 18632   130   0   0   0   0    0   0 240  343 133  7  1 92  0
```

The most important information to look for and to be aware of is if there are any blocked processes (second column), if there is much paging going on, and if the CPU spends too much time waiting for I/O (last column) or user processes. There should not normally be any blocked processes, there should be no or very little paging activity and the CPU should, at least for some periods, report some idle time (second last column).

iostat

The iostat utility reports the statistics for CPU and I/O utilization for system device such as TTY, disks, CD, and so on. The following is an example for the usage of `iostat`, where two sets of statistics are to be displayed with a one second interval:

```
# iostat 1 2

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait
          0.0          0.0              10.3      2.3       86.0       1.4

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           0.5       1.5       0.3    1213930   3563372
hdisk1           0.8       3.5       0.3    1847210    564560
hdisk2           0.3       0.5       0.3    2128330    246621
cd0              0.0       0.0       0.0        400         0
```

```
tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait
          0.0        353.0               22.0     11.0       66.5       0.5

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           0.0       0.0       0.0          0         0
hdisk1           0.1       0.2       0.1       6249      1024
hdisk2           0.1       0.4       0.2      82213    123820
cd0              0.0       0.0       0.0          0         0
```

Important information that can be gathered from `iostat's` output is the amount (in KB), rate (in KBps), and distribution of disk I/O among several disks. Similar to the vmstat utility, the percentage of CPU utilization for the four categories of user, system, idle, and I/O wait is reported as well. Note that the first reported set of statistics shows the activities since system start, which explains the higher numbers. Ideally, the disk I/O should be approximately equally distributed among the disks over time, provided they perform about the same and have similar I/O throughput characteristics.

sar (System Activity Report)

The `sar` utility shows the statistics of the CPU activity and also supports reporting the statistics about paging, queuing, TTY I/O, and much more by using the corresponding options. The following example shows a report of the CPU utilization and paging activity (three records with a five second interval).

```
# sar -ur 5 3
AIX HTTPServer 3 4 003825774C00    11/30/98

12:53:16    %usr    %sys    %wio   %idle
           slots cycle/s fault/s  odio/s

12:53:21      47      10       0      42
           11147    0.00    3.60    0.00

12:53:26      13       9       0      78
           11147    0.00    0.20    0.00

12:53:31       7       6       0      87
           11147    0.00    0.00    0.00


Average       23       8       0      69
Average    11147       0       1       0
```

This sample report shows some application-related CPU activity, causing some page faults.

netstat

This is a useful command to check traffic on the network interface. It supports many options that let you select and tailor the reports. It can also display information about the error values for each network interface. For example:

```
# netstat -anI tr0
Name  Mtu   Network    Address            Ipkts Ierrs    Opkts Oerrs  Coll
tr0   1492  link#2     8.0.5a.b9.39.22    1026188     0    36284     0     0
                       c0:00:00:04:00:00
tr0   1492  9.3.1      9.3.1.176          1026188     0    36284     0     0
                       224.0.0.1
```

The example above shows some configuration values and traffic statistics for the Token-Ring interface `tr0`. Note that no errors are reported for this interface.

ps

Displays statistics about the processes running in the system, such as process ID, memory and CPU utilization. This is a very handy tool to verify the status and resource consumption of running processes. The following example displays an excerpt from an output of the `ps` command.

```
# ps -ef
    UID   PID  PPID  C    STIME   TTY  TIME CMD
   root     1     0  0   Nov 24     -  0:11 /etc/init
ausres4  2160  3384  0   Nov 24     -  0:58 /usr/dt/bin/dtsession
   root  2348     1  0   Nov 24     -  0:00 /usr/dt/bin/dtlogin -daemon
   root  3200     1  0   Nov 24     -  0:00 /usr/vice/etc/afsd -nosettime
   root  3384  2348  0   Nov 24     -  0:00 dtlogin <:0>        -daemon
   root  3750     1  0   Nov 24     -  0:23 /usr/vice/etc/afsd -nosettime
   root  3888     1  0   Nov 24     -  0:00 /usr/vice/etc/afsd -nosettime
   root  4148     1  0   Nov 24     -  3:02 /usr/sbin/syncd 60
   root  4390     1  0   Nov 24     -  0:00 /usr/lib/errdemon
  ...
```

Other valuable information about each process can be displayed using the many options that are supported by this utility. For example, `ps gv` gives you accurate information about each process' memory utilization.

PTX (Performance Toolbox)

PTX is a very powerful AIX product (not shipped with the basic operating system) which provides a large set of specialized performance tools to monitor the system environment and a collection of diverse tools for identifying performance problems and bottlenecks. The functions supported by PTX handle the performance monitoring on a system or network and provide the data in a variety of graphical formats. PTX also has a feature that

lets you monitor remote machines via a network, displaying all information on a single graphic screen.

A more detailed description of PTX is beyond the scope of this book and you should refer to the product documentation for more information.

Other Utilities

Besides the most commonly used utilities described above, there are some more tools that can be utilized, such as `svmon` for checking a real memory usage, `iptrace` to collect the data between nodes on network, `tprof` and `netpmon` to find out how much CPU time a process is using, and `filemon` for monitoring file system activity.

7.2.2 Web Server

A benchmarking tool is provided with the IBM HTTP Server package for webmasters to analyze how capable the IBM HTTP Server is running on the environment they had configured. This useful tool is called `ab` (which stands for ApacheBench) and it resides in the /usr/lpp/HTTPServer/sbin directory. The following shows the syntax and options supported to run this program:

Syntax:

```
ab [options] [http://]hostname[:port]/path
```

Options:

`-n requests`	Number of requests to perform
`-c concurrency`	Number of multiple requests to make
`-t timelimit`	Seconds to max. wait for responses
`-p postfile`	File containing data to POST
`-T content-type`	Content-type header for POSTing
`-v verbosity`	How much troubleshooting info to print
`-V`	Print version number and exit
`-k`	Use HTTP KeepAlive feature
`-h`	Display usage information (this message)

Below is an example of how webmasters can simulate 20 users, each requesting a CGI script called `test.cgi` 10 times using port 80:

```
# ab -n 200 -c 20 www.CompanyA.com:80/scripts/test.cgi
```

A sample output after running this command is shown next. Note that the example scenario chosen here involves a CGI script that, as we will explain in 10.2.3, “CGI Performance Considerations” on page 217, results in a considerably lower response rate as compared to serving static HTML pages.

```
This is ApacheBench, Version 1.2
Copyright(c)1996 Adam Twiss,Zeus Technology Ltd,http://www.zeustech.net/
Copyright(c) 1998 The Apache Group, http://www.apache.org/

Benchmarking www.CompanyA.com (be patient).........
Server Software:        IBM_HTTP_Server/1.3.3
Server Hostname:        www.CompanyA.com
Server Port:            80

Document Path:          /scripts/test.cgi
Document Length:        8749 bytes

Concurrency Level:      20
Time taken for tests:   6.769 seconds
Complete requests:      200
Failed requests:        0
Total transferred:      1930708 bytes
HTML transferred:       1872286 bytes
Requests per second:    29.55
Transfer rate:          285.23 kb/s received

Connnection Times (ms)
              min   avg   max
Connect:        2     4    29
Processing:   430   579  1050
Total:        432   583  1079
```

The most important results that webmasters are probably interested in are the following three items shown:

```
Complete requests:      200
Failed requests:        0
Requests per second:    29.55
```

The utility can best be used to evaluate the effect of a certain configuration change by comparing the reported numbers before and after the change. Webmasters can vary the values for the options to impost more rules for the data collection using the ab utility.

Note that the ab utility cannot be used to test resources that are SSL enabled due to different protocol handshaking issues.

7.3 Scalability for the IBM HTTP Server

In most cases, the load on a new Web server cannot be exactly estimated or calculated. Even if the load history of a Web server is well known, a new application, potentially targeting new users, can dramatically increase the load within a short period of time.

While tuning does increase the performance of a Web server, other methods and plans need to be prepared if the load grows beyond expectations. The term *scalability* generally addresses the issue of growing beyond the capabilities of a single server. If a server is not powerful enough for the load it should handle, there are basically two solutions: Upgrade the hardware to a more powerful system (for example by adding more processors to a multi-processor system) or add other systems and create some kind of a single-system image (for example by adding nodes to an RS/6000 SP system). The first solution does not require any special setup other than some possible configuration changes to optimize performance on the new system. For planning purposes, however, Web server hardware should be selected so there is room for upgrading should it become necessary. The latter solution does require some special considerations. Adding additional server machines provides a path for almost unlimited growth, but it creates some new obstacles to overcome. The most obvious concern is that multiple servers will have multiple IP addresses associated with them. Thus, a user only accesses a particular server by specifying its hostname in the URL of a request, unless special methods are implemented that let multiple systems appear as if they were one single system. Such methods are available that also incorporate some means of load balancing among the servers. Another issue is that multiple servers must be able to access the same data, especially when Web applications are involved.

The following sections describe these considerations and possible solutions. Please bear in mind that the following is only a brief introduction to these topics. For more details, please visit `http://www.software.ibm.com/enetwork`, or study the redbook *Load-Balancing Internet Servers*, SG24-4993.

7.3.1 Load Balancing

As mentioned above, a single machine may not be enough to support the total transaction volume. When multiple servers are being used to handle the load, it becomes necessary that they all share the load and that there must be a load balancing mechanism in place.

The two most commonly used methods for load balancing and single system imaging are Round-Robin DNS (RR-DNS) and specialized vendor products,

such as the IBM eNetwork Dispatcher. Both methods are discussed in the sections that follow.

7.3.1.1 Round-Robin DNS

Round-Robin DNS is a relatively simple method of load balancing. A DNS (Domain Name System) server provides name to address resolution and is always involved when a hostname is included in a URL. A RR-DNS server has the capability of resolving one single hostname into multiple IP addresses, such that requests for a single URL (containing a hostname) are actually pointed to different Web servers. Figure 32 shows the principles of Round-Robin DNS.

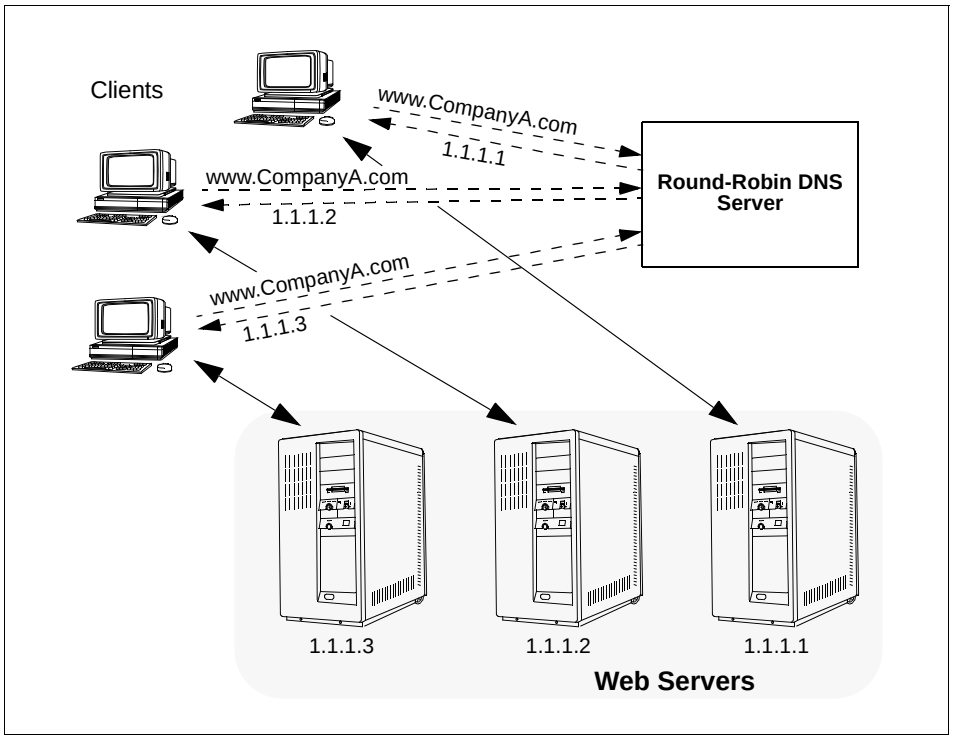


Figure 32. Principle of Round-Robin DNS

Clients in the upper left corner (see Figure 32) request a name resolution for the same hostname, but get different IP addresses to spread the load among the Web servers. In a simple configuration, the RR-DNS server will just cycle through the list of available servers. This works fine if the requests create about the same load on a server, and the servers have the same performance characteristics. This is, however, not real load balancing. Most actual

RR-DNS servers incorporate advanced algorithms for assigning addresses to names. Such algorithms may take into consideration a performance factor for each Web server or an actual status of the current load of each server. The advantage of RR-DNS is its easy implementation. The IBM eNetwork Dispatcher (see also the following section), together with a DNS server, can provide Round-Robin DNS services.

7.3.1.2 IBM eNetwork Dispatcher

The IBM eNetwork Dispatcher is a software running on IBM AIX and other platforms for the balancing of server load by routing TCP/IP session requests to different servers in a group of servers. This is accomplished by a function called advisors which query and evaluate the load of the servers on the network, and then transfer that information to the manager. The following figure, Figure 33, shows the eNetwork Dispatcher operation between the clients and the Web servers.

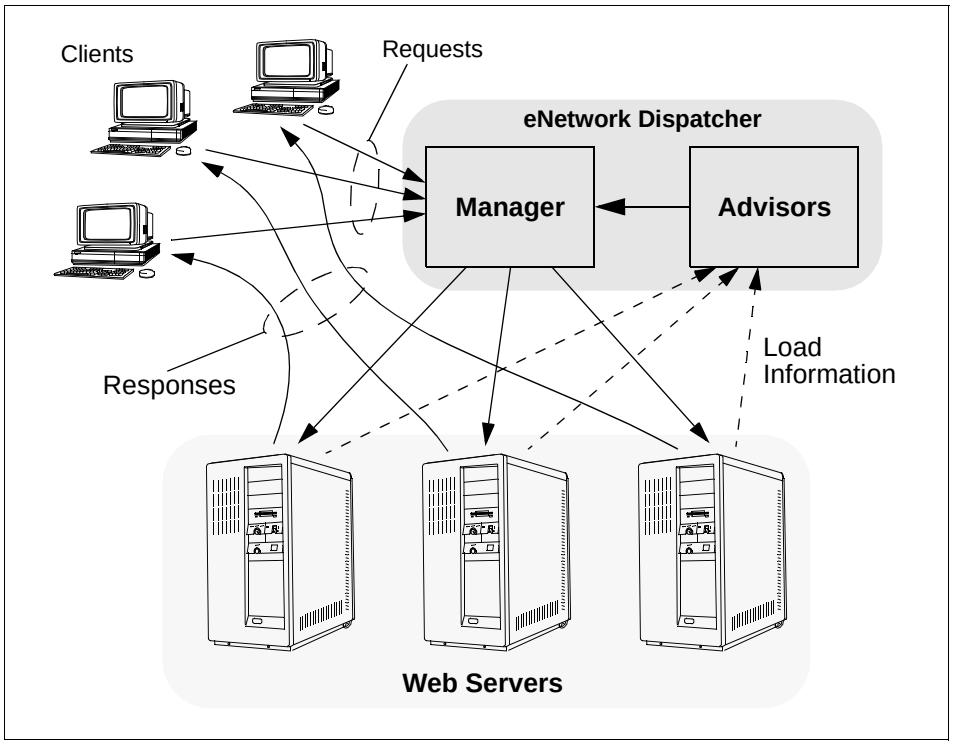


Figure 33. eNetwork Dispatcher Operation Flow

The manager, as part of the IBM eNetwork Dispatcher product, routes the clients' requests to the appropriate Web server based on current load

information received through the advisors. The Web servers process the requests and respond directly to the clients.

7.3.2 File Sharing

Once multiple Web servers are in place, providing a single-server image to process clients' requests, additional mechanisms must be implemented so that they all serve the same set of data. The most primitive (but not necessarily the worst) solution would be to distribute the data to each server's local disk(s) and keep them synchronized with simple file copy operations. In most cases, however, a shared filesystem is being used among the Web servers. Unfortunately, unless special provisions are being taken, a shared filesystem can add a new bottleneck if multiple servers access it at high rates. If that shared filesystem is being accessed through the same network interface as the Web traffic flows through, the total traffic easily doubles.

In the following three sections, solutions are briefly introduced and explained.

7.3.2.1 General Parallel File System (GPFS)

The General Parallel File System (GPFS) is a standards-based high performance, high-availability filesystem specifically designed for IBM RS/6000 SP systems. It features, among other aspects, client-side caching, read-ahead and write-behind, and large file block support. If multiple Web servers are implemented on an IBM RS/6000 SP system, the GPFS is the ideal filesystem for a high-performance Web server cluster. GPFS utilizes the high speed SP switch and does, therefore, not interfere with any other traffic on regular network interfaces, leaving the full bandwidth available for application traffic.

Besides the performance advantage, GPFS offers means for availability, such as data replication and self-recovery after error detection. It is, however, limited to an RS/6000 SP system environment and cannot be used among standalone RS/6000 systems.

More information about the GPFS can be found at `http://www.rs6000.ibm.com/software/sp_products`.

7.3.2.2 Distributed File System (DFS)

DFS is an advanced method for file sharing among different systems and even different system architectures. It overcomes most shortcomings of the popular NFS (Network File System) that is commonly used as a shared filesystem (see next section). DFS was developed from the popular Andrew File System (AFS) and is adopted and supported by the Open Software

Foundation (OSF) as a vendor-independent industry standard. DFS is very similar to AFS, and, with respect to Web servers and the discussion here, they can serve the same purpose with the same advantages over conventional file systems. In fact, AFS is part of the IBM WebSphere Performance Pack for the reasons mentioned here.

The major advantages of DFS are its scalability and performance. It has a unique protocol (on top of UDP) to manage the file and control information exchange between servers and clients. The superior performance results from its efficient, adaptive protocol and client-side caching. Scalability is achieved by its distributed database that stores and manages fileset location information. Users do not have to know the location of any file; DFS appears to them as one single file tree, no matter how large the file space is and how many DFS file servers there are. The DFS file tree is the same on every client that accesses DFS. Administrators manage the file space in units (called *filesets*) and assign them to individual DFS file servers. Moving filesets to different DFS file servers can be done online and even replicating filesets for improved availability is supported.

DFS for Personal Web Pages

DFS offers a tremendous advantage over other solutions in terms of shared file access. Assuming a Web server has access to the DFS file space, anyone can easily publish his or her own Web pages without any administrative overhead, such as file transfer to the Web server.

The IBM site in Austin, Texas, for example, runs a large DFS file space with more than 4 TB online storage. The site Web server, based on Apache, has access to this DFS file space, namely the user's home directories in DFS. Through the mechanism described in 5.3, "User Directories" on page 85, each user can publish his/her private Web pages just by saving files in a particular subdirectory of his/her home directory. There is no need for any file transfer or any other administrative action. As soon as the user saves a file, it is immediately available through the Web server to anybody. This is a very effective way for Web publishing when multiple authors are involved, especially when looking at the numbers. There are several thousand users with personal home directories in DFS in the IBM Austin site.

DFS file servers and clients are available from most major vendors for a variety of platforms. More information about DFS can be found at `http://www.transarc.com/Product/EFS/DFS`.

7.3.2.3 Network File System (NFS)

NFS is a technology developed by Sun Microsystems and shipped with AIX that provides the ability to transparently access the files on remote systems. The architecture is comprised of some layers such as XDR (external data representation) which is a common format used for data exchange between heterogeneous machines on a network, RPC (Remote Procedure Call) which is able to execute any procedure related to the process called on the remote machines, and UDP (User Datagram Protocol) which is a stateless data transfer protocol. As its use is relatively easy and it has been around for a while, NFS is being widely used. Although improvements are being introduced with a newer version, the traditional NFS implementation is limited in scalability and performance. Since a client needs to know the server where a file (or directory) is located, manageability, flexibility and recovery in large environments may become difficult. When multiple Web servers are operated using NFS as a shared filesystem, performance restrictions limit the overall size of such a solution.

Chapter 8. Building HTTP Server Modules

Extending the capabilities of the IBM HTTP Server can be accomplished with modules. This chapter provides a technical introduction to DSO modules. It also provides examples of how to build dynamically loadable modules from their source files. There is an exhaustive list of modules that people around the world have written for Apache, but to use these modules with the IBM HTTP Server, they will need to be built against the IBM HTTP Server header files. The module-build process will be shown as a common example, mod_info. This module extends the IBM HTTP Server such that its status can be queried using a Web browser and the appropriate URL.

8.1 The Programmer's View of DSOs

Section 2.2, "The Apache Server Model" on page 14 briefly introduced the DSO concept from the webmaster's point of view. A programmer, on the other hand, needs to know how DSO interoperates with the core Web server process. This section gives you a high-level introduction to how modules and the core server work together. Later sections of this chapter explain how such modules can be added to the IBM HTTP Server.

The Apache server has a *module table* containing *hooks* for the modules to attach to. Hooks are logical representations for the events when the server invokes program execution of the DSOs. In particular, the program code found in the modules dedicated to performing and reacting to these occasions, is executed. These program codes, or functions, are known as *handlers*, as illustrated in Figure 34 on page 178. Thus, at compile and link time, the handlers register themselves in the module table containing hooks so that the handlers and the hooks build the run-time connection between the IBM HTTP Server program and the modules. The HTTP server program then activates the respective DSO when necessary to leave the execution to the modules. This is all completely transparent to the clients (Web browsers and the users).

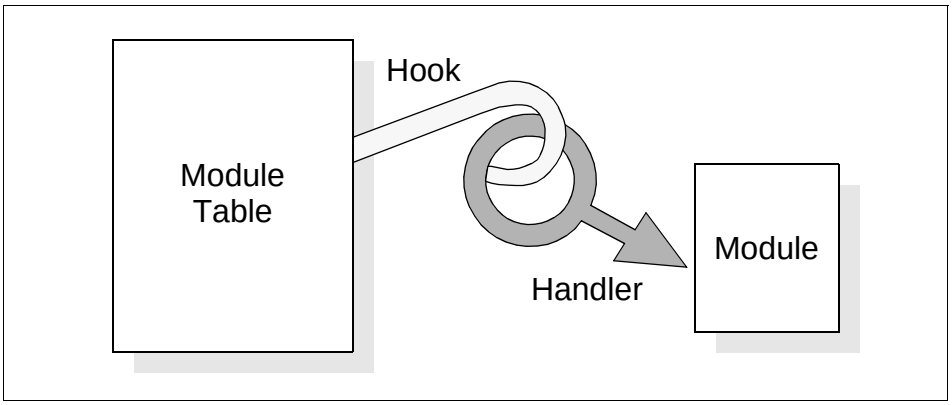


Figure 34. Hook and Handler Relation

As explained earlier, the modules register themselves into the module table when building (compiling/linking) the server core using a structure called *module*. Each module uses this structure to register its handlers for the hooks available in the server. As an example, the structure of the CGI module (mod_cgi) is shown below:

```
module MODULE_VAR_EXPORT cgi_module =
{
    STANDARD_MODULE_STUFF,
    NULL,                       /* initializer */
    NULL,                       /* dir config creater */
    NULL,                       /* dir merger,default override */
    create_cgi_config,          /* server config */
    merge_cgi_config,           /* merge server config */
    cgi_cmds,                   /* command table */
    cgi_handlers,               /* handlers */
    NULL,                       /* filename translation */
    NULL,                       /* check_user_id */
    NULL,                       /* check auth */
    NULL,                       /* check access */
    NULL,                       /* type_checker */
    NULL,                       /* fixups */
    NULL,                       /* logger */
    NULL,                       /* header parser */
    NULL,                       /* child_init */
    NULL,                       /* child_exit */
    NULL                        /* post read-request */
};
```

The Apache server currently supports no less than 18 different hooks (as can be seen in the module structure above) that modules can cling on in order to add some functionality to the server.

In the following section, each of these hooks is described in more detail. In addition, the respective standard modules registered to particular hooks are listed. Further discussion of each module can be found in 2.3, “Standard Modules in the Apache Server” on page 18.

Hooks, in general, can be grouped into two categories: those concerning the server’s environment, called the *Config Phase*, and those concerning the clients’ requests, called the *Request Phase*. The first six hooks discussed in the following section deal mainly with the server configuration and module initialization stage when the server starts up. In this stage, the server reads the appropriate configuration file(s) before it does any client request processing in the request phase. These configuration files include the httpd.conf and the .htaccess files. Following that, the modules become initialized since the server knows which of them are registered for the hooks in the module table.

8.1.1 Hooks for the Config Phase

Following is a list of the hooks (that apply to the server config phase) with their numbers, names, and a brief description:

Initialization (1) – The modules that cling to this hook are invoked after the server is configured and started in order to perform one-time setup steps of the environment at the module initialization process.

Create Directory Config (2) – There are two occasions when this hook invokes the modules that are registered to it. One occasion is during the configuration process when the server reads and processes the default setting for the main server’s directory configuration. The other occasion is during specific directory configuration, with reference to the directives defined in the .htaccess file or the server’s configuration file(s). In any case, if the server finds a module’s directive defined in these configuration files, the particular module that defines the directive is called to do the necessary configuration on the directory specified as the argument of the directive.

Merge Directory Configs (3) – This hook takes care of conflicts between directive usage in the parent and subdirectories. When the server hits such conflicts, the respective module is invoked and it resolves the conflicts to produce the most appropriate configuration for that directory. From then on, subsequent hooks will make use of this new configuration during the client’s request processing.

Create Server Config (4) – This hook invokes modules that perform configuration that affects the environment of the entire server. It is used, for

example, to set up the virtual hosts environment (more details on virtual hosts can be found in 5.1, “Virtual Hosts” on page 71).

Merge Server Configs (5) – Like the Directory Merger hook, this hook also fine-tunes and resolves any conflicts between servers.

Commands Table (6) – This hook points to a list of directives and their respective attributes defined in the modules. These attributes include the syntax, their default values, context, override flag, status, and so on. All this information is checked against those made in the configuration files during the configuration-reading process.

8.1.2 Hooks for the Request Phase

Most of the hooks described in the following section apply to the parsing and handling of clients’ requests. For better illustration of the client request parsing stage (see also 2.2.2, “Implementation of DSO in the Apache Server” on page 15) and the description of the hooks there, each request phase is shown in Figure 35 on page 181, along with a number tag attached to indicate their actual position in the module table.

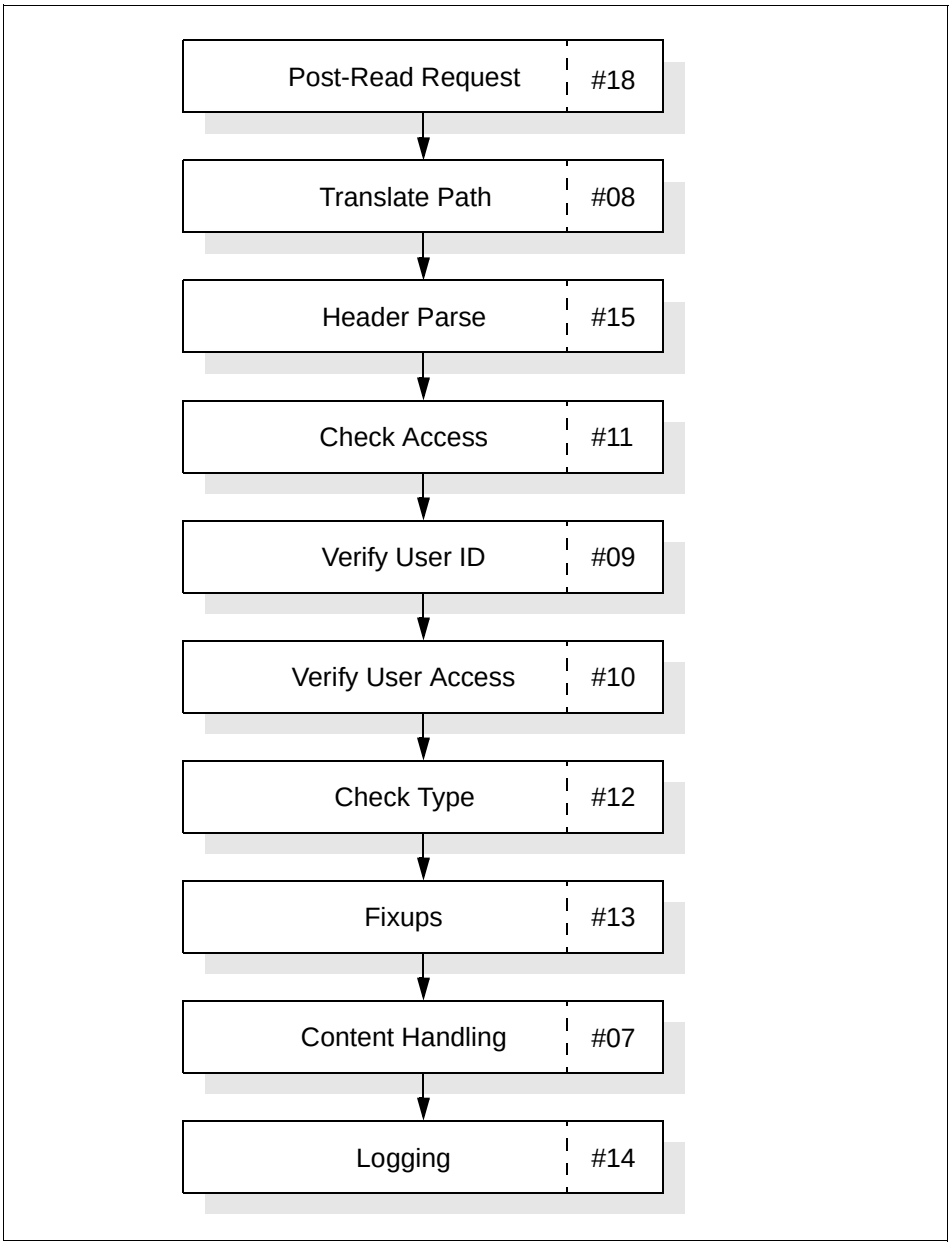


Figure 35. Client Request Parsing Process

Content Handling (7) – This hook points to a table containing the name and function of each handler, such that the server knows who to locate if the need

to perform a particular function arises later when serving the clients' requests. As the name implies, the functions here are responsible for massaging the content in the file based on the file properties.

Translate Path (8) – During the translation phase, the server calls any module registered for this hook in order to allow them to translate the URL into a filename. Once a translation is done, the server suppresses the rest of the requesting modules to prevent further redundant translation. However, if no module is interested in doing the translation, the core server translates it with reference to the *DocumentRoot* directive defined in the server configuration file(s).

Verify User ID (9) – This hook comes after the access checking phase (see #11 below) and checks the credentials of the users such as the user ID and password, against the authorization database defined in the server. The server stops processing other modules on this hook as soon as one module completes the validation. On the other hand, if no module performs this task, the server aborts the request with an error message sent to the client.

Verify User Access (10) – This is the last phase for any security verification to be done before the client's request is finally accepted. After knowing who is requesting, the server moves on to check whether the client has the access rights to obtain the requested document by comparing the credentials collected in the User Identification phase with the *Require* directive defined, for instance, in the .htaccess file in the specific directory (see also 6.2, "Basic Authentication" on page 118). Here again, there should be at least one module performing the validation, otherwise, the request is aborted and an error is returned.

Check Access (11) – This is the first phase where no credentials from the client are requested, but rather, based on information like the client's IP address, the server invokes modules to do a basic check of the client. The server returns a permission denial error message if any of the modules opposed the rights for this client.

Check Type (12) – After all the validation and verification of the client, the eligible client's request is passed on to the modules to determine the type of document requested. Thereafter, the one module that has completed the determination of the document type informs the core server so that the server can inform the client (Web browser) to act accordingly, for example to ask the user whether to save a file to disk or open for display.

Fixups (13) – Before the server processes the document and returns it to the client, the server offers this hook for any modules that wish to perform some

actions, such as making changes on the request or changing some environmental variables used in scripts.

Logging (14) – At this point, the client's request has already been handled, but this hook allows any modules to capture the events that happened throughout the request parsing process for logging or future references.

Header Parse (15) – The server invokes the modules registered to this hook to do a basic check at this early stage based on the request headers and translated filename. There is no standard module defined for this hook.

Child Init (16) – Modules registered for this hook are being called whenever a new child process is being spawned by the main process.

Child Exit (17) – This hook informs the registered modules before a child process terminates for them to perform necessary actions.

Post-Read Request (18) – This is the very first phase that checks the clients' requests after reading the request headers. This hook permits the invoked modules to make necessary decisions based on the raw request, but forbids them to make any modification at this point.

8.1.3 DSO Reference Lists

For your reference, the following two tables list the hooks as well as the respective standard modules registering to the hooks. Table 14 lists the hooks and modules applicable to the config phase.

Table 14. Module Matrix, Config Phase

Hook	Modules
Initialization (#1)	mod_log_agent, mod_log_config, mod_log_referer, mod_mime, mod_mime_magic, mod_rewrite, mod_status, and mod_unique_id.
Create Directory Config (#2)	mod_access, mod_actions, mod_alias, mod_auth, mod_auth_anon, mod_auth_db, mod_auth_dbm, mod_autoindex, mod_cern_meta, mod_digest, mod_dir, mod_expires, mod_headers, mod_imap, mod_include, mod_mime, mod_mime, mod_negotiation, and mod_usertrack.
Merge Directory Configs (#3)	mod_actions, mod_alias, mod_autoindex, mod_cern_meta, mod_dir, mod_expires, mod_headers, mod_imap, and mod_negotiation.
Create Server Config (#4)	mod_access, mod_alias, mod_cgi, mod_env, mod_headers, mod_info, mod_log_agent, mod_log_config, mod_log_referer, mod_mime, mod_mime_magic, mod_setenvif, mod_so, mod_speling, mod_userdir, and mod_usertrack.

Hook	Modules
Merge Server Configs (#5)	mod_alias, mod_cgi, mod_env, mod_headers, mod_info, mod_log_config, mod_mime_magic, and mod_setenvif.
Commands Table (#6)	mod_actions, mod_alias, mod_auth, mod_auth_anon, mod_auth_db, mod_auth_dbm, mod_autoindex, mod_cern_meta, mod_cgi, mod_digest, mod_dir, mod_env, mod_expires, mod_headers, mod_imap, mod_include, mod_info, mod_log_agent, mod_log_config, mod_log_referer, mod_mime, mod_mime_magic, mod_negotiation, mod_setenvif, mod_so, mod_speling, mod_status, mod_userdir, and mod_usertrack.

Table 15 lists the hooks and modules that apply to the request phase.

Table 15. Module Matrix, Request Phase

Hook	Modules
Content Handling (#7)	mod_actions, mod_asis, mod_autoindex, mod_cgi, mod_dir, mod_imap, mod_include, mod_info, mod_negotiation, mod_rewrite, and mod_status.
Translate Path (#8)	mod_alias, mod_rewrite, and mod_userdir.
Verify User ID (#9)	mod_auth, mod_auth_anon, mod_auth_db, mod_auth_dbm, and mod_digest.
Verify User Access (#10)	mod_auth, mod_auth_anon, mod_auth_db, mod_auth_dbm, and mod_digest.
Check Access (#11)	mod_access
Check Type (#12)	mod_mime, mod_mime_magic, mod_negotiation, and mod_rewrite.
Fixups (#13)	mod_alias, mod_cern_meta, mod_env, mod_expires, mod_headers, mod_negotiation, mod_rewrite, mod_speling, and mod_usertrack.
Logging (#14)	mod_log_agent, mod_log_config, and mod_log_referer.
Header Parse (#15)	–
Child Init (#16)	mod_rewrite and mod_unique_id.
Child Exit (#17)	mod_log_config.
Post-Read Request (#18)	mod_setenvif and mod_unique_id.

8.2 The Apache Information Module (mod_info)

The Apache Information Module (mod_info) provides a comprehensive view of the server configuration, including the modules that have been activated and certain configuration directives from the httpd.conf file. One of the great benefits of mod_info is that it provides the webmaster with the configuration parameters for each module. If, for example, you would like to know which URL Aliases you have defined for your server, you could query mod_info, which will query mod_alias behind the scenes. The output of mod_info provides you with the active Aliases for your currently running IBM HTTP Server. Figure 36 on page 186 is a sample output produced by mod_info. Note the scroll bar to the right, which is indicating that there is much more information available than is shown in the figure. Especially interesting (at least as long as you are not very familiar with the individual modules) is the list of supported directives per module. The output also lists the settings of all configuration directives and containers that are not pertinent to any specific module, such as ServerRoot, DocumentRoot, or any Directory directive.

Once mod_info is imbedded in the IBM HTTP Server, the information page, as shown in Figure 36 on page 186, can simply be displayed by appending *server-info* the Web server's root URL. Note that this may expose information about the Web server that is not meant to be available to anybody. Thus, mod_info is not normally available on production Web servers.

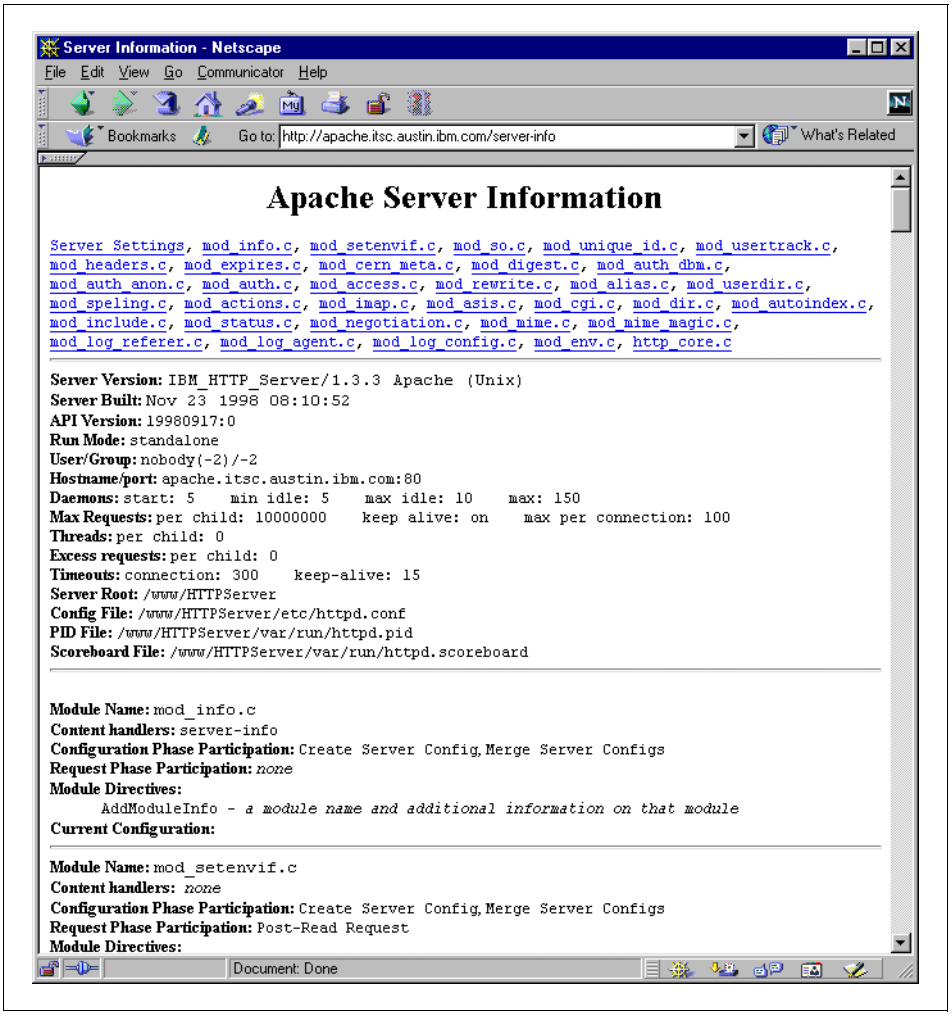


Figure 36. Sample Output from mod_info

8.2.1 Building the Apache Information Module (mod_info)

Since the Information Module does not come standard with the IBM HTTP Server, you will need to acquire the mod_info.c file. There are some security concerns to exposing the configuration of your server to others, thus the reason for not including mod_info in the standard IBM HTTP Server distribution.

In order to build the mod_info module (or any other modules), you will need to have an ANSI C compiler installed on the system.

Here are the steps for acquiring and building mod_info:

1. Change directories to the root directory of the IBM HTTP Server install (/usr/lpp/HTTPServer). Then, change to the apachesrc subdirectory.
2. Unzip the apache_tar.gz file using gunzip, which will create a new file apache_tar.

   If you do not have the GNU zip/unzip utilities, you may download the GNU source code for gzip/gunzip from many sources, such as, for example: `http://sunsite.unc.edu/pub/gnu/`, or the installable binaries from `http://www-frec.bull.com/`.
3. Then execute the command `tar -xvf apache_tar` to unpack the files in the tar file.
4. Change directories to the newly created apache-1.3/src/modules/standard directory.
5. From within this directory (see the last step), copy the mod_info.c file to the directory /usr/lpp/HTTPServer/example_module.
6. Change current directory to /usr/lpp/HTTPServer/example_module.
7. Edit the Makefile.exmpl file. Locate the line below and change it from:

   ```
   SRCFILE=mod_example
   ```

   to:

   ```
   SRCFILE=mod_info
   ```
8. Within the current directory, copy the file mod_example.exp to mod_info.exp.
9. Then edit mod_info.exp and change example_module to info_module.
10. Execute the command:

    ```
    # make -f Makefile.exmpl
    ```

    which will compile the module and produce the mod_info.so file. Make sure that the compilation did not abort with any errors.
11. Copy the newly built mod_info.so file to the /usr/lpp/HTTPServer/libexec directory.
12. Edit the httpd.conf file (in /usr/lpp/HTTPServer/etc unless otherwise configured) and add the excerpt below at the bottom of the file:

    ```
    <Location /server-info>
       SetHandler server-info
       Order deny,allow
       Deny from all
    ```

```
   Allow from <your domain>
</Location>
```

Replace <your domain> with your DNS domain name, or add any other access restrictions you wish.

Although a domain name can be used, for maximum security, specific IP addresses should be used. An example allowing access to only two specific IP addresses (1.1.1.1 and 1.1.1.2) would be:

```
<Location /server-info>
   SetHandler server-info
   Order deny,allow
   Deny from all
   Allow from 1.1.1.1 1.1.1.2
</Location>
```

13. In the httpd.conf file, use the LoadModule directive to load the newly built mod_info.so, and the AddModule directive to enable the mod_info module. The easiest way to accomplish this would be to copy the last LoadModule line in the httpd.conf file, and insert the LoadModule line for mod_info after it. Note that the AddModule directive must go below the ClearModuleList directive, since the IBM HTTP Server uses the ClearModuleList directive to prioritize the ordering of modules. To accomplish adding the AddModule line, place the AddModule line for mod_info after the last current AddModule line.

    An Example of this would be:

    ```
    # Other LoadModule lines here
    LoadModule info_module libexec/mod_info.so

    ClearModuleList
    # Other AddModule lines here
    AddModule mod_info.c
    ```

14. If your server is already running, then you can restart it with the command:

    ```
    # apachectl restart
    ```

    If the IBM HTTP Server is not running, you can start it with the command:

    ```
    # apachectl start
    ```

    (The `apachectl` executable is located in /usr/lpp/HTTPServer/sbin.)

15. If access has been granted to the client you are using with the above Allow directive, then you should then be able to request the URL `http://<your server>/server-info`. Make sure to replace `<your server>` with the hostname of your Web server.

16. You will then see the server configuration information about the loaded modules and their configuration parameters, as shown in Figure 36 on page 186.

    If, for some reason, you do not get the expected mod-info output page, then things to check are:

    - You should be accessing the server from a machine that has been granted access by the Allow directive in the httpd.conf file.
    - If you get any error messages when you start the server on the command line, or if the error log file (the default file name is /usr/lpp/HTTPServer/error_log) contains any errors, then these might give you a better idea of the problem. If the error message mentions something along the nature of an unrecognized handler, then you might have a typing error in the SetHandler line, or else you did not include both the LoadModule and AddModule directives in the httpd.conf file.

Chapter 9. Migration Considerations

This chapter provides guidelines for users who would like to migrate from their current Web server to the IBM HTTP Server. The descriptions that follow cover the migration from the IBM Internet Connection Secure Server (ICSS), Lotus Domino Go, and Netscape Communication Corporation's FastTrack and Enterprise Servers. Migration from other Web server products might be similar to those mentioned above. It is, however, almost impossible to explain every aspect of a migration, and thus, only the commonly used features are described in detail later.

Before we go into the details in the subsequent sections, some of the most obvious differences shall be listed here. Almost all commercial Web servers, including the IBM Internet Connection Secure Server, Lotus Domino Go Webserver, Netscape FastTrack and Netscape Enterprise Servers, have remote administration features with graphical user interfaces. For the IBM HTTP Server, such a feature might be available in a future release. A proxy function is available only as an add-on module to the IBM HTTP Server that is not included in the standard package. The Lotus Domino Go Webserver has powerful Java-based log reporting tools, a Java Servlet engine, SNMP management, a text search engine, multithreading, multiple language support, and more. These features are not (yet) included in the current version of IBM HTTP Server.

The main advantages of the IBM HTTP Server in comparison with other commercial Web servers are its modular structure, its industry standard open architecture (including the availability of source code), its flexible configuration, and numerous skilled specialists. Also, the IBM HTTP Server supports virtual hosts, which Lotus Domino Go and ICSS do not.

9.1 IBM ICSS and Lotus Domino Go Webserver

This section covers the migration from the IBM Internet Connection Secure Server (ICSS) and the Lotus Domino Go Webserver (DGW) together, because they have almost identical features (and the same code base).

9.1.1 Installation

You can install and run the IBM HTTP Server on the same machine that is running the IBM ICSS or Lotus DGW, as long as you keep them on different IP ports. For example, you can run Lotus DGW on the default port 80 and, at the same time, evaluate the IBM HTTP Server on port 8080. After completion

of the migration, you can stop (and eventually uninstall) Lotus DGW and switch the IBM HTTP Server to the default port.

9.1.2 Directory Structures

This section presents a brief comparison of the IBM HTTP Server and IBM ICSS/Lotus DGW default directory structures. You might find this information in Table 16 helpful when moving files between the servers or when adapting configuration parameters.

Table 16. Directory Structure (Comparison)

File, Directory	IBM HTTP Server	IBM ICSS, Lotus DGW
server root	`/usr/lpp/HTTPServer`	`/usr/lpp/internet/server_root`
binaries	`/usr/lpp/HTTPServer/sbin`	`/usr/sbin`
configuration	`/usr/lpp/HTTPServer/etc/httpd.conf`	`/etc/httpd.conf`
logs	`/usr/lpp/HTTPServer/var/log`	`/usr/lpp/internet/server_root/logs`
document root	`/usr/lpp/HTTPServer/share/htdocs`	`/usr/lpp/internet/server_root/pub`
CGI programs	`/usr/lpp/HTTPServer/share/cgi-bin`	`/usr/lpp/internet/server_root/cgi-bin`

9.1.3 Basic Configuration

Although the IBM HTTP Server's and IBM ICSS/Lotus DGW's configuration files have the same names, they use a slightly different syntax. The IBM HTTP Server configuration file is more complex and flexible. In this and in the following sections we will provide some guidelines about how to transfer Web server configuration information from the IBM ICSS or Lotus DGW to the IBM HTTP Server.

The most common configuration directives are almost the same for the subject Web servers. The following table, Table 17, lists the most important of these directives.

Table 17. Basic Directives (Comparison)

Directive Description	IBM HTTP Server	IBM ICSS, Lotus DGW
Directory where the server is installed	`ServerRoot`	`ServerRoot`
To do DNS lookups or not	`HostNameLookups`	`DNS-Lookup`
User ID under which the server runs	`User`	`UserID`
Group ID under which the server runs	`Group`	`GroupID`
Fully-qualified domain name of server	`ServerName`	`HostName`

Directive Description	IBM HTTP Server	IBM ICSS, Lotus DGW
Port on which the server is listening	`Port`	`Port`
IP addresses on which the server is listening	`BindAddress`	`BindSpecific`
File containing the process ID of the server	`PidFile`	`PidFile`
Name of users' public html directory (this feature is optional for the IBM HTTP Server)	`UserDir`	`UserDir`
Log file for request logging (this feature is optional for the IBM HTTP Server)	`TransferLog`	`AccessLog`
Log file for error logging	`ErrorLog`	`ErrorLog`
Default file name for requests that include only a directory part in the URL (this is an optional feature of the IBM HTTP Server)	`DirectoryIndex`	`Welcome`
Custom error messages	`ErrorDocument`	`ErrorPage`
Server administrator's e-mail address	`ServerAdmin`	`WebMasterEmail`

More details about each of the directive mentioned above can be found in the documentation of the corresponding Web server, or at the appropriate places in this redbook.

9.1.4 Request Mapping

The request mapping syntax is quite different between the IBM HTTP Server and the IBM ICSS/Lotus DGW Servers. The core part of the IBM HTTP Server has only one directive for request mapping: *DocumentRoot*. It is equivalent to *Pass /** in the IBM ICSS/Lotus DGW configuration. Equivalents of other request-mapping directives can be implemented using IBM HTTP Server modules.

The module mod_alias provides equivalents to the `Pass /something*`, `Redirect`, and `Exec` directives. Some examples of these directives follow.

The IBM ICSS/Lotus DGW directive

```
Pass /documentation* /webserver/documentation*
```

should be replaced by the following IBM HTTP Server directive

```
Alias /documentation /webserver/documentation
```

The IBM ICSS/Lotus DGW directive

```
Redirect  /other/*  http://www.other.org/*
```

should be replaced by
```
Redirect  /other     http://www.other.org
```

The IBM ICSS/Lotus DGW directive
```
Exec /cgi-bin/* /webserver/cgi-bin/*
```
should be replaced by
```
ScriptAlias /cgi-bin/ /webserver/cgi-bin/
```

The IBM HTTP Server module mod_rewrite allows you to migrate the Map and Fail directives. The usage of module mod_rewrite is quite complex and you should use it with caution. Below are some simple examples of how to migrate `Map` and `Fail` directives.

First, you must enable rewriting with the directive
```
RewriteEngine on
```

Then, use the IBM ICSS/Lotus DGW directive
```
Map /other*  /another*
```
should be replaced by
```
RewriteRule ^/other(*)  /another$1 [PT]
```

The IBM ICSS/Lotus DGW directive
```
Fail /expired/*
```
should be replaced by
```
RewriteRule ^/expired/* - [F]
```

The proxy module is not included in current version of IBM HTTP Server (see 3.1, “Product Contents” on page 33) so we do not describe migration of the proxy functionality.

9.1.5 Virtual Hosts

The virtual-hosting syntax of the IBM HTTP Server is even more different from IBM ICSS/Lotus DGW than request mapping. But it is not difficult to migrate the virtual hosts configuration to the IBM HTTP Server. The IBM ICSS/Lotus DGW syntax requires the administrator to put an IP address or server name at the end of the request mapping directive. On the other hand, the IBM HTTP Server uses the special section <VirtualHost> to describe the virtual host configuration. So the rule of thumb is:

- If there is an IP address or host name at the end of a request mapping directive, migrate this directive to an appropriate <VirtualHost> section.
- In all other cases, migrate it to the basic part of the configuration file.

For example, the following IBM ICSS/Lotus DGW configuration

```
Pass /* /webserver/CompanyA/* 1.2.3.4
Pass /* /webserver/CompanyB/* 1.2.3.5
Pass /* /webserver/default/*
```

should be replaced by the following fragment in IBM HTTP Server configuration

```
DocumentRoot /webserver/default
<VirtualHost 1.2.3.4>
DocumentRoot /webserver/CompanyA
   ...
</VirtualHost>
<VirtualHost 1.2.3.5>
DocumentRoot /webserver/CompanyB
   ...
</VirtualHost>
```

The IBM HTTP Server has more advanced virtual hosting features than the IBM ICSS/Lotus DGW Server, such as separate log files, access restrictions, and error messages for each virtual host. You might want to take advantage of them when migrating.

9.1.6 Authentication and Access Control

The IBM HTTP Server has a number of authentication modules, but none of them is directly compatible with the IBM ICSS or Lotus GWS Servers. You must select the IBM HTTP Server authentication module that suites the situation best and manually transfer all user information. The most similar authentication to IBM ICSS/Lotus GWS (so called basic authentication) is supported by mod_auth module. To manage user information, use the IBM HTTP Server utility `htpasswd` (see 6.2.3, “Authentication Files and Databases” on page 124).

The access-control features are quite similar among all subject Web servers, but the syntax is so different that it is very difficult to do corresponding configuration.

The following example will give you some ideas about how to migrate authentication and access-control configuration. The IBM HTTP Server modules mod_auth and mod_access are required for that configuration.

The following fragment of an IBM ICSS or Lotus GWS configuration file:

```
Protect /staff/* {
   ServerID StaffOnly
   Authtype Basic
```

```
   PasswdFile /webserver/security/users1
   GroupFile /webserver/security/groups1
   GetMask All@(1.2.3.*)
   PostMask All@(1.2.3.*)
   PutMask management,accounting
}
```

can be replaced by this fragment for the IBM HTTP Server:

```
<Location /staff>
   AuthName StaffOnly
   AuthType Basic
   AuthUserFile /webserver/security/users2
   AuthGroupFile /webserver/security/groups2
   <Limit GET POST>
      Order deny,allow
      Deny from all
      Allow from 1.2.3
      require valid-user
      Satisfy all
   </Limit>
   <Limit PUT>
      Order deny,allow
      Allow from all
      require group management accounting
   </Limit>
</Location>
```

Take into account that password and group file formats are incompatible between the Web servers and need to be migrated or recreated manually.

Access-control restrictions in the configuration file can be overwritten by special access control files in specific directories. To enable that feature, the IBM ICSS/Lotus GWS uses the ACLOverride On directive and .www_acl files. The corresponding directive in the IBM HTTP Server configuration is AllowOverride AuthConfig Limit and it uses .htaccess files.

9.1.7 Logging and Reporting

The IBM ICSS and Lotus GWS support separate log files for access, error, agent, cache access, and references logging. They also have a rich set of directives for log files handling and reporting. A major advantage of the IBM HTTP Server is the possibilities to create custom log files and have separate log files for each virtual host. The IBM HTTP Server log file handling and reporting could be done by third party utilities that can be found on the Internet.

The IBM HTTP Server supports basic error logging and standard modules for other logs: mod_log_config, mod_log_agent, and mod_log_referer. The module mod_log_config supports the TransferLog directive, which creates *Common Log Format* log files that are used by almost any Web server, including the IBM ICSS and Lotus GWS.

The following example shows how to create custom report files that are equivalent to those created by the IBM ICSS/Lotus GWS reference and agent log files.

```
CustomLog /webserver/logs/referer_log "%t \"%{Referer}i\""
CustomLog /webserver/logs/agent_log "%t \"%{User-agent}i\""
```

9.1.8 Web Applications

There are many ways to deploy Web applications using different technologies (see also Chapter 10, “Web Applications” on page 213). The following sections briefly overview the most important of them and the implications on migration.

9.1.8.1 CGI Scripts

CGI scripts depend mostly on environment variables that they get from the Web server. Most basic environment variables are the same for all Web servers. Thus, if you do not use some exotic environment variables, your CGI scripts should continue to work on the IBM HTTP Server without problems. Also, remember that the IBM HTTP Server allows you to define environment variables unconditionally with the PassEnv and SetEnv directives supported by the mod_env module, and conditionally with the SetEnvIf and BrowserMatch directives supported by the mod_setenvif module.

FastCGI scripts are also supported through the optional module mod_fastcgi (see 10.2.3, “CGI Performance Considerations” on page 217).

9.1.8.2 Server-Side Includes

Server-side includes on the IBM HTTP Server are supported by the mod_include module. You can enable .shtml file parsing by using the Options +Includes directive in the corresponding <Directory> section.

The IBM HTTP Server and the IBM ICSS/Lotus GWS support the same syntax for server-side includes. The feature sets differ only very little so you should not have serious problems during migration.

9.1.8.3 Image Maps

Image maps are supported by the IBM HTTP Server through the mod_imap module. To enable image maps, use the AddHandler imap-file map directive.

Although the map-file syntax and image-map handling are different in the IBM HTTP Server, it is very easy to migrate from IBM ICSS/Lotus GWS:

- Change all map files extensions to .map.
- Change all references in HTML documents from `HREF="/cgi-bin/htimage/somewhere/mapfile.txt"` to something similar to this: `HREF="/somewhere/mapfile.map"`
  (in this example `/somewhere/mapfile.map` is the map file name and location).
- Edit all map files to adjust the syntax:
  - Change the `rectangle` and `polygon` directives to `rect` and `poly`.
  - Move each URL in front of the coordinates.
  - Remove parentheses from the coordinates.
  - Change the radius of circles to coordinates of any point on the circle, which could most easily be done by adding the radius to one of the center coordinates.

The following example illustrates the changes that must be done.

This is an excerpt of an original file mapfile.txt for the IBM ICSS/Lotus DGW:

```
default                          http://www.CompanyA.com
rectangle (50,25) (80,75)        http://www.CompanyA.com/sales
circle    (100,130) 20           http://www.CompanyA.com/products
polygon   (10,10) (40,20) (5,15) http://www.CompanyA.com/support
```

This is the edited and renamed file mapfile.map for the IBM HTTP Server:

```
default http://www.CompanyA.com
rect    http://www.CompanyA.com/sales    50,25 80,75
circle  http://www.CompanyA.com/products 100,130 100,150
poly    http://www.CompanyA.com/support  10,10 40,20 5,15
```

9.1.8.4 API Programs

The IBM HTTP Server API is different from the APIs supported by the IBM ICSS and Lotus GWS. It is, therefore, not possible to easily migrate programs written at the API level since more than simple-call translation might be involved.

9.2 Netscape FastTrack and Enterprise Server

Since the Netscape FastTrack Server and the Netscape Enterprise Server provide GUI-administration, it is sufficient for some webmasters to rely on them completely without ever editing the flat configuration files. The IBM HTTP

Server, running on IBM AIX, at present only uses a flat file for configuration of the Web server and does not support an administration GUI. Thus, the migration dwells mainly on the content found in the flat files. The migration from the Netscape FastTrack Server and the Netscape Enterprise Server to the IBM HTTP Server is covered together because they have almost identical features in terms of configuration. The term *Netscape Web servers* is used in the following to represent the two servers as one single entity throughout the discussions.

In the sections that follows, we focus on some of the considerations in the areas of configuration directives that require manual editing, the directives used in the dynamic configuration files of the Netscape Web servers (using .nsconfig file) and those that are directly compatible in both servers (using the .htaccess file). Note that the .htaccess file is only supported beginning from Netscape FastTrack Server Version 3.01 and Netscape Enterprise Server Version 3.0.

9.2.1 Installation

You can install and run the IBM HTTP Server on the same machine that is running a Netscape Web server as long as you keep them on different IP ports. For example, you can run a Netscape Web server on the default port 80 and, at the same time, evaluate and configure the IBM HTTP Server on port 8080. After some time, you can sunset the Netscape Web server and, at the same time, switch the IBM HTTP Server to the default port.

9.2.2 Directory Structures

This section briefly compares the IBM HTTP Server's and Netscape Web servers' default directory structures. However, webmasters should be aware that the Netscape Web servers run an admin server to administer and configure the Web servers. Thus, the configuration files may span across different directories. The admin server has a directory called /usr/netscape/suitespot/admin-serv/config that was created as a default (note that another path might be specified during installation). This directory stores the configuration files. The main configuration file is called ns-admin.conf. The actual Web server (assuming only one) is configured after installation of the admin server and its configuration files are located in a directory called httpd-<server_name>/config, where <server_name> is the name of the Web server given at the time of installation. The two main configuration files for

each Netscape Web server are the magnus.conf and obj.conf, which are more related to the topic of discussion here.

Table 18. Directory Structure (Comparison)

File, Directory	IBM HTTP Server	Netscape Web Servers
server root	/usr/lpp/HTTPServer	/<ns-home>
binaries	/usr/lpp/HTTPServer/sbin	/<ns-home>/bin
configuration	/usr/lpp/HTTPServer/etc/httpd.conf	/<ns-home>/admin-serv/config, and /<ns-home>/httpd-<server_name>/config
logs	/usr/lpp/HTTPServer/var/log	/<ns-home>/httpd-<server_name>/logs
document root	/usr/lpp/HTTPServer/share/htdocs	/<ns-home>/docs
CGI programs	/usr/lpp/HTTPServer/share/cgi-bin	anywhere

Note: The <ns-home> directory listed in the right column of Table 18 above can be chosen upon installation. By default, it is either /usr/netscape/suitespot or /usr/ns-home.

9.2.3 Basic Configuration

This section focuses on the directives supported by the Netscape Web servers specified in the .nsconfig file. A tool provided to convert these files into .htaccess files in order to ease migration to the IBM HTTP Server is also covered.

Table 19. Basic Directives (Comparison)

Directive Description	IBM HTTP Server	Netscape Web Servers
Directory where the server is installed	ServerRoot	ServerRoot
To do DNS lookups or not	HostNameLookups	DNS
User ID under which server runs	User	User
Fully-qualified domain name of server	ServerName	Hostname
Port on which server is listening	Port	Port
Hostname that is sent back to client	ServerName	ServerName
IP addresses on which the server is listening	BindAddress	Address
File containing the process ID of the server	PidFile	PidLog

Directive Description	IBM HTTP Server	Netscape Web Servers
Name of users' public html directory (this feature is optional in IBM HTTP Server)	`UserDir`	`fn=unix-home (obj.conf)`
Log file for requests logging (this feature is optional in IBM HTTP Server)	`TransferLog`	`access (obj.conf)`
Log file for error logging	`ErrorLog`	`ErrorLog`
Default file name for requests that include only a directory part in the URL (this is optional feature of IBM HTTP Server)	`DirectoryIndex`	`fn=find-index (obj.conf)`
Custom error messages	`ErrorDocument`	`ErrorFile (.nsconfig)`

Table 19 on page 200 shows some basic directives found in both servers, mainly in the httpd.conf (IBM HTTP Server) and magnus.conf (Netscape Web servers). It can be seen that the Netscape Web servers may not have some directives defined, but it does cater the option for other representation in files like obj.conf and .nsconfig.

The list below illustrates the list of directives supported in the .nsconfig file:

AddType Assigns encoding to file extensions.

ErrorFile Assigns error messages other than default.

RequireAuth Performs user authentication using a userfile.

RestrictAccess Applies access control to resources.

A tool is provided with the Netscape Enterprise Server that converts an .nsconfig file into a .htaccess file (see description below). The following list shows the supported directives in the converted .htaccess file:

AuthName The authorization realm name for a directory.

AuthType The authentication type for that directory; only supports Basic type whereas for IBM HTTP Server, Basic and Digest are supported.

AuthUserFile The text file containing the usernames and encrypted password.

AuthGroupFile The text file containing the groups and members.

Limit Applied onto a method specified based on the following access control directives—*order*, *deny*, *allow*, and *require*.

Converting existing .nsconfig files to .htaccess files

As mentioned earlier, the Netscape Enterprise Server provides a tool to perform the conversion of .nsconfig files to .htaccess files. To run this script (known as htconvert), enter these parameters at the command prompt:

```
# <path to Perl> <path to htconvert> <path to obj.conf>
```

This script converts all .nsconfig files to .htaccess files. Perl must also be installed in order to run this. For more information, please refer to the administrator guides or the documentation for the respective Netscape Web server.

9.2.4 Request Mapping

Generally speaking, there are three concerns when it comes to mapping requests of directories from a specified URL request and most Web servers provide features to serve these concerns. In this section, the types of mapping are discussed and the implementations on both the Netscape Web server and IBM HTTP Server.

9.2.4.1 URL Mapping (Aliasing)

Aliasing is the mapping of a document directory's physical pathname to a defined alias so that the files can be referenced using that alias name. This is most often used for security purposes. The Netscape Web servers offer a GUI administration tool to facilitate the configuration of this feature. The resulting information can then be found in the obj.conf file as shown below:

```
<Object name="default">
  ...
  NameTrans fn="pfx2dir" from="/admin" dir="/webserver/admin"
  ...
</Object>
```

The IBM HTTP Server uses the Alias directive to perform the same function:

```
Alias /admin /webserver/admin
```

9.2.4.2 Forwarding URLs (Redirection)

Redirection is the mapping of a URL to another URL on the same server or to other servers. The Netscape Web servers offer the capability to map to a URL prefix or to a fixed URL. The implementation of both is very similar, as can be seen in the excerpt from the obj.conf file below:

```
<Object name="default">
  ...
  NameTrans fn="redirect" from="/other" url-prefix="http://www.other.org/"
  ...
```

```
  NameTrans fn="redirect" from="/other2" url="http://www.other.org/
  test/other2"
  ...
</Object>
```

This is how it is implemented in the IBM HTTP Server using the Redirect directive:

```
Redirect  /other     http://www.other.org
Redirect  /other2    http://www.other.org/test/other2
```

9.2.4.3 CGI Directory and CGI File Type

CGI directories and CGI file types are the two methods used by the server to identify CGI programs on the server. The Netscape Web servers allow a webmaster to indicate CGI programs either by specifying a directory that contains only CGI programs or by specifying CGI programs that belong to a certain file type. For the former method, all files present in the specified directory are considered CGI programs regardless of the file extensions, whereas in the latter method, programs can be in any directory, but must have a specific file extension, usually .cgi, .exe, or .bat.

Using a CGI Directory

The following shows the configuration settings in the obj.conf file for the assignment of a CGI directory for CGI scripts:

```
<Object name="default">
   ...
   NameTrans fn="pfx2dir" from="/cgi-bin" dir="/webserver/cgi_scripts"
   name="cgi"
   ...
</Object>

<Object name="cgi">
   ObjectType fn="force-type" type="magnus-internal/cgi"
   Service fn="send-cgi"
</Object>
```

The IBM HTTP Server uses the ScriptAlias directive for the equivalent implementation in the httpd.conf file as shown below:

```
ScriptAlias /cgi-bin/ /webserver/cgi_scripts/
```

Using CGI File Type

The following shows the configuration settings in the obj.conf file for the assignment of a CGI directory conforming to the file types mentioned above:

```
<Object ppath="/webserver/cgi-filetype/*">
```

```
   Service fn="send-cgi" type="magnus-internal/cgi"
</Object>
```

The IBM HTTP Server offers more flexibility in defining any file types to be CGI programs, as compared to the Netscape Web servers that restrict it to only the three file types mentioned above. However, webmasters are reminded of the effects of specifying file types as CGI programs, such as .exe file extensions, that might be provided for download, rather than execution. Also any non-CGI files with those extensions will result in errors when the server processes them as CGI programs.

The IBM HTTP Server uses a directive called AddHandler in the httpd.conf file for the specification of file types with extensions .cgi to be run as CGI programs. Ensure that the execution of CGI programs are enabled using the Options directive, as shown below with the AddHandler directive:

```
Options ExecCGI
AddHandler cgi-script cgi
```

9.2.5 Virtual Hosts

There are three ways in which the Netscape Web servers can be configured for hosting multiple Web servers. These three ways are briefly introduced below. Note that for the Netscape Web servers, the definitions for the virtual servers or hosts can be found in the file obj.conf or the magnus.conf in the directory /usr/netscape/suitespot/httpd-ServerA/config.

9.2.5.1 Hardware Virtual Servers

Hardware virtual servers is the mapping of multiple IP-addresses to Web servers that share the same configuration information. The following shows how they are defined in the configuration file obj.conf:

```
<Object name="default">
   ...directives...
   NameTrans fn="document-root" address="1.2.3.4" root="/CompanyA"
   ...directives...
</Object>
```

The equivalent notation for hardware virtual servers in the IBM HTTP Server is IP-based virtual hosts (see also 5.1, “Virtual Hosts” on page 71) and the following clip from the configuration file shows how it is implemented:

```
<VirtualHost www.CompanyA.com>
    ServerName www.CompanyA.com
    DocumentRoot /webserver/CompanyA/
</VirtualHost>
```

```
<VirtualHost www.CompanyC.com>
    ServerName www.CompanyC.com
    DocumentRoot /webserver/CompanyC/
</VirtualHost>
```

9.2.5.2 Software Virtual Servers

Software virtual servers is the mapping of one IP address to multiple Web servers that share the same configuration information. The following shows a sample of the virtual hosts definition in the file obj.conf:

```
<Object name="default">
   ...
   <Client urlhost="www.CompanyA.com">
      NameTrans fn="home-page" path="/webserver/CompanyA/"
   </Client>
   <Client urlhost="www.CompanyC.com">
      NameTrans fn="home-page" path="/webserver/CompanyC/"
   </Client>
   ...
</Object>
```

The equivalent notation for software virtual servers in the IBM HTTP Server is name-based virtual hosts. This is how it is implemented in the httpd.conf file:

```
NameVirtualHost 1.2.3.4
<VirtualHost 1.2.3.4>
    ServerName www.CompanyA.com
    DocumentRoot /webserver/CompanyA/
</VirtualHost>

<VirtualHost 1.2.3.4>
    ServerName www.CompanyC.com
    DocumentRoot /webserver/CompanyC/
</VirtualHost>
```

9.2.5.3 Multiple Instances

Multiple instances refer to the mapping of unique IP addresses per Web server that use individual configuration information. Each of these servers has a config directory and similar sets of configuration files, and each can be considered an individual server. The file magnus.conf is identical in all the virtual servers shown below.

For the virtual host called CompanyA:

```
ServerID CompanyA
ServerName www.CompanyA.com
Address 1.2.3.4
```

```
...
ErrorLog /webserver/CompanyA/logs/errors
PidLog /webserver/CompanyA/logs/pid
...
```

For the virtual host called CompanyC:

```
ServerID CompanyC
ServerName www.CompanyC.com
Address 1.2.3.5
...
ErrorLog /webserver/CompanyC/logs/errors
PidLog /webserver/CompanyC/logs/pid
...
```

The similar configurations required in the IBM HTTP Server httpd.conf file are as follows:

```
<VirtualHost 1.2.3.4>
   ...
   ServerName www.CompanyA.com
   ErrorLog /webserver/CompanyA/logs/error_log
   TransferLog /webserver/CompanyA/logs/access_log
   ...
</VirtualHost>

<VirtualHost 1.2.3.5>
   ...
   ServerName www.CompanyC.com
   ...
   ErrorLog /webserver/CompanyC/logs/error_log
   TransferLog /webserver/CompanyC/logs/access_log
   ...
</VirtualHost>
```

The main consideration here for the IBM HTTP Server is whether or not the virtual hosts share the same or different configuration files. The IBM HTTP Server offers the flexibility to share configuration files based on either one or multiple IP addresses. The webmaster needs to put the configuration parameters inside the <VirtualHost> directive to enforce specific actions on that particular host, or place them outside that directive to apply these configurations to all the hosts defined.

For instance, this is how to configure two hosts to share common log files:

```
...
ErrorLog /webserver/logs/error_log
TransferLog /webserver/logs/access_log
```

```
<VirtualHost 1.2.3.4>
   ...
   ServerName www.CompanyA.com
   ...
</VirtualHost>

<VirtualHost 1.2.3.5>
   ...
   ServerName www.CompanyC.com
   ...
</VirtualHost>
```

9.2.6 Authentication and Access Control

The basic principles of authentication and access control are similar in most Web servers. The IBM HTTP Server offers several authentication modules to suit different implementations, which is overviewed in 2.3.1, “Security Modules” on page 20. On the other hand, the Netscape Web servers also offer basic authentication of users similar to what the mod_auth module of the IBM HTTP Server supports. As with the IBM HTTP Server, the Netscape Web servers support different ways of authentication and access control through the administration GUI, using the .nsconfig file, or using the .htaccess file.

If the configuration was created exclusively using the administration GUI, the generated ACL files are different from any files supported by the IBM HTTP Server, and thus, migration does require some individual work.

The .nsconfig file contains similar configurations as the .htaccess file used in the IBM HTTP Server. The following shows a fragment of the .nsconfig file used by the Netscape Web servers:

```
<Files /staff/*>
RestrictAccess method="(GET|POST)"type="deny"
RestrictAccess method="(GET|POST)"type="allow" ip="1.2.3.*"
RestrictAccess method="(PUT)"type="deny"
RestrictAccess method="(PUT)"type="allow" ip="*"
RequireAuth file=/webserver/security/users2 realm=StaffOnly
userlist=valid-user
</Files>
```

The following shows a sample of an .htaccess file containing the typical directives. It is very similar to that used by the IBM HTTP Server (see example below):

```
<Limit GET POST>
   order deny,allow
```

```
   deny from all
   allow from 1.2.3
   require valid-user
</Limit>
<Limit PUT>
   order deny,allow
   allow from all
   require group management accounting
</Limit>
AuthName StaffOnly
AuthUserFile /webserver/security/users2
AuthGroupFile /webserver/security/groups2
```

The equivalent version of the .htaccess on the IBM HTTP Server can be implemented with the help of the modules mod_auth and mod_access. The following is a sample of an .htaccess file enforcing access control using the equivalent directives as above:

```
<Location /staff>
   AuthName StaffOnly
   AuthType Basic
   AuthUserFile /webserver/security/users2
   AuthGroupFile /webserver/security/groups2
   <Limit GET POST>
      Order deny,allow
      Deny from all
      Allow from 1.2.3
      require valid-user
      Satisfy all
   </Limit>
   <Limit PUT>
      Order deny,allow
      Allow from all
      require group management accounting
   </Limit>
</Location>
```

When either the .nsconfig or the .htaccess files are used in the Netscape Web servers, the userfile and the groupfile are compatible to the ones used by the IBM HTTP Server, provided that they are of the following simple form:

Userfile format:

```
username:password
username:password
...
```

Groupfile format:

```
groupname:username username ...
groupname:username username ...
...
```

Access control restrictions in the configuration file can be overwritten by special access control files in specific directories. The Netscape Web servers use their acl-directives, such as Default or Always. On the other hand, the IBM HTTP Server uses directives like AllowOverride, AuthConfig, and Limit found in the .htaccess files. However, for the webmasters using the GUI method of the Netscape Web servers, manual porting to .htaccess files should be done. The user files are of a different format.

9.2.7 Logging and Reporting

The IBM HTTP Server has the same flexibility in customizing log files as the Netscape Web servers, besides performing standard logging functions. Basically the Netscape Web servers offer two log files: access and error log files. For the IBM HTTP Server, the two files are also present. The logs file are normally found on the Netscape Web servers in the directory /usr/netscape/suitespot/httpd-<server_name>/logs. For the IBM HTTP Server, the default log file directory is /usr/lpp/HTTPServer/var/logs and the log files found there are access_log and error_log.

The access log files of the Netscape Web servers can be configured in three formats: common logfile format, flexible log format, or customizable format. Generally, the configuration settings of any of these are found in the obj.conf file. The following is an example of the default settings for access logging of the Netscape Web servers:

```
Init fn="flex-init" access="/nscape/suitespot/httpd-CompanyA/logs/access"
format.access="%Ses->client.ip% - %Req->vars.auth-user% [%SYSDATE%]
\"%Req->reqpb.clf-request%\" %Req->srvhdrs.clf-status%
%Req->srvhdrs.content-length%"
```

The following shows the default settings present in the httpd.conf file of the IBM HTTP Server:

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog /webserver/log/access_log common
```

For the Netscape Web servers, there is a log-analyzer tool to analyze the log data collected. As for IBM HTTP Server, log files can be configured to be handled by any other third party utilities for analysis. Such utilities are available on the Web (see also 5.7.3, “Customizing the Log Format” on page 101).The mod_log_config module supports the TransferLog directive for the creation of log files that adhere to the *Common Log Format* (CLF) standard,

which is compatible with other Web servers. In addition, other modules, like mod_log_agent and mod_log_referer, support logging of additional information such as the browser agent used and the reference links, respectively. This is also available in the Netscape Web servers, too. Just uncomment the # character for this particular attribute, which is a default in the httpd.conf file of the IBM HTTP Server as shown below:

```
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
CustomLog /<log dir>/referer_log referer
CustomLog /<log dir>/agent_log agent
```

Or do it directly as shown:

```
CustomLog /<log dir>/agent_log "%t \"%{User-agent}i\""
CustomLog /<log dir>/referer_log "%t \"%{Referer}i\""
```

The IBM HTTP Server also supports the creation of separate log files for each virtual host defined on the system. Likewise, it is not difficult to locate these log files on the Netscape Web servers since each host is defined under a separate directory tree beneath the admin server, and their respective obj.conf file states the rules and location of these log files.

9.2.8 Web Applications

There are various technologies used to implement Web applications on different Web servers. The following sections present some general considerations when migrating these applications to the IBM HTTP Server.

9.2.8.1 CGI Scripts

CGI scripts depend mostly on environment variables that they get from the server. The basic environment variables are the same on most Web servers, including the Netscape Web servers and the IBM HTTP Server. Two common CGI variables not supported by the Netscape Web servers are GATEWAY_INTERFACE and SERVER_SOFTWARE. Other than these, the CGI scripts should work on the IBM HTTP Server without problems if no other special environment variables are used. On the other hand, the IBM HTTP Server allows you to define environment variables unconditionally with the PassEnv and SetEnv directives supported by the mod_env module and conditionally with the SetEnvIf and BrowserMatch directives supported by the mod_setenvif module.

FastCGI scripts are also supported by the optional module mod_fastcgi (see also 10.2.3, “CGI Performance Considerations” on page 217).

9.2.8.2 Server-Side Includes

Server-side includes are supported on the Netscape Web servers as well as on the IBM HTTP Server. Basically, the Netscape Web servers use the ObjectType directive in the obj.conf file to enable the parsing of .shtml files. As for the IBM HTTP Server, the Options directives with the +Includes parameter in the <Directory> directive can be used to permit the use of server-side includes, and the following must be specified to indicate which file extensions are to be parsed:

```
AddHandler server-parsed .shtml
AddType    text/html     shtml
```

9.2.8.3 Image Maps

The IBM HTTP Server and the Netscape Web servers all support the server-side image mapping, as well as client-side mapping. Traditionally, server-side image maps are necessary for old browsers. The Netscape Web servers use the Service directive with the Service-class function called *imagemap* to handle server-side image maps. For instance, below is a configuration settings found in the obj.conf file:

```
Service type=magnus-internal/imagemap method=(GET|HEAD) fn=imagemap
```

As for the IBM HTTP Server, image maps are implemented using the mod_imap module. To enable image maps, use the AddHandler imap-file map directive. For more information about image mapping in the IBM HTTP Server, see 10.5, “Image Maps” on page 220.

Chapter 10. Web Applications

The Web has quickly evolved to an instrument where not only static HTML pages are available from their providers. Business applications, such as a classic merchandise catalog and ordering application, presented very attractive business opportunities for most companies.

This chapter serves as an introduction to Web-application concepts and tools used to build applications that webmasters might be concerned about. For more specific Web application development information for developers we refer you to other pertinent literature, including redbooks (see, for example, B.1, “International Technical Support Organization Publications” on page 227).

10.1 Concepts

Each day the Internet becomes more and more business-oriented. To use the World Wide Web for business, static HTML pages are not enough. Special applications are required to process users’ input and integrate the Web server with other information systems. Such programs that extend the Web beyond passive content browsing are called Web applications.

Web application examples can be search engines, such as Yahoo! (`http://www.yahoo.com`) or Altavista (`http://www.altavista.com`), online shops (such as Amazon.com, `http://www.amazon.com`), parcel tracking services, such as FedEx (`http://www.fedex.com`) or UPS (`http://www.ups.com`), or Web-based system management tools, like the IBM AIX 4.3 Java-based system management tool *websm*.

Usually, Web applications process user data supplied by *HTML forms* (see the example in the following figure, Figure 37 on page 214). Forms can contain entry fields, selection lists, check boxes and other controls. Each form has a specific URL of an application that handles the form data. After filling in the form, a user is typically required to click on a submit button on the form. At that time, the browser sends the form data in a well defined format to the Web server. The server subsequently decodes the data and passes it on to a specified program for processing. The application may respond with an HTML page that was specifically constructed for that particular instance.

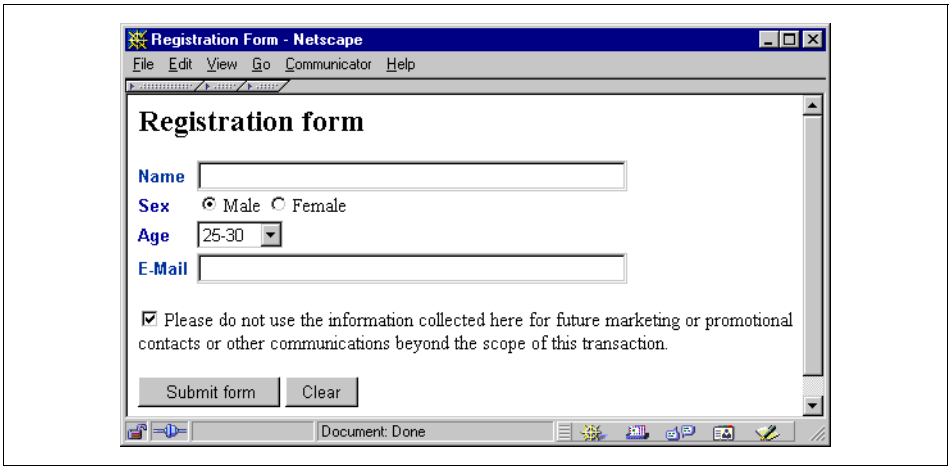


Figure 37. HTML Form Example

Applications (programs) that handle data supplied by Web clients can be either external (CGI programs) or internal (IBM HTTP Server modules). The next sections overview both variants.

10.2 CGI Programs

The Common Gateway Interface (CGI) specification was introduced to enable and standardize the interface between Web servers and external programs. The CGI is a relatively simple, platform and language independent, industry-standard interface for Web application development. Programs that make use of the CGI standard are commonly called *CGI programs*.

The basic principle of CGI is that a Web server passes client request information to CGI programs in system environment variables (and in some cases through standard input or command line arguments) and all standard output of CGI programs is returned to Web clients. This allows for easy writing of Web applications in almost any programming language, but has some performance and security drawbacks (see 10.2.3, “CGI Performance Considerations” on page 217 and 10.2.4, “CGI Security” on page 218). Another deficiency to overcome (if necessary) is that the HTTP protocol itself is stateless and any application that requires more than one step to complete a task needs to send all related information in each step to keep track of the steps.

Although widely used in many Web applications, at the time of writing this book there was no official CGI standard available. Two commonly referenced

sources of information are http://www.golux.com/coar/cgi, and the NCSA Web page at http://hoohoo.ncsa.uiuc.edu/cgi. There is, however, a CGI specification draft RFC available at http://www.ietf.org (search for draft-coar-cgi-v11-01).

10.2.1 Server Configuration

The IBM HTTP Server supports CGI programs through its standard module mod_cgi, together with the modules mod_alias and mod_mime. Make sure to have the corresponding AddModule and LoadModule directives in the server configuration file, which are put there by default.

There are two general ways to enable CGI program execution:

- Declare a whole directory as a CGI programs directory
- Associate a file extension with CGI programs

Each of these serves different purposes, mainly related to security issues (see 10.2.4, “CGI Security” on page 218). When there is a designated directory (or directories) for CGI programs, these programs can be better controlled. When CGI programs are recognized by their filename extension, they can reside anywhere in the file tree and it is generally more difficult to assure that some erroneous CGI program in some user’s directories do not open a back door to the system.

If you want to declare that all files in a particular directory are CGI programs, use the ScriptAlias directive, as in the following example:

```
ScriptAlias /cgi-bin/ /usr/lpp/HTTPServer/share/cgi-bin/
```

In this example, after receiving a request for /cgi-bin/search.pl, the Web server executes the program /usr/lpp/HTTPServer/share/cgi-bin/search.pl and sends the standard output of it back to the Web client. The ScriptAlias directive can be used in the main configuration file, including as part of a <VirtualHost> section, but not in a <Directory> section or a .htaccess file.

If you would like to associate a filename extension with CGI programs, use the AddHandler directive. For example:

```
AddHandler cgi-script cgi
```

This directive instructs the Web server to treat each file with the extension .cgi as a CGI program and it will, therefore, attempt to execute it rather than simply read and send it back to the browser as it would with an ordinary HTML file.

Additional CGI execution control is possible with the Options ExecCGI directive (more about the Options directive can be found in 4.10, “Options” on page 67). This option allows you to enable or disable CGI program execution in individual subdirectories. For example, if you would like to enable execution of all files with a filename extension of .pl in the /www/html/perl directory, use the following directives:

```
<Directory /www/html/perl>
   AddHandler cgi-script pl
   Options +ExecCGI
</Directory>
```

The ExecCGI option can be used anywhere in the main configuration file and also in .htaccess files.

To disable CGI program execution in a particular subdirectory, use the Options -ExecCGI directive. This has no effect on directories specified with the ScriptAlias directive.

10.2.2 Environment Variables

Environment variables are the main means of communication between the Web server and a CGI program. A full description of CGI environment variables can be found, for example, at
`http://hoohoo.ncsa.uiuc.edu/cgi/env.html`

There is small Perl script (/usr/lpp/HTTPServer/share/cgi-bin/printenv) included in IBM HTTP Server distribution which displays the names and values of all CGI environment variables passed to it. A sample output of that script is displayed in Figure 38 on page 217.

If you are uncertain about the variables and/or their values, you can replace your CGI program temporarily with this Perl script to examine the variable in a certain context. Perl needs to be installed on your system in the /usr/local/bin directory in order to make this script work. If you do not have Perl installed, the following Korn shell script does almost the same thing:

```
#!/usr/bin/ksh

echo "Content-type: text/html\n\n"
export | while read line; do
  echo "$line<BR>"
done
```

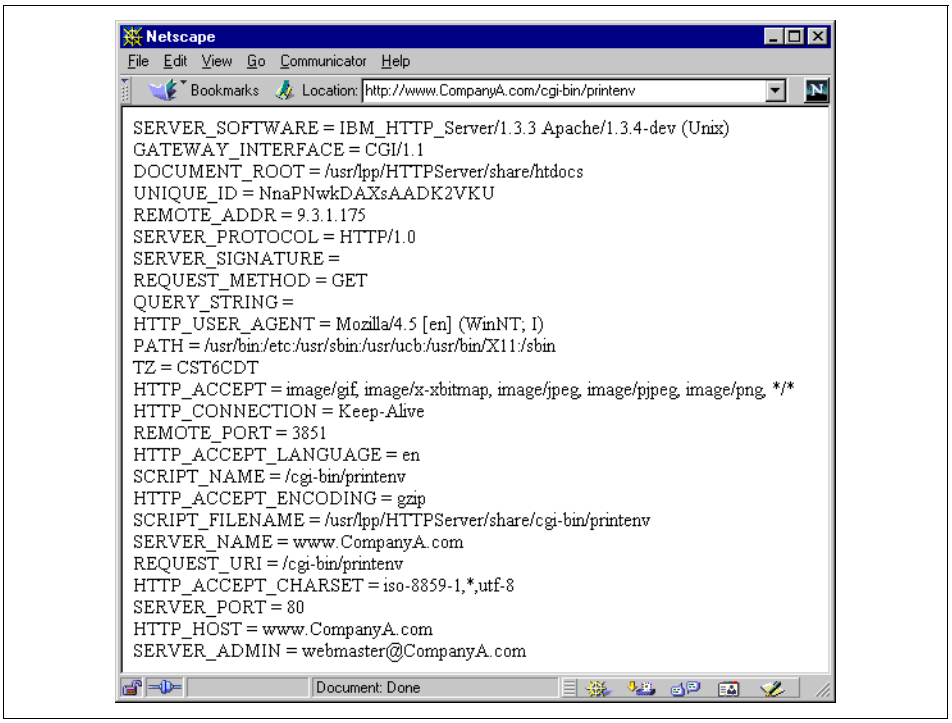


Figure 38. CGI Environment Variables

10.2.3 CGI Performance Considerations

A major concern with CGI performance is the fact that a CGI program is started on each client request. This includes additional disk and operating system activity to create the new process. Quite often, CGI program initialization, such as connecting to a database management system, also takes some time that adds to the response time users experience with such applications.

A possible improvement to this problem is called the FastCGI interface. This CGI extension allows you to start CGI programs only once, and then reuse them for subsequent requests. This significantly improves Web server performance and maintains backward compatibility with old CGI programs.

The FastCGI interface is supported through the mod_fastcgi module, which is not a standard module shipped with the IBM HTTP Server (or Apache). It can be downloaded from the Internet. More information about FastCGI and the mod_fastcgi module can be found at `http://www.fastcgi.com`.

10.2.4 CGI Security

CGI programs greatly increases the capabilities of a Web server, but, on the other hand, they introduce new security risks. A CGI program has the power of doing everything that any other program on that system can do, for example, it can display the contents of the /etc/passwd file. Because of this, all CGI programs must be carefully reviewed by reliable and qualified programmers. Also, it is not wise to allow CGI program execution from within users' public directories (see 5.3, "User Directories" on page 85).

One of the security precautions is to run all CGI programs under some special user other than root. This user is specified in the configuration file by the User directive. By default, it is set to user nobody, but it is better to create a special user for this purpose (see also 3.6, "Initial Setup" on page 42).

Another recommendation is to keep all CGI programs in one (or a few) specially designated directories, specified by the ScriptAlias directive. That allows you better control of these programs. If it is necessary to execute CGI programs from within other directories, it should be verified that only trustworthy people have write access to these directories.

If the AllowOverride FileInfo Options (or AllowOverride All) directive is used in the configuration file, CGI program execution can be enabled by the following directives in the .htaccess file:

```
AddHandler cgi-script cgi
Options +ExecCGI
```

Thus, for security reasons, the preferable directive is AllowOverride None.

10.3 Modules

Modules are a very powerful way to create Web applications, though not necessarily easy. Modules can influent client request processing in almost any step. The IBM HTTP Server has special C language APIs (Application Programming Interface) for additional modules. A comprehensive API description can be found at `http://www.apache.org/docs/misc/API.html`. Also, Chapter 8, "Building HTTP Server Modules" on page 177 gives you an introduction to writing modules.

Because of its complexity, API programs (modules) are not widely used to create Web applications directly. A more common way is to write modules that provide their own, simpler, platform-independent and task oriented API. Examples of such modules are the Perl language interpretation module mod_perl (see 10.6.3, "Perl" on page 222) or the IBM WebSphere Application

Server's Java Servlet Engine (see 10.7, "WebSphere Application Server" on page 223).

Modules must be compiled to shared libraries. They must also be properly listed in the Web server's configuration file. See Chapter 8, "Building HTTP Server Modules" on page 177 for more information on how to implement custom modules.

10.4 Server-Side Includes

The server-side includes (SSI) feature allows for modifying of HTML documents before they are sent to clients. This opens huge additional opportunities for creating dynamic Web pages, but also has some impact on server performance, and even on server security if used improperly.

SSI commands control the inclusion of other files into HTML files, conditionally remove parts of HTML files, and even execute external programs. For example, the following SSI instruction will be replaced with the content of the file header.html before the HTML document is sent to the client:

```
<!-- #include virtual="header.html" -->
```

The next example illustrates how to include a part of a document conditionally. In this example, the link to internal documents will be included only when the request is submitted from an IP address in the 1.2.3.* subnetwork range:

```
<!-- #if expr="$REMOTE_ADDR = /^1.2.3./" -->
   <a href="internal.html">Internal Information</a>
<!-- #endif -->
```

In the following example, the standard output of the specified command `cat /doc/include` replaces the SSI command itself:

```
<!-- #exec cmd="cat /doc/include" -->
```

These examples certainly do not show all the powerful capabilities of SSI. More SSI features can be found in the documentation for the mod_include module.

10.4.1 Server Configuration

Server-side includes are handled by the standard IBM HTTP Server module mod_include. First, you need to associate the file extension .shtml with the

SSI handler. This can be done by adding the following directives to the server configuration file or the .htaccess file:

```
AddType text/html .shtml
AddHandler server-parsed .shtml
```

Then, the SSI feature can be enabled by the following directive in the corresponding section of the configuration file or the .htaccess file (more about the Options directive can be found in 4.10, “Options” on page 67):

```
Options +Includes
```

With this, every file within the scope of the definition with an extension of .shtml will be processed by the SSI module before sending it to the Web client.

10.4.2 Security Considerations

SSI provides, not only additional functionality, but, unfortunately, introduces some security risk. Some confidential files, for example /etc/passwd, can be included into a generated HTML document. The concept of SSI also allows you to execute any shell command and CGI program, so all security considerations applicable to CGI also apply to SSI.

Some general recommendations regarding SSI security follow:

- If you have enabled SSI for some directories, be sure that only trustworthy people have write access to these directories.
- If you prefer to not allow shell commands and CGI programs execution in SSI, use the following directive:

  ```
  Options +IncludesNOEXEC
  ```

- If you would like to disable SSI, use the directive:

  ```
  Options -Includes
  ```

Note that the Options All directive allows SSI processing, including shell commands and CGI programs.

10.5 Image Maps

An image map is a single image on a Web page that contains multiple links. By clicking on different image parts you can get different Web pages. There are two different implementations of image maps:

- Client-side image maps
- Server-side image maps

Client-side image map regions are defined in HTML code and are resolved by the browser. They were introduced by Netscape Navigator Version 2 and were then included as standard in HTML Version 3.2. For more information about client-side image maps, see the appropriate HTML documentation. Client-side image maps are not further explained in this book.

Server-side image maps have existed since HTML Version 2. In this method, pixel coordinates of a mouse click (in relation to an image) are sent to the Web server and resolved to a URL there. The IBM HTTP Server uses the standard module mod_imap to handle server-side image maps.

Server-side image map handling can be enabled by the directive:

```
AddHandler imap-file map
```

Here, `map` is a file extension of image map files. The contents of this file describe regions on an image, related URLs and other instructions. Regions can have rectangle, circle, polygon or point shapes. The closest point region is used when no other regions are satisfied. A default URL can be specified, which will be used in case the given coordinates do not fit to any specified region. The origin of coordinates (x, y) is the upper-left corner of an image.

Let's look at an example. The image map file sample.map in the document root directory of the Web server www.CompanyA.com contains:

```
default /help/images.html
rect    /products.html 50,25 80,75
poly    http://www.CompanyB.com/info 10,10 40,20 5,15
circle  mailto:webmaster@CompanyA.com 100,130 100,150
```

The current HTML document on the Web browser contains the following fragment, displaying some sort of a graphic:

```
<A HREF="/sample.map">
<IMG ISMAP SRC="/menu.gif"></A>
```

If the user clicks on the coordinates `60,50` within that graphic, his request for `http://www.CompanyA.com/sample.map?60,50` will be redirected to the URL `http://www.CompanyA.com/products.html`.

Of course, there are more image map configuration possibilities. For a more detailed description, we refer you to the original mod_imap module documentation.

10.6 Web Applications Development Languages

There are many languages used to write Web applications. Some are widely used (like Perl or C), others are used in IBM-oriented environments (such as REXX or IBM Net.Data macros). Here, we provide an overview of the most popular of them.

In general, programming languages which need to be compiled (like C or C++) are more complex to use and programs written in these languages typically run faster. On the other hand, scripting languages (like Perl or C shell) are more easy to use and maintain, but run slower.

10.6.1 C

C is a general-purpose programming language. It can be used for writing CGI programs and IBM HTTP Server modules. C programming requires higher programming skills than script programming. Generally, it takes more time to develop and debug programs in C as compared to scripting languages. On the other hand, C is more suitable for big and complex projects. The whole IBM HTTP Server and the additional modules are written in C. Performance of compiled C programs is usually much better than scripts.

10.6.2 Shell Script

Shell scripting is the usual way to write small programs in any UNIX environment.The two most popular shell scripting languages in UNIX are the Korn shell and the C shell, which have a different syntax but very similar capabilities. Though programming in shell script is quite easy, performance and functionality are insufficient in some cases.

10.6.3 Perl

Probably the most popular script language for Web application development is Perl. Besides the usual advantages of interpreted languages such as fast and easy programming, Perl has powerful textual data manipulation capabilities and extensible features. There are many Perl-extension libraries available on the Internet.

To improve Perl script performance on the IBM HTTP Server, the mod_perl module was developed, which has a built-in Perl language interpreter.

More information about the Perl language, extensions and the mod_perl module can be found at `http://perl.apache.org`, or `http://www.perl.com`.

10.6.4 Java

Java is a very popular, platform-independent, and object-oriented programming language. Primarily, it has been used for creating client-side Web applications. As the popularity of this language grew, it was used as a server-side application development language. Java inherited many of its features from the C++ language. It is relatively complex and requires good programming skills.

Java programs require a special environment, the Java virtual machine, to run. They also require additional interfaces to other information systems (such as databases). The IBM WebSphere Application Server provides the necessary environment for Java programs and integrates with the IBM HTTP Server. More about this product can be found in 10.7, “WebSphere Application Server” on page 223.

10.6.5 PHP

PHP (Hypertext Preprocessor) is an HTML-embedded scripting language similar to server-side includes, but much more powerful. It allows quick and easy development of dynamic HTML pages. In general, PHP has better performance than CGI scripts.

PHP is supported by Apache servers through the mod_php module. This module and more about the PHP language can be found at: `http://www.php.net`.

10.6.6 REXX

REXX is a procedural programming language created for IBM mainframe computers and later ported to other platforms. It is quite popular in IBM-related environments. Similar to other interpreted languages, REXX programming is relatively easy and performance is average.

10.7 WebSphere Application Server

The IBM WebSphere Application Server is a complete runtime environment for Java-based Web applications. It includes a Java Servlet Engine, Deployment Manager, Connections Services, Applications Services and the IBM HTTP Server itself. The advanced version also includes the Enterprise Java Beans (EJB) Engine and Database Server for EJB.

Java Servlet

A Java servlet is analogous to a Java applet. The difference is that an applet runs on the Web client and communicates with the server by TCP/IP protocol. A servlet runs directly on a server and communicates with the server through an API. A servlet can also communicate to other applications on a server and access other information systems (for example, database management systems) directly through an API. Servlets and applets communicate between themselves using HTTP protocol.

The Java servlet engine is implemented in the IBM HTTP Server as the dynamically loaded module mod_app_server. This ensures better performance compared to the CGI approach and allows stronger control of client request processing.

Connection services supports Java servlet access to relational databases and other external data sources. Application Services, as part of the WebSphere Application Server, allows you to monitor and manage the WebSphere Application Server using a graphical user interface.

More about the IBM WebSphere Application Server can be found at `http://www.software.ibm.com/webservers/appserv`.

Appendix A. Special Notices

This publication is intended to help professionals who need to plan for and implement the IBM HTTP Server on RS/6000 based on the Apache server. The information in this publication is not intended as the specification of any programming interfaces that are provided by the WebSphere or the Apache product. See the PUBLICATIONS section of the IBM Programming Announcement for the IBM WebSphere Application Server V2.0 product for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate

them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX ®	AIX/6000 ®
AS/400 ®	eNetwork
HACMP/6000	IBM ®
RS/6000 ®	WebSphere

The following terms are trademarks of other companies:

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks or registered trademarks of Microsoft Corporation.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

Appendix B. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

B.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How to Get ITSO Redbooks” on page 229.

- *TCP/IP Tutorial and Technical Overview*, GG24-3376
- *The Basics of IP Network Design*, SG24-2580
- *Load-Balancing Internet Servers*, SG24-4993-00
- *Learning Practical TCP/IP for AIX V3.2/V4.1 Users: Hints and Tips for Debugging and Tuning*, SG24-4381
- *Protect and Survive Using IBM Firewall 3.1 for AIX*, SG24-2577
- *Building the Infrastructure for the Internet*, SG24-4824
- *The Technical Side of Being an Internet Service Provider*, SG24-2133
- *Safe Surfing: How to Build a Secure WWW Connection*, SG24-4564
- *A Guide to the Internet Connection Servers*, SG24-4805
- *Enterprise Web Serving with the Lotus Domino Go Webserver for OS/390*, SG24-2074
- *How to Secure the Internet Connection Server for MVS/ESA*, SG24-4803
- *AS/400 e-commerce: Internet Connection Servers*, SG24-2150
- *AS/400 Internet Security : Protecting Your AS/400 from HARM in the Internet*, SG24-4929
- *Unleashing AS/400 Applications on the Internet*, SG24-4935
- *Accessing CICS Business Applications from the World Wide Web*, SG24-4547
- *World Wide Web Access to DB2*, SG24-4716
- *Understanding IBM RS/6000 Performance and Sizing*, SG24-4810

B.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs:

CD-ROM Title	Collection Kit Number
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbook	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044

CD-ROM Title	Collection Kit Number
AS/400 Redbooks Collection	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SK2T-8041
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037

B.3 Other Publications and Links

These publications are also relevant as further information sources:

AIX Installation Guide, SC23-4112

AIX Commands Reference, SBOF-1877

Apache Server For Dummies, Ken A. L. Coar, ISBN 0-7645-0291-3

A good source for Requests for Comments (RFCs) can be found at:

`http://www.isi.edu/rfc-editor/rfc.html`

The Apache Week offers a good amount of related information at:

`http://www.apacheweek.com`

The IBM development and support teams maintain a newsgroup at:

`ibm.software.websphere.http-servers at news.software.ibm.com`

The HTTP protocol versions and features are explained at:

`http://www.w3.org/Protocols`

For AIX performance tuning, see the:

IBM AIX Performance Tuning Guide, SC23-2365

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** http://www.redbooks.ibm.com/

  Search for, view, download or order hardcopy/CD-ROM redbooks from the redbooks web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this redbooks site.

  Redpieces are redbooks in progress; not all redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders via e-mail including information from the redbooks fax order form to:

	e-mail address
In United States	usib6fpl@ibmmail.com
Outside North America	Contact information is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl/

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl/

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl/

This information was current at the time of publication, but is continually subject to change. The latest information for customer may be found at http://www.redbooks.ibm.com/ and for IBM employees at http://w3.itso.ibm.com/.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and redbooks by accessing the IBM Intranet Web site at http://w3.itso.ibm.com/ and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may also view redbook. residency, and workshop announcements at http://inews.ibm.com/.

IBM Redbook Fax Order Form

Please send me the following:

Title	Order Number	Quantity

First name	Last name	
Company		
Address		
City	Postal code	Country
Telephone number	Telefax number	VAT number

☐ Invoice to customer number _____

☐ Credit card number _____

Credit card expiration date	Card issued to	Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

List of Abbreviations

AFS	Andrew File System
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ATM	Asynchronous Transfer Mode
CA	Certificate Authority
CD-ROM	Compact Disc/ Read-only Memory
CERN	European Laboratory for Particle Physics
CGI	Common Gateway Interface
CIDR	Classless InterDomain Routing
CLF	Common Log Format
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CRL	Certificate Revocation List
DBM	Database Manager
DCE	Distributed Computing Environment
DES	Data Encryption Standard
DFS	Distributed File System
DNS	Domain Name System
DSO	Dynamic Shared Objects
EJB	Enterprise Java Beans
FRCA	Fast Response Cache Accelerator
FTP	File Transfer Protocol
GIF	Graphic Interchange Format
GPFS	General Parallel File System
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPD	HTTP Daemon (Web server daemon)
IAB	Internet Architecture Board
IBM	International Business Machines Corporation
ICSS	Internet Connection Secure Server
ID	Identificator
IETF	Internet Engineering Task Force
I/O	Input/Output
IP	Internet Protocol
ISAPI	Internet Server Application Programming Interface
ISDN	Integrated Services Digital Network
ISO	International Organization for Standardization
ISP	Internet Service Provider
ITSO	International Technical Support Organization
JDBC	Java Dabatase Connectivity

JPEG	Joint Photographic Experts Group
NCSA	National Center for Supercomputing Applications
NFS	Network File System
NSAPI	Netscape Server Application Programming Interface
PCT	Private Communication Technology
PKCS	Public Key Cryptography Standard
RAM	Random Access Memory
RFC	Request for Comments
RR-DNS	Round-Robin DNS
RSA	Rivest, Shamir and Adleman
SHA	Secure Hash Algorithm
SMIT	System Management Interface Tool
SMP	Symetric Multiprocessing
SPEC	Standard Performance Evaluation Corporation
SSA	Serial Storage Architecture
SSI	Server Side Includes
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
TLS	Transport Level Security
URI	Uniform Resource Identificator
URL	Uniform Resource Locator
WebDAV	World Wide Web Distributed Authoring and Versioning
WWW	World Wide Web

Index

Symbols

.htaccess 83, 94, 119, 123, 150, 164, 179, 207, 215
.nsconfig 207
.shtml 197, 211
.www_acl 196
/etc/inittab 52
/etc/passwd 119
/etc/rc.httpd 52
/etc/rc.shutdown 53
/usr 49
/usr/local/bin 125
/usr/lpp/HTTPServer 37
/usr/lpp/HTTPServer/etc 55
/usr/lpp/HTTPServer/etc/httpd.conf 43
/usr/lpp/HTTPServer/share/htdocs 57
/usr/lpp/HTTPServer/var/log/access_log 39
/usr/lpp/HTTPServer/var/log/error_log 39, 44
/var 49

Numerics

401, 403, 404 (error codes) 92
  401 127
  403 116

A

ab 169
abbreviations 231
access control (see also authorization) 112
access.conf 6, 57
access_log 12, 39, 209
AccessFileName 59, 124
acronyms 231
AddDescription 84
AddHandler 197, 204, 215
AddLanguage 89
AddModule 16, 56, 85, 87, 106, 215
advanced configuration 71
AFS 86
aliasing 202
Allow 122, 123
AllowOverride 59, 123, 164
Andrew File System (AFS) 174
Apache 5, 11
  building blocks 11
  current version 2
  features 11
  future 8
  GUI enhancement 9
  history 1
  name 2
Apache Group 2, 12
Apache Week 228
apache_tar.gz 37
apachectl 38, 44, 45, 50, 54, 77
apxs 16
asymmetric cryptography 117
ATM 154
authentication 13, 112, 114, 130
  basic 118
  digest 127
AuthGroupFile 120
AuthName 120
authorization 92, 112, 115
AuthType 120
AuthUserFile 120, 124
availability 112

B

Base64 115, 136
basic authentication 118
basic configuration 49
BrowserMatch 197, 210
brute-force 114
building blocks 11
byte ranges 12

C

C (programming language) 222
CacheGcInterval 107
CacheRoot 107
CacheSize 107
caching proxy 106
canonical name 75
CBC 135
CERN 1
Certificate Authority (CA) 115, 136
certificates
  client 146
  from a trusted CA 139
  from an unknown CA 142

revocation lists 148
root 142
self-signed 136
CGI 13, 150, 203, 214
FastCGI 210, 217
cgi-bin 38
CheckSpelling 106
ClearModuleList 56
CNAME 75
commands
apachectl 77
dbmmanage 38, 125
htdigest 38, 128
htpasswd 38, 124
Communicator (Netscape) 90, 108
config phase (hooks) 179
configuration
advanced 71
basic 49
configuration file 6
connections 12
containers 61
content negotiation 13, 87, 95
cookies 115
CoreDumpDirectory 151
CPU 156
CRL 148
cryptography 7, 112, 117, 130
CustomLog 50, 101

D

data confidentiality 112
data integrity 112, 130
DBM files 13, 125
dbmmanage 38, 125
DCE/DFS 86, 174
decryption 117, 133
default directory structure 37
Deny 122, 123
DER 136
DES 135
DES-EDE3-CBC 135
DFS 86, 174
digest authentication 118, 127
directives 17, 58
AccessFileName 59, 124
AddDescription 84
AddHandler 197, 204, 215
AddLanguage 89
AddModule 16, 56, 215
Alias 202
Allow 122, 123
AllowOverride 59, 123, 164
AuthGroupFile 120
AuthName 120
AuthType 120
AuthUserFile 120
BrowserMatch 197, 210
CacheGcInterval 107
CacheRoot 107
CacheSize 107
CheckSpelling 106
ClearModuleList 56
CoreDumpDirectory 151
CustomLog 50, 101
Deny 122, 123
Directory 59, 62, 77, 119
DirectoryIndex 80, 164
DirectoryMatch 62
DocumentRoot 50, 56, 66, 76, 182, 193
ErrorDocument 93
ErrorLog 50, 56, 76, 78, 101
Files 62, 63, 122
FilesMatch 62
FollowSymLinks 116, 150, 163
Group 43, 56
HeaderName 82
HostnameLookups 97, 115, 162
IdentityCheck 98
IfDefine 62
IfModule 62
IndexIgnore 83
IndexOptions 83, 84
IndexOptions FancyIndexing 81
IndexOptions ScanHTMLTitles 84
IndexOptions SuppressHTMLPreamble 83
KeepAlive 160
KeepAliveTimeout 161
LanguagePriority 89
Limit 121
ListenBacklog 160
LoadModule 16, 56, 215
Location 62, 63, 119
LocationMatch 62
LockFile 50
LogFormat 102
LogLevel 100, 134

MaxClients 158
MaxKeepAliveRequests 161
MaxRequestsPerChild 159
MaxSpareServers 159
MinSpareServers 159
Name_or_Address 76
NameVirtualHost 76
Options 163
Options ExecCGI 86
Options Includes 86
Options IncludesNOEXEC 86
Options Indexes 80, 85
Options MultiViews 89
PassEnv 197, 210
PidFile 50
Port 55
ReadmeName 82
Redirect 203
Require 120, 122
RLimitCPU 162
RLimitMEM 162
RLimitNPROC 162
Satisfy 122
Script 96
ScriptAlias 203, 215, 218
SendBufferSize 162
ServerAdmin 56, 76, 93
ServerAlias 77
ServerName 56, 76, 144
ServerPath 78
ServerRoot 50, 56, 122
ServerType 55
SetEnv 197, 210
SetEnvIf 197, 210
SnoreBoardFile 50
SSLCachePortFilename 145
SSLCipherSpec 134
SSLClientAuth 145, 148
SSLClientAuthRequire 148
SSLFakeBasicAuth 148
SSLServerCert 145, 146
StartServers 160
SymLinksIfOwnerMatch 116, 150, 163
Timeout 161
TransferLog 76, 78, 101, 209
User 43, 55, 218
UserDir 85, 116, 151
VirtualHost 78, 119, 145, 194
Directory 59, 62, 77, 119
directory indexing 80
directory structure
default 37
recommended 49
DirectoryIndex 80, 164
DirectoryMatch 62
disk I/O 157
disk requirements 37
Distributed Computing Environment (DCE) 126
Distributed File System (DFS) 174
DNS setup 75
DocumentRoot 50, 56, 66, 76, 182, 193
Dynamic Shared Objects (DSO) 11, 14
implementation 15

E

EDE3 135
encryption 7, 21, 113, 117, 125, 130, 133, 135
eNetwork Dispatcher 173
Enterprise Java Bean Engine 31
environment variables 197, 210, 216
error code 92
error logging 12
error messages (customizing) 91
error_log 12, 39, 44, 209
ErrorDocument 93
ErrorLog 50, 56, 76, 78, 101
Ethernet 154
European Laboratory for Particle Physics (CERN) 1
ExecCGI 86

F

fake basic authentication 148
fancy indexes 81
FastCGI 210, 217
Fcntl.pm 125
file uploading 95
filemon 169
Files 62, 63, 122
filesets 33
FilesMatch 62
fingerprint 114, 117
firewall 113
fixes for the IBM HTTP Server 36
FollowSymLinks 116, 150, 163
forbidden (error) 92
FRCA 7

FrontPage (Microsoft) 96
FTP 95
future of Apache 8

G

General Parallel File System (GPFS) 174
GIF images 38
graphic images 13
Group 43, 56
gskre301 33
gskrf301 33
gskru301 33
GUI (future) 9
gunzip 187
gzip 125

H

handlers 177
  type-map 87
handshake (SSL) 130
hash 117
HeaderName 82
history of Apache 1
hooks 177
host command 75
HostnameLookups 97, 115, 162
htconvert 202
htdigest 38, 128
htdocs 38
HTML forms 213
htpasswd 38, 124, 195
HTTP 115
HTTP Engine 31
HTTP server 2
HTTP/1.0 118, 228
HTTP/1.1 3, 12, 72, 74, 78, 127, 160, 228
http_server.base 33
http_server.modules 33
http-analyze 103
httpd 38
HTTPd (NCSA) 1
httpd.conf 6, 38, 55, 57, 116, 143, 158, 179
httpd.conf.sample.ssl 38, 146
hypertext 1

I

IBM HTTP Server (introduction) 6
icons 38
ICSS 191
Ident daemon 98
IdentityCheck 98
IETF 97, 129, 130
ifconfig 75
IfDefine 62
IfModule 62
ikeyman 39, 136
image maps 197, 220
IncludesNOEXEC 86
index.html 13, 80
IndexIgnore 83
indexing (directory) 80
IndexOptions 83, 84
  FancyIndexing 81
  ScanHTMLTitles 84
  SuppressHTMLPreamble 83
inetd 54, 55
init 52
inittab 52
installation 39
  filesets 33
  prerequisites 36
installp image 39
Internet 1, 111
Internet Architecture Board (IAB) 130
Internet Explorer (Microsoft) 78, 90
Internet Service Provider (ISP) 72, 113
intranet 111
iostat 166
IP-based virtual hosts 72
iptrace 169
ISAPI 3
ISO 111

J

JAVA_HOME 136

K

KeepAlive 160
KeepAliveTimeout 161
Kerberos 126
key database 136
key management 112
Korn shell 216

L

LanguagePriority 89
languages (multiple) 87
language-specific documents 13
Limit 121
ListenBacklog 160
LoadModule 16, 56, 85, 87, 106, 215
Location 62, 63, 119
LocationMatch 62
LockFile 50
log files 37
LogFormat 102
logging 183
LogLevel 100, 134
logresolve 97, 163
lslpp 42
lynx 46

M

MAC 133
magic.default 38
magnus.conf 200
man pages 38
MaxClients 158
MaxKeepAliveRequests 161
MaxRequestsPerChild 159
MaxSpareServers 159
MD5 118, 127, 133, 136
messages 91
Microsoft FrontPage 96
Microsoft Internet Explorer 78, 90, 129
migration 191
mime.types 38
MinSpareServers 159
mkitab 52
module table 177
modules 4, 18, 35
- mod_access 20, 123, 195, 208
- mod_actions 26
- mod_alias 21, 67, 193, 215
- mod_app_server 224
- mod_asis 29
- mod_auth 20, 119, 195, 207, 208
- mod_auth_anon 20
- mod_auth_db 21, 35, 120
- mod_auth_dbm 21, 35, 125
- mod_autoindex 26, 80, 82
- mod_cern_meta 29
- mod_cgi 27, 178, 215
- mod_dav 97
- mod_digest 21, 128
- mod_dir 22
- mod_env 24, 197, 210
- mod_example 36, 38
- mod_expires 29
- mod_fastcgi 197, 217
- mod_headers 30
- mod_ibm_ssl 144
- mod_imap 27, 197, 211, 221
- mod_include 28, 197, 219
- mod_info 18, 24, 36, 177, 185
- mod_log_agent 25, 197, 210
- mod_log_config 25, 197, 209
- mod_log_referer 25, 197, 210
- mod_mime 22, 87, 215
- mod_mime_magic 23, 38
- mod_negotiation 23, 87
- mod_perl 218, 222
- mod_php 223
- mod_proxy 36, 106
- mod_rewrite 23, 78, 194
- mod_setenvif 28, 197, 210
- mod_so 30, 36
- mod_speling 28, 106
- mod_status 25, 46
- mod_unique_id 26
- mod_userdir 24, 85
- mod_usertrack 26

Mosaic 127
multilingual error messages 94
multiple language support 87
MultiViews 88

N

Name_or_Address 76
name-based virtual hosts 72
NameVirtualHost 76
Navigator (Netscape) 90
NCSA 1
Netcraft Ltd. 2
netpmon 169
Netscape 129, 191, 198
- Communicator 90
- Enterprise Server 198
- FastTrack Server 198
- Navigator 90

netstat 168
network I/O 158
network interface 72, 75
NFS 86
nobody user account 42, 150
nonce 128
non-repudiation 112
not found (error) 92
ns-admin.conf 199
nslookup command 75

O

obj.conf 200
ObjectType 211
Options 163
  ExecCGI 86
  Includes 86
  IncludesNOEXEC 86, 150
  Indexes 80, 85
  MultiViews 89
options 67
OS/2 Warp 3

P

packaging 33
paging space 157
PassEnv 197, 210
Perl 95, 125, 216, 222
persistent connections 12
PHP 223
PidFile 50
ping 75
PKCS10 134
Port 55
POST method 95
preface xiii
prerequisites 36
privacy 112, 130
Private Communication Technology (PCT) 129
processes (main, children) 43
product packaging 33
proxy 106
proxy client configuration 108
proxy server products 108
ps 43, 44, 168
PTX (Performance Toolbox) 168
public-key cryptography 117
PUT method 95

R

rc.httpd 52
rc.shutdown 53
RC2 135
RC4 135
README file 38
Readme.httpserver 37
ReadmeName 82
recommended directory structure 49
redirection 202
request phase (hooks) 179
Require 120, 122
restarting (the IBM HTTP Server) 54
retina 114
RFC 228
  1945 92
  2068 92
  2291 97
  931 98
  draft for CGI 215
RLimitCPU 162
RLimitMEM 162
RLimitNPROC 162
robot 100
robots.txt 100
root authority 42
rotatelogs 101, 103
RSA 135

S

sar 167
Satisfy 122
scalability 171
scope 61, 116
Script 96
ScriptAlias 203, 215, 218
sections 61
  parsing rules 64
security 111
  .htaccess file 123
  authorization 115
  basic authentication 118
  basic elements 111
  CGI programs 218
  cryptography 117
  digest authentication 127
  logical 113
  physical 112

SSL 129
terms 134
security audit 112
self-signed certificates 136
SendBufferSize 162
server side includes (SSI) 150
ServerAdmin 56, 76, 93
ServerAlias 77
ServerName 56, 76, 144
ServerPath 78
ServerRoot 50, 56, 122
server-side includes (SSI) 197, 211, 219
ServerType 55
servlet 224
Servlet Engine 31
SetEnv 197, 210
SetEnvIf 197, 210
SHA-1 118, 133, 135
shared object 36
shell scripts 222
sidd 145
SMIT 6, 39
SnoreBoardFile 50
srm.conf 6, 57
SSL 7, 11, 33, 129
SSLCachePortFilename 145
SSLCipherSpec 134
SSLClientAuth 145, 148
SSLClientAuthRequire 148
SSLFakeBasicAuth 148
SSLServerCert 145, 146
stability 13
standalone 54, 55
standard modules 18
starting and stopping the HTTP server 52
StartServers 160
strict.pm 125
support for the IBM HTTP Server 36
survey 2
svmon 169
symbolic links 150
SymLinksIfOwnerMatch 116, 150, 163
symmetric cryprography 117
syslog 101
system configuration 8

T

TCP/IP setup 75
telecommunication 1
Timeout 161
tprof 169
TransferLog 76, 78, 101, 209
translation 182
Transport Layer Security (TLS) 129
Triple DES 135
type maps 87

U

uninstallation 47
UNIX 3
UNIX password file 125
updates to the IBM HTTP Server 36
uploading files 95
User 43, 55, 218
UserDir 85, 116, 151

V

variables 216
variants 87
virtual hosts 12, 71, 204
VirtualHost 78, 119, 145, 194
vmstat 166

W

Web Traffic Express 108
WebDAV 96
websm 213
WebSphere 5, 11, 30, 33, 218, 223
Application Server 5
Windows 95 and NT 3

X

X.509 132, 136
XSSI 95

ITSO Redbook Evaluation

IBM HTTP Server Powered by Apache on RS/6000
SG24-5132-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.ibm.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?
_ **Customer** _ **Business Partner** _ **Solution Developer** _ **IBM employee**
_ **None of the above**

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

_____

_____

_____

_____

What other redbooks would you like to see published?

_____

_____

_____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

_____

_____

_____

_____

SG24-5132-00
Printed in the U.S.A.

IBM HTTP Server Powered by Apache on RS/6000

SG24-5132-00