Pace Web Server:

A Pure Java Web Server
with a
Servlet Container

by
Priya Srinivasaraghavan

Submitted in partial fulfillment
of the requirements for the degree of
M.S. in Computer Science

at

School of Computer Science and Information Systems

Pace University

December 2003

We hereby certify that this dissertation, submitted by Priya Srinivasaraghavan, satisfies the dissertation requirements for the degree of *M.S. in Computer Science* and has been approved.

_____-_____
Dr. Lixin Tao Date
Chairperson of Dissertation Committee

_____-_____
Dr. Narayan Murthy, Chair, CS Dept. Date
Dissertation Committee Member

_____-_____
Dr. Mehdi Badii Date
Dissertation Committee Member

_____-_____
Dr. Susan M. Merritt Date
Dean, CSIS

School of Computer Science and Information Systems
Pace University 2003

Abstract

Pace Web Server:
A Pure Java Web Server
with a
Servlet Container

by
Priya Srinivasaraghavan

Submitted in partial fulfillment
of the requirements for the degree of
M.S. in Computer Science

December 2003

The Internet has had a profound impact on almost all facets of life both for the individual as well as for businesses. It has transformed how business is conducted all over the world. For individuals from employees to students it has provided a rich treasure trove of information from education to entertainment to interaction. For businesses, it has provided a low cost way to advertise and sell their products and services to their customers.

An essential ingredient to the Internet has been the web server, a computer that dishes out the now all too familiar web pages. Though hardly visible or perceivable to the actual user, the web server is an essential part in any interaction between a web user and the provider. Web servers have evolved from simple distributors of hyperlinked pages to complex servers handling advanced technologies like Servlets, JavaServer Pages and Active Server Pages, as well as support for additional services.

In this thesis, I present a Java web server with an integrated Servlet container, which is completely written in Java, lightweight and simple to install and configure. The main purpose of this project is to provide a simple, complete, and well-documented Web server as a vehicle for student study and research on Internet technologies. The Pace Web Server is a multi-threaded server that can handle basic web pages, Servlets and any CGI programs. As it is written in Java, it runs on any platform that has a Java Virtual Machine. In addition, there is also a GUI console available on Windows to control and configure the server without dealing with the configuration files manually.

Acknowledgements

I would like to express my sincere gratitude to my advisor, Dr. Lixin Tao. Throughout my thesis, he provided me with valuable guidance and help. He always found time for me in the middle of his busy schedule to go over the details and provide suggestions and direction. Without his help and constructive criticism, I cannot imagine completing my thesis. Whenever I was stuck and needed direction, he was there for me to guide me through.

I would like to thank the department Chairperson, Dr. Narayan Murthy, for giving me the opportunity to do my thesis and his review and comments.

I would also like to thank my research committee member, Dr. Mehdi Badii for his guidance and support.

Finally, I would like to thank my husband for his patience and moral support throughout my graduate studies and my family for providing me the encouragement and motivation.

पोथी पढ़ पढ़ जग मुआ पंडित भया न कोय
ढाई आखर पेम का पढ़े सो पंडित होय। ।

A person who has read many books is not intelligent.
A person who can understand others is intelligent.

Kabir, a famous Indian medieval poet

Table of Contents

Abstract iii

Acknowledgements iv

List of Tables ix

List of Figures x

Chapter 1 Introduction 1

Chapter 2 Literature Survey 4

2.1 Apache Web Server 12

2.2 Microsoft Internet Information Services (IIS) 13

2.3 Comparison of the Popular Web Servers 14

2.4 Application Server 15

Chapter 3 Elements of a web server 21

3.1 The HyperText Transfer Protocol 21

3.2 HTTP Versions 22

3.3 HTTP request 22

3.3.1 HTTP Request 23

3.3.2 Simple get request 24

3.3.3 Full get request 24

3.3.4 Full get request with headers 24

3.3.5 Post request 24

3.3.6 Head request 25

3.3.7 request URIs and the virtual paths 25

3.3.8 URI encoding 26

3.4 HTTP Reponses 27

3.4.1 Simple responses 27

3.4.2 Full response 28
3.4.3 Full response with headers 28
3.4.4 HTTP response codes 28
3.4.5 MIME types 29
3.5 The Common Gateway Interface 30
3.5.1 Environment variables 31
3.5.2 CGI input 32
3.5.3 CGI output 32
3.5.4 CGI header parsing 32
3.6 Servlets 33
3.6.1 The Advantages of Servlets Over “Traditional” CGI 35
3.6.2 Basic Servlet Structure 38
3.6.3 The Servlet Life Cycle 39
3.6.4 Initialization Parameters 41
3.6.5 Servlet Equivalent of CGI Variables 42
Chapter 4 Pace Web Server Design 46
4.1 Architecture of Pace Web Server 47
4.2 Components of the server 50
4.2.1 Components of the web server 50
4.2.2 Servlet Container 59
4.2.3 Session Management 63
4.2.4 Messages and Error Logging 65
4.2.5 GUI Management Console 67
4.2.6 Help Files for the console 83
4.2.7 Javadoc for sources 84
4.3 Other Features of the Pace Web Server 84
Chapter 5 Installation and Configuration 85

5.1 System Requirements 85
5.2 Installation 85
5.3 Running the Pace Web Server 87
5.4 Command line compiling and invocation 91
5.4.1 Windows 91
5.4.2 Linux [Redhat 7.1] 93
5.5 Using Integrated Development Environments 95
5.5.1 Oracle9i JDeveloper 95
5.5.2 Borland JBuilder 7 99
5.6 Configuration and Management 101
5.6.1 PWS.conf 101
5.6.2 Servlet.conf 107
5.6.3 Session.conf 111
Appendix A. PWS Configuration Files 114
Appendix B. Scripts for compilation and execution 116
Appendix C. Acronyms and Abbreviations 122
References 123

List of Tables

Table 1 List of Web Servers - Oct 2003 5
Table 2 Comparison Matrix of Apache, IIS, Sun ONE servers 14
Table 3 List of available Application Servers 17
Table 4 Comparison of popular commercial application servers 20
Table 5 HTTP response codes 29
Table 6 CGI versus servlet comparison 38
Table 7 Status codes sent most often by Pace Web Server 58

List of Figures

Figure 1 Netcraft survey of web servers 11
Figure 2 Web Server request-response flow 21
Figure 3 Request URI's and virtual paths 26
Figure 4 URI encoding and translation 27
Figure 5 CGI Processes 30
Figure 6 Basic Stages of Pace Web Server 48
Figure 7 Servlet API and their Implementation Classes in PWS 59
Figure 8 Popup menus available via the Windows System Tray Icon 68
Figure 9 General Settings panel of the Console 69
Figure 10 General Settings panel – restart message 71
Figure 11 Servlet Settings panel of the Console 73
Figure 12 Servlet Settings panel – Add Servlet dialog 75
Figure 13 Servlet Settings panel – Individual servlet settings 78
Figure 14 Delete Confirmation message after a Servlet is deleted 80
Figure 15 Session Settings panel of the Console 81
Figure 16 Settings panel – Help Window 83

Chapter 1

Introduction

The 1990s saw an explosion of the Internet and related technologies. The Internet became a household name and its benefits reached millions of people worldwide. It revolutionized the way personal and business affairs are conducted. The email has become an essential mode of communication and is replacing most of phone, fax and other modes of communication. Small and big businesses alike have opened a web storefront in addition to their brick-and-mortar stores. The process of order taking and communications about order status, fulfillment and payments are now done over the Internet with drastic reduction in time taken to complete the entire order-to-cash cycle.

An essential ingredient with the Internet is the Web Server, one of the Internet technology components, that serves as a go between among the multitude of clients or buyers that request a service and the business or provider of such services. The Web servers originally served only static pages to client browser applications. However, the Web servers have evolved from being a simple server to one that includes several independent and/or inline modules that handle dynamic content creation and other services. The Web server is one of the key components in any e-commerce Enterprise Web Application.

The most popular Web servers in the market today are Apache and Microsoft Internet Information Services (IIS). They are very powerful and provide a lot of useful features. Together, these two servers command a major portion of the Web server market. At the same time these servers are also very complex and require a lot of effort to set up,

configure and maintain. Therefore, these servers are not suited well for teaching and research purposes.

In this thesis, I present the Pace Web Server, a Java Web server with an integrated Servlet container, which is completely written in Java, lightweight and simple to install and configure. The main purpose of this project is to provide a simple, complete, and well-documented Web server as a vehicle for student study and research on Internet technologies. The server handles basic Web pages, Servlets and any CGI programs. As it is written in Java, it runs on any platform that has a Java Virtual Machine. In addition, there is also a GUI console available on Windows to control and configure the server without dealing with the configuration files manually.

My thesis starts with a look at the basic concepts and workings of a Web Server and a Servlet Container to understand how the web servers operate, the common challenges and some of ways in which the challenges are overcome, and how the Servlet technology is used in generating and presenting dynamic content. I also compare some of the popular open source and commercial Web servers and some available benchmarks. In the chapter on Pace Web Server design, I outline the architecture of the Web server including the Web server components, and the Servlet container. I explain the multithreaded connection management and delegation, Servlet session management features, CGI capabilities, logging features and the GUI management console of the Web server. The later sections describe the installation instructions, how the code can be compiled and built in various IDEs as well as command line, details of the Web server configuration files, and the ongoing management guidelines for the server. The appendices list

examples of configuration files that are necessary for the operation of the server, and actual scripts that can be used for compilation and running the server.

One of the primary objectives is to understand the basic HTML protocols, the interaction between a client browser and a server that serves static and dynamically created pages, how hyperlinked references work to create random or ad-hoc links to related content from any page. Another related objective is to understand the networking features in the Java programming language by implementing the server in pure Java. Lastly, by implementing a Servlet container, the objective is to understand the Servlet API specification, the relation between a Web server and the Servlet container, request siphoning from the Web server to the Servlet container, how Servlets are spawned to serve client requests, and how responses are sent back to the client.

Chapter 2

Literature Survey

The original use of the Internet was to serve static pages that were stored on a server machine to client programs, usually a browser application that requested the pages by sending a request. Consequently, the original Web Servers were server programs that only served static HyperText Markup Language (HTML) files. Though HTML was powerful in presenting content and did not require any major software installation on the client end, they lacked the strengths of traditional client/server programs and could not interact well with legacy programs. The web servers also could not directly interact with databases where most of the data needed for real time interaction was stored. As the interchange between servers and clients grew over time and became increasingly complex there was a need for additional capabilities on the part of the Web server to handle dynamic content creation from data stored in databases and interaction with legacy programs.

The Web servers were enhanced with capabilities to run small scripts or programs for dynamic content generation on the server side. One of the initial approaches was to use the Common Gateway Interface (CGI) protocol. The Web server upon receiving data from the client via HTML forms executed the CGI scripts to create the dynamic output. However, the CGI turned out to be too slow and required a lot of work, both on the part of the developer writing the scripts and on the Web server. Later, a new set of technologies like Servlets, JavaServer Pages (JSP), and Active Server Pages (ASP) were added to the Web Servers that overcame most of the drawbacks with the CGI programs.

These technologies introduced a new framework for building server side scripts and programs that were capable of generating and sending dynamic HTML pages to the client and capable of interfacing with various legacy programs and databases on the server end.

The following sections describe the most commonly available Web servers along with others available in the market.

Table 1 lists the available Web servers in the market today as chronicled in ServerWatch [18] as of October 2003. The most popular Web servers are Apache, Microsoft IIS, and Sun ONE.

Table 1 List of Web Servers - Oct 2003

Name	Server Type	OS/Platform	Minimum Price
4D WebSTAR	web	Macintosh	$399
AOLserver	web	All Unix All Windows	Free
Apache	web	All Unix All Windows NetWare OS/2	Free
BadBlue	web	All Windows	Free
Baikonur Web App Server	web	Windows 95 Windows NT 4.0	Free
Commerce Server/400	web	AS/400	Free
Covalent Enterprise Ready Server	web	HP-UX Linux Solaris	$1495

Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed

		Windows 2000 Windows XP	
Domino Go Webserver	web	OS/2 Unix Windows 95 Windows 98 Windows NT 4.0	Free
ESAWEB	web	VM/CMS	$3800
Enterprise WebServer for NetWare	web	Novell NetWare	Free
GoAhead WebServer	web	Linux NetWare Solaris Windows 2000 Windows 95 Windows 98	Free
Hawkeye	web	Linux	Free
Java Server	web	HP-UX IRIX Linux OS/2 Solaris Windows 95 Windows NT 4.0	Free
Jigsaw	web	Java_VM Solaris Windows 95 Windows 98 Windows NT 4.0	Free
Microsoft Internet Information Services	web	Windows Server 2003	Free
Microsoft Site Server	web	Windows NT 4.0	$1239
RapidControl for Web	web	BSDI	$1239

Field Code Changed

		Digital UNIX FreeBSD HP-UX IRIX Linux MS-DOS NetBSD SCO OpenServer Solaris Windows 3.x Windows 95 Windows NT 4.0	
RapidSite	web	SGI IRIX Unix	$1239
RomPager Embedded Web Server	web	AS/400 BSDI Be OS Digital UNIX Embedded FreeBSD HP-UX IBM AIX IRIX Java_VM Linux Lynx MS-DOS MacOS X Server NetBSD OS/2 QNX SCO OpenServer Solaris VMS Windows 2000 Windows 3.x Windows 95 Windows 98 Windows CE Windows NT 4.0 Windows XP Windows ME	$1239

Field Code Changed

Field Code Changed

Field Code Changed

Field Code Changed

Roxen WebServer	web	Digital UNIX FreeBSD HP-UX IBM AIX IRIX Linux Macintosh OS/2 Solaris Windows 2000 Windows NT 4.0	Free
Savant	web	Windows 2000 Windows 95 Windows 98 Windows NT 4.0 Windows ME	Free
Servertec Internet Server	web	All Windows HP-UX IBM AIX Linux Solaris Unix	$100
Shadow Web Server	web	MVS	$100
SimpleServer:WWW	web	Windows 2000 Windows 3.x Windows 95 Windows 98 Windows NT 4.0 Windows XP Windows ME	Free
Stronghold Secure Web Server	web	BSDI Digital UNIX FreeBSD HP-UX IBM AIX IRIX Linux NetBSD SCO OpenServer	$995

Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed

		Solaris	
Sun ONE Web Server (formerly iPlanet Web Server)	web	Digital UNIX HP-UX IRIX Linux Solaris Windows 2000 Windows NT 4.0	$1495
Tcl Web Server	web	Linux Macintosh Unix Windows NT 4.0	Free
URL Live!	web	Windows 2000 Windows 98 Windows NT 4.0 Windows XP Windows ME	Free
Viking	web	Windows 2000 Windows 95 Windows 98 Windows NT 4.0 Windows XP	$100
WN Web Server	web	BSDI Digital UNIX FreeBSD HP-UX IBM AIX IRIX Linux NetBSD SCO OpenServer Solaris	Free
WebBase	web	Windows 2000 Windows 95 Windows 98 Windows NT 4.0	$995
WebSite	web	All Windows	$300

Field Code Changed

Xitami	web	BSDI Be OS Digital UNIX FreeBSD HP-UX IBM AIX IRIX Linux NetBSD OS/2 SCO OpenServer Solaris VMS Windows 2000 Windows 95 Windows 98 Windows NT 4.0 Windows XP Windows ME	Free
Zeus Web Server	web	BSDI Digital UNIX FreeBSD HP-UX IBM AIX IRIX Linux MacOS X Server NetBSD SCO OpenServer Solaris	$1700
iTools	web	MacOS X Server	$349
vqServer	web	BSDI Be OS Digital UNIX FreeBSD HP-UX IBM AIX IRIX Java_VM Linux Macintosh	Free

Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed

		NetBSD OS/2 SCO OpenServer Solaris Windows 95 Windows 98 Windows NT 4.0	

There are several organizations that track what each web site or domain in the Internet uses as their web servers and provide periodic updates to their findings. One popular survey is the one conducted by Netcraft [11] every month. Figure 1 below lists the findings for October 2003.

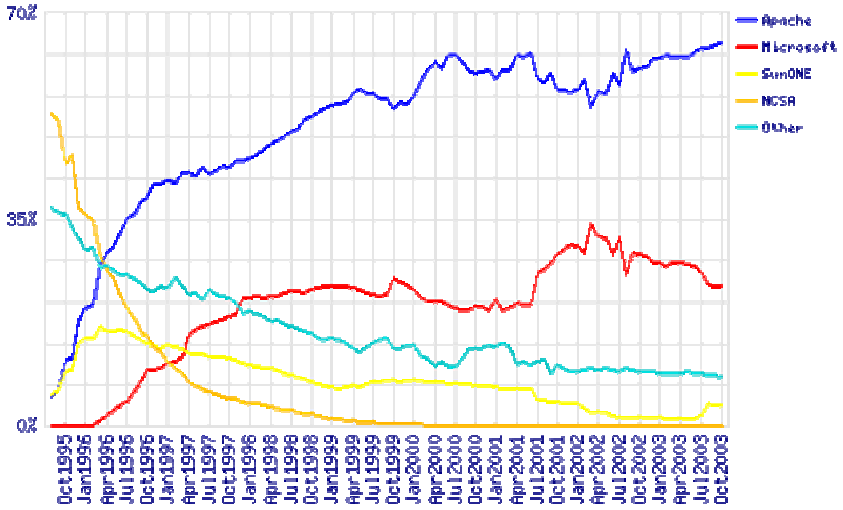

Figure 1 Netcraft survey of web servers

It is evident from the survey numbers that the two most popular web servers are Apache and Microsoft IIS web servers with Sun ONE a distant third.

2.1 Apache Web Server

The Apache web server from The Apache Software Foundation [5] is an open source based web server that has become the number one Web Server on the Internet with more than 60% of the sites using it as their server. It started out as a replacement for the NCSA HTTP Server but has become the most popular one today. The server is written to be very scalable using a multi-threaded or multi-process model depending on the operating system environment. The server is also extensible by way of add-on modules. It also supports several scripting languages. It is offered on a wide variety of platforms and most of all it is free. The server is configured by one or more directives or commands within several configuration files. The latest version of the server on Windows also provides a small monitoring application that can be used to manage the server.

Apache 2.0, the current release, is a major rewrite from the previous versions, the latest of which is 1.3. It is now available on a variety of platforms, including Windows, Mac OS, OS/2, etc. It now has Unix- and Windows-specific execution models that make the best use of the underlying OS. The core of the system is the Apache Portable Runtime (APR), which enables the Apache core to run on any system with a C compiler. A number of multi-processing modules (MPMs) then provide the support for the actual accepting and processing of requests. Under Unix, this can be the traditional "forked process" model, a newer threaded model, or a hybrid of both to achieve the best performance. Under Windows, this uses a threaded model. Please refer to ServerWatch comparison document [17] for more details.

2.2 Microsoft Internet Information Services (IIS)

Microsoft IIS is the Web server platform from Microsoft. IIS 6.0 is the latest version of the web server that is part of the Windows Server 2003 operating system. It is a key component of the Windows Server 2003 application platform that enables the development and deployment of Web sites, Web Applications, and Web Services. The Application server role in Windows Server 2003 consists of Internet Information Services (IIS) 6.0, Microsoft .NET Framework, ASP.NET, ASP, UDDI Services, COM+, and Microsoft Message Queuing (MSMQ) products. IIS performs the functions of a Web server while other components provide Application server and other functions. IIS provides static content; dynamic content via ASP, ASP.NET, Server Side Includes; and Web Services. It is nicely integrated with the Microsoft .NET framework.

IIS is administered from either the Configure Your Server wizard or the Add/Remove Components application. IIS 6.0 also includes a new Web based administration console called the Remote Administration Tool. It also includes a graphical interface for configuring application pools or Web, FTP, SMTP & NNTP sites. The interface can also be used to configure IIS security, performance and reliability features. In addition, the interface also enables the creation or deletion of sites, create virtual directories, etc. In previous versions, this was called the Internet Service Manager. Source: IIS Overview from Microsoft [9]

In version 6.0 several new features were introduced including a new fault tolerant request processing architecture for robust and actively managed runtime processing, increased reliability and scalability by a new process isolation model, called worker isolation model, and finally performance enhancements such as kernel mode queuing and caching. More information can be obtained from the Microsoft technical overview document on IIS 6.0 [10]

2.3 Comparison of the Popular Web Servers

Table 2 presents a comparison of the three popular web servers from ServerWatch findings [16].

Table 2 Comparison Matrix of Apache, IIS, Sun ONE servers

Feature	Apache	IIS	Sun ONE
Act as an HTTP Proxy Server	X	-	-
Can require password	X	X	X
Can write to multiple logs	X	X	X
Comes with SNMP agent	-	X	-
GUI based setup	-	X	X
GUI based maintenance	-	X	X
Built-in scripting language	X	X	X
Includes full source code for server	X	-	-
Includes own API	X	X	-
Integrated certificate server	-	X	-

Feature	Apache	IIS	Sun ONE
Remote maintenance	-	X	X
Scripting languages built-in or as modules	X	X	X
Search engine	-	-	X
Supports IPv6	X	-	X
Supports Microsoft ISAPI	-	X	-
Supports Non-IP Intensive Virtual Servers	X	X	X
Supports SNMP (1, 2c or secure v3) for management	-	-	X
Supports SSL encryption in hardware	-	-	X
Supports SSL v. 3	-	X	X
Supports WebDAV	X	X	-

2.4 Application Server

Though the web servers evolved into more powerful servers with the addition of technologies like Servlets, JSP, and ASP, they still lacked support for building scalable and reliable enterprise web applications. For building such applications there is a need for extensive enterprise-quality server-side common services including transparent networking (abstracting networking away from Web application programmers), thread pooling, database connection pooling, session management, data persistency, transaction control, security, load balancing, caching, directory services, message queuing, and data sharing as in-memory objects for reduced database accesses.

A new class of servers emerged with support for most or all of the above requirements. This new class of servers was called Application Servers.

Examples of Application Servers include the commercial ones like Oracle 9iAS, BEA WebLogic Application Server, IBM WebSphere, Microsoft's Application Server services on Windows and open source implementations like JBoss J2EE application server. Table 3 lists the common application servers available in the market today and Table 4 compares the most popular application servers according to ServerWatch. An even more detailed analysis and matrix is available at TheServerSide [25].

Table 3 List of available Application Servers

Name	Server Type	OS/Platform	Minimum Price
BEA WebLogic Server	application	HP-UX Linux Solaris Windows 2000 Windows NT 4.0	$10000
Borland AppServer	application	HP-UX IBM AIX Red Hat Linux Solaris Windows 2000 Windows NT 4.0	$399
ColdFusion	application	All Windows HP-UX Red Hat Linux Solaris SuSE Linux	$799
Delano e-Business Interaction Suite	application	Windows NT 4.0	$50000
Flash Communication Server MX	application	Macintosh Windows 2000 Windows 98 Windows NT 4.0 Windows XP Windows ME	$499
HAHTsite	application	HP-UX IBM AIX Solaris Windows NT 4.0	$2000
JBoss	application		Free

JRun	application	All Windows Compaq Tru64 Unix HP-UX IBM AIX Red Hat Linux SGI IRIX Solaris	$795
Oracle Application Server	application	All Unix Linux Solaris Windows 2000 Windows NT 4.0	$10000
Orion	application	Linux Unix Windows 2000 Windows NT 4.0	Free
PowerTier for J2EE	application	HP-UX IBM AIX Solaris Windows 2000 Windows NT 4.0	$7500
Pramati Server	application	Red Hat Linux Solaris Windows 2000 Windows NT 4.0	$5000
Sybase Enterprise App Server	application	HP-UX IBM AIX Red Hat Linux Solaris Windows 2000 Windows NT 4.0	$1995
Total-e-Server	application	Java_VM Linux Solaris Windows NT 4.0	$30000

Versata Business Logic Server	application	Unix Windows NT 4.0	$2995
WebApp Server	application	Windows 2000 Windows NT 4.0	$495
WebObjects	application	HP-UX MacOS X Server Solaris Windows 2000 Windows NT 4.0	$699
WebSphere	application	IBM AIX Linux Solaris Unix Windows 2000 Windows NT 4.0 Windows Server 2003	$2000
Witango	application	Linux MacOS X Server Solaris Windows 2000 Windows NT 4.0 Windows XP	$1279
eXtend	application	HP-UX IBM AIX Red Hat Linux SCO OpenServer Solaris Windows 2000 Windows NT 4.0	$295

Table 4 Comparison of popular commercial application servers

	BEA WebLogic Server	Oracle Application Server	WebSphere
Server Type	APPLICATION	APPLICATION	APPLICATION
Latest Version	7.0	9i Release 2	5.0.2
Price Detail	10000 Commercial-ware: $10,000 per CPU, 90-day evaluation available	10000 Oracle9iAS Release 2 Standard Edition, $10,000 per processor; Oracle9iAS Enterprise Edition, $20,000 per processor; Oracle9iAS Personalization and Oracle9iAS Wireless are available as options to Oracle9i Application Server Enterprise Edition and are each priced at an additional $10,000 per processor	2000 Basic Edition, $8,000 per CPU; Enterprise Edition, $25,000; Express Edition, $2,000; Network Deployment package, $12,000; Edge Server package, $6,250.
Rating	5	5	5
Size	87 MB	7 CDs	152 MB
Vendor	BEA Systems	Oracle Corporation	IBM

Chapter 3

Elements of a web server

3.1 The HyperText Transfer Protocol

HTTP is a simple application-layer protocol that enables message passing between Web clients, most often web browsers, and Web Servers.

An HTTP *conversation,* as illustrated in Figure 2, between client and server consists of a client request and a server response over a single TCP connection. The client initiates the conversation with an HTTP request to the server. The server then fulfills the request, for example by returning an HTML document or sending an error message as the response, and then closes the connection. Refer to Hughes et al., [2], and World Wide Web Consortium (W3C) HTTP protocol web page [31] for more details.

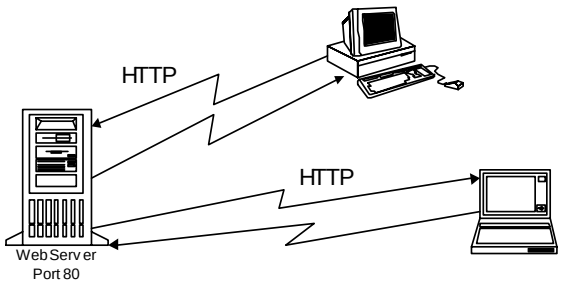

Figure 2 Web Server request-response flow

The HTTP protocol is fundamentally a stateless protocol. Each client request is serviced by the server independently of any other request from the same client or from a different

client. No information about the client is maintained at the server after a connection has been closed. In the last decade, several improvements have been made to enhance the HTTP protocol for stateful sessions with the client. These include cookies and hidden HTML form fields. For more details on the Cookie definitions and protocols refer to Cookie documentation from Netscape [12].

3.2 HTTP Versions

The current version of HTTP protocol is HTTP/1.1 with several extensions proposed in recent times. For a complete discussion of the latest proposals please refer to the World Wide Web Consortium pages on HTTP [31].

3.3 HTTP request

An HTTP request is a request from a client for a document or more generically, a resource, on the server. For example, if the web server has a page called mypage.html then the client requests the page by sending a request for that page and mentions the name of the page. The client may also send some additional information along with the name of the document or resource so the web server can service the request more efficiently. This additional information is in the form of attribute-name = attribute-value pairs. The additional information are collectively called request Headers.

In response, the web server sends a status code and message about whether the request was processed successfully or not, the content type or the type of the document, for example, simple text or HTML or images, etc., and the body of the document or resource

requested. The server may also send some additional information as name=value pairs about the resource and these are called response Headers.

The following discussion on HTTP Request, HTTP Response and CGI protocol information is summarized from Hughes et al., [2].

3.3.1 HTTP Request

The request line is the first line of a request:

```
GET /document.html HTTP/1.0
```

The first element of the request line is called the *method*; it specifies what action the server is to perform on the resource. The second element is the *request URI*, which denotes the resource in the question. A third element, which is present in the HTTP 1.0 and the 1.1 request, indicates the *version* of HTTP understood by the client.

HTTP requests come in two forms: simple and full. Simple (HTTP 0.9) requests consist of only one of the two methods (get or *post*) and a request URI. Full requests can employ an additional method, called *head*. Full requests always include the client HTTP version as a third element of the request line. Full request may also follow the request with various headers, which give more information about the request and the client.

3.3.2 Simple get request

A simple get request follows the old HTTP 0.9 specification, which, while still in use, is essentially, obsolete.

```
GET /document.html[CRLF]
```

3.3.3 Full get request

A full get request follows the HTTP 1.0 (or later) specification. It includes the HTTP version number in the request. The request must be followed by a blank line.

```
GET /document.html HTTP/1.0[CRLF]
```

3.3.4 Full get request with headers

HTTP/1.0 supports optional headers in the request. The following request tells the server the type of browser being used and the requests that the document only be returned if it has been modified more recently than the specified date. This particular header allows the browser to use a cached version of the document if the original has not changed. The request must be followed by a blank line.

```
GET /document.html HTTP/1.0[CRLF]
User-Agent: Surfer/1.01 libhttp/0.1[CRLF]
If-Modified-Since: Sun, 20 Oct 1996 04:07:51 GMT[CRLF]
```

3.3.5 Post request

A post request allows the client to include a significant body of data in a request. Post is used, for example, to submit a large body of information to a CGI script or to upload a file to a Web server, to be processed by a target script. The content-length header is mandatory with a post request; the request headers must be followed by a blank line.

```
POST /cgi-bin/code.cgi HTTP/1.0[CRLF]
Content-type: application/octet-stream[CRLF]
Content-length: 2048[CRLF]
[LF]
body
```

Some Web servers and Web proxies also require that the post body be followed by a CRLF sequence; this should not be counted in the content-length header.

3.3.6 Head request

Head requests return only the headers of the resource, or the headers of an error message if an error occurs. This is useful to find out the information about a document without the expense of transmitting the actual document. This method is illegal in a simple request since HTTP 0.9 does not support head requests.

```
HEAD /index.html HTTP/1.0[CRLF]
[LF]
```

3.3.7 request URIs and the virtual paths

The main function of the request URI portion of the request line(the second token) is to specify a virtual path to the resource that the client is requesting.

A URI consists of both a virtual path and an optional query string, separated by a query character (?):

```
/cgi-bin/code.cgi?query-string
```

The virtual path is a path like string that identifies a document or service being requested; the query string is some optional additional information that will be supplied to a dynamic resource. The virtual path is *virtual* because it always uses a / path-element separator, independent of the client or server operating system. Although it looks like a path, it will not be an absolute path on the Web server; it may refer to CGI script or other dynamic resources, and will almost certainly be translated according to aliasing rules to prevent external users from accessing arbitrary documents on the host machine. Figure 3 from Hughes et al., [2] illustrates this concept.

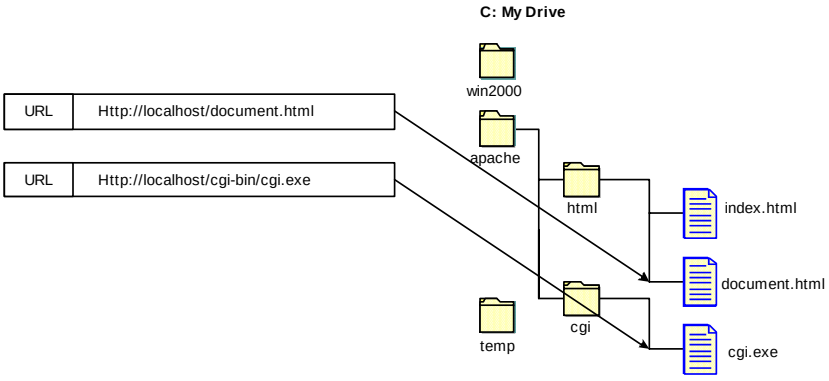

Figure 3 Request URI's and virtual paths

3.3.8 URI encoding

The request URI portion of the request line may be sent from the client in an encoded form; this allows arbitrary textual characters to be transmitted in unambiguous ASCII format. Therefore, before the server can process it, the request URI must be decoded to

change any + characters to space characters. In addition, certain characters are encoded to a hexadecimal value, denoted by a preceding %

Because of the fact that the raw request URI from the client may contain '..' and '.' path elements, it must finally be canonicalized to remove these before translation to the physical file system location. Failure to do so allows a client to access data outside the server's document root, which is a serious security threat; for example a client could request / ../ ../passwd which would refer to a document outside of the Web server's HTML directory. Figure 4 from Hughes et al., [2] illustrates this concept.

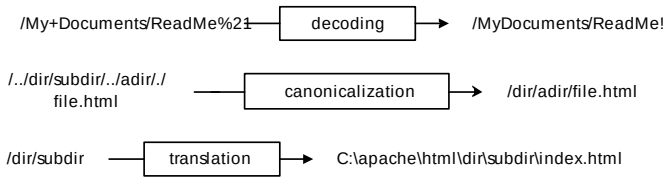

Figure 4 URI encoding and translation

3.4 HTTP Reponses

The Web server's response varies with the type of request and whether or not the request could be serviced

3.4.1 Simple responses

A simple response is only returned in response to an HTTP 0.9 request. It consists of the body of the requested resource with no headers.

```
body
```

3.4.2 Full response

A full response includes a status line followed by the body of the document. The status line consists of a HTTP version of the response and a status code, which indicates how successfully the request was serviced.

```
HTTP/1.0 200 OK[CRLF]
```

3.4.3 Full response with headers

A full response also may include some headers that include additional information about the server and requested document, such as its content type, whether it is compressed, and when it was last modified.

```
HTTP/1.0 200 OK[CRLF]
Server: Apache/1.2b11[CRLF]
Content-type: text/html[CRLF]
Content –encoding: x-gzip[CRLF]
[LF]
body
```

3.4.4 HTTP response codes

Table 5 lists the status codes that are included with HTTP 1.0 response. The most common status code is 200, which means that the request was serviced successfully; 301 means that the document has moved (the response headers will include the new location); 404 means the document was found, and so on.

Table 5 HTTP response codes

Code	**Meaning**
200	OK
201	Created
202	Accepted
204	No Content
301	Moved Permanently
302	Moved Temporarily
304	Not Modified
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not found
500	Internal Server Error
501	Not implemented
502	Bad Gateway
503	Service Unavailable

3.4.5 MIME types

Multipurpose Internet Mail Extensions (MIME) is a mechanism, originally designed for email, to associate a type with message so that the message received will understand how to decode / view it. MIME is defined in RFC 1521. The seven top-level MIME type defined in RFC 1521 are *text, image, audio, video, multiparts, application, and message*.

The most common subtypes associated with the HTTP include *text/html, text/plain, image/gif, image/jpeg, and application/octet-stream.*

HTTP 1.0 and 1.1 support MIME typing as the means for servers to indicate the type of information contained in a response. To do this, the server sends a content-type header with the MIME type and subtype of the data being returned.

3.5 The Common Gateway Interface

CGI programs are the most widespread type of software for generating dynamic Web content. CGI enables developers to write custom request-handling software that automatically interfaces with any CGI- complaint Web server. A CGI or CGI script is a piece of software, usually written in Perl, C/C++, or shell scripting language that conforms to the CGI standard. For security reasons, CGIs are usually restricted to a central server directory, history called *cgi-bin.*

When the server receives a request for a URI that refers to a CGI, it creates a process to execute the CGI, providing the CGI with certain information about the request. The server then forwards the CGI's output back to the client as the HTTP request. The server finally closes down the client connection when the CGI finishes executing. This concept is illustrated in Figure 5 below.

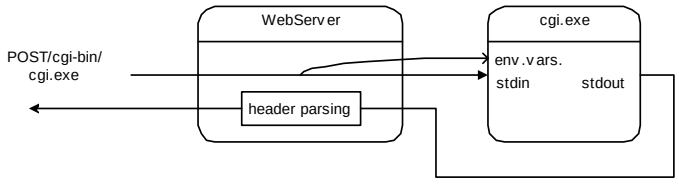

Figure 5 CGI Processes

3.5.1 Environment variables

When the server launches a CGI process, it provides it with data required by the CGI specification. The majority of this information is contained in environment variables, the most interesting of which are:

QUERY_STRING This variable contains the portion of the client's request following the ? symbol; it is not decoded by the server. The query string is usually used by a client using the get request method in order to pass information to the CGI. For example, in the following request:

```
GET /cgi-bin/process.cgi? name=jim&address=pine+haus
```

The QUERY_STRING variable will contain `name=jim&address=pine+haus`

PATH_INFO The variable contains the portion of the request beyond the pathto the CGI. This information is decoded by the server and may be canonicalized as well. For example, in the following request:

```
GET /cgi-bin/colorchange.cgi/blue/red
```

The PATH_INFO variable will contain /blue/read.

*REQUEST_METHOD*This variable contains the method used in the request; that is, GET, POST, or HEAD.

CONTENT_TYPE This variable may contain the content type of the information being passed to the server in a post request. Data encoded from a HTML form has content type application/x-www-form-urlencoded

CONTENT_LENGTH This variable will contain the value of the `Content-Length` request header. This header is mandatory for a post request; it indicates the volume of data included in the client's request. The CGI will be able to read this volume of data (in bytes) from its input stream.

Request header All request headers from the client are translated into environment variables by changing the instances of the – character into _ and prepending HTTP_ to each variable name. For example, `Content-Length` is also supplied to CGI script as the variable `HTTP_CONTENT_LENGTH`

3.5.2 CGI input

A script executed with the post method has the opportunity to access QUERY_STRING and PATH_INFO variables, just like a script executed using get. In addition, it receives the body of the client's post request as its standard input, and may read `CONTENT_LENGTH` bytes from the source.

3.5.3 CGI output

The CGI writes its output to the server on its standard output. The server then directs this output back to the client after header processing, if any.

3.5.4 CGI header parsing

In a HTTP1.0 response, the client expects a standard header preceding any response from the server. The server can handle the issue of the headers for the CGI's response is one of two ways. The default behavior is to parse any headers that the CGI sends, insert a status line and any other headers the server wants, and send the whole resulting header to the client followed by the remaining body of the CGI's response.

The other option is for the server to simply send the content produced by the CGI directly to the client without parsing its headers at all. This is called no parse headers (NPH). NPH CGIs are responsible for explicitly sending the appropriate status line and headers to the client themselves. The server treats any CGI that begins with a special filename designation (typically nph-) as an NPH program.

3.6 Servlets

Servlets are Java technology's equivalent to Common Gateway Interface (CGI) programming. A Servlet is a small piece of Java code loaded by the web server when a client request comes in. The servlet code can access other resources on the server side like a database or a transaction management system or any other resource without the restrictions of the Java sandbox model. In contrast, a Java applet program on the browser can only open a connection back to the server where the original request came from and cannot maintain multiple links to multiple hosts or services.

A Servlet's function, according to Hall [1], is:

1. ***Read any data sent by the user.***

This data usually originates in a form on a Web page, but could also come from a Java applet or a custom HTTP client program.

2. ***Look up any other information about the request that is*** **embedded in the HTTP request.**

This information includes details about browser capabilities, cookies, the host name of the requesting client, and so forth.

3. ***Generate the results.***

This process may require talking to a database, executing an RMI or CORBA call, invoking a legacy application, or computing the response directly.

4. ***Format the results inside a document.***

In most cases, this involves embedding the information inside an HTML page.

5. ***Set the appropriate HTTP response parameters.***

This means telling the browser what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

6. ***Send the document back to the client.***

This document may be sent in text format (HTML), binary format (GIF images), or even in a compressed format like gzip that is layered on top of some other underlying format.

A Servlet is most useful when the HTML page generated by the web server is dynamic in that either it depends on the data sent by the user, or is derived from a database that is constantly changing or any other form of dynamic data.

For example, consider a simple web page counter that tracks the number of times a web server has been accessed by clients. If the counter has to be extended to remember how

many times a particular client has accessed the web server, it requires a database where the client id and number of times accessed is stored. The information about the client can be stored as a cookie or a client's information, say, a username and password is also stored in the database. A servlet can be used very conveniently in this instance to access the database and provide the results to the client.

3.6.1 The Advantages of Servlets Over "Traditional" CGI

Java servlets are more efficient, easier to use, more powerful, more portable, safer, and cheaper than traditional CGI and many alternative CGI-like technologies. Source: Hall [1].

Efficient

With traditional CGI, a new process is started for each HTTP request. If the CGI program itself is relatively short, the overhead of starting the process can dominate the execution time. With servlets, the Java Virtual Machine stays running and handles each request using a lightweight Java thread, not a heavyweight operating system process. Similarly, in traditional CGI, if there are *N* simultaneous requests to the same CGI program, the code for the CGI program is loaded into memory *N* times. With servlets, however, there would be *N* threads but only a single copy of the servlet class. Finally, when a CGI program finishes handling a request, the program terminates. This makes it difficult to cache computations, keep database connections open, and perform other optimizations that rely on persistent data. Servlets, however, remain in memory even after they

complete a response, so it is straightforward to store arbitrarily complex data between requests.

Convenient

Servlets have an extensive infrastructure for automatically parsing and decoding HTML form data, reading and setting HTTP headers, handling cookies, tracking sessions, and many other such high-level utilities.

Powerful

Servlets support several capabilities that are difficult or impossible to accomplish with regular CGI. Servlets can talk directly to the Web server, whereas regular CGI programs cannot, at least not without using a server-specific API. Communicating with the Web server makes it easier to translate relative URLs into concrete path names, for instance. Multiple servlets can also share data, making it easy to implement database connection pooling and similar resource-sharing optimizations. Servlets can also maintain information from request to request, simplifying techniques like session tracking and caching of previous computations.

Portable

Servlets are written in the Java programming language and follow a standard API. Consequently, servlets written for, say, I-Planet Enterprise Server can run virtually unchanged on Apache, Microsoft Internet Information Server (IIS), IBM WebSphere, or StarNine WebStar.

Secure

One of the main sources of vulnerabilities in traditional CGI programs stems from the fact that they are often executed by general-purpose operating system shells. Therefore, the CGI programmer has to be very careful to filter out characters such as back quotes and semicolons that are treated specially by the shell. A second source of problems is the fact that some CGI programs are processed by languages that do not automatically check array or string bounds. Therefore, programmers who forget to do this check themselves open their system up to deliberate or accidental buffer overflow attacks. Servlets suffer from neither of these problems. Even if a servlet executes a remote system call to invoke a program on the local operating system, it does not use a shell to do so. And of course array bounds checking and other memory protection features are a central part of the Java programming language.

Inexpensive

There are a number of free or very inexpensive Web servers available that are good for "personal" use or low-volume Web sites. However, with the major exception of Apache, which is free, most commercial-quality Web servers are relatively expensive, but adding a servlet is very cheap. This is in contrast to many of the other CGI alternatives, which require a significant initial investment to purchase a proprietary package.

Table 6 below provides a comparison between Servlets and CGI in terms of performance and other properties from Orfali and Harkey [3].

Table 6 CGI versus servlet comparison

Feature	HTTP/Servlet	HTTP/CGI
Reliable communications	Yes	Yes
State across invocations	Yes (with great difficulty)	No
Parameter marshalling	No	No
Interface descriptions	No	No
Dynamic discovery	No	No
Parameter data typing	No	No
Performance	Slow (55.6 msec Ping)	Very Slow (827.9 msec Ping)
Security	Yes	Yes (Via SSL or S-HTTP)
Transactions	No	No

3.6.2 Basic Servlet Structure

Servlets can handle both `GET and POST` requests. To be a servlet, a class should extend `HttpServlet` and override `doGet` or `doPost methods`, depending on whether the data is being sent by `GET` or by `POST`. Usually if the same processing needs to be done for both GET and POST on of the doGet or doPost is coded to call the other method or both can call a same additional method that does the work.

Both of these methods take two arguments: an `HttpServletRequest` and an `HttpServletResponse`. The `HttpServletRequest` has methods by which we can find out about incoming information such as form data, HTTP request headers, and the client's

hostname. The `HttpServletResponse` lets us specify outgoing information such as HTTP status codes (200, 404, etc.), response headers (`Content-Type`, `Set-Cookie`, etc.), and, most importantly, helps obtain a `PrintWriter` used to send the document content back to the client. For simple servlets, most of the effort is spent in `println` statements that generate the desired page. Form data, HTTP request headers, HTTP responses, and cookies .

Servlets could, in principle, be used to extend mail, FTP, or other types of servers. Servlets for these environments would extend a custom class derived from `GenericServlet`, the parent class of `HttpServlet`. In practice, however, servlets are used almost exclusively for servers that communicate via HTTP (i.e., Web and application servers).

3.6.3 The Servlet Life Cycle

When the servlet is first created, its `init` method is invoked. The init method is invoked by the Servlet Container and is done only once during servlet initialization. This is a place where one-time setup code like opening a database connection or retrieving properties can be coded. Once the servlet is loaded, each user request results in a thread that calls the `service` method of the previously created instance. Multiple concurrent requests normally result in multiple threads calling `service` simultaneously. A servlet can implement a special interface that stipulates that only a single thread is permitted to run at any one time by extending the SingleThreadModel. It is then the responsibility of the servlet container to make sure that only one request is served at a time by one thread. The `service` method then calls `doGet`, `doPost`, or another `do`*`Xxx`* method,

depending on the type of HTTP request it received. Finally, when the servlet container or the server decides to unload a servlet, it first calls the servlet's `destroy` method so that any resources can be relinquished gracefully.

The init Method

The `init` method is called when the servlet is first created and is *not* called again for each user request. So, it is used for one-time initializations, just as with the `init` method of applets. The servlet can be created when a user first invokes a URL corresponding to the servlet or when the server is first started, depending on how the servlet is registered with the Web server. This is usually a feature of the web server.

The second version of `init` is used when the servlet needs to read server-specific settings before it can complete the initialization. For example, the servlet might need to know about database settings, password files, server-specific performance parameters, hit count files, or serialized cookie data from previous requests. The second version of `init` uses the ServletConfig parameter to perform these tasks.

The service Method

Each time the server receives a request for a servlet, the server spawns a new thread and calls `service`. The `service` method checks the HTTP request type (`GET`, `POST`,

PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc., as appropriate.

The doGet, doPost, and doXxx Methods

These methods contain the real meat of the servlet functionality. Almost all the time it is only the doGet or doPost is overridden. However, if necessary, a servlet can also override doDelete for DELETE requests, doPut for PUT, doOptions for OPTIONS, and doTrace for TRACE.

The destroy Method

The server may decide to remove a previously loaded servlet instance, perhaps because it is explicitly asked to do so by the server administrator, or perhaps because the servlet is idle for a long time. Before it does, however, it calls the servlet's destroy method. This method gives our servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

3.6.4 Initialization Parameters

A servlet can access certain initial arguments or parameters during startup. A most common need is to access the username and password information to get a database connection, any kind of properties that tell the servlet to behave in a particular fashion, or

simply a location where it can find the image and other resources, or to find out whether it should write debug messages or not. The init-arguments can also be used for internationalization.

The servlet initialization arguments are provided by the container wrapped in the ServletConfig object when the servlet is first initialized by the container. The parameters are specified in the servlet.properties file per the Servlet API 2.1 specifications. In later servlet API versions the initial arguments are specified in the web.xml file.

The initial arguments or initialization parameters are specified as name=value pairs as in the example below.

```
Name=Priya
Degree=MS
University=Pace
```

Within the init method, a servlet can request a value for a particular named parameter [getInitParameter()] or get an enumeration of the parameters [getInitParameterNames()]

3.6.5 Servlet Equivalent of CGI Variables

For each standard CGI variable, this section, from Hall, [1],summarizes its purpose and the means of accessing it from a servlet, assuming `request` is the `HttpServletRequest` supplied to the `doGet` and `doPost` methods.

AUTH_TYPE If an `Authorization` header was supplied, this variable gives the scheme specified (`basic` or `digest`). Access it with `request.getAuthType()`.

CONTENT_LENGTH For `POST` requests only, this variable stores the number of bytes of data sent, as given by the `Content-Length` request header. Technically, since the `CONTENT_LENGTH` CGI variable is a string, the servlet equivalent is

`String.valueOf(request.getContentLength())` or `request.getHeader("Content-Length")`. We'll probably just call `request.getContentLength()`, which returns an `int`.

CONTENT_TYPE `CONTENT_TYPE` designates the MIME type of attached data, if specified. Access `CONTENT_TYPE` with `request.getContentType()`.

DOCUMENT_ROOT The `DOCUMENT_ROOT` variable specifies the real directory corresponding to the URL `http://host/`. Access it with `getServletContext().getRealPath`. Also, we can use `getServletContext().getRealPath` to map an arbitrary URI (i.e., URL suffix that comes after the hostname and port) to an actual path on the local machine.

HTTP_XXX_YYY Variables of the form `HTTP_HEADER_NAME` were how CGI programs obtained access to arbitrary HTTP request headers. The `Cookie` header became `HTTP_COOKIE`, `User-Agent` became `HTTP_USER_AGENT`, `Referer` became `HTTP_REFERER`, and so forth. Servlets should just use `request.getHeader`.

PATH_INFO This variable supplies any path information attached to the URL after the address of the servlet but before the query data. Since servlets, unlike standard CGI programs, can talk directly to the server, they don’t need to treat path information

specially. Path information could be sent as part of the regular form data and then translated by `getServlet- Context().getRealPath`. Access the value of `PATH_INFO` by using `request.getPathInfo()`.

PATH_TRANSLATED `PATH_TRANSLATED` gives the path information mapped to a real path on the server. Again, with servlets there is no need to have a special case for path information, since a servlet can call `getServletContext().get-RealPath` to translate partial URLs into real paths. This translation is not possible with standard CGI because the CGI program runs entirely separately from the server. Access this variable by means of `request.getPathTranslated()`.

QUERY_STRING For `GET` requests, this variable gives the attached data as a single string with values still URL-encoded. We use `request.getParameter` to access individual parameters.

REMOTE_ADDR This variable designates the IP address of the client that made the request, as a `String` (e.g., `"198.137.241.30"`). Access it by calling `request.getRemoteAddr()`.

REMOTE_HOST `REMOTE_HOST` indicates the fully qualified domain name (e.g., `whitehouse.gov`) of the client that made the request. The IP address is returned if the domain name cannot be determined. We this variable with `request.getRemoteHost()`.

REMOTE_USER If an `Authorization` header was supplied and decoded by the server itself, the `REMOTE_USER` variable gives the user part, which is useful for session tracking in protected sites. Access it with `request.get-RemoteUser()`. For decoding `Authorization` information directly in servlets.

REQUEST_METHOD This variable stipulates the HTTP request type, which is usually `GET` or `POST` but is occasionally `HEAD`, `PUT`, `DELETE`, `OPTIONS`, or `TRACE`. Servlets rarely need to look up `REQUEST_METHOD` explicitly, since each of the request types is typically handled by a different servlet method (`doGet`, `doPost`, etc.). Access this variable by means of `request.getMethod()`.

SCRIPT_NAME This variable specifies the path to the servlet, relative to the server's root directory. It can be accessed through `request.getServletPath()`.

SERVER_NAME `SERVER_NAME` gives the host name of the server machine. It can be accessed by means of `request.getServerName()`.

SERVER_PORT This variable stores the port the server is listening on. Technically, the servlet equivalent is `String.valueOf(request.getServerPort())`, which returns a `String`. We usually just want `request.getServer-Port()`, which returns an `int`.

Chapter 4

Pace Web Server Design

The Pace Web Server is a lightweight Web server and Servlet container written entirely in Java. The web server employs a multi-threaded architecture to service the requests from client connections by maintaining a pool of threads. The parameters to configure the web server are specified on one or more configuration files. There is a GUI management console available to configure the web server parameters. The parameters can either be edited directly in the configuration files or the GUI console can be used. In Windows platform the web server uses a third-party library to integrate with the operating system to provide easy access to the console via the system tray. However, since the web server is written in Java it works in any operating system where an implementation of the Java Virtual Machine is available. The web server can also run any CGI programs written in Perl or other scripting language as well as CGI style programs written in C or C++.

The Servlet Container conforms to the Servlet API version 2.1 specifications. Any number of servlets can be added to the web server. The servlets are set up by specifying the class name, the full path to the class and an alias that can be used in the URL of the browser. The same servlet can be invoked using multiple aliases. Any servlet added via the console is automatically loaded by the server and does not require a shutdown/restart of the server. The console or the configuration file can be used to specify one or more servlet initargs (initialization arguments) for the servlets registered with the server.

4.1 Architecture of Pace Web Server

The basic stages of the Pace Web Server can be classified as Initialization and Waiting for Requests, Parsing Requests, Serving Static HTML pages or Invoking Servlets to serve the requests, Interaction with the Management Console, and Termination.

The basic stages are represented in the following flowchart, Figure 6.

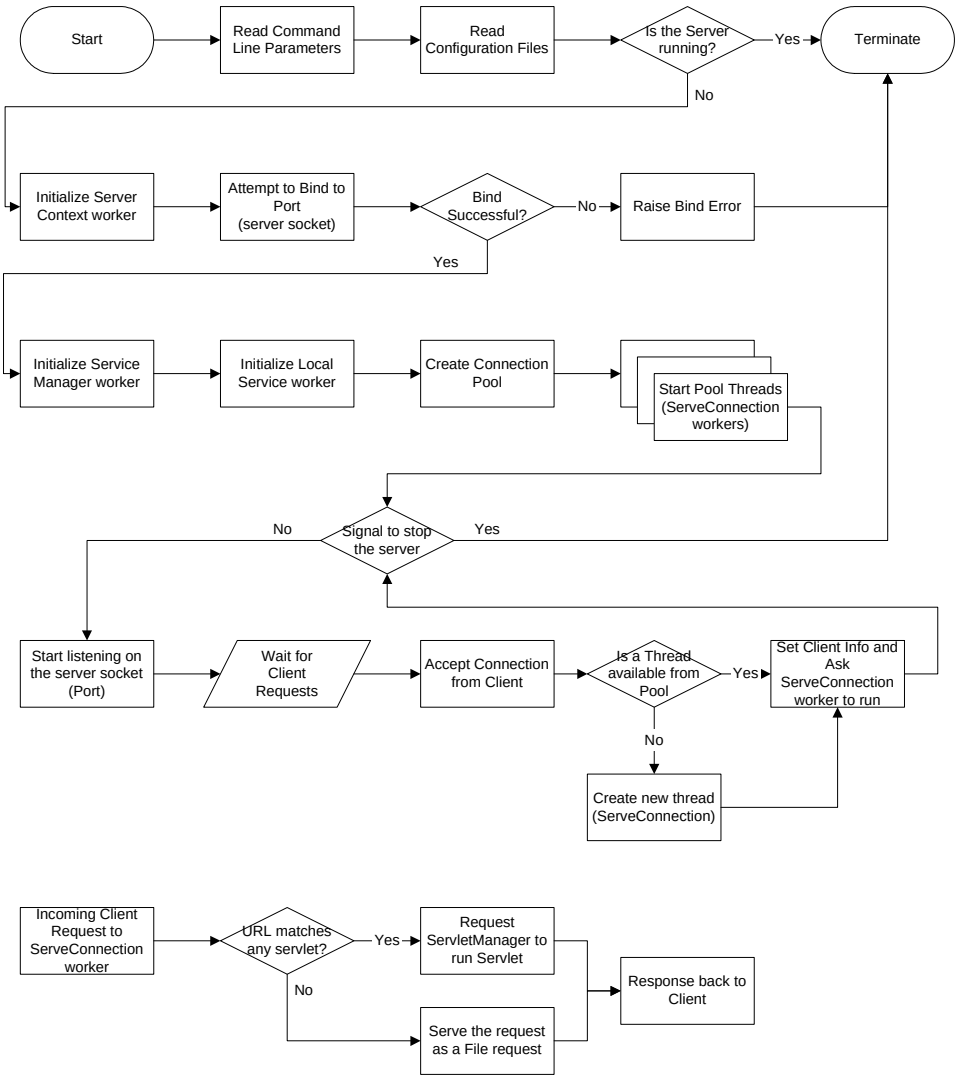

Figure 6 Basic Stages of Pace Web Server

During initialization, the main process, PWS, first reads the command line parameters that tell the process where to find the configuration files. The process then reads the basic web server configuration, session configuration, and servlet configuration files. If another instance of the web server is already running then the process stops by informing

the user that another instance is already running. Otherwise, the process then instantiates the ServerContext that is the thread or worker that takes care of the server. The ServerContext worker attempts to bind to the web server port and if that fails it returns an error. If the bind is successful, the subsequent worker classes are instantiated. The ServiceManager worker initializes a LocalService worker and one or more Remote Service workers as specified in the configuration. The LocalService worker creates a pool of ServeConnection threads and maintains this pool in a stack. The ServeConnection threads are the ones that accept the connection from the server and process the request and finally send output or response back to the client. At this stage the web server is ready to accept connections from the client on the port.

The basic function of the ServerContext is to wait for client requests unless the server is told to stop running either via the console or via command line. Once a client request comes in then it calls ServiceManager to select a service that can service the client request. The ServiceManager then calls the LocalService to respond or serve the client request. The LocalService worker attempts to find any free thread from its stack of ServeConnection workers and if it can find a free worker it provides the worker with the client socket and other information for it to process. If it cannot find a free worker from the stack, meaning the stack is empty, it creates a new worker and runs it to process the client request.

The main server process also instantiates a ServletManager and a SessionManager to initialize and maintain the servlets and sessions respectively.

The ServeConnection worker reads the input from the client socket and proceeds to parse the input line by line. First the headers are read and parsed. The headers are then saved into a Hashtable. After processing all the headers the actual request, which is the URL, is read. It tries to match the pattern to any one of the pre-defined patterns for the set of servlets registered with the server. If there is a match then that servlet is run. Otherwise, it is assumed to be a request for a File and the corresponding file is served to the client. If the file is a directory then the contents of the directory is sent back if directory indexing is set and the directory does not contain the default files.

The listening of the requests from clients continues until a stop signal is sent to the server. This can be done from the tray icon in Windows platform. Once the signal is sent, the server socket or port is closed and the server is shutdown.

4.2 Components of the server

In this section, I describe all the components that make up the server. I describe the Web server components, the Servlet Container details, CGI Support, GUI Management Console and Help files.

4.2.1 Components of the web server

4.2.1.1 PWS

PWS.java is the main class for the server. This is the class that is run either from the command line or via an IDE. It accepts a command line parameter to determine the

location of the configuration files. It then attempts to read the configuration files from this directory. If the option is not specified, it defaults to the *current directory* the server is running so that the command line parameter is made optional. The configuration files are assumed to be in the "conf" subdirectory. If it cannot read the configuration files it raises an error and quits. Once it reads the configuration files successfully, it initializes the main worker for the web server, the ServerContext worker, and then runs that thread. A net outcome of these changes is that once the code for the web server is uncompressed by anyone who installs it, she can run the web server successfully without any changes and it will run with the default options.

In Windows platform, PWS also initializes the tray icon option. A third party library, Jeans [27], enables the server to have an icon available in the System Tray for providing options to view the GUI console, shutdown or restart the server. An additional benefit by using the icon is to determine if another instance is already running or not. If another instance is running PWS will not start a second instance but quit.

The following code intercepts the Windows callback return codes properly and hence it recognizes if any prior instance is already running or not.

```
long result = WindowsTrayIcon.sendWindowsMessage(progName, 1234);

if (result == -1)
{
  // Show our main window
    WindowsTrayIcon.initTrayIcon(progName);
    try
    {
```

```
            (new  PWSconfFrame()).setVisible(true);
        }
        catch (TrayIconException e)
        {
            System.out.println("PWS: Error: "+e.getMessage());
        }
        catch (InterruptedException e)
        {
        }
    }
    else if(result != -1)
    {
      System.out.println("PWS:  The  Pace  Web  Server  is  already
    running");
      System.exit(1);
    }
```

Within the ServerContext, I have separated the initial bind to the port and the actual listening to client requests so that any bind errors can be raised back to the PWS process. This way, regardless of whether it is running on Windows or Linux or any other platform, the server attempts to bind to the specified port for the web server first during initialization of ServerContext. If it cannot bind itself it raises an error. This is most probably the case when another instance is running on the same port and is a good indicator of such occurrence.

If I do not trap this exception, then when the server is started in Linux with another instance already running we will not get a correct error message and the second instance

will not do anything. This way the server behaves somewhat consistently no matter where it is run. PWS.java initializes the ServerContext thus:

```
try
{
  server = new ServerContext(confs);
}
catch( BindException be )
{
  System.out.println("PWS: Unable to Bind to port " +
confs.getInteger("port",80) +
    "\n Another instance of PWS may be running already");
  be.printStackTrace();
  throw be;
}
```

The ServerContext.java constructor does not catch the bind exception, it is declared as throwing it so that it will be caught in PWS.java.

```
public ServerContext(Configurations confs) throws BindException,
IOException
{
 ---
 ---
    serverSocket = new ServerSocket(
port,MAX_NUMBER_OF_CONNECTIONS);
    System.out.println("Server: Listening to port " + new
Integer(port).toString());
 ---
 ---
```

```
}
```

In addition, PWS provides some helper methods to retrieve the configuration files for other threads in the server. For example, `getServletConf()` and `getSessionConf()` will retrieve the servlet and session configuration files respectively.

There is a user-friendly logging mechanism to identify from which module the messages are written from and also make sure that all error messages are correctly raised and written to the system output or to the PWS log file. The log file determination and verification of whether log file can be written to, are done up front so that if some error happens the process will quit instead of generating some runtime exceptions. If messages are not correctly identified by their origin, it might result in confusion during troubleshooting. For example, the output without a friendly log message may look like:

```
Starting PWS Server...Web Server is Online!
ServiceCount....1
Listening to port 8989
```

Though this is a simple example, in reality if messages appear without any indication of which module or where they originated from, it will be difficult to track or fix. The log messages in PWS are listed with the source or module name and other pertinent location information along with the actual message text.

```
PWS: Starting PWS Server...PWS: Web Server is Online!
```

```
ServiceManager: ServiceCount....1
Server: Listening to port 8989
```

The various log methods are all overridden such that there is only one implementation and all other methods call the single implementation in turn.

4.2.1.2 ServerContext

ServerContext is the main worker process in the Pace web server. It initializes the server socket and listens for incoming connections. During initialization it instantiates a ServiceManager worker to which it hands off the client socket on accepting an incoming request. The ServerContext worker maintains the infinite loop waiting for client requests. Upon shutdown instruction it closes the server socket and exits out of the loop. The service manager initializes the LocalService worker that actually maintains the pool of threads.

4.2.1.3 LocalService

The LocalService worker process handles all the incoming requests. It creates a pool of threads that are determined by the RESERVED_NUMBER_OF_CONNECTIONS parameter. The threads are instances of ServeConnection worker threads. The pool is maintained in a stack and whenever an incoming connection comes in the top most thread in the stack is popped out and it handles the request. When the stack is empty it creates a new thread and assigns it to the new request. Once the request processing is complete the

threads are pushed back into the stack for the next request. The priority of the main ServerContext thread and the child ServeConnection threads can be set in the PWS.conf file manually or via the console.

4.2.1.4 ServeConnection

ServeConnection is the actual worker thread that handles the incoming request by parsing the request and sending back the response. The parserequest() is the main method that handles the incoming connection from the client. The stages can be outlined as follows:

Request-Header: It first reads the headers and determines the type of the request. The usual requests are either GET or POST or HEAD.

HTTP Version: It also determines from the first line of the request the HTTP versions (0.9 or 1.0 or 1.1) and chooses the appropriate response headers. After this step it reads in all the request headers like “If-Modified-Since” and so on. It also keeps the connection on the socket open if the client sends a “Keep-Alive” request in the header. This way the client and server need not open connection again and again if there are multiple GET requests from the client. The most often case is a page that contains images and other additional data that requires a separate roundtrip to the server. If the socket is kept open it performs much better than a separate open/close phase. The Keep-Alive is specified only for HTTP/1.1 versions and for all other HTTP versions in the header the ServeConnection worker sets the Keep-Alive to false.

Cookies: The next step it does is to read the incoming cookies. The cookie can contain the session information if a session is set by a servlet using the mechanism of a HTTPSession. The session management is described later. The entire list of cookies are read in and maintained in a table internally.

Whenever the session information comes in the header the session is retrieved with the help of the Session Manager and its last accessed time is updated.

Servlet or File?: Based on the request URI it then decides if this request is for a Servlet or a simple File Request. First, it decodes the encoded URL and then calls a helper method from the ServletManager module to determine if this matches any of the URL patterns mapped for the servlets registered with the server. Please refer to the next section on servlet management for more details.

If the request is for a known servlet, it then gets a handle to the servlet and calls its service() method.

Otherwise this is assumed to be a file-request. It then proceeds to determine if this is a simple file or a directory. If directory, then it searches for default files. The default files are listed in the main PWS configuration file. Most often this is either “index.html” or “index.htm.” The server can understand a list of default files and it goes through this list in the order specified in the configuration file and the first one found is sent back to the client. If none of the default files are found it sends a directory listing if the directory-indexing parameter is set to true.

File restrictions: The server does not send the following files to the client:

- A non-existent file is replied with a 404 Not-Found error.
- Any non-readable or hidden files (hidden files are set using the properties in Windows) is refused with a 403 Forbidden
- While sending the directory listing, none of the hidden files are listed.

Table 7 lists the statuses most often sent back by the Pace Web Server.

Table 7 Status codes sent most often by Pace Web Server

Status Code - Status Message	**Scenario**
200 OK	When everything was processed successfully
304 Not-Modified	Sent when a GET operation found the resource and is available but not modified since the date sent by the client. Most often it is done when the client browser has cached a page or image and would only want it if it had changed since the cache date
400 Bad Request	The request was improperly formed like required headers missing or bad/malformed url etc.
403 Forbidden	When the request can be processed successfully but was denied because of permissions or file is hidden or unreadable, etc.
404 Not Found	When the file or resource requested cannot be found on the server
501 Not Implemented	When the request type is not implemented on the server.
500 Internal Server Error	Any internal errors encountered by the server. Usually if there is a problem running a servlet.

4.2.2 Servlet Container

Figure 7 below depicts the various interfaces in the Servlet APIs and their implementation classes in PWS.

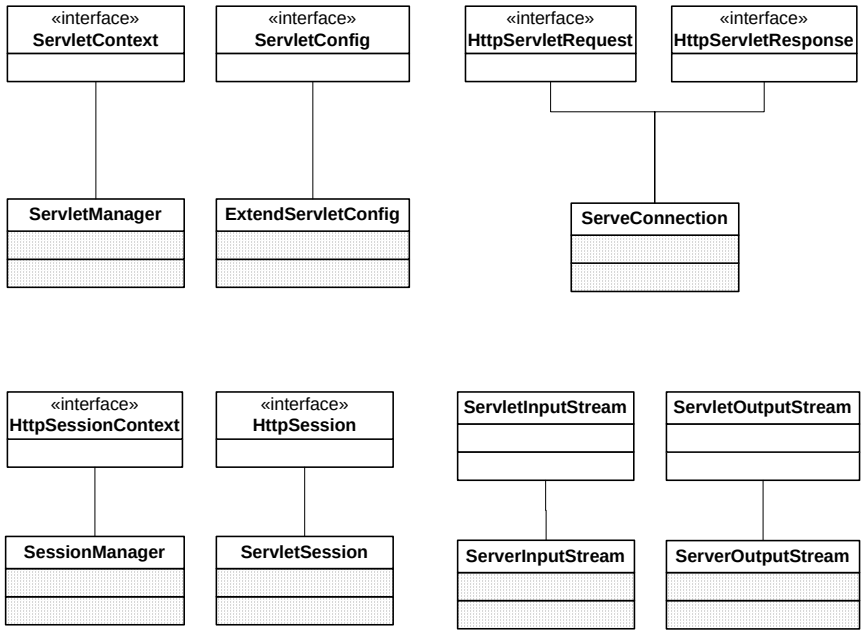

Figure 7 Servlet API and their Implementation Classes in PWS

4.2.2.1 ServletManager

The servlet manager is the main worker module that manages the servlets in the web server. It implements the methods in the ServletContext interface. It maintains two sets of tables. One, a list of all servlets and their url patterns or mappings. Second, a list of the servlets and their fully qualified class names. The mapping of url patterns is used to set up an alias path or alias paths for the servlets. For example, in my sample files, I have

defined a servlet called DateServlet and have set up the mapping as servlet/DateServlet in addition to the DateServlet pattern. When a request comes in like http://localhost:8989/servlet/DateServlet this request is sent to DateServlet for processing. The mapping lets the user put her classes in any directory but present a uniform path to the clients for all the servlets.

Field Code Changed

The servlet manager uses the servlet configuration file, Servlet.conf, to read and register the servlets. These servlet are preloaded and are available as soon as the web server is up and running. Additional servlets can be added while the server is running using the GUI console and existing servlets can be removed as well using the console.

The ServletManager provides methods to add any servlet, remove any servlet and also calls the servlet's destroy() method when necessary. As it implements the ServletContext it provides the servlets with the useful methods to write to the servlet container log file, getting the MIME types, etc.

The servlet manager also adds some default servlets. At present the only one it adds is the CGIServlet. The CGI servlet handles the requests for CGI programs. The CGI Servlet is registered with the following URL patterns:

```
*.cgi, *.pl, *.bat, and *.exe
```

There are some minor variations in how the CGI scripts are run by the CGIServlet on Windows platform. On other platforms the CGI scripts are run as is.

*.cgi & *.pl: These are run using the "Perl" interpreter. On Windows the scripts are run as:

```
Perl <cgi-script-name>
```

The Perl interpreter is assumed to exist in the PATH environment variable. This is necessary in order to successfully process any POST requests as the CGIServlet sends the user form information via the input stream to the script.

On other platforms the cgi-script is run as is.

```
<cgi-script-name>
```

*.bat : The Windows Batch file is run as indicated below and this applies only to the Windows Platform.

In Windows NT, 2000 or XP the following applies.

```
CMD /C <batch-file-name>
```

In Windows 95 or 98 it is run as

```
COMMAND /C <batch-file-name>
```

Also note that since the CGIServlet runs *.bat files, it is possible to set up a script with a *.bat name in Linux, for example, and get it run as a CGI script.

*.exe : Any executable script is run as is as a CGI program. I provide a couple of C programs compiled as executable in the Windows platform. I have also compiled the same C file into an executable for Linux as well.

4.2.2.2 CGIServlet

CGIServlet handles requests for CGI scripts. The scripts can be written in Perl or C/C++ executables and for simple scripts that need not have to do lot of processing. Even MS-DOS batch files can be run as a CGI script. The CGI process invocation is controlled by the Enable-CGI parameter to the PWS.

The CGIServlet parses the request parameters and the headers from the client browser and formats them and create all the environment variables and command line parameters before invoking the requested CGI script as required by the CGI protocol.

The CGI program is run using the `Runtime.exec()` call in Java.

It provides the CGI program or script, among other things, with the following:

- The environment variables like PATH, SERVER_NAME, SERVER_PORT etc.
- The query string is passed in as the QUERY_STRING environment variable for any GET requests
- For POST requests, the input stream from the client is read and passes it as the input stream for the CGI script. The length of the input data is sent as the CONTENT_LENGTH parameter.

As with the CGI protocol, the response from the CGI program is sent without any pre or post processing by the CGIServlet. It is the responsibility of the CGI program itself to send the proper headers with the status and the content. The bare minimum output that needs to be sent to the client is as follows. An empty line is required after the content-type output.

```
HTTP/1.0 200 OK
Content-type: text/html

<html>
<head><title>...</title></head>
<body>
...
...
</body></html>
```

4.2.3 Session Management

The Servlet API requires the implementation of a HTTP Session Interface. This interface lets any servlet create and maintain some user data so that it can be accessed across requests from the client browser within the same session. A client session is the time interval or period a user interacts with a particular host navigating across one or more pages within the same domain. The session is considered active until such time either the session has been timeout by the server because of the user being idle or the user having intentionally closed the browser. The session is to provide the servlet a place to store any Java Objects or primitives, each associated with a name, called the attribute name.

4.2.3.1 SessionManager

The Pace Web Server has a SessionManager that manages the session objects on behalf of the servlets and also has other useful helper methods. The sessions are maintained internally in a Hashtable and each time it is requested it is inspected to make sure it is still active and only then it is returned either to the ServeConnection worker or the servlet that

requests it. The session manager reads the session related parameters from the session configuration file called Session.conf. I have provided the configuration details in a later section.

The session manager uses cookies to maintain information about the client session. The only information written to a cookie is the session id identified by the token ***PWSSessionID***. This id is generated using the Java random number generator and applying additional operations so as to maintain the uniqueness among the session identifiers created.

When a servlet requests the session object, it is created if it is not already available and the session identifier is written to the response cookie. The cookie is by default written with a path of "/", which means any and every page from the Pace Web Server can access the cookie. Once the cookie is written back to the response, the browser sends it back with every request as long as the request is for the same domain and path that is valid.

The sessions are periodically checked to make sure they have not been idle beyond the timeout period set up in the configuration file. There is a thread created that runs in the background that goes over the sessions and compares the last accessed time of each session with the current system time. If the interval is beyond the timeout period then it is removed from the internal table. Any objects present in the session data is inspected and if the object has implemented the session bound listener it notifies the object upon "unbound" event.

4.2.3.2 ServletSession

The HttpSession interface is implemented by the ServletSession module. It provides the methods necessary for a servlet to set and unset any values in the session data. It also provides a method to update the last accessed time whenever a request from a client comes in with this session identified via the session identifier cookie. Anytime the servlets set or unset objects in the session data, the last accessed time is not changed. It is only when a client request comes in with the session identifier the session's last accessed time is changed. It provided additional helper methods as required by the interface to get a list of all attributes and values for attributes when requested by the servlet. A session is considered new until such time the client accesses the same session the next time after it is created.

4.2.4 Messages and Error Logging

The Pace Web Server maintains a Log file, PWS.log, for writing useful messages and in case of any errors encountered writing the error message and the stack trace of such errors. The logging of detailed diagnostic and error messages is controlled by a configurable parameter ***Log*** in the PWS.conf configuration file. If the parameter is set to true, then detailed messages are written to the log file and if set to false no detailed messages are written. This can be useful to keep an audit trail of the activities of users accessing the web server and also to troubleshoot any problems, if any.

The same log file is also made available to the servlet manager so that any log messages from the individual servlets via the ServletContext.log() calls can be written. The servlet log messages are also redirected to the PWS.log file transparently.

The following is a sample from the log file with the log parameter set to true.

```
[Sun Nov 16 10:05:38 EST 2003] PWS: Starting PWS Server...


[Sun Nov 16 10:05:38 EST 2003] edu.pace.web.servlets.Counter: init

[Sun Nov 16 10:05:38 EST 2003] Counter: Writing to ServletContext.Log
=>  Counter is initialized

[Sun Nov 16 10:05:38 EST 2003] ServletManager: Servlet Counter added
successfully.


[Sun Nov 16 10:05:38 EST 2003] edu.pace.web.servlets.CgiServlet: init

[Sun Nov 16 10:05:38 EST 2003] edu.pace.web.servlets.CgiServlet: init

[Sun Nov 16 10:05:38 EST 2003] edu.pace.web.servlets.CgiServlet: init

[Sun Nov 16 10:05:38 EST 2003] edu.pace.web.servlets.CgiServlet: init


[Sun Nov 16 10:42:00 EST 2003] ServletConsole: AddServlet: Servlet Name
= DateServlet

[Sun Nov 16 10:42:00 EST 2003] ServletConsole: AddServlet: Class Name =
edu.pace.web.servlets.DateServlet

[Sun Nov 16 10:42:00 EST 2003] edu.pace.web.servlets.DateServlet: init

[Sun Nov 16 10:43:42 EST 2003] ServletConsole: AddServlet: New Servlet
DateServlet added successfully.


[Sun Nov 16 10:46:21 EST 2003] ServletConsole: RemoveServlet: Servlet
does not exist!

[Sun Nov 16 10:47:14 EST 2003] edu.pace.web.servlets.DateServlet:
destroy

[Sun Nov 16 10:47:14 EST 2003] removing non null value in
[servlet.DateServlet.code]
```

```
[Sun Nov 16 10:47:14 EST 2003] removing non null value in
[servlet.DateServlet.urlpattern]

[Sun Nov 16 10:47:14 EST 2003] removing non null value in
[servlet.DateServlet.initarg]

[Sun Nov 16 10:48:24 EST 2003] ServletConsole: RemoveServlet: Servlet
DateServlet removed successfully.


[Sun Nov 16 10:48:44 EST 2003] ServletConsole: AddServlet: Servlet Name
= DateServlet

[Sun Nov 16 10:48:44 EST 2003] ServletConsole: AddServlet: Class Name =
edu.pace.servlets.DateServlet

[Sun Nov 16 10:48:44 EST 2003] Class not found:
edu.pace.servlets.DateServlet

[Sun Nov 16 10:48:44 EST 2003] ServletConsole: AddServlet: Unable to
create new Servlet DateServlet

[Sun Nov 16 10:48:44 EST 2003] edu.pace.servlets.DateServlet

java.lang.ClassNotFoundException: edu.pace.servlets.DateServlet

   at java.net.URLClassLoader$1.run(URLClassLoader.java:200)


[Sun Nov 16 10:53:01 EST 2003] Console: Shutting down the server...
```

4.2.5 GUI Management Console

There is a GUI Management Console available to manage the web server. It provides a GUI application to configure the server parameters like ServerSocket port, the size of the reserved connection pool, the root directory from which the web server serves files, and so on. The console also allows the user to manage the servlet configuration. The user can add new servlets, view the existing servlets and their parameters and update them, if necessary, and also remove an existing servlet. The console can be used to set the session parameters as well. The console provides user-friendly error messages and is built with the platform specific look and feel using Java Swing facilities.

Currently the console is supported only within Windows platform. It uses a third party library Jeans TrayIcon, [27], to provide an icon in the System Tray with a few useful popup menus. The popup menus provide an option to make the console visible, and options to either restart or shutdown the web server. As soon as the web server is started it enables an icon to be visible in the system tray and the web server enables a few popup menus via calls to this library.

The console itself is written in Java and can run on any platform but the interface with the system tray in Windows provides the icon and the associated popup menu. With a few modifications it can be made to work in other platforms without the help of the System Tray icon application.

The following sections describe the various options available in the console.

4.2.5.1 System Tray integration

Figure 8 Popup menus available via the Windows System Tray Icon

The Tray Icon library provides convenient popup menus to display the console, restart the web server and also exit the server. It also has an option to view an “About PWS” dialog.

4.2.5.2 The General Settings panel

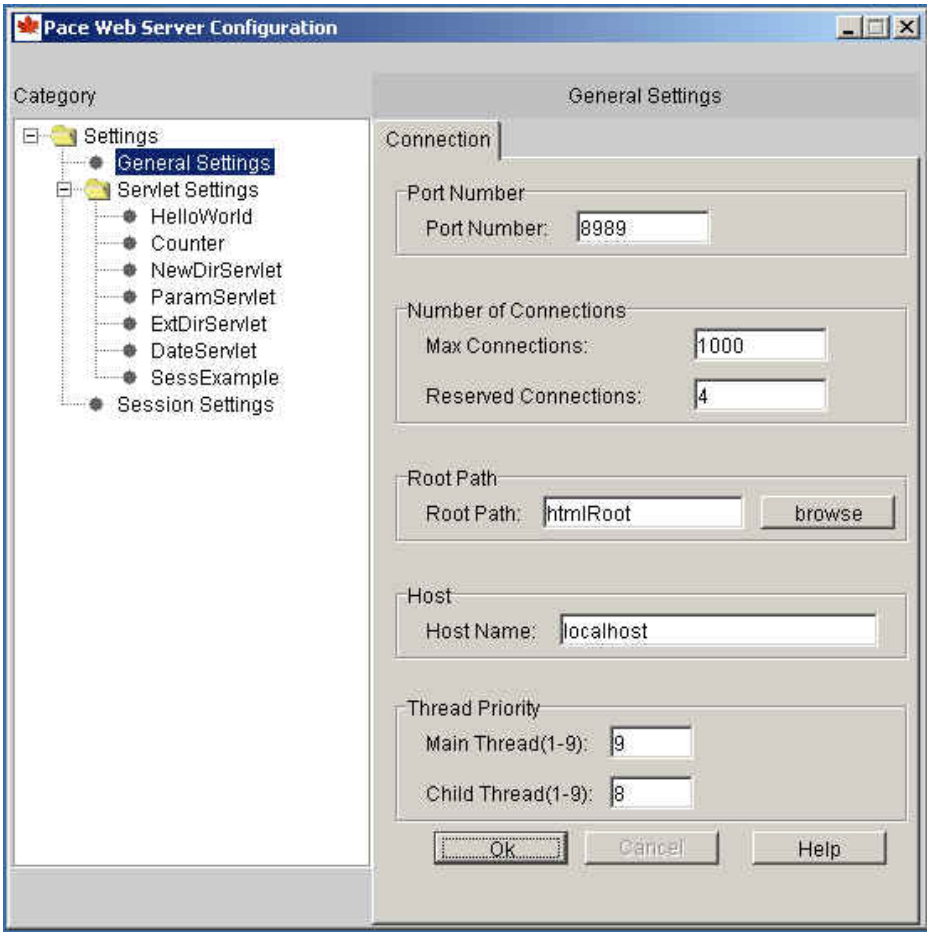

Figure 9 General Settings panel of the Console

The general settings panel as seen in the Figure 9 above provides an option to set the following parameters.

- The port number the web server listens to
- The reserved and maximum number of connections to be set up inside the server

- The RootPath for the web server
- The Host name which the clients can use to access the web server
- The thread priorities for the main thread (ServerContext worker) and the child threads (ServeConnection worker threads.)

Once the user updates the parameters for the general configuration file, the web server needs to be restarted. A dialog, as shown in Figure 10 below, informs the user that the web server needs to be restarted.

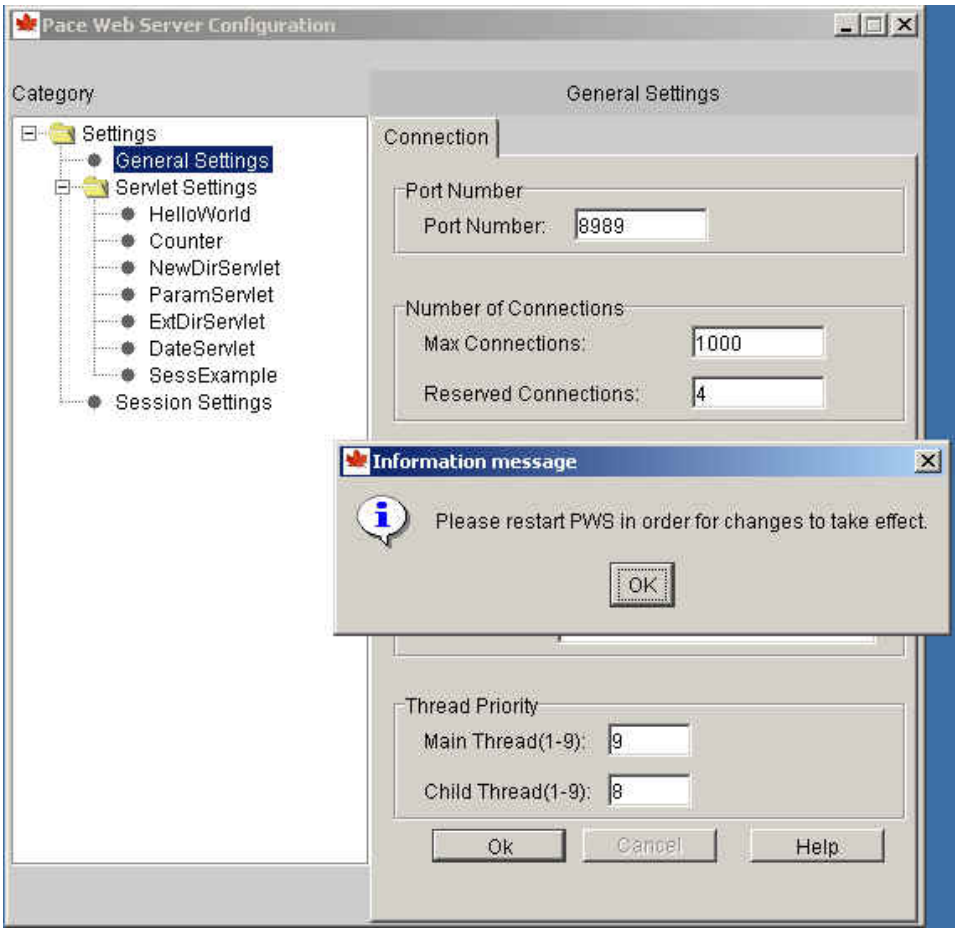

Figure 10 General Settings panel – restart message

The console validates the parameters entered by the user. It does the following validations:

- All the parameters cannot be left empty
- Port number must be greater than 0
- Reserved and Maximum connections must at least be 1 or more
- The main thread and child threads priority can only be between 1 and 9 inclusive
- Only numerical values are allowed for the port, connection and thread settings.

The console provides the following buttons:

OK Button: When the OK button is clicked, the console validates the parameters and saves the changes to the configuration file. It also displays the restart required information message to the user.

Cancel Button: When the Cancel button is clicked, the settings are reverted back to the original values before the changes were made.

Help Button: When the Help button is clicked, the Help information for the General Settings is displayed in a separate window.

Browse Button: When the Browse button is clicked, it brings up the File dialog where the root path for the server can be selected.

4.2.5.3 The Servlet Settings panel

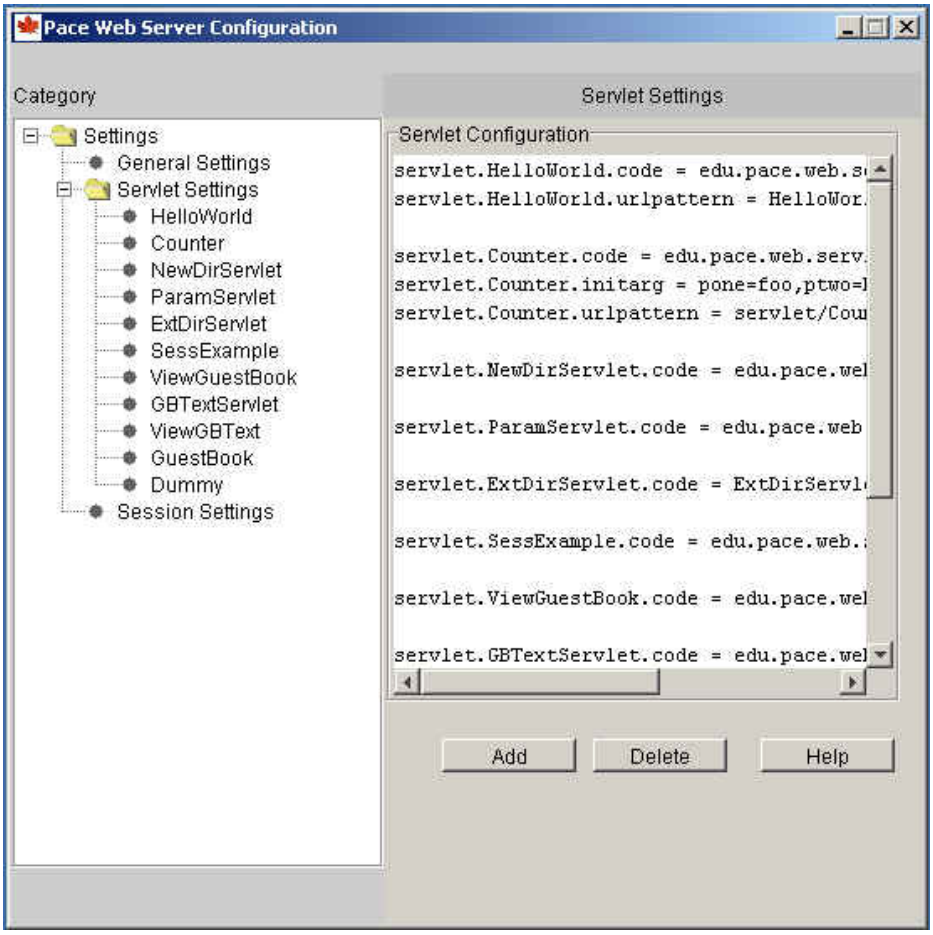

Figure 11 Servlet Settings panel of the Console

The servlet configuration panel, Figure 11 above, lists the servlet configurations on the right side. The node for the servlet settings has all the servlets registered via the configuration file as child nodes as seen in the figure above. The individual settings for each of the servlets can be viewed and modified by clicking that servlet's node. It

provides options to Add a servlet, Delete a servlet and finally displaying the help file related to the servlet settings. It also provides a Help button, which when clicked, opens the Help for servlet settings in a separate window.

Adding a Servlet

The servlet panel provides an option to Add or Delete a servlet. The Add button invokes the add servlet dialog, as in Figure 12 below.

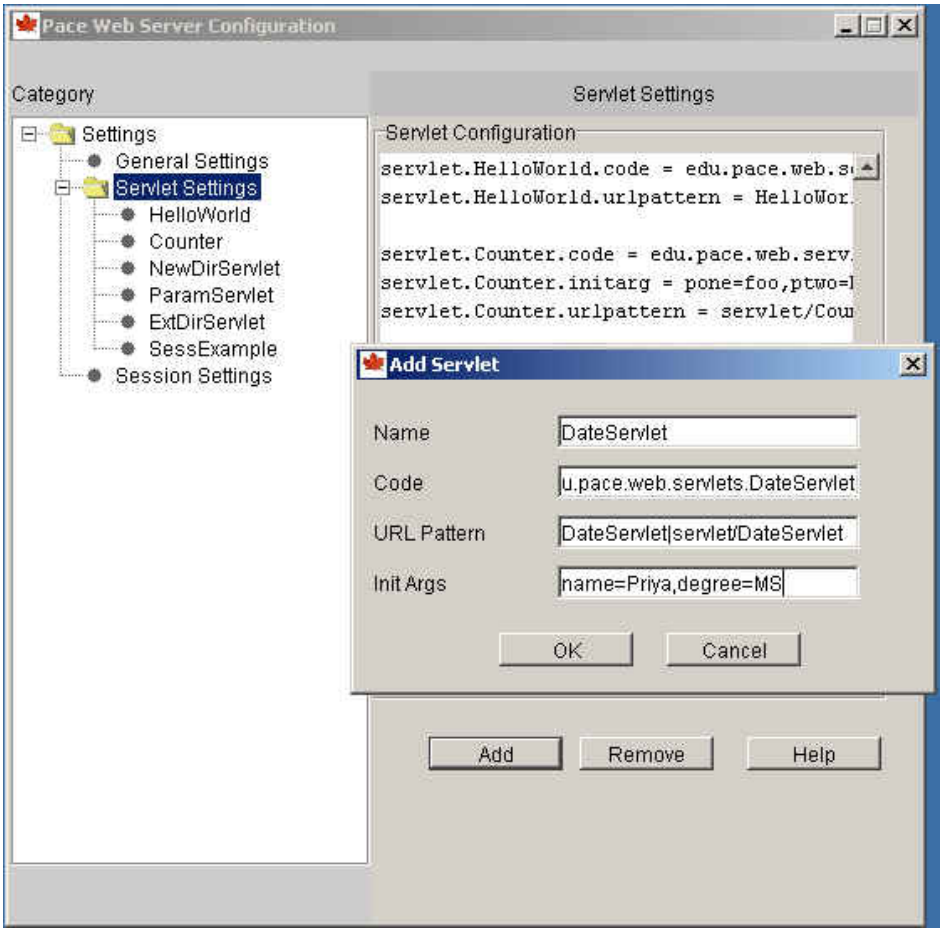

Figure 12 Servlet Settings panel – Add Servlet dialog

The add servlet dialog lets the user enter the servlet name, the fully qualified path name to the servlet class, the URL pattern and any initial arguments. It does the following validations:

- The servlet name cannot be empty and cannot be one of the already existing servlets
- The servlet class name must be entered. It will be validated by the server that it can be loaded and is available at the specified path. For example, edu.pace.web.servlets.Counter. The class must be accessible from the classpath of the server. For example, if a servlet NewDirServlet is created in the package edu.pace.web.newdir then the path need not be changed as it is already available in the classpath of the server. On the other hand, if a servlet ExtDirServlet is created under C:\extdir then this path needs to be part of the CLASSPATH before the server is restarted. Otherwise a class-not-found error is raised.
- The URL mapping is optional. If not specified the name of the servlet will be used as the mapping. Multiple mappings can be specified by delimiting each by a "|" symbol. For example, if DateServlet needs to be accessed by both DateServlet alias and servlet/DateServlet alias, it should be specified as "DateServlet|servlet/DateServlet
- The initial arguments are optional. If specified it will be passed to the servlet upon initialization. The parameters are specified as name=value parameters and multiple parameters are separated by a ",". For example, the init-args could be "name=Priya, degree=MS, university=PACE" and so on.

Note: As soon as the servlet is added it can be accessed by any client. The server need not have to be restarted for the new servlet to take effect.

Also, it is possible to have the same servlet class mapped with two very different servlet names.

As soon as the servlet is added, the tree structure on the left is refreshed automatically to reflect the newly added servlet.

Deleting a Servlet

The panel also provides an option to delete an existing servlet. A dialog opens up for the user to enter the name of the servlet to be deleted. The name entered is validated against the list of servlets that exist at that time. If a valid name is entered then that servlet is deleted immediately, that is, no further requests for the URL mappings for that servlet is allowed and a Not-Found error is thrown back to the client. The servlet class itself is not removed if there are other servlets referencing this class and if there are no other references, then the class is also removed from the server.

Note: Upon deletion, the servlet mapping is immediately removed and the server need not be restarted for the changes to go into effect.

Individual Servlet Settings

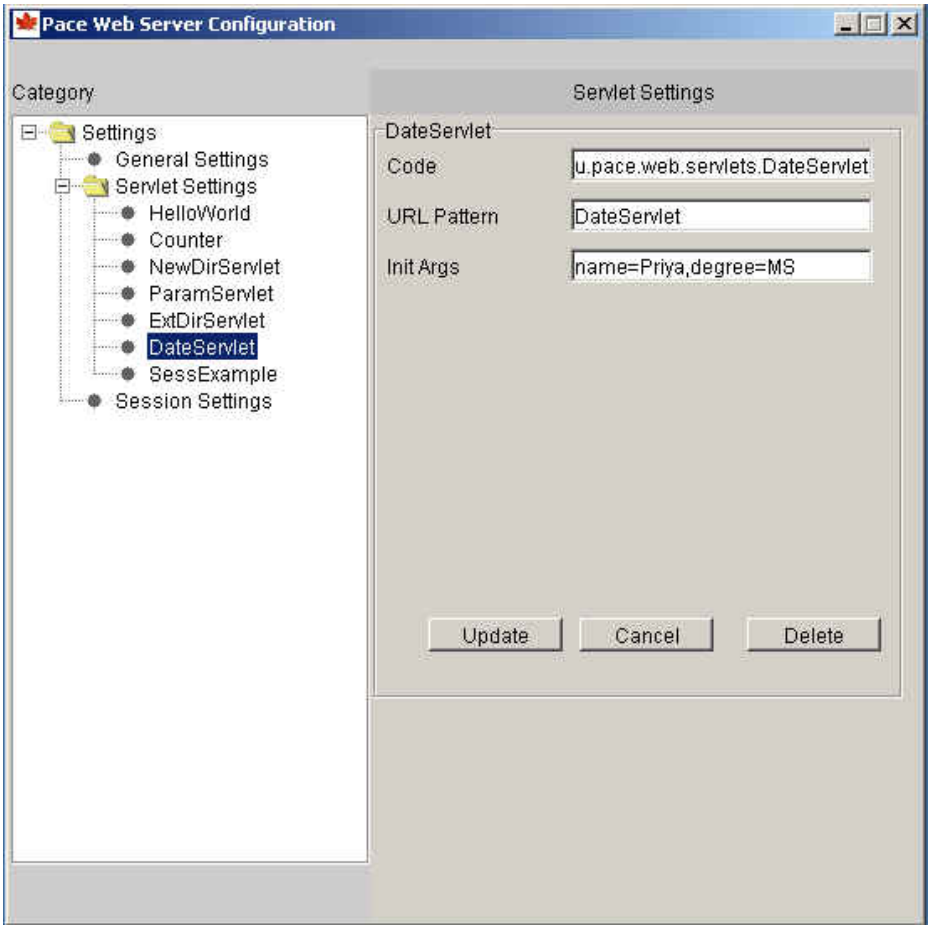

Figure 13 Servlet Settings panel – Individual servlet settings

The right panel lists the individual servlet's settings when a particular servlet is selected on the left pane. The right pane, as in Figure 13 above, shows the servlet name as the border title, servlet class name, the servlet URL mapping, and the initial arguments to the servlet, if any.

Only the URL mapping can be updated for an existing servlet. It validates to make sure that it is not left null. The current displayed servlet can be updated or deleted from this panel. It provides the following buttons:

Update Button: When the Update button is clicked, it updates any changes made to the URL pattern.

Cancel Button: When the Cancel button is clicked, the settings are reverted back to the original values before any changes were made.

Delete Button: The Delete button is used to delete the currently selected servlet. A confirmation message is shown to the user, as in Figure 14 below.

Note: When the URL mapping is updated, the server need not be restarted to use the new mapping and it works automatically at the next client request.

Similarly, the server need not be restarted when a servlet is deleted. Also, the tree structure on the left panel is refreshed automatically to remove the deleted servlet from the list.

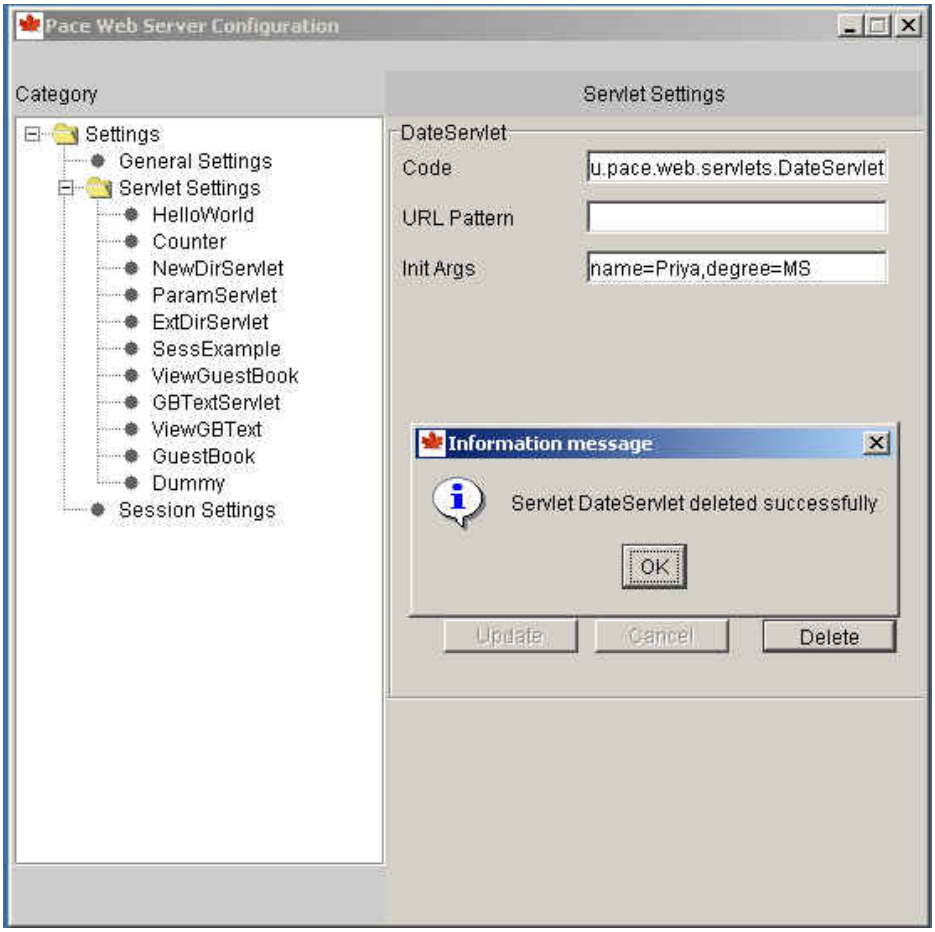

Figure 14 Delete Confirmation message after a Servlet is deleted

4.2.5.4 The Session Settings panel

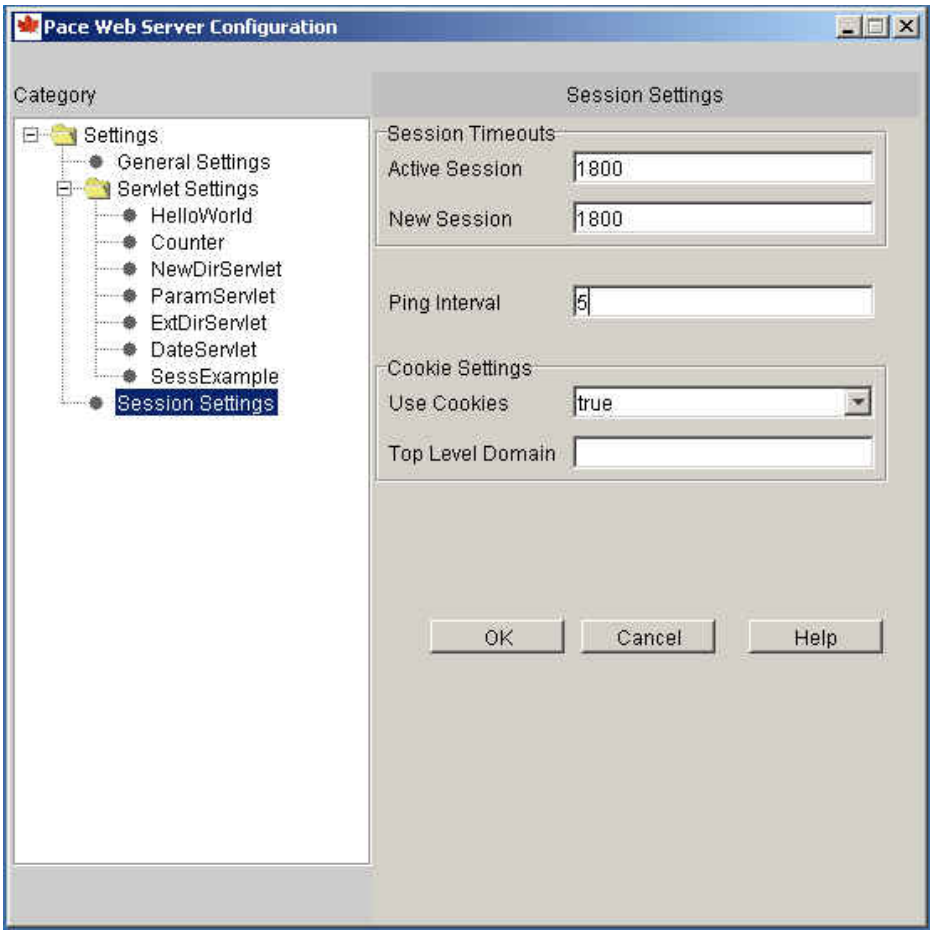

Figure 15 Session Settings panel of the Console

The session settings panel displays the session settings as read from the Session.conf configuration file. It displays the session timeout in seconds, new session timeout in seconds, how often the Session Manager checks for idle sessions (ping interval), enable/disable cookies for session management and the domain written to the cookies.

For a detailed explanation of these parameters refer to the configuration and management section 5.6.3. The panel does the following validations:

- The timeout, new timeout, and ping interval values must be entered by the user
- The timeout values must be greater than 0.
- The ping interval must be greater than 0.
- Only numbers can be entered in the numeric fields.
- The timeout and new timeout values must be at least twice the value of the ping interval parameter.

It provides tooltip help for the various fields and provides the following buttons:

OK Button: When the OK button is clicked, the changes are saved back to the Session configuration file.

Cancel Button: When the Cancel button is clicked, the settings are reverted back to the original values before the changes were made.

Help Button: When the Help button is clicked, the help for the session settings is displayed in a separate window.

Note: When the session settings are changed, they are updated automatically and the server need not be restarted.

4.2.6 Help Files for the console

The console displays help for the various options that are configurable via the console. These help files are retrieved from the Pace Web Server via HTTP rather than statically reading it from the file system. Whenever help is invoked, the help Java frame identifies the URL of the help file and accesses the contents from the Pace Web Server itself. Figure 16 below shows the help displayed for the general settings panel of the console.

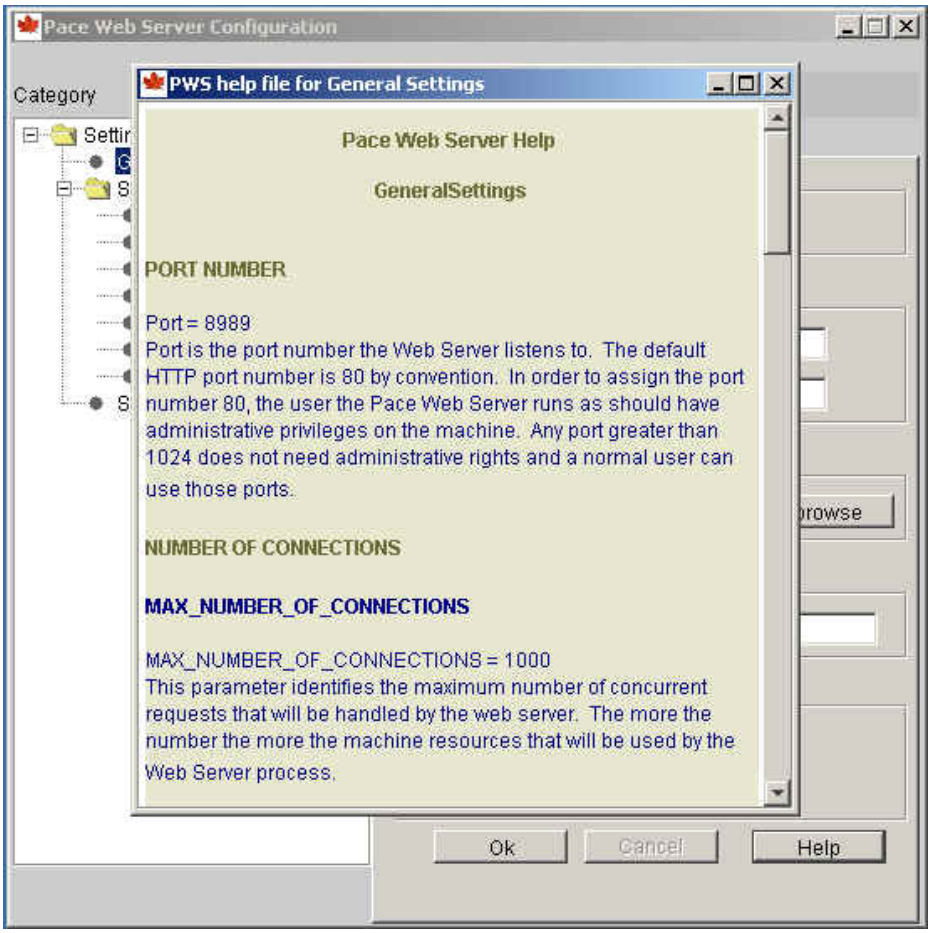

Figure 16 Settings panel – Help Window

4.2.7 Javadoc for sources

The Javadoc for the web server source files can be accessed from the javadoc/index.html page from the Pace Web Server itself. For example, if the server is running on the local machine, the javadoc can be accessed from http://localhost:8989/javadoc/index.html

4.3 Other Features of the Pace Web Server

The following is a list of other features provided by the Pace Web Server.

- The default index page of the Web Server highlights the web server features and provides link to documentation and sample files.
- The Web server comes with a set of sample files demonstrating that it can handle HTML files, CGI scripts in Perl, Batch File and C program executables, Servlets, HTML with Applets, and a simple application that connects to a database.

Chapter 5

Installation and Configuration

5.1 System Requirements

The server is written entirely in Java and can therefore run on any platform that has a Java Virtual Machine Implementation. The server is built and tested using JDK 1.3.1 but should work with any Java version 1.2 and above. The server has been tested in both Windows 2000 and Linux (Redhat 7.1.) However, the GUI management console is available only in Windows as it is integrated with the Windows System Tray.

5.2 Installation

The entire source and compiled code for the Pace Web Server is bundled into a single zip archive, PaceWeb.zip. It contains the Java sources, configuration files, sample HTML and other CGI files, sample servlets, and necessary resource files like images. The zip file has the following structure:

```
PaceWeb
   classes
   conf
   htmlRoot
      examples
   lib
   log
   src
      edu
```

```
pace
   web
      net
      service
      servletmanagement
      servlets
      utils
      win32
```

Depending on the Integrated Development Environment (IDE) used to compile and build the code there might be additional directories, for example, bak (created by JBuilder, by default), and public_html (created by JBuilder and JDeveloper.) Though these are created by the IDE tools they do not affect the build nor running of the server. In addition, while running the server from within these IDEs there are other files created directly under the main PaceWeb directory for runtime support. The IDE specific files are listed in their corresponding section.

The following sections outline how to install, compile and run the server using various options. We start with the simple command line build and run. Throughout these sections I assume that the zip file is uncompressed to the `C:\` drive or `/home/priya` in Windows and Linux respectively. However, regardless of where the archive is unzipped, the Web server is configured for compilation and execution with the default options from the directory to which they are unzipped. The configuration files are assumed to be in the conf sub-directory and the HTML Root is set to the htmlRoot sub-directory. No changes are necessary to run the web server with the default configuration and options.

5.3 Running the Pace Web Server

The web server can be run from the command line or from within an IDE. In a production environment, it will be run from the command line. Follow the steps below.

- Unzip the archive using WinZip or other program in Windows and gunzip or unzip in Linux.
- Edit the PWS.conf file found in conf sub directory and modify the following parameters. (For more information, refer to the configuration and management section 5.6.1)
    - HOST: Edit the host to the appropriate name. The default is localhost. If you want to access the Web server from other machines, you have to provide the local machine's IP address or the fully qualified domain name of the machine as registered in the Domain Name Servers (DNS).
    - PORT: Edit the port to whatever port you want the web server to listen on. The default is *8989.*
    - ROOT_PATH: Edit the Root Path to specify where the web server should look for files and directories for any client request. The default is *htmlRoot*. It is recommended that the default value be used.

- Run the web server either using the supplied script `startserver.bat` (in Windows) or `startserver.sh` (in Linux). Alternately, it can also be run manually.

- To run using the supplied script, open the script `startserver.bat` in a text editor like Notepad. Edit the JAVA_HOME parameter to point to the location of JDK 1.3.1 or other appropriate version in your machine. I have mine in C:\jdk1.3.1. The script automatically sets the necessary CLASSPATH and PATH variables. To run the server, just type

  `Startserver`

- To manually run the web server

  - Include the path to Java in the PATH environment variable,

    `set PATH=C:\JDK1.3.1\bin;%PATH%`

  - Include the current directory (`C:\PaceWeb`) and the required libraries in the CLASSPATH

    ```
    set
    CLASSPATH=.\classes;.\lib\servlet.jar;.\lib\jeans178a.
    zip;%CLASSPATH%
    ```

  - Then run the main class as below:

    ```
    C:\PaceWeb> java edu.pace.web.PWS -c .
    ```

    The same command should also work in Linux.

```
/home/priya/PaceWeb$ java edu.pace.web.PWS -c .
```

- Once the web server is running, point a browser window to the following URL

  http://localhost:8989

Note: In order to run the CGI scripts written in Perl you need to download and install Perl in your machine where the Web Server is running. Perl software for Windows can be downloaded from ActiveState [4]. In Linux, Perl is already pre-loaded in Redhat and installed in /usr/bin

If required, the web server can also be started as a Windows Service in Windows platform. You can use any of the open source/shareware tools available that can install and run any program or batch script as a Windows Service. One of the tools is SRVSTART [19], a simple but powerful utility program.

The server understands the following command line parameters:

```
-c <basepath to conf files>
```

The base path tells the server where to find the various configuration files, where to find the log file to write to if logging is enabled, the icon image file for the server.

The configuration files are read as

<basepath>\conf\PWS.conf

<basepath>\conf\Servlet.conf and

<basepath>\conf\Session.conf

The log file, where PWS writes trace, diagnostic, and error messages will be

<basepath>\log\PWS.log

The icon image file for the various windows will be

<basepath>\images\leaf.gif

For example, if the server is started as below

```
C:\PaceWeb>java edu.pace.web.PWS –c C:\my\folder
```

then it will look for the PWS.conf configuration file in the directory `C:\my\folder\conf`. The complete file name will then be `C:\my\folder\conf\PWS.conf`. Similarly for other files.

Full path to Servlet.conf will be `C:\my\folder\conf\Servlet.conf` and

Full path to Session.conf will be `C:\my\folder\conf\Session.conf`

Full path to log file will be `C:\my\folder\log\PWS.log`

Similarly it applies to Linux with the appropriate file separators ("/" is the file separator in Linux as compared to "\" in Windows)

Note: Although I have listed some directory names as I used it, the code can be unzipped and run from anywhere in the file system.

5.4 Command line compiling and invocation

5.4.1 Windows

I use Windows 2000 as my operating system but the code should work identically in Windows NT or Windows XP Professional versions though I have not tested in those versions. For the steps below, I assume that I have uncompressed the zip archive to C:\ drive though the files can be uncompressed to any location. The files will be created in PaceWeb directory and its subdirectories as indicated in the previous section.

The following steps are required to compile and run the server.

1. Unzip the PaceWeb zip archive using WinZip utility. Use C:\ as the destination folder to extract the contents to and check the "Use Folder Names" option and "All Files" option.

2. Open a command window (Start > Programs > Accessories > Command Prompt or Start > Run and then type cmd.exe in the dialog and click ok)

3. Change directory to `C:\PaceWeb`

4. To run the web server, run the script `startserver.bat` as below

```
C:\PaceWeb>startserver
```

Note 1: All the scripts assume that the Java JDK is available in `C:\jdk1.3.1` directory. If it is installed in a different directory then change the setting for JAVA_HOME inside the batch files.

Note 2: By default the root directory from which the server sends html pages is set to `htmlRoot` sub-directory from the base directory. If the HTML root directory is moved elsewhere or if it needs to be pointed to some other directory it has to be changed in the PWS.conf file.

In addition, to compile all the sources at once, run the script `makeall.bat` as below:

```
C:\PaceWeb>makeall
```

To set the environment only for compiling one or more files individually, run the script *`envset.bat`* as below

```
C:\PaceWeb>envset
```

Once the environment is set, any file can be compiled by specifying the full path to the source Java file as in the example below (all parameters must be in the same line):

```
C:\PaceWeb>javac -verbose -d classes -sourcepath src
src/edu/pace/web/servlets/HelloWorld.java
```

Note: The –verbose option prints detailed information as the Java compiler builds the file. It is optional and need not be specified.

Refer to the appendix for a listing of the batch files.

5.4.2 Linux [Redhat 7.1]

I have tested the compilation and execution in Redhat Linux 7.1. However, the same shell scripts should work in any version of Linux/Unix as long as the Java compiler and JVM are available. The scripts in Linux are generic and written using the basic shell commands that must be available in any version of Linux or Unix.

The compilation and execution steps are very similar to Windows except that the path and directory information is different. For the steps below, I am using my home directory, `/home/priya`, to uncompress and run the server though the files can be uncompressed to any location. Obviously, in the steps below substitute *`priya`* with the your actual login name.

1. Open a terminal window. In KDE it is the `Konsole` window. In GNOME, it is simply the `gnome-terminal`.
2. Change directory to the user home directory, if not already there, by simply running the `cd` command without any parameters.
3. Unzip the PaceWeb zip archive using *`unzip`* or *`gunzip`* utilities.

Formatted: Bullets and Numbering

4. Change directory to /home/priya/PaceWeb

5. Enable execute permissions on startserver.sh and makeall.sh scripts (type the entire command in the same line) if not already available
   `/home/priya/PaceWeb $ chmod u+x startserver.sh makeall.sh`

6. To run the web server, run the script startserver.sh as below
   `/home/priya/PaceWeb $ startserver.sh`

Note 1: All the scripts assume that the Java JDK is available in `/usr/local/jdk1.3.1` directory. If it is installed in a different directory then change the setting for JAVA_HOME inside the shell scripts.

Note 2: By default the root directory from which the server sends html pages is set to `htmlRoot` sub-directory. If the HTML root directory is moved elsewhere or if it needs to be pointed to some other directory it has to be changed in the PWS.conf file.

In addition, to compile all the sources at once, run the script `makeall.sh` as below:

```
/home/priya/PaceWeb $ makeall.sh
```

To set the environment only for compiling one or more files individually, source, do not run, the script envset.sh as below:

```
/home/priya/PaceWeb $ . envset.sh
```

Note: “Sourcing” a shell script merely sets the environment variables and other settings in the current shell whereas executing a script will run the script in a child shell forked from the parent or current shell.

Once the environment is set, any file can be compiled by specifying the full path to the source Java file as in the example below (all parameters must be in the same line):

```
/home/priya/PaceWeb $ javac -verbose -d classes -sourcepath src
src/edu/pace/web/servlets/HelloWorld.java
```

Note: The –verbose option prints detailed information as the compiler builds the file. It is optional and need not be specified.

Refer to the appendix for a listing of the scripts.

5.5 Using Integrated Development Environments

5.5.1 Oracle9i JDeveloper

I used JDeveloper in Windows 2000 for my entire development, debugging and testing. It is one of the best IDE tools available for Java, web and J2EE development. It provides a lot of wizards to create, run, test, and debug code. It has an integrated debugger by which you can set breakpoints, view the variable and instance values during execution and so on. I used the version 9i (version 9.0.3) for my work.

In JDeveloper you create a Workspace and then a Project within a workspace to assemble all the related code. It uses the J2EE compliant directory structure for the Java source files along with any HTML or JSP files, if any. Refer to Oracle JDeveloper webpage [14] for more details on how to download JDeveloper, tutorials on how to use JDeveloper to create Java applications, etc.

I have included the JDeveloper workspace and project files as a part of the PaceWeb.zip archive. Within JDeveloper the project can be opened simply using the steps below:

1. Click File > Open
2. Navigate to C:\PaceWeb directory in the File Open dialog
3. Select PaceWeb.jws file and click OK.

Now the project is ready for compilation and running.

To compile all the files in the Project

Right Click on PaceWeb.jpr and use the context menu to select “Rebuild PaceWeb.jpr”.

It should take a few seconds to a minute to compile all the files depending on the processing power of the computer.

To run the pace web server,

Right Click on PaceWeb.jpr and use the context menu to select “Run PaceWeb.jpr”.

The Pace Web Server should be up and running.

In addition to the PaceWeb.jws and PaceWeb.jpr project files, JDeveloper creates the following files during compilation and build. These are necessary only for JDeveloper and are not required while building and running from the command line. I have indicated JDeveloper created directories in bold and JDeveloper created files underlined to differentiate from regular files and directories.

```
PaceWeb
   PaceWeb-data-sources.xml
   PaceWeb-oc4j-app.log
   PaceWeb-oc4j-app.xml
   WebAppRunner.html
   public_html
      WEB-INF
         web.xml
```

The following instructions are to create a new project in JDeveloper from the source files without using the existing workspace and project files. [*JDeveloper IDE online help]*

To create a new workspace:

1. From the main menu, choose File | New or, with any workspace or project node selected, right-click and choose New. The New Gallery opens.
2. In the Categories tree, select General.
3. In the Items list, double-click Workspace.
4. In the New Workspace dialog that appears, enter a directory path and a filename. You can elect also to open the workspace in a new navigator or to populate it with a new empty project. Note that the default directory path stores workspace and

project files in a folder entitled mywork within the JDeveloper directory structure. You can also choose to store your files outside of this structure, by specifying such in the dialog.

5. Click OK.

Alternately, to bypass the New Gallery, select the Workspaces node in the Navigator, right-click, and choose New Workspace

To create a new empty project and add it to the selected workspace:

1. In the Navigator, select the workspace within which the project will appear.
2. From the main menu, choose File | New, or right-click and choose New. The New Gallery opens.
3. In the Categories tree, select General.
4. In the Items list, double-click Empty Project.
5. In the New Project dialog, enter the project's path and filename. To select an existing directory, click Browse. For more information, press F1 or click Help from within the dialog.
6. Click OK.

A new empty project appears in the Navigator. It inherits whatever default properties you've already set. To alter project properties for this particular project, right-click on the filename and choose Project Settings.

Alternately, to bypass the New Gallery, select the workspace in the Navigator, right-click, and choose New Empty Project.

Once an empty Project is created, you can add existing source files. The following figures show the project settings required to build and run the pace web server code. The command line parameters “-c .” are optional.

Note: In order to compile and run properly the following two options must be set correctly.

(a) In the libraries section, include the jeans178a.zip and servlet.jar files in the project settings

(b) In the Runner options make sure you select hotspot as the JVM.

5.5.2 Borland JBuilder 7

Borland's JBuilder is another one of the best IDE for Java, web and J2EE development similar to JDeveloper. I tried out JBuilder 7 to make sure my code can be built using any IDE. I used JBuilder 7 Personal Edition for my testing purposes. Refer to Borland JBuilder webpage [6] for more information on IDE and tutorials.

Again with JBuilder, you need to create a Project and add all the source files. JBuilder also provides a lot of wizards to create, run, test, and debug code. It also has an integrated debugger by which you can set breakpoints, view the variable and instance values during execution and so on.

I have included the JBuilder project files as a part of the PaceWeb.zip archive. Within JBuilder the project can be opened simply using the steps below:

1. Click File > Open Project
2. Navigate to C:\PaceWeb directory in the File Open dialog
3. Select PaceWeb.jpx file and click OK.

Now the project is ready for compilation and running.

To compile all the files in the Project

Right Click on PaceWeb.jpx and use the context menu to select “Rebuild”.

It should take a few seconds to a minute to compile all the files depending on the processing power of the computer.

To run the pace web server,

Go to the menu Run > Run Project

The Pace Web Server should be up and running.

Alternately, a new project can be created in JBuilder and the existing source files can be added to create the project. The command line parameters “-c .” are optional.

Note: In order to compile and run properly the following two options must be set correctly.

(a) In the required libraries section, include the jeans178a.zip and servlet.jar files in the project settings

(b) In the Run options create a new Runtime Configuration (I have named it PaceWeb) in the project settings.

In addition to the PaceWeb.jpx project file, JBuilder creates the following files during compilation and build. These are necessary only for JBuilder and are not required while building and running from the command line. I have indicated JBuilder created directories in bold and JBuilder created files underlined to differentiate from regular files and directories.

```
PaceWeb
   Bak
   classes
      package cache
   PaceWeb.jpx.local
   PWSlibraries.library
```

5.6 Configuration and Management

The Pace Web Server is configured by one or more configuration files as listed below. The configuration files should be available inside the conf sub-directory. The files are simple text files with name=value pairs of properties similar to Java properties files. Some of the property names occur only once, for example, the port number, and others can occur many times with each time listing a different value, for example, servlet names.

The configuration parameters in the files can be either edited manually in the appropriate files or they can be updated or added as the case may be using the GUI console available.

For each configuration file parameter, the first line is the default value. All possible values are listed beneath the description. Unless otherwise specified, the parameters are mandatory and cannot be listed more than once.

5.6.1 PWS.conf

This is the main configuration file that lists the server name, web server port number, etc.

5.6.1.1 HOST

```
HOST = localhost
```

Field Code Changed

The HOST parameter identifies the name of the server where the web server is running. If you want to access the Web server from other machines, you have to provide the local machine's IP address or the fully qualified domain name of the machine as registered in the Domain Name Servers (DNS).

5.6.1.2 PORT

```
port = 8989
```

The Port parameter is the port number the Web Server listens to. The default HTTP port number is 80 by convention. In order to assign the port number 80, the user who starts the Pace Web Server should have administrative privileges on the machine. Any port greater than 1024 does not need administrative rights and a normal user can use those ports.

5.6.1.3 ROOT_PATH

```
ROOT_PATH = htmlRoot
```

The ROOT_PATH identifies the base directory from which the web server will read and send the files and/or execute relevant scripts. The web server makes sure that any requests that go higher up the directory tree (the parent directory to this root directory, its

parent and so on) will be denied. Since the web server reads all files in this and its subdirectories it makes sense to ***not*** set this to a place where there are sensitive or confidential files.

5.6.1.4 RESERVED_NUMBER_OF_CONNECTIONS

```
RESERVED_NUMBER_OF_CONNECTIONS = 4
```

The Web Server pre-allocates a set of threads that will handle the client connections. The number of threads allocated during initialization is specified by this property. The more the number the more the machine resources that will be used by the Web Server process during initialization and these resources will not be freed until the server is shutdown.

5.6.1.5 MAX_NUMBER_OF_CONNECTIONS

```
MAX_NUMBER_OF_CONNECTIONS = 1000
```

This parameter identifies the maximum number of concurrent requests that will be handled by the web server. The larger the number is, the more the machine resources that will be used by the Web Server process.

5.6.1.6 MainThreadPriority

```
MainThreadPriority = 9
```

The MainThreadPriority parameter identifies the priority of the main web server thread. Possible values are between 1 and 9 inclusive, 9 being the highest and 1 being the lowest. It is better to have the main thread priority at least 1 higher than the child thread priority.

5.6.1.7 ChildThreadPriority

```
ChildThreadPriority = 8
```

The ChildThreadPriority parameter identifies the priority of the child worker threads that handle the requests from clients. Possible values are between 1 and 9 inclusive, 9 being the highest and 1 being the lowest. It is better to have the main thread priority at least 1 higher than the child thread priority.

5.6.1.8 Default_Filename

```
Default_Filename = index.html
Default_Filename = index.htm
```

The Default_Filename parameter identifies the default page that will be sent if the request either ends with a trailing "/" or a directory name on the server is specified. This can be set up multiple times with multiple names. The server will try to find the default files one by one in the order they are set up in the configuration file and the first available file will be sent. This will also be the file sent when the base name of the web server is sent as the URL, for example, http://localhost:8989/

Field Code Changed

Even if directory indexing parameter is set to true, the server will first go over the default files first before sending the directory contents back to the client.

5.6.1.9 Servlet_Configuration_File

```
Servlet_Configuration_File = Servlet.conf
```

The Servlet_Configuration_File parameter identifies the name of the servlet configuration file that is used by the Web Server. The file identified here is read by the server to determine the Servlets registered with the Server.

5.6.1.10 Session_Configuration_File

```
Session_Configuration_File = Session.conf
```

The Session_Configuration_File parameter identifies the name of the Session configuration file that is used by the Web Server. The file identified here is read by the server to determine the Session settings for any Servlet Sessions created.

5.6.1.11 serverName

```
serverName = PWS
```

The serverName parameter is the Name of the server identified in the Response Headers sent by the Pace Web Server. There is no need to change this value.

5.6.1.12 serverVersion

```
serverVersion = 1.0
```

The serverVersion parameter is the version of the server identified in the Response Headers sent by the Pace Web Server. There is no need to change this value.

5.6.1.13 Log

```
Log = true
```

When the Log parameter is set to true, the Web Server logs detailed trace messages about requests, errors and other pertinent information. The Log parameter should be set to true only during development or during troubleshooting.

5.6.1.14 ENABLE_CGI

```
ENABLE_CGI = true
```

The ENABLE_CGI parameter enables the support for CGI scripts within the Pace Web Server. Setting this parameter to true will enable CGI support and any requests for running CGI scripts are carried out successfully.

5.6.1.15 Directory_Indexing

```
Directory_Indexing = true
```

If a file name requested by a client is a directory on the server then the server lists the directory contents if this parameter is set to true. If the parameter is set to false, then the directory listings are disabled and a forbidden error is sent back. If the directory contains any of the default files specified by the Default_Filename parameter then that file is sent back to the client regardless even if this parameter is set to true.

5.6.2 Servlet.conf

The servlet configuration file identifies the servlets that are registered with the web server along with the complete path to the servlet class file. It also lists the optional aliases for the servlets, any initialization arguments that are made available to the servlet via the ServletContext getInitParameter() calls.

There are four property names that are used for every servlet. The properties are:

5.6.2.1 servlet

```
servlet = <any valid Java identifier>
```

The servlet parameter identifies the name of the servlet. This is a mandatory parameter for a servlet. The name should be a legal Java language identifier. This is the name that will be used internally to refer to the servlet as well as listed in the GUI management console.

Once the name is specified, it is also used in the other property names. Once used for a particular servlet class, it should not be reused for any other servlet class.

If no url pattern property is specified then the name of the servlet is used as the url pattern to match incoming requests.

Please refer to section 5.6.2.5 for an example.

5.6.2.2 servlet.<servlet-name>.code

```
servlet.<servlet-name>.code = fully.qualified.class.name
```

The servlet.servlet-name.code parameter identifies the fully qualified class name of the servlet using which the web server resolves and instantiates the class for this servlet. Please refer to section 5.6.2.5 for an example.

5.6.2.3 servlet.<servlet-name>.urlpattern

```
servlet.<servlet-name>.urlpattern = <servlet-alias1>|<servlet-alias2>|…
```

The servlet.servlet-name.urlpattern parameter identifies the name mappings or aliases of this servlet that can be specified on the request URL. The webserver matches the incoming requests to one of the URL patterns specified here and if there is a match then that servlet is run to respond to the client request. If no URL pattern is specified then the name of the servlet is used as the url pattern to match incoming requests.

Any number of mapping names can be specified for a single servlet separated by a | symbol. Make sure that no two servlets have the same mapping or url pattern.

Please refer to section 5.6.2.5 for an example.

5.6.2.4 servlet.<servlet-name>.initarg

```
servlet.<servlet-name>.initarg = name1=value1|name2=value2|…
```

or

```
servlet.<servlet-name>.initarg = name1=value1
servlet.<servlet-name>.initarg = name2=value2
```

The servlet.servlet-name.initarg parameter identifies one or more servlet initialization parameters for the named servlet. The initialization parameters are made available to the servlet during its initialization via the ServletContext() Interface. The init-args are specified by themselves as name=value pairs. Any number of name=value pairs can be specified for a servlet. The init-args can be specified either on the same line or on multiple lines each for the same property.

Please refer to section 5.6.2.5 for an example.

5.6.2.5 Example of a servlet configuration setting

A sample servlet configuration file that I have included with the zip file archive is listed below.

```
servlet = HelloWorld
```

```
servlet.HelloWorld.code = edu.pace.web.servlets.HelloWorld
servlet.HelloWorld.urlpattern = HelloWorld|servlet/HelloWorld

servlet = DateServlet
servlet.DateServlet.code = edu.pace.web.servlets.DateServlet
servlet.DateServlet.initarg = pone=foo
servlet.DateServlet.initarg = ptwo=bar
servlet.DateServlet.initarg = pthree=foobar
servlet.DateServlet.initarg = pfour=barfoo
```

In the sample file above, the HelloWorld servlet is run using the code found in edu.pace.web.servlets.HelloWorld class file and the DateServlet is run using the code found in edu.pace.web.servlets.DateServlet.

The HelloWorld servlet is matched to any requests in the form of http://localhost:8989/HelloWorld or http://localhost:8989/servlet/HelloWorld. There are no initialization parameters specified for the HelloWorld servlet.

Field Code Changed

Field Code Changed

On the other hand, the DateServlet does not have a mapping name and hence only the http://localhost:8989/DateServlet is matched to this servlet. When the DateServlet is initialized by the servlet container inside PWS, the parameters pone, ptwo, pthree, and pfour are made available during its init() call via the ServletContext interface. The value of the parameters will be foo, bar, foobar, barfoo respectively.

Field Code Changed

5.6.3 Session.conf

The session configuration file identifies the session related parameters for the web server. These sessions are Servlet Sessions that are created for any servlet that requests HTTP Session objects. The properties that can be configured for the sessions are as follows:

5.6.3.1 session.timeout

```
session.timeout = 1800
```

The session timeout parameter indicates the maximum time an idle session is kept inside the web server. Once a session is idle beyond the timeout period, it is inactivated and any further requests for the session after this interval is denied. Any objects that are stored in the session by the servlets are lost when the session is deactivated. The value is specified in seconds and the default value is 1800 seconds (30 minutes.) If this parameter is not specified in the configuration file then the default value is assumed.

5.6.3.2 session.newtimeout

```
session.newtimeout = 1800
```

The new session timeout parameter indicates the maximum idle time a newly created session is kept inside the web server. Once a new session is idle beyond the new session timeout period, the web server inactivates it. Again, any objects that are stored in the session by the servlet are lost when the session is deactivated. A session remains in a “new” status until the next request from the same client comes in with a reference to this

session in the request header. In addition, a new session is also checked against the active session timeout value to determine if it needs to be deactivated.

For example, if a servlet creates a session and puts some data inside it, the session still remains in a “new” status until the next request from the same client comes in. If this “new session” times out before the next request from the same client, the data put in by the servlet is lost.

The value is specified in seconds and the default value is 1800 seconds (30 minutes.) If this parameter is not specified in the configuration file then the default value is assumed.

5.6.3.3 session.checkFrequency

```
session.checkFrequency = 5
```

The session check frequency parameter determines how often the web server’s session manager worker verifies the active sessions to make sure that none of them remain active beyond their idle time period as specified by the two timeout parameters. The value is specified in seconds and the default value is 5 seconds. If this parameter is not specified in the configuration file then the default value is assumed.

The active session timeout and the new session timeout values must be at least twice the session check frequency duration.

5.6.3.4 session.useCookies

```
session.useCookies = true
```

The session useCookies parameter determines whether the web server uses cookies in managing the sessions. If the parameter is set to true then the web server sets the session identifier in a cookie while assembling the response from the servlets. The session identifier is retrieved from subsequent incoming requests from the client browser to lookup the session the client is participating with the web server. The sessions themselves and their contents are maintained inside the web server. The value is set as either “true” or “false.” The default value is “true.” If the parameter is not specified in the configuration file then the default value is assumed.

5.6.3.5 session.topleveldomain

```
session.topleveldomain = .pace.edu
```

The session top-level domain parameter determines the domain name that is set in the Cookies that are sent to the browser whenever a servlet creates an HTTP Session object. The domain parameter set during the creation of the cookie tells the client browser to return the cookie to other hosts within the same domain. Only hosts within a specific domain can set a cookie for its domain.

Appendix A.

PWS Configuration Files

I have listed the basic configuration files for the Pace Web Server below. These can and should be modified appropriately to the user installation.

PWS.conf

```
serverName = PWS
RESERVED_NUMBER_OF_CONNECTIONS = 4
Default_Filename = index.html
Default_Filename = index.htm
Servlet_Configuration_File = Servlet.conf
port = 8989
MAX_NUMBER_OF_CONNECTIONS = 1000
serverVersion = v1.0
Log = true
ROOT_PATH = htmlRoot
Session_Configuration_File = Session.conf
ENABLE_CGI = true
MainThreadPriority = 9
Directory_Indexing = true
ChildThreadPriority = 8
HOST = localhost
```

Servlet.conf

```
servlet.HelloWorld.code = edu.pace.web.servlets.HelloWorld
servlet.HelloWorld.urlpattern = HelloWorld|servlet/HelloWorld

servlet.Counter.initarg = pone=foo
servlet.Counter.initarg = ptwo=bar
servlet.Counter.initarg = pthree=foobar
servlet.Counter.initarg = pfour=barfoo
servlet.Counter.code = edu.pace.web.servlets.Counter
servlet.Counter.urlpattern = servlet/Counter|Counter
```

```
servlet.DateServlet.code = edu.pace.web.servlets.DateServlet
servlet.DateServlet.urlpattern = DateServlet
servlet.DateServlet.initarg = name=Priya
servlet.DateServlet.initarg = degree=MS

servlet.ParamServlet.code = edu.pace.web.servlets.ParamServlet
servlet.NewDirServlet.code = edu.pace.web.newdir.NewDirServlet
servlet.SessExample.code = edu.pace.web.servlets.SessExample

servlet.ExtDirServlet.code = ExtDirServlet

servlet = HelloWorld
servlet = Counter
servlet = NewDirServlet
servlet = ParamServlet
servlet = ExtDirServlet
servlet = DateServlet
servlet = SessExample
```

Session.conf

```
session.useCookies = true
session.newtimeout = 1800
session.checkFrequency = 5
session.timeout = 1800
```

Appendix B.

Scripts for compilation and execution

makeall.bat – Compilation batch file for Windows

```
@echo off
REM ============================================================
REM Pace Web Server Build Script
REM Author: Priya Srinivasaraghavan
REM ============================================================

IF "%OS%"=="Windows_NT" SETLOCAL

SET JAVA_HOME=C:\JDK1.3.1

IF NOT EXIST %JAVA_HOME%\bin\javac.exe GOTO :nojava

SET PACEWEB_BASE=.
SET
CLASSPATH=%PACEWEB_BASE%\classes;%PACEWEB_BASE%\lib\servlet.jar;%
PACEWEB_BASE%\lib\jeans178a.zip;%PACEWEB_BASE%\Extdir;%CLASSPATH%

SET PATH=%JAVA_HOME%\bin;%PATH%

echo Classpath is %CLASSPATH%
echo.
echo Path is %PATH%
echo.

javac -d classes -sourcepath src src/edu/pace/web/PWS.java
javac -d classes -sourcepath src
src/edu/pace/web/servlets/HelloWorld.java
javac -d classes -sourcepath src
src/edu/pace/web/servlets/DateServlet.java
javac -d classes -sourcepath src
src/edu/pace/web/servlets/Counter.java
javac -d classes -sourcepath src
src/edu/pace/web/servlets/Dummy.java
javac -d classes -sourcepath src
src/edu/pace/web/servlets/GBTextServlet.java
javac -d classes -sourcepath src
src/edu/pace/web/servlets/GuestBook.java
javac -d classes -sourcepath src
src/edu/pace/web/servlets/ParamServlet.java
```

```
javac -d classes -sourcepath src
src/edu/pace/web/servlets/SessExample.java
javac -d classes -sourcepath src
src/edu/pace/web/servlets/ViewGBText.java
javac -d classes -sourcepath src
src/edu/pace/web/servlets/ViewGuestBook.java

echo.
echo Compilation Completed
echo.

goto end

:nojava
echo Unable to find javac.exe in the Java Home directory
[%JAVA_HOME%] specified
echo %JAVA_HOME%\bin\javac.exe : No such file exists
echo exiting...

:end
IF "%OS%"=="Windows_NT" ENDLOCAL
```

makeall.sh – Compilation shell script for Linux

```
#!/bin/sh
# ============================================================
# Pace Web Server Build Script
# Author: Priya Srinivasaraghavan
# ============================================================

JAVA_HOME=/usr/local/jdk1.3.1_04
export JAVA_HOME

if [ ! -f $JAVA_HOME/bin/javac ]; then
  echo "Unable to find javac in Java Home directory ( $JAVA_HOME
) specified."
  echo "$JAVA_HOME/bin/javac : No such file exists"
  echo "Exiting..."
  exit
fi

PACEWEB_BASE=.
export PACEWEB_BASE

CLASSPATH=$PACEWEB_BASE/classes:$PACEWEB_BASE/lib/servlet.jar:$PA
CEWEB_BASE/lib/jeans178a.zip:$PACEWEB_BASE/ExtDir:$CLASSPATH
export CLASSPATH
```

```
echo Classpath is $CLASSPATH
echo
echo Path is $PATH
echo
echo "java version is"
$JAVA_HOME/bin/java -version
echo

javac -d classes -sourcepath src src/edu/pace/web/PWS.java
javac -d classes -sourcepath src
src/edu/pace/web/servlets/HelloWorld.java
javac -d classes -sourcepath src
src/edu/pace/web/servlets/DateServlet.java
javac -d classes -sourcepath src
src/edu/pace/web/servlets/Counter.java
javac -d classes -sourcepath src
src/edu/pace/web/servlets/Dummy.java
javac -d classes -sourcepath src
src/edu/pace/web/servlets/GBTextServlet.java
javac -d classes -sourcepath src
src/edu/pace/web/servlets/GuestBook.java
javac -d classes -sourcepath src
src/edu/pace/web/servlets/ParamServlet.java
javac -d classes -sourcepath src
src/edu/pace/web/servlets/SessExample.java
javac -d classes -sourcepath src
src/edu/pace/web/servlets/ViewGBText.java
javac -d classes -sourcepath src
src/edu/pace/web/servlets/ViewGuestBook.java

echo
echo Compilation completed
echo
```

startserver.bat – Execution batch file for Windows

```
@echo off
REM ============================================================
REM Pace Web Server Execution Script
REM Author: Priya Srinivasaraghavan
REM ============================================================
IF "%OS%"=="Windows_NT" SETLOCAL

SET JAVA_HOME=C:\JDK1.3.1
rem SET JAVA_HOME=C:\JBuilder9\jdk1.4
```

```
IF NOT EXIST %JAVA_HOME%\bin\java.exe GOTO :nojava

SET PACEWEB_BASE=.
SET
CLASSPATH=%PACEWEB_BASE%\classes;%PACEWEB_BASE%\lib\servlet.jar;%
PACEWEB_BASE%\lib\jeans178a.zip;%PACEWEB_BASE%\ExtDir;%CLASSPATH%

SET PATH=%JAVA_HOME%\bin;%PATH%

echo Classpath is %CLASSPATH%
echo.
echo Path is %PATH%
echo.

TITLE Pace Web Server
java -hotspot edu.pace.web.PWS -c .

goto end

:nojava
echo Unable to find java.exe in the Java Home directory
[%JAVA_HOME%] specified
echo %JAVA_HOME%\bin\java.exe : No such file exists
echo exiting...

:end
IF "%OS%"=="Windows_NT" ENDLOCAL
```

startserver.sh – Execution shell script for Linux

```
#!/bin/sh
# ============================================================
# Pace Web Server Execution Script
# Author: Priya Srinivasaraghavan
# ============================================================

JAVA_HOME=/usr/local/jdk1.3.1_04
export JAVA_HOME

if [ ! -f $JAVA_HOME/bin/java ]; then
  echo "Unable to find java in Java Home directory ( $JAVA_HOME )
specified."
  echo "$JAVA_HOME/bin/java : No such file exists"
  echo "Exiting..."
  exit
fi
```

```
PACEWEB_BASE=.
export PACEWEB_BASE

CLASSPATH=$PACEWEB_BASE/classes:$PACEWEB_BASE/lib/servlet.jar:$PA
CEWEB_BASE/lib/jeans178a.zip:$PACEWEB_BASE/ExtDir:$CLASSPATH
export CLASSPATH

echo Classpath is $CLASSPATH
echo
echo Path is $PATH
echo
echo "java version is"
$JAVA_HOME/bin/java -version
echo

$JAVA_HOME/bin/java edu.pace.web.PWS -c .
```

envset.bat – Environment setup batch file for Windows

```
@echo off
REM ============================================================
REM Pace Web Server Environment Setup Script
REM Author: Priya Srinivasaraghavan
REM ============================================================

SET JAVA_HOME=C:\JDK1.3.1

IF NOT EXIST %JAVA_HOME%\bin\javac.exe GOTO :nojava

SET PACEWEB_BASE=.
SET
CLASSPATH=%PACEWEB_BASE%\classes;%PACEWEB_BASE%\lib\servlet.jar;%
PACEWEB_BASE%\lib\jeans178a.zip;%PACEWEB_BASE%\ExtDir;%CLASSPATH%

SET PATH=%JAVA_HOME%\bin;%PATH%

echo Classpath is %CLASSPATH%
echo.
echo Path is %PATH%
echo.

goto end

:nojava
```

```
echo Unable to find java.exe in the Java Home directory
[%JAVA_HOME%] specified
echo %JAVA_HOME%\bin\java.exe : No such file exists
echo JAVA_HOME setting may be incorrect
echo exiting...

:end
echo.
```

envset.sh – Environment setup shell script for Linux

```
# ============================================================
# Pace Web Server Environment Setup Script
# Author: Priya Srinivasaraghavan
# ============================================================

JAVA_HOME=/usr/local/jdk1.3.1_04
export JAVA_HOME

if [ ! -f $JAVA_HOME/bin/java ]; then
  echo "Unable to find java in Java Home directory ( $JAVA_HOME )
specified."
  echo "$JAVA_HOME/bin/java : No such file exists"
  echo "JAVA_HOME setting may be incorrect"
  echo "Exiting..."
  exit
fi

PACEWEB_BASE=.
export PACEWEB_BASE

CLASSPATH=$PACEWEB_BASE/classes:$PACEWEB_BASE/lib/servlet.jar:$PA
CEWEB_BASE/lib/jeans178a.zip:$PACEWEB_BASE/ExtDir:$CLASSPATH
export CLASSPATH

PATH=$JAVA_HOME/bin:$PATH

echo Classpath is $CLASSPATH
echo
echo Path is $PATH
echo
echo "java version is"
$JAVA_HOME/bin/java -version
echo
```

Appendix C.

Acronyms and Abbreviations

ASP: Active Server Pages

CGI: Common Gateway Interface

HTML: HyperText Markup Language

HTTP: HyperText Transfer Protocol

JSP: JavaServer Pages

Perl: Practical Extraction and Report Language

References

Books

[1] Marty Hall, *Core Servlets and JavaServer Pages*, Java 2 Platform, Enterprise Edition Series, Sun Microsystems Press

[2] Merlin Hughes, Michael Shoffner, and Derek Hamner, *Java Network Programming*, 2nd edition, Manning Publications Company, May 1999

[3] Robert Orfali & Dan Harkey, *Client/Server Programming with JAVA and CORBA*, 2nd Ed., 1998, John Wiley & Sons.

Internet Web Sites & Resources

[4] Active Perl – Perl for Windows, http://www.activestate.com/Products/ActivePerl/

[5] Apache Software Foundation, http://www.apache.org

[6] Borland JBuilder, http://www.borland.com/jbuilder/

[7] CPAN – Comprehensive Perl Archive Network, http://www.cpan.org/

[8] Microsoft, IIS 6.0 Help, http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/windowsserver2003/proddocs/standard/iiswelcome.asp

[9] Microsoft, IIS Overview, http://www.microsoft.com/windowsserver2003/iis/evaluation/default.mspx

[10] Microsoft, Technical Overview of IIS, http://www.microsoft.com/windowsserver2003/docs/IISOverview.doc

[11] Netcraft – Web Server Survey, Oct 2003, http://news.netcraft.com/archives/web_server_survey.html

[12] Netscape – Client Side State – HTTP Cookies, http://wp.netscape.com/newsref/std/cookie_spec.html

[13] Operating System Names and Codes in Java, http://lopica.sourceforge.net/ref.html#os-arch-codes

[14] Oracle9i JDeveloper – IDE for Java Development, http://otn.oracle.com/products/jdev/index.html

Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed

[15] Perl.com – The source for Perl, http://www.perl.com/

[16] ServerWatch -- Comparison of Apache, IIS and SunOne Servers, http://www.serverwatch.com/tutorials/article.php/3074841

[17] ServerWatch – Apache Server Features, http://www.serverwatch.com/stypes/servers/index.php/15877

[18] ServerWatch – List of available Web servers, Oct 2003, http://www.serverwatch.com/stypes/compare/index.php/compare_d2Vi

[19] SRVSTART, Installing programs as Windows NT Services, http://www.nick.rozanski.com/services.htm

[20] Sun Java Servlet Development Kit 2.1, Servlet Archives, http://java.sun.com/products/servlet/archive.html

[21] Sun Java Servlets - http://java.sun.com/products/servlet/

[22] Sun Java Tutorial http://java.sun.com/docs/books/tutorial/index.html

[23] Sun JDK - Java Development Kit http://java.sun.com/

[24] Sun JDK API documentation http://java.sun.com/j2se/1.3/docs/api/index.html

[25] TheServerSide Application Server Matrix, http://www.theserverside.com/reviews/matrix.jsp

[26] TheServerSide http://www.theserverside.com

[27] Windows Tray Icon with Java, http://jeans.studentenweb.org/java/trayicon/trayicon.html

[28] World Wide Web Consortium, CGI – common gateway interface, http://www.w3.org/CGI/

[29] World Wide Web Consortium, Home Page, http://www.w3c.org

[30] World Wide Web Consortium, HTML Home Page, http://www.w3.org/MarkUp/

[31] World Wide Web Consortium, HTTP Protocol, http://www.w3.org/Protocols/

Field Code Changed

Other References

[32] Lee, Qing Jiang, *Pure Java Implementation of a Scalable Application Server*, Master's Thesis report, Concordia University, 2001