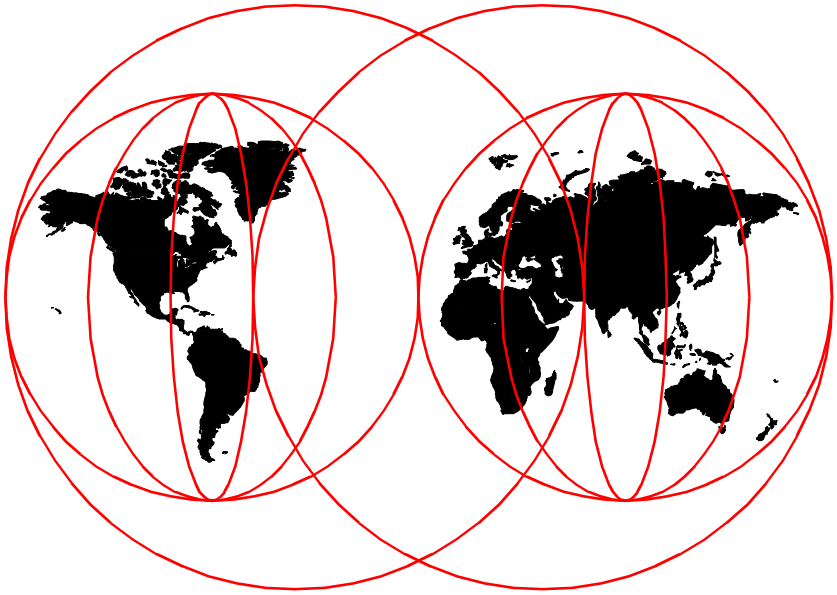
Linux for WebSphere and DB2 Servers

Jakob Carstensen, Herman Chen, Doug Marker, Dan Cornell, Paul Zikopoulos



International Technical Support Organization

www.redbooks.ibm.com

SG24-5850-00

SG24-5850-00

International Technical Support Organization

Linux for WebSphere and DB2 Servers

October 1999

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix B, “Special notices” on page 275.

First Edition (October 1999)

This edition applies to the following products:

- WebSphere Application Server 2.0.3
- DB2 Universal Database 6.1
- VisualAge for Java 3.0 (technical evaluation)
- IBM HTTP Server
- Apache Web Server 1.3.6
- IBM Java JDK 1.16
- Blackdown JDK 1.17 V.3
- Caldera OpenLinux 2.2 and 2.3
- Red Hat Linux 6.0
- SuSE Linux 6.1
- TurboLinux 3.6

Note

This book is based on a pre-GA version of a product and may not apply when the product becomes generally available. We recommend that you consult the product documentation or follow-on versions of this redbook for more current information.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HZ8 Building 678
P.O. Box 12195
Research Triangle Park, NC 27709-2195

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Preface

This redbook describes how to implement WebSphere Application Server, DB2 Universal Database, VisualAge for Java, Apache Web Server and IBM HTTP Server on the Linux operating system. We use four different distributions of the Linux operating system: Caldera OpenLinux, Red Hat Linux, SuSE Linux and TurboLinux.

Furthermore the book provides a discussion of Java servlets including servlet programming model, design patterns for e-commerce using servlets and a comprehensive discussion of WebSphere Application Server security.

The book is divided into five parts.

In the first part we discuss IBM's commitment to the Linux operating system and provide installation procedures for the different applications.

In the second part we discuss Web programming models, and include an overview of Java servlets, the advantages of servlets, and servlet programming.

Part 3 discusses WebSphere Application Server security, Enterprise Java Beans (EJB) and Extended Markup Language (XML). Also, part 3 discusses design patterns for e-commerce using servlets.

Part 4 discusses how to access 2- and 3-tier databases using DB2 Universal Database.

Part 5 contains three samples that will help to test your access to DB2. Part 5 also includes an appendix on how to install IBM HTTP Server with SSL.

The team that wrote this redbook

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

Jakob Carstensen is an Advisory Specialist for Netfinity Servers at the International Technical Support Organization, Raleigh Center. He manages residencies and produces redbooks. His most recent publication was *Implementing Oracle Parallel Server on Netfinity Servers*. Before joining the ITSO, he worked in Denmark both for the IBM PC Institute teaching TechConnect and Service Training courses and for IBM PSS performing level-2 support of Netfinity products. He has a Bachelor of Electronic Engineering degree and has worked for IBM for the past nine years.

Herman Chen is an IT Specialist for the IBM Global e-business solutions and a SUN-certified Java programmer from Endicott, New York. He has worked as an application developer on numerous e-business projects. During his recent project, he worked on a team that delivered a Web-enabled solution, e-business Accelerator, for small and medium businesses. His interests are in designing and developing applications to enhance business processes. He has a Bachelor's degree in Computer Science and a Master's degree in Business Administration from the State University of New York. He has worked for IBM for the past four years.

Doug Marker is Director of Research and Development at Internet Age Pty Ltd., Sydney Australia. He has over 32 years of experience in Computing. His areas of expertise include designing and building e-commerce systems plus technologies including UNIX/Linux, RDBMS, OOT (Java and Smalltalk). He has written and presented extensively on e-commerce and emerging Internet technologies.

Dan Cornell is a Vice President of Rare Medium, Inc., an Internet solutions firm helping clients develop e-commerce Internet strategies, improve business processes, develop marketing communications, branding strategies, and interactive content using Internet-based technologies and solutions. He directs technical production efforts in the San Antonio office, specializing in server-side Java Web application development. Previously, Dan was Vice President of Engineering for Atension, Inc. where he led their technical development team and architected Atension's internal engineering practices. Prior to his work with Atension, Dan developed simulation applications for the Air Force with Southwest Research Institute. He has a Bachelor of Science degree in Computer Science, and has published several papers on topics ranging from data security to high-end graphical simulations.

Paul Zikopoulos is a senior member of IBM's DB2 Universal Database Development team located in Toronto, Canada. As a team member for the past five years, he has written over 25 books on DB2 and its family of products. A certified DB2 Advanced Technical Expert and certified Database Administrator, he is a key player in the development of DB2 and its documentation.

Thanks to the following people from the ITSO Center in Raleigh for their invaluable contributions to this project:

Marco Pistoia
Barry Nusbaum
David Watts
Linda Robinson
Tate Renner
Shawn Walsh
Gail Christensen
Margaret Ticknor
Mike Haley

Thanks to the following IBM employees:

Peyen Fong, Marketing Manager, Linux Software
Jonathan Prial, Director, Integrated Solutions and Linux Marketing
Julie Briddon, Marketing Communications
Susan Williams, DB2 Platform Development
Brian Brandt, DB2 Install Development
Felix Lee, DB2 Install Development
Pamela Burnside, DB2 Install Development Manager
Scott Baily, Toronto Information Development
Doug Foulds, Information Development
Blair Adamache, DB2 Development
Brian Brandt, DB2 Development
Arnold Goldberg, WebSphere Development
Scott Sams, WebSphere Development

Thanks to the following people from Atension, Inc.:

Michael Brinkman
Daniel Glover
Adam Houghton

Thanks also to Mark Olberg from Caldera Systems.

Comments welcome

Your comments are important to us!

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

- Fax the evaluation form found in “ITSO redbook evaluation” on page 291 to the fax number shown on the form.
- Use the online evaluation form found at `http://www.redbooks.ibm.com/`
- Send your comments in an Internet note to `redbook@us.ibm.com`

Part 1. Introduction

This part introduces the Linux operating system and IBM's commitment to migrate software products to run on this new and exciting platform.

Chapter 1. The Linux operating system

Linux is a freely distributed, modifiable operating system (OS) that runs on most of the popular microprocessors on the market today, including Intel 80x86 and IBM PowerPC. Linux is free because it is licensed under the General Public License (GPL) which is a contract agreement allowing end users to freely use the software under certain restrictions. These restrictions prevent unscrupulous companies from stealing open source code into their own propriety software and call it their own code. For more details on GPL see `http://www.fsf.org`.

"Free speech, not free beer" - Richard Stallman

The Linux kernel, the underlying program interfacing and running the computer hardware, was created by Linus Torvalds in 1991. He first wrote the kernel on the Intel 80386 microprocessor and distributed the code worldwide. He invited programmers from around the world to improve the code. Whenever new code is written, Torvalds and other programmers closely involved with the kernel decide whether to incorporate the new changes into the operating system.

Many programs incorporated into the operating system were already freely distributed on the Internet. The majority of these programs are the efforts of Richard Stallman, the well-known advocate of free software and the creator of GPL - a license model adopted by Linux. The purpose of GPL as created by Richard Stallman is to ensure free software stay open sourced and allow new programs to retain under one ownership, the public. Hence, "free speech, not free beer".

1.1 Commercializing Linux

Under GPL, Linux evolves rapidly as enthusiastic programmers from around the world contribute source code to be incorporated into the official release version controlled by Linus Torvald, Alan Cox, and other programmers intimate with the development of the Linux kernel.

According to International Data Corp, Linux grabbed a 17.2 percent share of server operating units shipped in 1998 - a startling 212.5 percent growth rate from 1997. The support announced by IBM, followed by other companies such as HP, Compaq and Dell, along with a growing number of Linux distributors such as Caldera, Red Hat, SuSE, and TurboLinux, helped bring Linux into the mainstream.

According to a Gartner Group/Datapro 1998 survey, customer satisfaction (in terms of interoperability, cost of ownership, ease of management, and performance) with the Linux operating system running at business sites was rated significantly higher than the Solaris, HP-UX, NetWare, Windows NT, and other UNIX platforms.

1.2 Best things on the Internet are free?

While most product quality and improvements are driven by economic factors such as price, competition, and more competition, open software such as Linux and the Internet seem to be the exceptions and instead are driven by free and open standards.

The majority of Internet Web sites today are powered by a free Web server software called Apache. According to an ongoing NetCraft study, more than half of Web servers run Apache, which runs on all major platforms, and the number keeps growing (see Figure 1). Major Internet companies such as Yahoo, HotWired, Internet Movie Database and Microsoft Network (MSN) deploy Apache for its stability, scalability and performance. In addition, according to SiteMetrics Corp., 35 percent of 53,265 U.S. companies with revenue over $10,000,000 use Apache, including the number 1 and 5 most-popular Web sites, Yahoo and Geocities respectively, which have over 100,000,000 hits per day each.

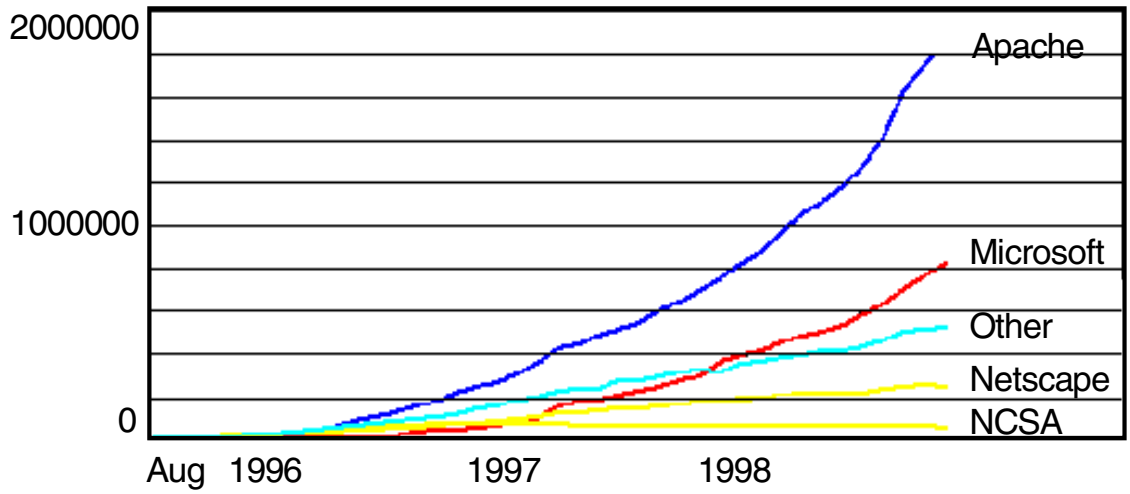


Figure 1. Apache has a large growing base; even more use Apache derivatives, such as IBM HTTP Server

1.3 Linux performance, stability, and security

Linux is a UNIX-like operating system. It offers stability and performance, and is a secure system. There is general consensus that Linux is one of the most secure operating systems in the market because of its "virus-proof" design. Linux is a UNIX clone that offers a multiuser environment. This environment prevents an individual user from running a virus-prone program such as a Trojan Horse program and infecting the entire system. In other words, a multiuser environment offers multiple levels of security protection.

A general misconception about the lack of security in open source software is that open source code invites crackers to penetrate the system more easily than closed proprietary code. Nothing can be further from the truth.

Commercial vendors prefer to sow fear, uncertainty, and doubt among its end users. They mislead end users to believe proprietary software is more secure because crackers will not know how to work around their software. But crackers, by definition, seek ways to circumvent loopholes and they constantly do so with or without you and the commercial vendors knowing. Security holes thrive only when they are hidden.

Open source software promotes security by allowing everyone to review the source code where potential security holes may exist. Because the code is open to everyone, security holes do not remain hidden for long. When a hole is exposed, a fix is available sooner than most commercial software fixes.

Open source also promotes security by giving system administrators the knowledge to take accountability for their own systems. Administrators need to be accountable for their systems because security is an on-going concern, not accomplished by simply applying service packs. Administrators can analyze their system’s security strengths and weaknesses by gaining more insights into the source code the system runs. The system is only as secure as the weakest link of a chain.

1.4 Ease of use

Linux has accelerated its development efforts in desktop GUI with the release of two major desktop environments: GNU Network Object Model Environment (GNOME) and K Desktop Environment (KDE). These desktop environments offer flexible ways end users can customize their desktop look and feel.

For instance, KDE setup wizard (see Figure 2) allows you to select any one of the four themes: MacOS, KDE Default, Windows, BeOS.

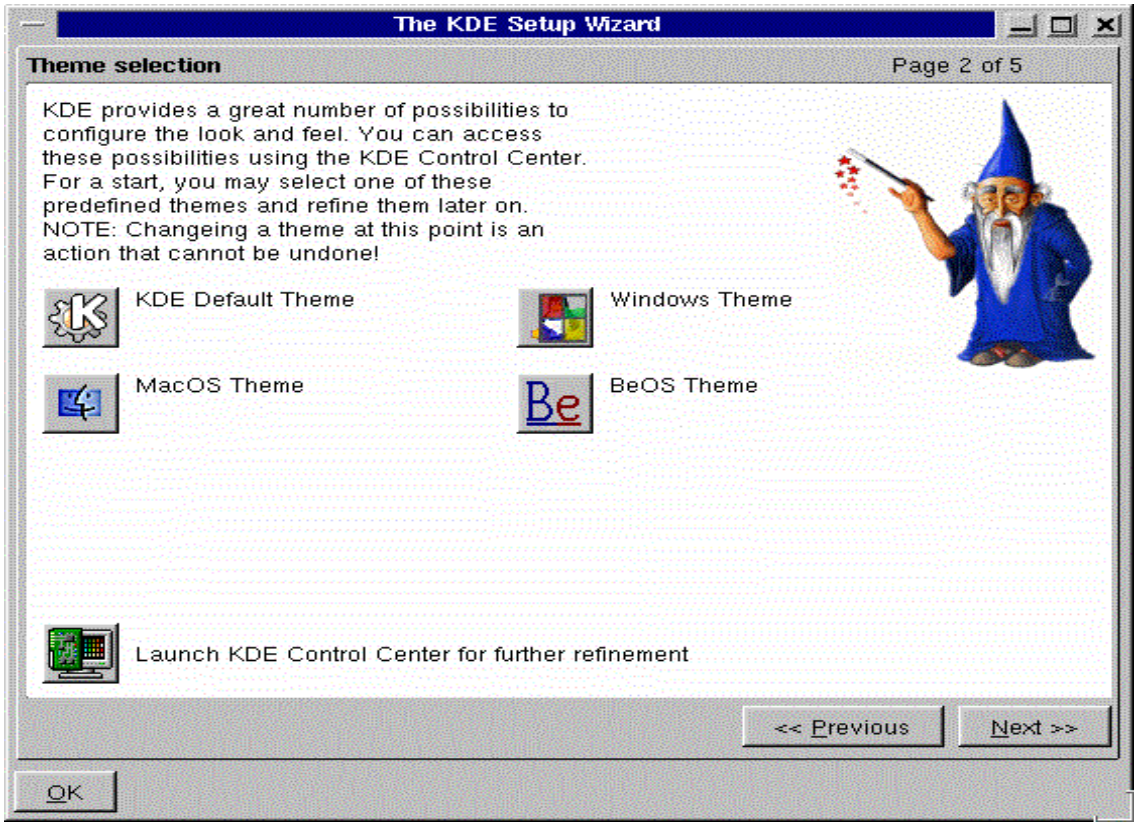


Figure 2. KDE setup wizard offers four themes for users to choose the look and feel they like

Note

Setup Wizard in Figure 2 is shown in the Windows theme, which provides the familiar look and feel of the Windows desktop.

In Figure 3, the MacOS theme lets a Linux user enjoy the popular look and feel of the Macintosh desktop.

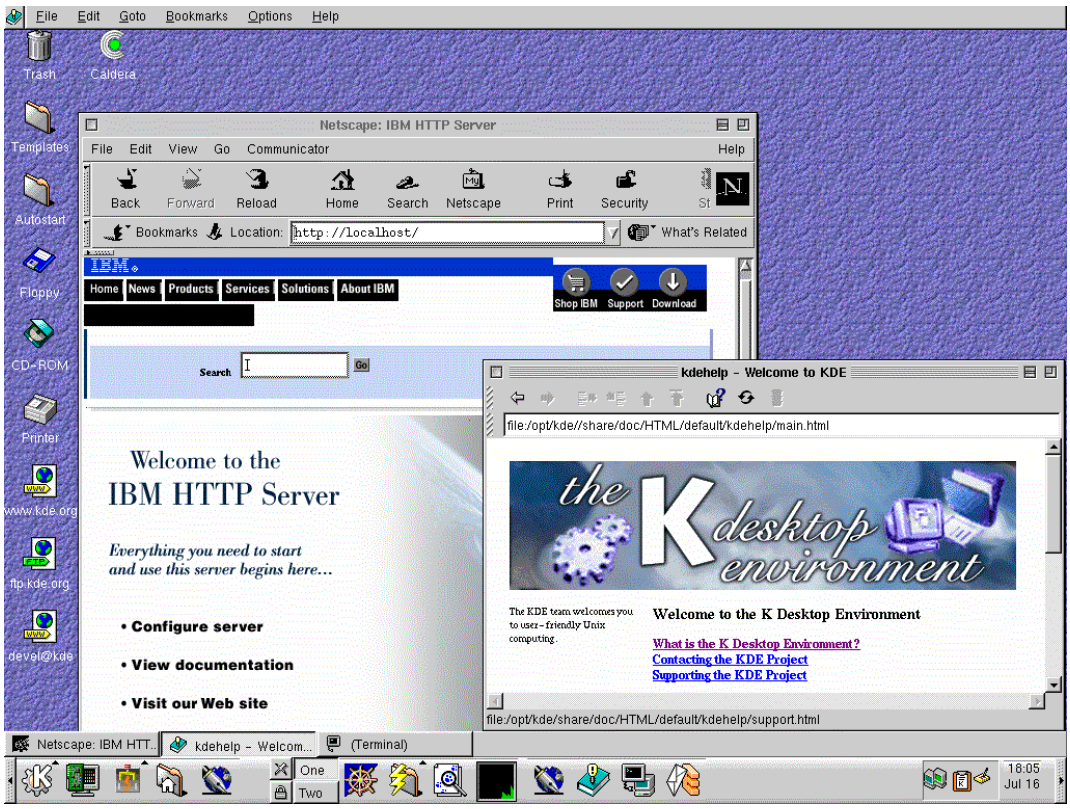


Figure 3. The MacOS theme offers the popular Macintosh desktop environment

1.4.1 Summary

Linux is an alternative OS for deploying corporate computers, both as a desktop and as a server.

The Linux community, people who develop and support Linux, has given the computing industry a viable choice in operating systems. It does not have to be an all-or-nothing proposition replacing your existing computing environment. Most enterprises already have a heterogeneous environment that Linux can play in.

More information can be found online at the following Web sites:

- IBM's introduction to Linux:

  `http://learn.ibm.be/linux/`

- The Cathedral and the Bazaar:

http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar.html

- General Public License:

  http://www.gnu.org/copyleft/gpl.html

- The oldest Linux resource on the Net:

  http://www.metalab.unc.edu/Linux/

- Computer Emergency Response Team, a central location for the latest security news and patches:

  http://www.cert.org

- A complete list of hardware ports:

  http://www.ctv.es/USERS/xose/linux/linux_ports.html

- NetCraft Survey of Apache Web Server Market Share:

  http://www.netcraft.com/survey

- SiteMetrics Corporation Survey of Internet Servers:

  http://www.sitemetrics.com/serversurvey/index.htm

- Survey: Linux Is No Fad:

  http://www.zdnet.com/sr/stories/news/0,4538,2309670,00.html

Chapter 2. IBM's commitment to Linux

IBM is fully committed to an open platform environment for running e-business applications in the most flexible ways. With the increased demand from IBM customers and business partners, IBM is offering full hardware and software support to the Linux operating system.

IBM is committed to supporting its customers' choice of platform and operating systems - a commitment IBM extends to the support of Linux, the open-source operating system.

To meet increased demand for this dynamic operating system, IBM is providing the most comprehensive hardware, software and support solutions based on products from its Personal Systems Group, Software Group, IBM Server Group, IBM Global Services and education services.

As part of IBM's continuing commitment to support the operating system of its customers' choice, IBM has teamed with leading commercial Linux distributors Caldera Systems, Red Hat, SuSE, and TurboLinux to port, test and certify the performance of IBM offerings running on various Linux distributions, enabling you to exploit the full potential of Linux.

IBM's support of Linux is a natural extension of the IBM Software Group cross-platform strategy of supporting customers' heterogeneous environments and supporting open standards. Look for key software applications, including DB2 Universal Database, WebSphere, Lotus Domino, MQSeries, Host On-Demand and Advanced File System, to integrate with your existing systems, enabling you to build, deploy and manage e-business Linux-based applications more rapidly and cost-effectively.

IBM Netfinity servers are earning a reputation as a reliable platform for Linux implementation, offering stability, performance, proven reliability, and low cost for total solutions. With support available worldwide, the combination of Netfinity servers and Linux applications offers you affordable, enterprise-class systems, backed by IBM's extensive experience, skills and resources to address your technical needs.

Please see `http://www.ibm.com/linux/` for further details.

Chapter 3. The IBM Application Framework for e-business

The IBM Application Framework for e-business uses open standards technology in the industry to support an open platform environment. This environment adopts the Java programming language to provide a write-once-and-run-anywhere methodology where developers can build on a new or existing computing framework.

3.1 e-business: merging the Internet and IBM technology

The IBM Application Framework for e-business is an information technology architecture defining a Web-centric software development platform based on Internet-standard technology. This open platform enables Web developers and system integrators to build new Internet solutions and to integrate multiplatform, multivendor systems. See Figure 4.

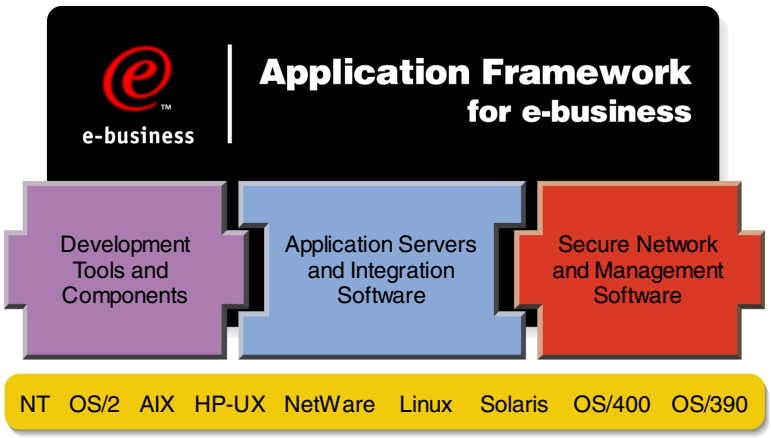


Figure 4. The IBM Application Framework for e-business provides an open platform for developers

Internet technology is the focal part of the IBM Application Framework for e-business. The information superhighway created a virtual world where everyone can connect from anywhere, at anytime, with any device. This became a reality when the computing industry adopted Internet-standard technology such as HTML, XML, and Java. The open standards Internet technology used by the Application Framework enables IT professionals to build innovative Web solutions independent of specific vendor hardware and software.

Web solutions reinforce the implementation of a network computing model where application users can run any hardware device to access dispersed information on the Web. These devices are called thin clients, which means they are mainly used for presenting information and have little or no application logic that processes the information. The processing of business application logic is run on the server machine connected to the thin client. This model provides universal access for the thin clients using various devices such as a Web browser, desktop computer, PDA, set-top box, mobile phone, information appliance, and other devices. See Figure 5.

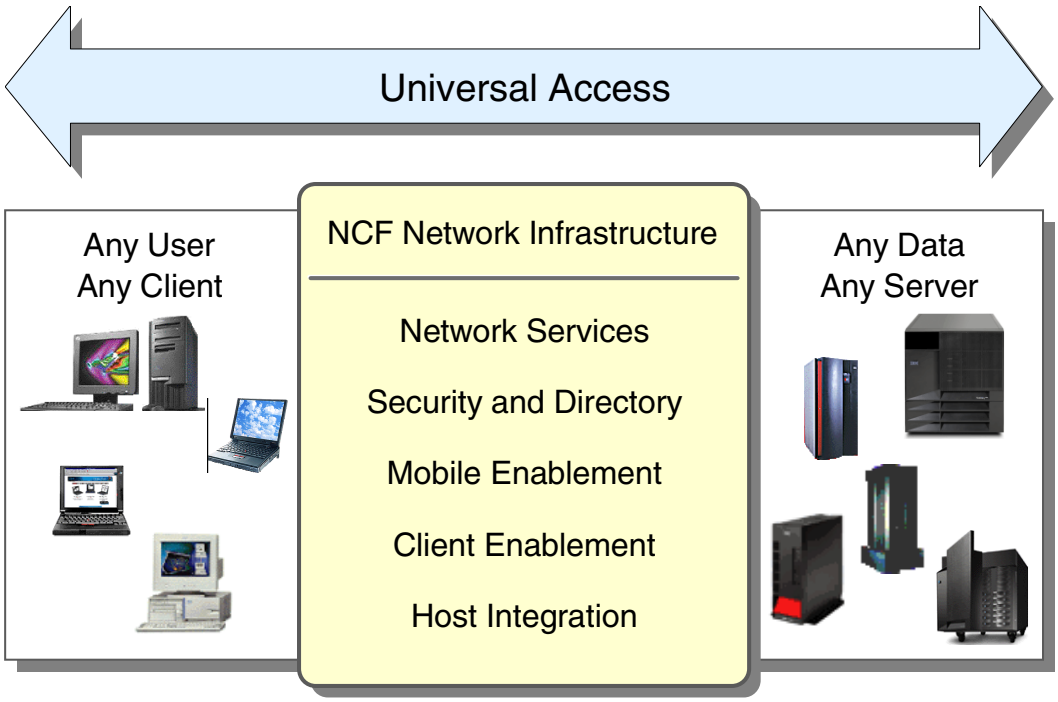


Figure 5. The Internet using open standards technology makes universal access a reality

3.2 Foundations of the IBM Application Framework for e-business

The Application Framework is built from modular layers of general services and tasks. These service layers span horizontally across your entire network, working in unity to provide an environment of high availability and scalability as shown in Figure 6:

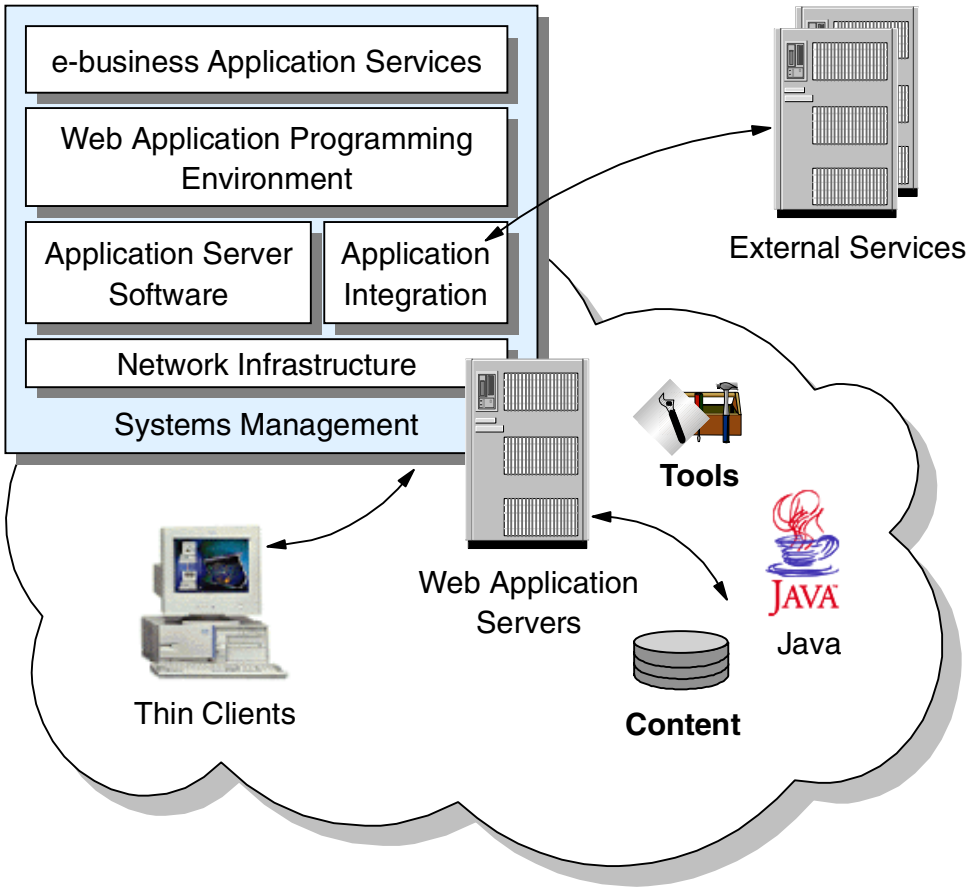


Figure 6. The open and modular architecture design of the IBM Application Framework for e-business provides a complete end-to-end open platform to develop e-business solutions

The building blocks of this illustration are two components that glue the Application Framework's software services and hardware infrastructure seamlessly. They are the Java programming language and the software connector components.

3.3 Java everywhere

Java was created by an IBM partner, Sun Microsystems Inc, as the unifying programming language for a platform-independent computing environment. It

is a fast and powerful object-oriented approach to run software on different computing platforms or networks.

3.4 Connectors are bridges between networks

Connectors link existing heterogeneous computers to communicate with each other. They are software written using industry open standards such as TCP/IP and IIOP protocols. One type of connector service is a gateway software that links an application Web server to a client computer communicating using HTTP protocol. An application server needs to be flexible; the IBM Application Framework provides a complete set of services to help businesses collaborate effectively, maintain and access information efficiently, and conduct transactions reliably in both intranet and extranet environments.

Connectors are Java libraries and supporting tools that support access from the Framework's programming environment to various external data and application servers. They shield the programmer from the usually proprietary network protocols needed to access these servers. IBM provides a comprehensive set of e-business connectors to a wide range of existing application and data servers, including CICS, Encina, IMS, Domino, MQ, DB2 and other relational databases. Connectors also links different computer networks, allowing them to interoperate on a common level. See Figure 7.

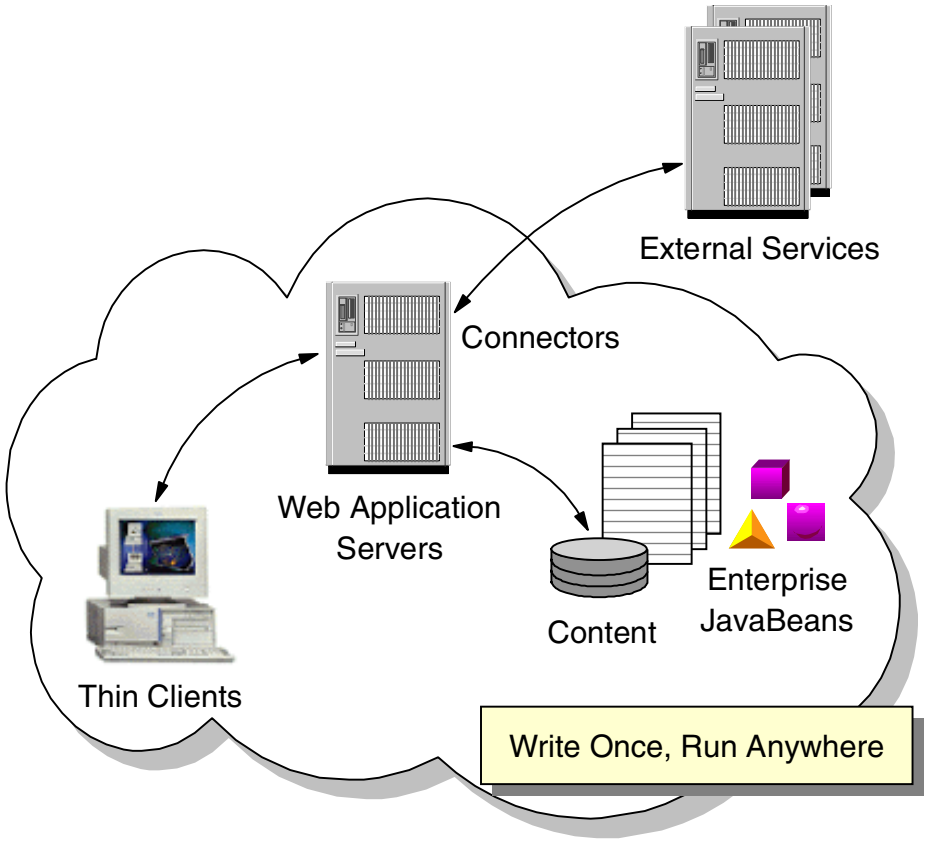


Figure 7. Java and connectors make a hetergeneous computing environment work seamlessly

3.4.1 Some key reasons for using connectors

Here is a list of key reasons for using connectors:

1. Software linking existing information systems to the Web
2. Helps leverage proven customer applications
   - Minimal change to back-end applications
   - Quicker time to market
   - Portability across multiple systems and servers
3. Leverages existing enterprise investments
4. Powerful Web integration for:
   - IBM and Lotus e-business application servers
   - Major application subsystems

5. Gateways for Go and others:
    - Net.Data (links relational databases using JDBC, SQL, ODBC)
    - CICS Internet Gateway (transaction-based servers and applications)
    - MQSeries Internet Gateway (guaranteed message delivery)
    - DCE Encina Lightweight Gateway (transaction-based DCE/Encina)
    - IMS Connectors (IMS systems, applications, hierarchical databases)
    - Host On-Demand (emulation access Java applet, 3270, 5250, VTxxx)
6. Gateways for Domino - the Domino.Connect family
    - NotesPump (relational databases)
    - MQSeries Enterprise Integrator
    - LSXs (MQSeries, SAP R/3, Baan, etc.)

3.5 Service modules in the Application Framework for e-business

The Framework consists of six independent service modules for building an e-business:

- e-business application services
- Web application programming environment
- Application server software
- Application integration
- Network infrastructure
- Systems management

These services support the deployment of business applications across intranets, extranets, and the Internet. They lay out a complete groundwork for which a solution provider creates e-business applications using secure, open, reusable Internet technology. A system integrator can add application server software, application integration, network infrastructure and system management services for various platforms including AIX/UX, Linux, SunOS, Windows NT to an existing computing environment. A Web developer can implement an e-commerce solution using the Framework's e-business application services and Web application programming environment. See Figure 8.

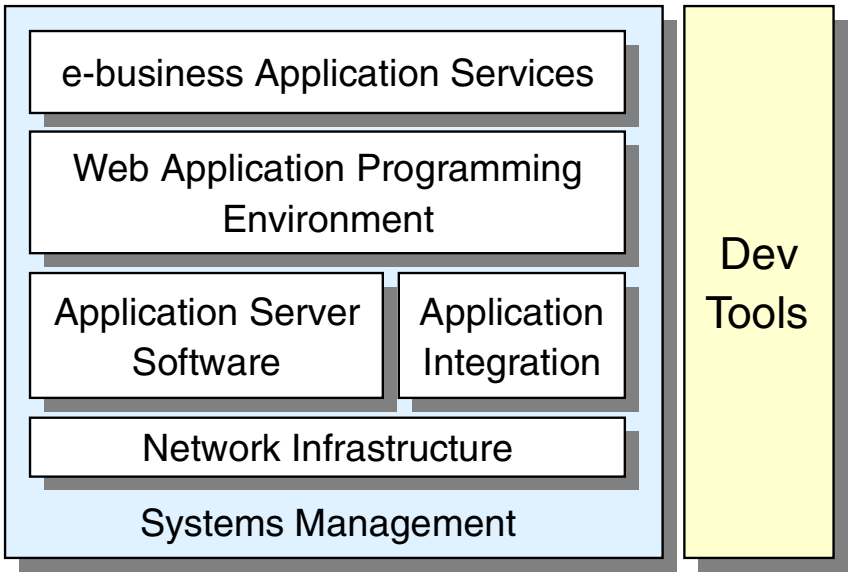


Figure 8. A complete set of services defines the e-business Web application server architecture

3.5.1 The e-business application services

These are Web applications built for any industry solutions. They generally contain high-level application logics and Graphics User Interface (GUI) presentations. Web-enabled payment, catalog, and order management services of an e-commerce site contain both the business application logic and the GUI presentation of the Web site design.

3.5.2 The Web application programming model using IBM software

The IBM WebSphere Application Server, which runs on an HTTP Web server such as IBM HTTP Server, is the central point for integration in the Web application programming environment. It is the first entry point to server functionality supporting HTTP requests from Web clients. It supports the mechanism for locating and invoking business logic by implementing servlets on the server, and it acts as connector gateways that link the Web to existing enterprise applications, data, and systems. It coordinates, collects, and assembles static and dynamic contents. These contents are provided by the Web application programming environment either through IBM software development products or any third-party development tools that comply with industry programming standards. For a complete list of IBM software products visit `http://www.software.ibm.com`.

These programming standards include HTML, XML, Java, and JavaBeans, for example Enterprise JavaBeans (EJB).

Enterprise JavaBeans provides dynamic binding of an application's business logic to the back-end data storage, such as DB2, and transaction service, such as CICS, Web clients, existing data and applications, the network infrastructure, and the server platform.

3.5.3 Summary

The IBM Application Framework for e-business allows developers to employ open standard technology contributed by different vendors to ensure the developers and IT solution providers that their solution will be easy to migrate and they will have independence from a single vendor, the freedom to choose, the flexibility to modify, etc. It also defines or demonstrates the communication gaps between different hardware and software systems. It also predicts or introduces the future of the network computing world in which IT changes are occurring today and tomorrow. The mobile, wireless, thin client computing model and the server-centric computing model are emphasized by the explosion of the Internet and the emergence of e-commerce applications.

Benefits of adopting the Application Framework include:

- Write once, run anywhere
- Lower development expense
- Quicker time to market
- Higher quality business applications

It incorporates an application model that dynamically adapts the application's business logic to a variety of clients, platforms, data storage, and transaction implementations. It provides a network infrastructure that enables the deployment of consistent, coherent, and scalable solutions - standards-based, scalable, server-centric technology to leverage an existing system.

The IBM Application Framework architecture helps you to become an e-business.

Chapter 4. WebSphere, VisualAge for Java and DB2

This section describes how to use IBM WebSphere, VisualAge for Java and DB2 in the IBM Application Framework for e-business. See Figure 9:

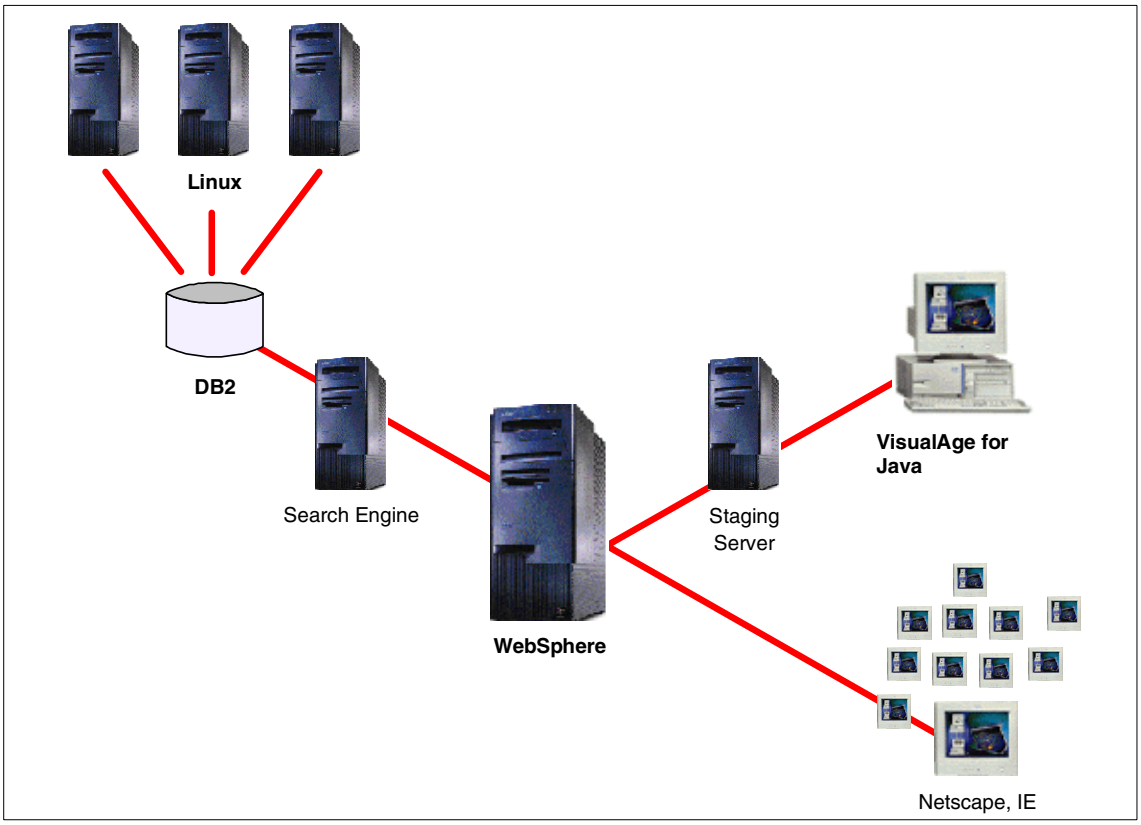


Figure 9. Web programming development using WebSphere, DB2, and VisualAge for Java

4.1 Foundation of the IBM Application Framework for e-business

The IBM Application Framework has a unifying programming model using the Java programming language. Java is an object-oriented programming language for building a Web-based application that can be written once and run on any operating system platform. In many cases, the Java program can be executed on another platform without recompiling. All IBM products in this Framework use the Java programming language to deliver complete server-side Web applications. In addition to WebSphere, VisualAge, and

DB2, IBM San Francisco and Lotus e-Suite facilitate the development of client-side Web applications.

4.2 VisualAge for Java

VisualAge for Java is an award-winning Java application development environment for building Java applications, applets, servlets, and JavaBean components. It offers exceptional developer productivity, ease-of-use and powerful features such as a built-in high-performance compiler, connections to more enterprise systems, and team programming support. Please see a detailed product description on the IBM Web site, `http://www.software.ibm.com`.

4.2.1 Methods of delivery

The IBM Application Framework for e-business contains all the necessary methods that form the Web application programming environment for writing server-side Web application. Using VisualAge for Java Integrated Development Environment (IDE) provides an easy approach to develop these methods. These methods use open standard technology including:

- Servlet for processing business logic on the Web application server.
- JavaServer Pages for separating application development from Web graphical design work.
- A flexible application development environment for adopting pluggable, reusable components such as applets and JavaBeans or selecting a third-party development tool.
- Connectors for bridging the applications with applications that access existing or external systems and with those applications that access data storage.
- Required Web network infrastructure and system management.

Together, these elements provide a secure and complete platform to deploy new application or to extend an existing application to the Web.

In Figure 10 on page 21, there are two interrelating models: client and Web application server models.

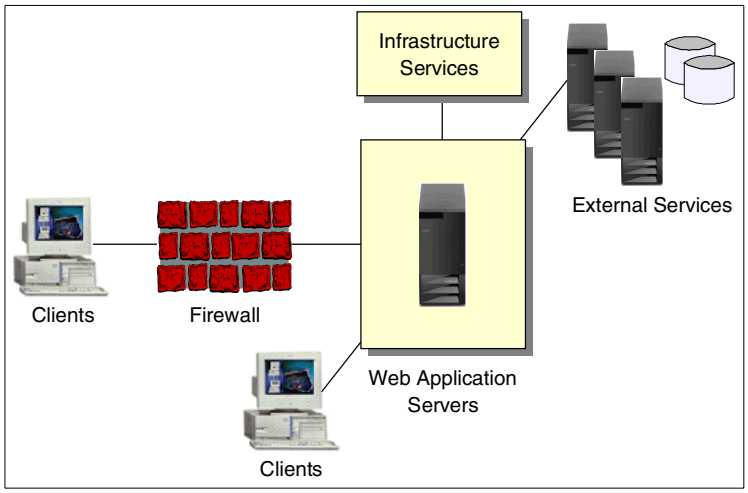


Figure 10. Web application topology consists of client and server models

The client model consists of GUI presentation modules allowing developers to define/refine presentation data created/processed by the Web server model. The Web application server model consists of server services that deliver data to the client side. These services are: Java services, database services, collaboration services, and application integration services. With VisualAge for Java, the developers can easily create the modules of the client model. These modules are typically applets, JavaBeans. In the enterprise edition, VisualAge for Java provides Enterprise JavaBean support for accessing data storage such as DB2.

With WebSphere, the developers can create server-side services using Servlets, Java Server Pages (JSPs), Enterprise JavaBeans (EJBs), and Extensible Markup Language (XML) technologies.

4.3 IBM WebSphere Application Server

IBM WebSphere Application Server is a Java-based Web application deployment environment for server-side applications and JavaBeans. It helps customers deploy and manage Web-based applications ranging from simple Web sites to powerful e-business solutions. The WebSphere Application Server helps solve real e-business problems by offering a portable Java-based execution and management environment for server-side critical business logic while enabling powerful Web transactions. The WebSphere Application Server components cane be part of a multitiered architecture or network topology with the WebSphere Application Server and associated

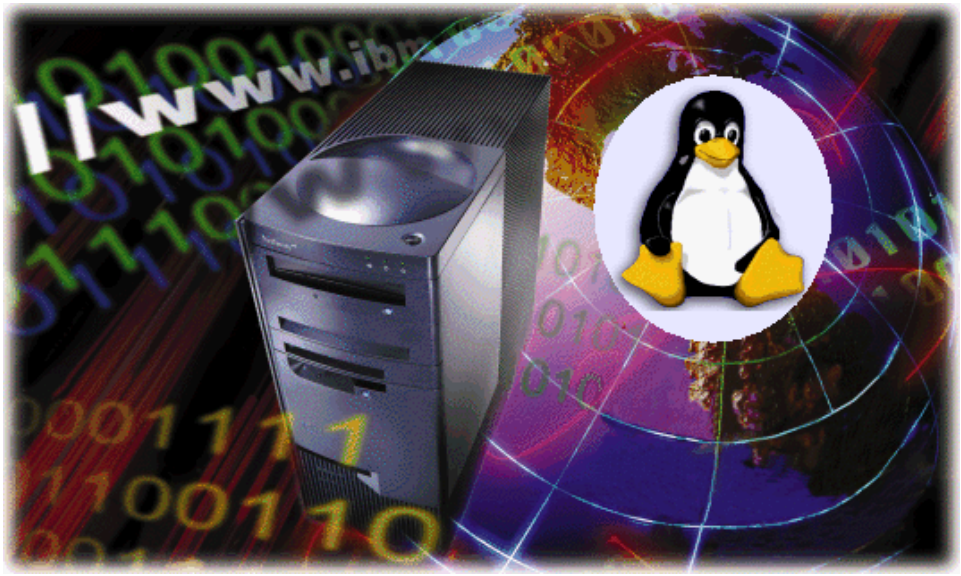
Web servers residing within the logical middle-tier along with a collection of middleware components and services. Please see a detailed product description at the IBM Web site: http://www.software.ibm.com.

Enterprise JavaBeans components provides dynamic binding of new business logic to the underlying DB2 data storage, as well as to clients, existing data and applications, the network infrastructure, and the server platform.

4.4 DB2 Universal Database- the foundation for e-business

DB2 Universal Database V6.1 is the industry's first multimedia, Web-ready relational database management system, strong enough to meet the demands of large corporations and flexible enough to serve medium-sized and small businesses. Please see a detailed product description at the IBM Web site: http://www.software.ibm.com.

Chapter 5. Hardware and software setup



This chapter explains the steps to install IBM's IBM HTTP Server, Apache, WebSphere Application Server, DB2 and VisualAge for Linux on various Linux distributions. Because each of these Linux distributions comes at different code levels, and with quite different levels of products and product mixes, success with one Linux install does not guarantee similar success with another Linux.

The suite of software installed during this project included the following:

- Red Hat 6.0, Caldera OpenLinux 2.2 + OpenLinux 2.3 beta, TurboLinux 3.6
- Apache 1.3.6 and the IBM HTTP Server (Apache 1.3.6 based)
- Java JDK (both IBM jdk116 and Blackdown jdk117_v3)
- IBM WebSphere Application Server 2.0.3
- IBM DB2 Universal Database 6.1
- VisualAge for Java for Linux Version 3.0 - Technical Evaluation

Note

Each Linux distribution is different. A procedure that works with one may well not work with another. Read distribution-related comments carefully.

A list of useful Linux and product Web sites:

Linux:	`www.linux.org`
Linux how-to's:	`www.linux.org/help/howto.html`
Red Hat:	`www.redhat.com`
Caldera:	`www.caldera.com`
TurboLinux:	`www.turbolinux.com`
SuSE Linux:	`www.suse.com`
IBM WebSphere:	`www.software.ibm.com/webservers/`
IBM DB2:	`www.software.ibm.com/data/db2/linux/`
IBM VA for Java:	`www.software.ibm.com/ad/vajava/`
IBM Alphaworks:	`www.alphaworks.ibm.com`

During this project we made use of the following news sites. They are publicly accessible on the IBM news server at `news.software.ibm.com`

- `ibm.software.websphere.application-server`
- `ibm.software.db2.udb.linux`

5.1 Initial setup guidelines

At the time of this project, some of the products were still available only in beta code. While each product ran as expected, we did experience install anomalies. While these will be resolved in future releases, we have mentioned them here.

Note

The steps provided below are a quick guide to the sections that follow. We advise you not to start your install until you have read through this chapter. Each of the following steps is explained in more detail in this chapter.

5.1.1 Recommended install sequence checklist

1. Prepare your hardware - ensure the minimum configurations are available.
2. Install your chosen Linux distribution. Add your name as a normal user.
3. Familiarize yourself with the preferred X-Window environment (KDE or GNOME).
4. To install Java, WebSphere, or DB2, you need to log in as the root user.
5. **Caldera**: Remove the old Java versions (kaffe and jdk rpms).
6. **Caldera**: Remove Apache 1.3.4 and install IBM HTTP Server 1.3.6.
7. Install Java (we used both IBM jdk116 and Blackdown jdk117_v3).
8. Install WebSphere 2.0.3 using an X-Terminal and the `rpm -i` command.
9. Restart your computer. The Apache HTTPD daemon should start normally.
10. Use Netscape to test Apache and WebSphere.
11. Use Netscape to test WebSphere admin http://yourhost:9527.
12. Install DB2 from a Linux login console (rather than an X-Window).
13. Log in to DB2 (db2inst1) and run `db2start` and `db2jstrt 8083` commands.
14. In DB2, add Java environment variables for jdk117_v3 into db2profile script.
15. Log in to DB2 from an X-Term and run the command 'db2cc 8083'.
16. To allow Apache / WebSphere to see DB2, patch the startup script.
17. Log in as yourself and Install VisualAge for Java in your home directory.
18. From your X-Window run VisualAge for Java per the install notes.
19. Run the sample programs provided in this book.

5.2 Hardware setup

In this project we used four IBM Netfinity 3000 computers as workstations and one IBM Netfinity 5000 as a LAN server running a shared DB2. We installed the full suite of software on each machine. Later we varied the configurations by moving functions between machines (See Figure 11 on page 27).

5.2.1 Netfinity 3000

Model:	8476-21U
Processor:	Pentium II 400/100 Mhz
Memory:	128 MB
SCSI:	Adaptec AHA 7895 dual channel ultra wide
Disk:	4 GB
Video Adapter:	Matrox AGP G200 with 8 MB RAM
Network:	Intel EtherExpress PRO100 on planar

5.2.2 Netfinity 5000

Model:	8659-2SY
Processor:	Pentium II 400/100 Mhz
Memory:	128 MB
SCSI:	Adaptec AHA 7895 dual channel ultra wide
Disk:	4 GB
Video Adapter:	Matrox AGP G200 with 8 MB RAM. We also used:
Video Adapter:	S3 Virge with 4 MB RAM
Network:	AMD Am79C972 on planar

The actual minimum disk space needed for installing a full configuration of the software was approximately 2.6 GB. The minimum processor speed we recommend is a Pentium II 200 Mhz. The minimum monitor screen we recommend is a 17inch monitor with a video adapter that can support 16-bit color at 1024 x 768 pixels. The minimum LAN speed: 10 Mbps.

At times we experienced some challenges with the different Linux distributions and the video adapters. We often had to do a custom install and experiment.

5.3 Lab LAN setups

Figure 11 and Figure 12 on page 27 show the LAN setup in the ITSO lab.

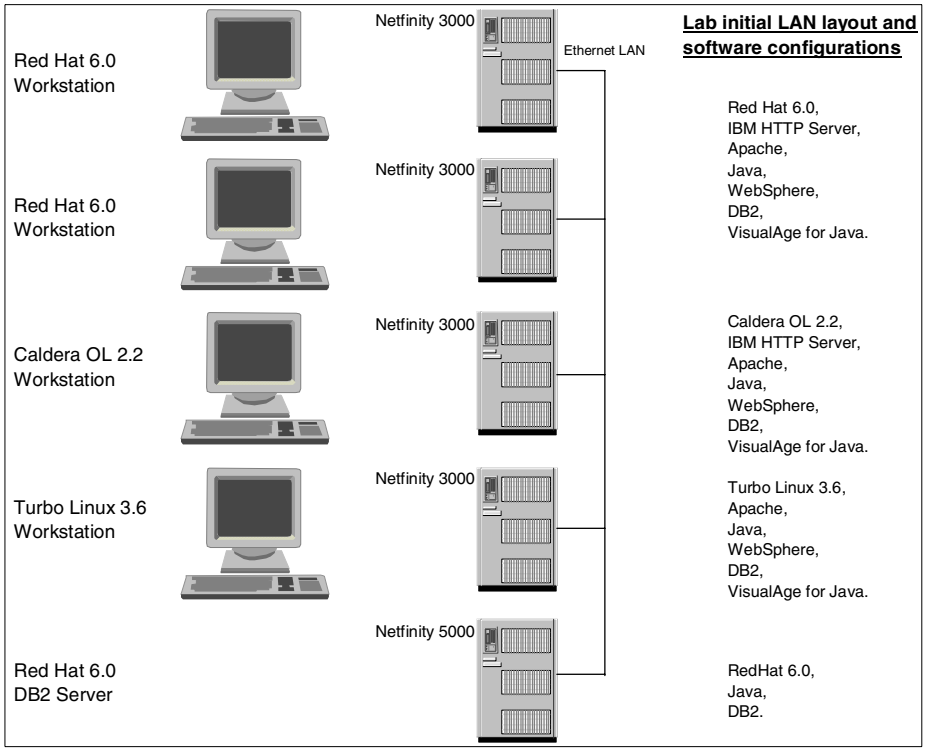


Figure 11. Initial LAN setup in lab

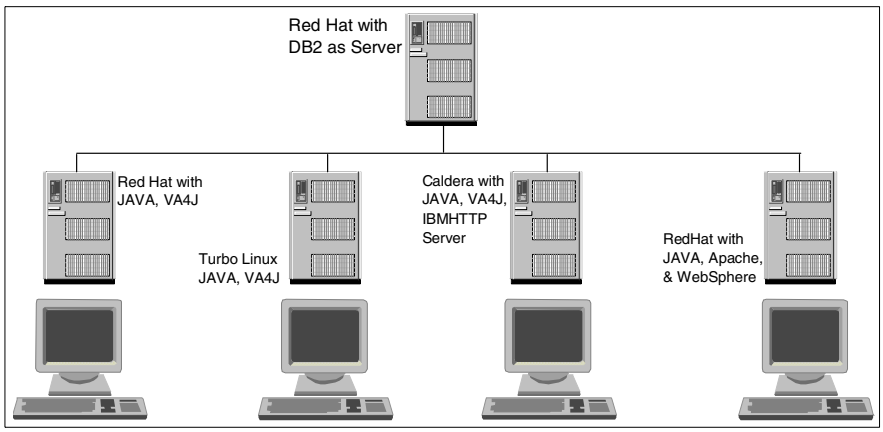


Figure 12. Later lab LAN setup

5.4 Linux - installation and configuration

The following sections cover the basic steps needed to install Linux from:

1. Red Hat Linux 6.0
2. Caldera Open Linux 2.2, Open Linux 2.3 (Beta)
3. Turbo Linux 3.6

(SuSE Linux 6.2 was not available at the time of this project.)

5.4.1 Installing Linux

The first task is to partition your computer's disk. Some Linux systems will do this for you while others require you to allocate file systems during the install.

Configuring a Linux disk has to do with the number of disks installed, your prior experience, individual preference, and the available install options. You should refer to your Linux install book for its advice and comments. You can also try the disk partitioning we used in the labs. See Table 1.

Red Hat. You will be required to partition the disk as part of the Red Hat install. You will be asked which partitioning tool you want to use (Fdisk or DiskDruid). We recommend DiskDruid, which is easy to learn and very forgiving.

Caldera. Offers to format the entire disk for you but will also allow you to choose your own settings. If you let Caldera auto-allocate the partitions, it will only allocate three partitions: swap, home (/home), and root (/).

Turbo Linux. As with Red Hat, you will be required to partition the disk. The tools supplied with TurboLinux are not quite as user friendly as Red Hat's DiskDruid, nor as forgiving of mistakes. An error in TurboLinux can mean having to start an install all over again from the beginning. Be careful.

Table 1. Linux partition sizes used in the lab on a 4 GB hard disk

Name	Mount Point	Size in MB	Type	Comment
home	/home	1600	Linux	User's home directories
root	/	2000	Linux	Linux directories and products
swap		128	swap	Must be as big as installed RAM
var	/var	250	Linux	Logging area

Originally we allocated /, /usr, /usr/doc, /tmp, /var, and /home but because of the way many products install, we ended up only using these partitions.

This layout is a good starting point even if you later choose to split the disk into more partitions. An advantage in keeping root (/) as a separate filesystem is to prevent the other subdirectories for users (/home), logging (/var) and temporary data (/tmp) from filling up the root filesystem or accidentally corrupting root files and locking up or damaging Linux.

Be aware that if you do a lot of work logged in as the user root (also called superuser) and this activity is performed while logged in via the X-Window system, there is an added risk of filling up the Linux root filesystem without realizing you have done so. The root user's home directory is /root, which sits in the / filesystem and not the /home filesystem where users normally go. The GNOME GUI warns you of the danger of logging into X as root.

If you have two disks you could place either / or /home or /usr on disk 2.

5.4.2 Graphics adapter setup

All Linux systems will try to configure your graphics adapter so they can automatically start the X-Window software on bootup. In the past Linux would boot up to a text console window and you had to manually start the X-Window system. The Linux products we installed all give you the option of having X-Window as the default post-boot login window. While in X-Window you can switch to any of the six or more virtual text consoles by using the Ctrl+Alt+F1 (F1 to F6) keys. On Caldera you can also access additional virtual consoles by pressing F8 to F12. These virtual consoles are used to log startup messages. These can prove helpful if you are having any kind of problems with software not starting properly. For example, pressing Ctrl+Alt+F12 brought us to a screen with an error message advising us that WebSphere was incompatible with Apache 1.3.4. during our Caldera WebSphere install.

In most cases you should be able to get your graphics adapter working okay. As a rule of thumb, on Linux, X-Windows works best at 1024 x 768 or better resolution. A resolution of 800 x 600 is workable, but you may find some windows go off the screen. At 640 x 480, X-Windows is close to useless.

A good brand of adapter with as little as 2 MB RAM can be used to support Linux X-Windows but the best results are obtained with 4MB or more of video RAM. Your card should offer 1024 x 768 or better resolution with 16-bit color. If you are forced to run the card as low as 256 colors, you may lose a lot of impact.

Notebook Install

As a side project, this team also installed Red Hat 6.0 and the products on an IBM ThinkPad 760 that had 2.8GB disk and 96Mb RAM and the panel display set to 1024 x 768. The software ran well on this notebook. We also did a dual Windows/Linux install on an IBM Thinkpad 600 Notebook with 6.4 GB and only a 1024 x 768 panel display. This ran very well.

You can expect some challenges with setting up and running a PCMCIA network card but we succeeded on both machines. Essential reading is at `http://hyper.stanford.edu/~dhinds/pcmcia/ftp/doc/PCMCIA-HOWTO.html` That document explains how to set up the following two files that are crucial for PCMCIA to work: `/etc/pcmcia/network.opts` and `/etc/sysconfig/pcmcia`.

5.5 Product installation preparation

Follow these instructions before installation:

Caldera 2.2 and 2.3 only

Before you can install Java or Apache or IBM's HTTP Server you must remove the preinstalled Java and Apache products that are at the wrong level to work with WebSphere. Remove them in this sequence:

- rpm -e apache
- rpm -e apache-docs
- rpm -e kaffe
- rpm -e jdk-shared (may only be in Caldera 2.3)
- rpm -e jdk-static
- rpm -e jdk
- rm -rf /etc/httpd (removes any residual configuration files)

5.6 Java - Installation and configuration

During this project the Java versions that were current for Linux included jdk116 from IBM (`http://www.ibm.com/alphaworks`) and jdk117_v3 from Blackdown (`http://www.blackdown.org/`). We installed both IBM's jdk116 and

Blackdown jdk117_v3 (plus the jdk117_v3 native threads pack). In time these versions will change. jdk116 (with native threads) is required for WebSphere.

To install IBM's jdk116 (which comes as part of WebSphere), wait until you are ready to install WebSphere and follow the directions there (see 5.12, "WebSphere Application Server - installation and configuration" on page 51). Also, it is quite okay to set up more than one Java on your computer. We set up both jdk116 and jdk117_v3 on every install. We did this because of problems we experienced where WebSphere needed jdk116 and the DB2 Control Center needed jdk117.

Blackdown jdk117_v3 is in two files. These are jdk_1.1.7-v3-glibc-x86.tar plus jdk_1.1.7-v3-glibc-x86-native_tar.gz (which is an add-on pack). The Blackdown jdk archive files should be copied to directory /opt then un-zipped and un-tarred extracting themselves into the directory (named jdk117_v3).

The following steps are the recommended way to install jdk117_v3 on your computer. The installing directory /opt was chosen because that is where WebSphere automatically installs jdk116. To be consistent we also placed jdk117_v3 in the /opt directory.

5.6.1 Java install steps

1. Copy the two downloaded jdk117 archive files to /opt.
2. If either file has the .gz suffix, then unzip it with the `gunzip` command.
3. Next run `tar -xvf` against each file. This un-tars them into jdk117_v3.
4. Create a file in your home directory called java117 and add lines:
   - `export JAVA_HOME=/opt/jdk117_v3`
   - `export PATH=$PATH:$JAVA_HOME/bin`
   - `export THREADS_FLAG=green`
5. Create a file in your home directory called java116 and add lines:
   - `export JAVA_HOME=/opt/ibm-jdk-l116`
   - `export PATH=$PATH:$JAVA_HOME/bin`
   - `export THREADS_FLAG=native`

5.6.2 Java testing and setup

In your home directory run commands:

```
source java117
java -version
```

The above source command takes the java117 you created and uses it to set your working environment variables to the correct values needed for Java 117. To test that Java can be found and that it runs properly run the `java -version` command. You can further check the settings of the environment variables with the following `set` command:

```
set | more
```

You can switch between the different Java versions as needed, by just running the `source` command against the particular Java you want to activate. Again, we needed these two different Java versions in order to get all the products to work correctly. This may be fixed by the time you read this book. If so, you can just use the one recommended Java version.

We strongly recommend that you do not put your Java environment variables in the system profile files (/etc/profile for Red Hat, TurboLinux and SuSE - /etc/config.d/shells/profile for Caldera), as some of the Java environment variables (such as THREADS_FLAG) can override the settings needed by running programs such as DB2 and WebSphere, even though products such as WebSphere also keep their own paths and settings in their own profile files (see part of WebSphere's bootstrap.properties file in Figure 24 on page 56).

5.7 Apache - installation and configuration

This is the easiest part of the setup for Red Hat and TurboLinux (and SuSE) as these distributions come with Apache 1.3.6 already installed and running.

Caldera Apache Reinstall: You should have already deleted the Apache 1.3.4 version that is pre-installed with Caldera 2.2 and 2.3 beta (this was covered in 5.5, "Product installation preparation" on page 30). Your first decision at this point is to decide which replacement Apache to install: IBM HTTP Server 1.3.6 or the standard Apache 1.3.6 distribution. If you choose the standard Apache distribution, you may well have to install it as source and compile it to get a proper working product. We have tested the IBM HTTP Server and it installs easily and works well with WebSphere 2.0.3.

Apache and IBM HTTP Server: Because we had troubles trying to get some standard Apache 1.3.6 setups working with WebSphere 2.0.3.,we removed Apache 1.3.6 and installed IBM HTTP Server. Our most successful testing was using IBM HTTP Server.

Visual RPM installer

Some Linux distributions allow clicking on an `*.rpm` module to install. If you prefer the visual RPM install technique you can use it with safety on both the stdlibc++ rpm module and then the IBM HTTP Server RPM module. However, it is important to remember later when in the WebSphere install chapter that WebSphere install may fail if using a visual installer, since WebSphere may need to issue messages to you that visual installers miss.

5.7.1 IBM HTTP Server install steps

1. If on **Caldera 2.2 or 2.3,** you need to download the prerequisite software module called libstdc++-compat-2.8.0-1.i386.rpm. To do this, go to `ftp://ftp.calderasystems.com/pub/openlinux/2.3/contrib/RPMS/`. The module can be applied to both 2.2 and 2.3 versions of Caldera. If you have the Caldera OpenLinux 2.3 install CD, you can obtain the module directly from the CD in the col/contrib/RPMS directory. Note the DB2 install also requires the libstdc++ module.
2. Next install the stdlibc++ and IBM HTTP Server modules using either the visual install method or manually using the following X-term commands:
   1. Create `/home/ihs_install` and put the IBM Server RPM in it.
   2. `rpm -i libstdc++-compat-2.8.0-1.i386.rpm` (**Caldera only**)
   3. `rpm -i IBM_HTTP_Server-1.3.6-1.i386.rpm`

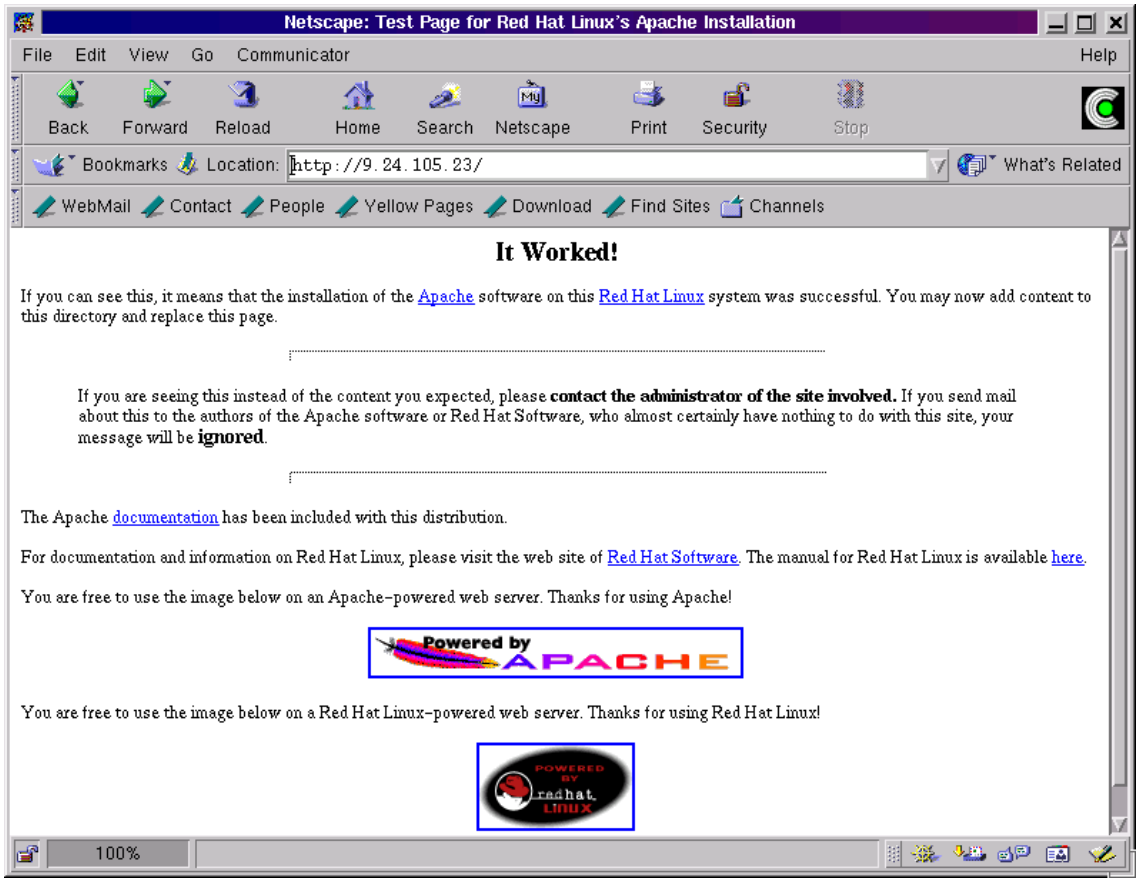


Figure 13. Generic Apache Web page

5.7.2 Testing Apache

Apache: Figure 13 shows the Web page you should see if your Apache is running normally (`http://localhost/`). Try this as soon as your Apache or IBM HTTP Server install is complete and you have Apache up and running.

IBM HTTP Server: Figure 14 on page 35 shows the IBM HTTP Server Web page working.

Remember that the only real difference between IBM HTTP Server 1.3.6 and standard Apache 1.3.6 is that IBM has added a lot of ease-of-use features. Also the IBM HTTP Server seems to integrate better with the WebSphere Application Server.

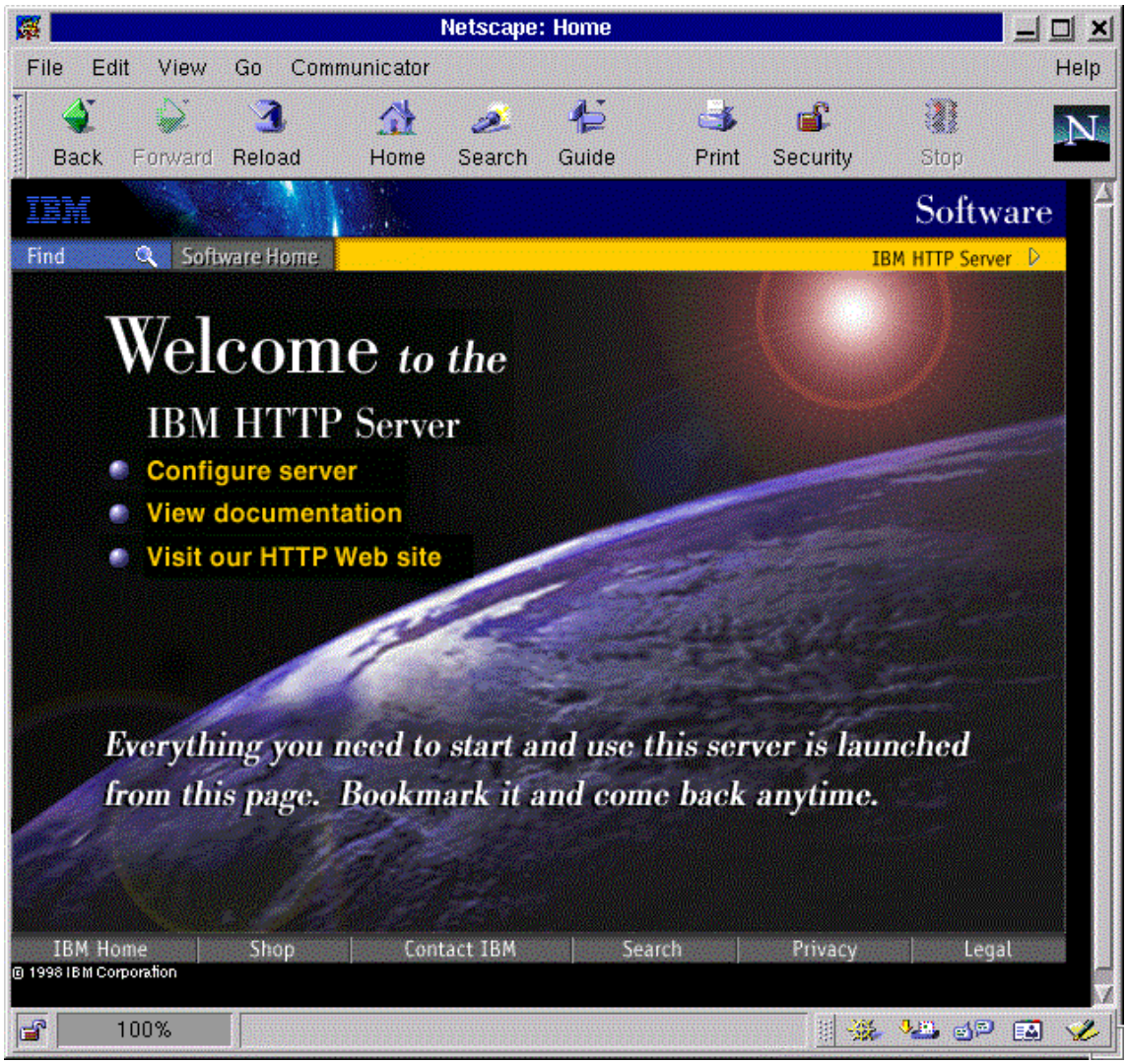


Figure 14. IBM HTTP Server

5.7.3 Setting up IBM HTTP Server startup script

When standard Apache is installed it sets up a startup script called httpd in /etc/rc.d/init.d and adds this script to various Linux runlevels. Usually you will see httpd as a start in runlevels 3, 4 and 5 and as a stop in runlevel 2. The runlevels represents different phases of a Linux bootup and run state. A normal Linux after bootup goes to runlevel 5. As a further example, running the `halt` command Invokes a runlevel of 0. See Figure 16 on page 38.

The /etc/rc.d/init.d directory is where all runlevel scripts are placed and is used to start Linux and Product services during bootup, as Linux goes through its various startup runlevels. It is beyond the scope of this book to explain how and why runlevels function the way they do, other than to say it was derived from UNIX System V and is an integral part of Linux. So if you want IBM HTTP Server to start automatically at boot time, then you will need to copy the IBM HTTP Server startup script (apachectl) to /etc/rc.d/init.d and go through the process of setting up the required runlevels following the administration instructions for your Linux distribution and those below.

5.7.4 Getting IBM HTTP Server (apachectl) to start at boot time

5.7.4.1 Non-Caldera distributions

The following steps should work for all distributions **except Caldera**, which unfortunately does not contain any visual runlevel editor.

1. Copy file /opt/IBMHTTPServer/bin/apachectl to /etc/rc.d/init.d
2. Invoke the runlevel editor for your Linux system. This can be achieved by opening an Xterm and running either tksysv or ksysv if available.
3. Add the script apachectl to start in runlevels 2 3 and 4 using priority 85.
4. Add the script apachectl to stop in runlevel 2 using priority 15.

Test it by restarting your Linux and seeing that the httpd daemon starts.

5.7.4.2 Caldera (but works on all other) distributions

1. Copy file /opt/IBMHTTPServer/bin/apachectl to /etc/rc.d/init.d
2. Add the following links:
   - ln -s /etc/rc.d/init.d/apachectl /etc/rc.d/rc0.d/K15apachectl
   - ln -s /etc/rc.d/init.d/apachectl /etc/rc.d/rc1.d/K15apachectl
   - ln -s /etc/rc.d/init.d/apachectl /etc/rc.d/rc2.d/K15apachectl
   - ln -s /etc/rc.d/init.d/apachectl /etc/rc.d/rc3.d/S85apachectl
   - ln -s /etc/rc.d/init.d/apachectl /etc/rc.d/rc4.d/S85apachectl
   - ln -s /etc/rc.d/init.d/apachectl /etc/rc.d/rc5.d/S85apachectl

The above links create an entry for apachectl at every runlevel. At levels 0, 1 and 2, apachectl is (K)illed. The 15 sets the priority or sequence in which it is killed. In runlevels 3, 4 and 5, apachectl is (s)tarted and given priority or sequence 85, which ensures it starts after other more important tasks (such as the network). Using 15 for kill and 85 for start are good choices, but other numbers could have been chosen providing you know what you are doing.

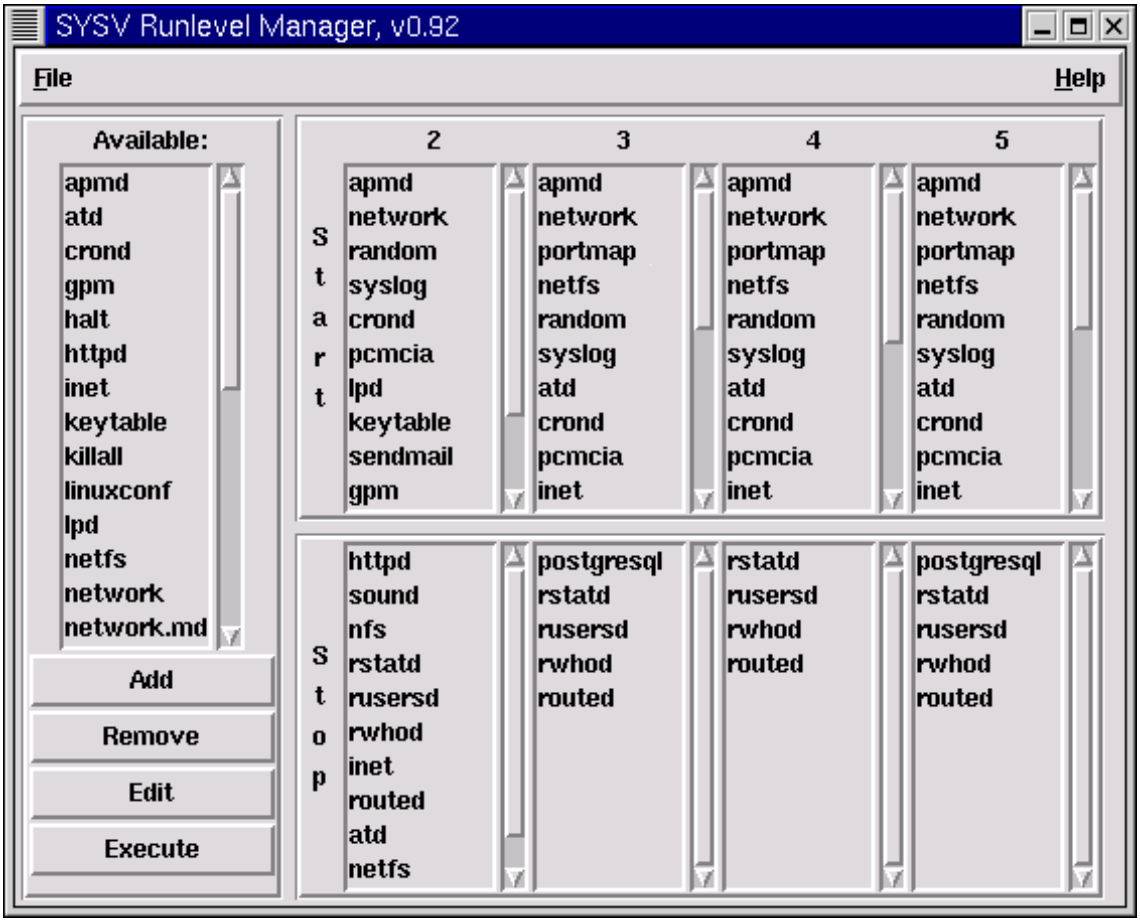


Figure 15. Standard SysV runlevel editor tool from Red Hat 6.0

The details in Figure 16 on page 38 may not be easily readable, but the image was included to show you another version of a typical runlevel editor. These vary from one Linux distribution to another and from one X desktop to another. The one that seems common to all Linux distributions is the one shown in Figure 15. If you can locate that one on your Linux system, then use it. A sure way to prove you have it is to invoke it in an Xterm window with command `tksysv`. The tool shown in Figure 16 can be invoked with the Xterm command `ksysv`. However, these applications are normally invoked from the menu bar of your desktop environment, if you can manage to identify them. This takes a lot of familiarity, and the actual menu to use varies from Linux distribution to distribution and desktop to desktop. Running the Xterm commands mentioned makes it much easier.

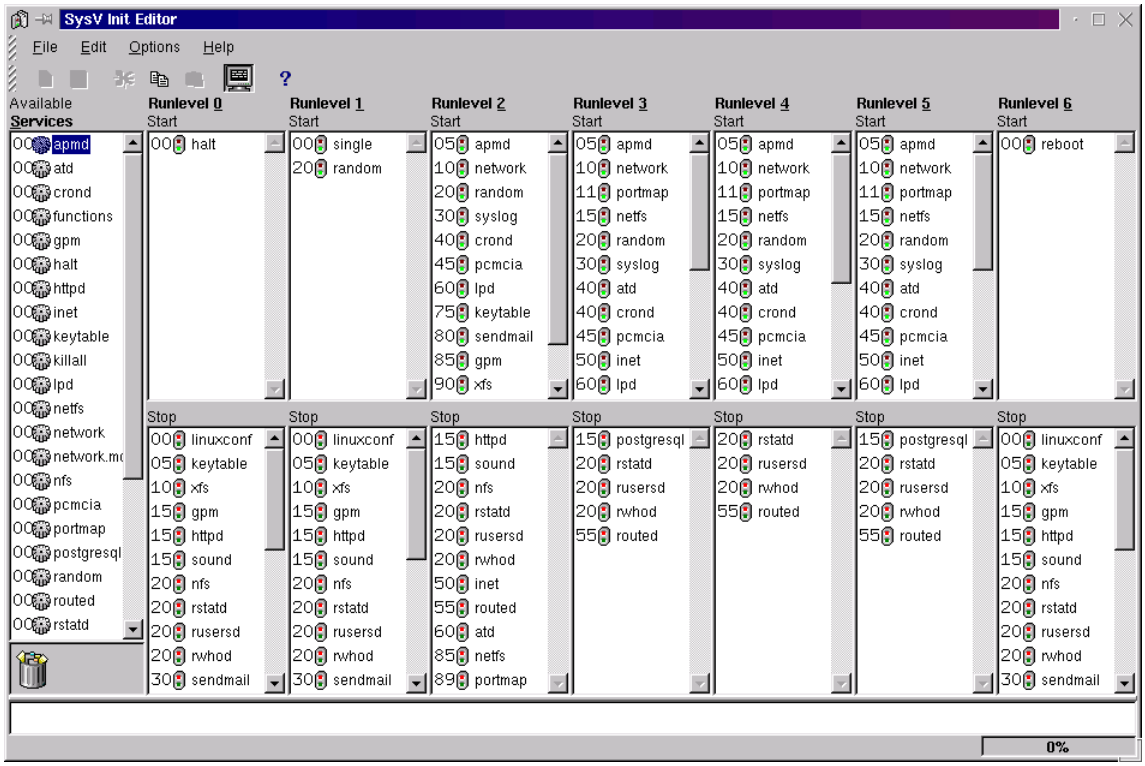


Figure 16. Another SysV runlevel editor used to set runlevels for startup scripts and tasks

5.7.5 Detecting Apache problems

Apache and Red Hat: Any problems with Apache will show up during the boot in that the HTTPD task will not start. It will sit at this process for several minutes before giving up and carrying on with the bootup. Once up, any attempt to access `http:/localhost/` from an Internet browser will fail.

Apache and TurboLinux: Same as for Red Hat.

Apache and Caldera: If Apache fails it will not be obvious during bootup as it is with Red Hat. The boot will seem normal but Apache will not be running when you try to access `http://localhost/`. Check the Caldera bootup messages by switching to virtual console F12 (Ctrl+Alt+F12). Also look at virtual consoles. Because each Linux distribution (such as Red Hat, Caldera, TurboLinux or SuSE) can and may put the Apache configuration files in different places, it is important to refer to the documentation that came with your particular Linux system or Apache product release.

5.8 Installing and configuring DB2 Universal Database

The following describes how to install DB2 Universal Database Version 6.1 on a Linux-based workstation that is running one of the following distributions:

- Caldera OpenLinux Version 2.2 or Version 2.3
- Red Hat Version 5.2 or Version 6.0
- TurboLinux Version 3.6
- SuSE Version 6.1

The instructions in this section assume that you install and configure your DB2 product using all the defaults provided by the DB2 Installer program. The DB2 Installer program is a menu-based installation program that assists you in accomplishing difficult installation and configuration tasks. The DB2 Installer program will automatically create an instance, create the Administration Server, and configure them for communications, allowing you to get DB2 up and running in very little time.

This also assumes that you have met all of the hardware and software requirements outlined in 5.4, “Linux - installation and configuration” on page 28.

We recommend, for the purposes of this book, that you do not maintain previous users or copies of DB2 on your workstation. If you need to remove DB2, see 5.11, “Deinstalling DB2 Universal Database” on page 48. If you maintain a user that is one of the default user accounts created by a default DB2 installation, the DB2 Installer program may not be able to successfully create the default instance or the Administration Server.

5.9 Before you begin

Before you can install DB2 on a Linux-based workstation, ensure that your distribution of Linux meets the following requirements:

1. Linux kernel 2.0.35 or higher
2. RPM (Red Hat package manager)
3. pdksh package (public domain Korn shell)
4. glibc Version 2.0.7 or higher
5. libstdc++ Version 2.8.0

Red Hat package manager

Despite its name, the Red Hat package manager (RPM) is a tool used on all distributions. This can cause some confusion. Each Linux distribution described in this chapter comes with the required RPM package installed as part of the operating system.

Since the whole Linux phenomenon is relatively new and based on an open and ever-improving code base (without a clear set of standards), each Linux distribution differs from one vendor to another. Consequently, the distribution you install may not be DB2-ready out of the box and may not come with some of the packages that DB2 requires to run.

glibc and shlibs

Some distribution even have their own naming conventions. For example, SuSE Version 6.1 refers to the required glibc package as shlibs.

This section will tell you the out-of-the-box preparation work that you need to do with your particular Linux distribution to enable it for DB2 Version 6.1. Once your Linux workstation is enabled for DB2, this section will take you through a basic installation and configuration where you can install DB2, create a sample database, and access data from this database using SQL.

As the Linux operating system matures, and relationships strengthen between IBM and the distribution vendors, you can expect to see many of these problems disappear. Until then, save yourself a lot of time by carefully reading this chapter. You will be on your way to using DB2 in no time at all.

The following sections describe the steps that you must take to prepare your Linux workstation for a DB2 Installation. Each distribution-specific section will note the Linux installation type that was used. The installation type that you installed may be different. If a package was included in the installation type used in this chapter, but does not exist on your installation (because you selected a different installation type), you can install the required package from the distribution CD-ROM.

5.9.1 Caldera OpenLinux Version 2.2 or Version 2.3

Installing DB2 on the Caldera Open Linux (or simply Caldera) Version 2.2 All Recommended Packages distribution was somewhat challenging. Most of the

problems that you will encounter have been fixed for the Caldera Version 2.3 release. (We used the Standard Installation type for Caldera Version 2.3.)

The pdksh package, required for the DB2 Installer program to run, is missing from Caldera Version 2.2 and is supplied on the Caldera Version 2.3 distribution media (though it was not part of the Standard Installation). Unfortunately, Caldera's pdksh package is not compatible with DB2. In the coming months, you should expect to see this problem fixed and a compatible pdksh package available for download from Caldera's ftp site at `ftp.calderasystems.com/pub/`. For now, you can get this package (called pdksh-5.2.13-3.i386.rpm) from the University of North Carolina's MetaLab site at: `www.metalab.unc.edu/pub/Linux/distributions/redhat/redhat-6.0/i386/RedHat/RPMS/`.

Because this is a Red Hat package, you will receive a dependency error on the glibc package when you try to install it. The glibc package is automatically installed with both versions of Caldera. You can therefore ignore this error; it is a result of different naming conventions between distribution vendors. To bypass the error, you will have to install this package using the no dependencies option (for example, the `rpm -i --nodeps` command). Refer to your Linux documentation for information on how to install this package using RPM.

Both Caldera Version 2.2 and Version 2.3 are installed with the libstdc++ 2.9.0 library. As previously mentioned, DB2 requires libstdc++ 2.8.0 to run (at the time this book was written, more recent versions of this package were not supported). Caldera Version 2.3 comes with the required libstdc++ 2.8.0 library (called libstdc++-compat-2.8.0-1.i386.rpm). This library is available on the Caldera Version 2.3 distribution media (located in the /col/contrib/RPMS directory) and can be installed using RPM. Caldera Version 2.2 does not ship with this library. You can download this library from their ftp site at `ftp://ftp.calderasystems.com/pub/openlinux/2.3/contrib/RPMS/`. Refer to your Linux documentation for information on how to install this package using RPM.

Internally, DB2 requires a file called libcrypt.so.1. Problems that arose from US export laws caused Caldera Version 2.2 to ship without this file. This problem has been fixed for Caldera Version 2.3. To resolve this problem on a workstation that is running Caldera Version 2.2, download the package called glibc-crypt-2.1-3i.i386.rpm from the Linux Land ftp site at: `ftp://ftp.linuxland.de/pub/OpenLinux/crypto/2.2/RPMS/`. Refer to your Linux documentation for information on how to install this package using RPM.

You are now ready to install DB2 on a workstation that is running Caldera Open Linux Version 2.2 or Version 2.3.

5.9.2 Red Hat Linux Version 5.2 or Version 6.0

The Red Hat Version 5.2 or Version 6.0 Server distributions are the easiest to enable for DB2. The only pitfall from the installation is the missing pdksh package that is required to run the DB2 Installer program. This package is available on the Red Hat Version 5.2 and Version 6.0 distribution media (in the /RedHat/RPMS directory) and can be installed using RPM. Refer to your Linux documentation for information on how to install this package using RPM.

You are now ready to install DB2 on a workstation that is running Red Hat Version 5.2 or Version 6.0,

5.9.3 TurboLinux Version 3.6

The TurboLinux Version 3.6 Workstation distribution had some hurdles to overcome in order to enable it for DB2. However, there is a downloadable fix that is available from the Web. Save yourself a lot of trouble and download this fix. The fix is a compressed and tarred file called tl36_instfix.tar.Z (the second character in the downloadable fix is the letter 'l', not the number '1') and is available at `ftp://ftp.software.ibm.com/ps/products/db2/tools/`. A README file for this fix (called tl36_instfix.readme.txt) can be viewed from this URL and includes complete instructions on how to implement this fix.

When you have downloaded the DB2 Installer program fix, you need to add the pdksh package. This was the only package that was missing from the base workstation installation. This package is available on the Turbo Linux distribution media (located in the /TurboLinux/RPMS directory) and can be installed using RPM. Refer to your Linux documentation for information on how to install this package using RPM.

You are now ready to install DB2 on a workstation that is running TurboLinux Version 3.6.

5.9.4 SuSE Linux Version 6.1

The SuSE Version 6.1 Network Oriented System distribution is fairly simple to enable for DB2, but initially confusing. Thanks to our overcome frustrations, you will not have to worry about the confusing part. We installed the SuSE Network Oriented System and noticed that some of the users that the DB2 Installer program would try to create already exist. As it turns out, SuSE packages a Beta version of DB2 Universal Database Version 5.2 with the SuSE Version 6.1

distribution (DB2 Universal Database is not part of the Network Oriented System installation type). To facilitate the installation, SuSE creates the default DB2 users during any installation of their distribution. This causes some problems when performing a default DB2 Version 6.1 installation because the default user names cannot be created. To make matters worse, we could not find any information on the passwords assigned to the users (they are not the standard DB2 default passwords) and some of the settings that SuSE implements for DB2 do not work. It is quite confusing, and in the end, you are far better off to remove the users created by the default SuSE installation and then proceed to install DB2. To remove the default DB2 users, enter the following commands:

```
userdel db2inst1
userdel db2fenc1
userdel db2as
```

DB2 requires glibc Version 2.0.7 or higher to run. The SuSE distribution comes with this package; however, it refers to this package by a different name. SuSE calls this package shlibs. This causes problems because the DB2 Installer program fails to recognize the existence of the required glibc package, and ultimately fails. A "dummy" glibc RPM is included on the DB2 CD-ROM. If you install this package, the installation will run smoothly. The package is called glibc-2.0.7-0.i386.rpm and is located in the /db2/install/dummyrpm directory on the DB2 CD-ROM. Refer to your Linux documentation for information on how to install this package using RPM.

You are now ready to install DB2 on a workstation that is running SuSE Version 6.1.

5.10 Performing the installation

Now that your Linux workstation is DB2-enabled, you are ready to install DB2 Universal Database Version 6.1. Sometimes, display problems can occur when running the DB2 Installer program. To refresh the current screen at any time, press Ctrl+L. To avoid most potential display problems, install DB2 through a virtual console session (a terminal window outside of the graphical interface that most Linux distributions are installed with). To change to a virtual console session, press Ctrl+Alt+F1. To change back to the graphical interface, press Ctrl+Alt+F7 [1] (your particular Linux distribution may differ; refer to your Linux documentation for more information).

To install DB2, perform the following steps:

1. Log on to the Linux workstation as *root*.

[1] On Caldera Open Linux the graphical console session usually resides in F8.

2. Insert the DB2 Universal Database Version 6.1 CD into the CD-ROM drive.
3. Mount the CD by entering the following command:

   ```
   mount -t iso9660 -o ro /dev/cdrom /mnt/cdrom
   ```

   The /mnt/cdrom path is the default CD-ROM mount point for any Linux distribution (except SuSE) as defined by the /etc/fstab file. On SuSE Version 6.1, by default, the CD-ROM must be mounted on the /cdrom directory.

Note

Even if you are installing DB2 on a workstation that is running Turbo Linux Version 3.6, you still need to mount the CD-ROM. The image created by the install fix links to the DB2 CD-ROM.

4. Change focus to the mounted CD-ROM directory.

   If you are installing DB2 for Turbo Linux Version 3.6, you would change to the directory that you created for the installation fix (for example, /tmp/image). Refer to the README provided with the downloadable fix for more information.
5. Enter the `./db2setup` command to begin the installation. The Install DB2 V6 window opens:

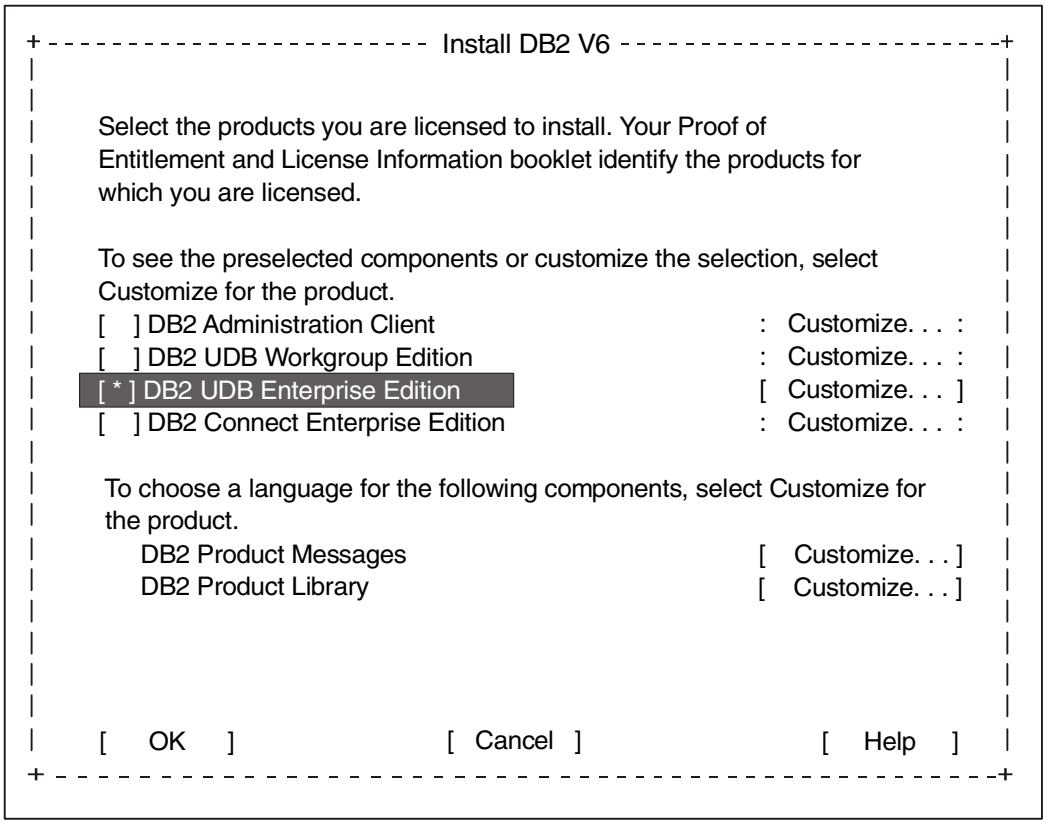


Figure 17. DB2 install screen

6. Press the Tab key to change the highlighted option and the Enter key to select or deselect an option. For more information or assistance during the installation of DB2, select **Help**.

   From the product list, select the DB2 product that you want to install. To install DB2 Universal Database Enterprise Edition, select **DB2 UDB Enterprise Edition** and then **OK**. If you are not planning to access data that resides on a host (MVS, VM, OS/390) or AS/400 machine, you can select **DB2 UDB Workgroup Edition**. DB2 Enterprise Edition contains all of the functionality available with DB2 Workgroup Edition, but also includes the bundled software required to access host or AS/400-based DB2 databases.

Note

If you want to install the DB2 documentation, select **DB2 Product Library** and then **Customize.** You will be given the option to install the documentation in a variety of languages.

For the purposes of this example, select **DB2 UDB Enterprise Edition** and then **OK**. The Create DB2 Services window opens.

7. Select the **Create a DB2 Instance** option. This option will create an instance and a user that will have System Administrative (SYSADM) authority on the instance. Enter a password for the db2inst1 user and verify it by retyping this password in the field provided.

   The SYSADM user has complete control over an instance and any databases that reside within it. For more information on DB2 authorities, refer to your DB2 product documentation.

8. Select the **Properties** option.
9. Select the **Create a sample database for a DB2 Instance** option and then **OK**.
10. Select **OK**.
11. You will be asked to create a user that will be used to execute user-defined functions (UDFs) and stored procedures. Enter a password for the db2fenc1 user, verify it by retyping the password in the field provided, and select **OK**. You are returned to the main Create DB2 Services window.
12. Select the **Create the Administration Server** option. Enter a password for the db2as user, verify it by retyping this password in the field provided, and select **OK.**

    The Administration Server is used to automate the configuration of connections to DB2 databases and to enable the remote administration tools. For more information on the Administration Server and the available administration tools, refer to your DB2 product documentation.

13. A pop-up window will appear informing you of the DB2 system name for your computer. Select **OK**.
14. You are returned to the Create DB2 Services window. Select **OK.**
15. The Summary Report window opens. This window lists all of the options that you have selected for installation. To review the entire list,

use the cursor keys to scroll the contents. When you are finished, select **Continue**.

16. If you are satisfied with the installation and configuration selections that you have made, select **OK**. If you would like to change some of the settings that you selected, select **Cancel**.
17. When the installation completes, the Status Report window opens. This window lists all of the actions performed by the DB2 Installer program and notes if each task was completed successfully. To review the entire list, use the cursor keys to scroll the contents.You can review a log of installation activity by selecting **View Log**. When you are finished, select **OK**, then **Close**, and dismiss any remaining windows by selecting **OK**.

5.10.1 Verifying the installation

Now that you have successfully installed DB2, verify the installation by accessing data from the sample database (called SAMPLE) that was created during the installation.

To access data from the SAMPLE database, perform the following steps:

1. Log on to the system as a user that has System Administrative (SYSADM) authority on the instance. You created this user account when you installed DB2. The default username was db2inst1.
2. Ensure that the database manager has been started by entering the `db2start` command[2].
3. Enter the following commands to connect to the sample database and retrieve a list of all the employees that work in department 20:

```
db2 connect to sample
db2 "select * from staff where dept = 20"
```

4. Enter the following command to reset the database connection:

```
db2 connect reset
```

Congratulations! You have successfully installed and configured DB2 on your Linux workstation and are ready to start using the world's leading database on the world's most utilized platform for e-commerce.

You should be quite proud of yourself, as this is not a trivial task. Many computer people around the world will appreciate (or envy) your new skill.

[2] During a default DB2 installation, the *db2inst1* instance is configured to automatically start after the installation completes.

The remainder of this book will show how to implement a basic e-business solution using DB2, WebSphere, and Linux.

5.11 Deinstalling DB2 Universal Database

During our experiments, we found it useful to not only know how to install a product, but how to remove a product as well. The steps in this section describes the basic tasks in removing a DB2 installation from your Linux workstation.

5.11.1 Step 1. Stop and remove the Administration Server

You must stop the Administration Server, if it exists, before you remove a DB2 product from your workstation. For the purposes of this book, we recommend that you drop the Administration Server as well, although this is not a requirement. Some users may choose to back up the Administration Server for later use. For more information, refer to your DB2 product documentation.

To stop and remove an Administration Server, perform the following steps:

1. Log on to the system as the user account that is associated with the Administration Server (by default, this is the db2as username).
2. Ensure that this is the user account that owns the Administration Server by entering the following command:

   ```
   dasilist
   ```

   This command should return output that matches the name of the user account that is associated with the Administration Server and that you are logged in as. If it doesn't, then you are not logged on to the system with the correct user account. If this command returns no output, an Administration Server does not exist on your workstation.
3. Stop the Administration Server be entering the following command:

   ```
   db2admin stop
   ```
4. Log out.
5. Log on to the system as a user with root authority.
6. Remove the Administration Server by entering the following command in the /usr/IBMdb2/V6.1/instance directory:

   ```
   ./dasidrop ASNAME
   ```

   where `ASNAME` is the name of the user account that is associated with the Administration Server (by default, this is the db2as username).

7. Optionally, remove the user account associated with the Administration Server by entering the following command:

   ```
   userdel db2as
   ```

8. Log out.

You have successfully dropped the Administration Server.

5.11.2 Step 2. Stop and remove any instances

You must stop all instances before you remove a DB2 product from your workstation. For the purposes of this book, we recommend that you drop all instances, although this is not a requirement. Some users may choose to back up their instances for later use. For more information, refer to your DB2 product documentation.

To stop and remove any instances on your system, perform the following steps:

1. Log on to the system as a user with root authority.
2. Obtain a list of all the instances on your system by entering the following command in the /usr/IBMdb2/V6.1/instance directory:

   ```
   ./db2ilist
   ```

   This command should return a list of all instances that are on your workstation. Make note of this list, as you will need to remember these names in order to remove all of the instances. If this command does not return any output, then there are no instances on your workstation.
3. Log on and out of the system as the instance owner for *each* instance that was returned to you when you ran the `db2ilist` command in the previous step. As you log on to each instance, enter the following command:

   ```
   db2stop
   ```

4. Log back on to the system as a user with root authority.
5. Remove each instance from the system by entering the following command in the /usr/IBMdb2/instance directory:

   ```
   ./db2idrop INSTNAME
   ```

   where `INSTNAME` is the name of each instance that you want to drop.

   The `db2idrop` command removes the instance specified from the list of instances and removes the /INSTHOME/sqllib directory (where INSTHOME is the home directory of the instance owner).

6. Optionally, remove any instance's associated user account (if used only for that instance) by entering the following command:

   ```
   userdel INSTNAME
   ```

   where `INSTNAME` is the name of the instance; for example, `db2inst1`.

7. Log out.

You have successfully dropped all of the instances on your workstation.

5.11.3 Step 2. Deinstall DB2

Now that you have stopped and removed the Administration Server and all instances, perform the following steps to deinstall DB2:

> **Note**
>
> You must stop any outstanding DB2 processes before removing DB2 from your workstation.

1. Log on to the system as a user with root authority.
2. Insert the DB2 Universal Database Version 6.1 CD into the CD-ROM drive.
3. Mount the CD by entering the following command:

   ```
   mount -t iso9660 -o ro /dev/cdrom /mnt/cdrom
   ```

   The /mnt/cdrom path is the default CD-ROM mount point for any Linux distribution (except SuSE) as defined by the /etc/fstab file. On SuSE Version 6.1, by default, the CD-ROM must be mounted in the /cdrom directory.

4. Change focus to the mounted CD-ROM directory.

   If you are installing DB2 for Turbo Linux Version 3.6, you would change to the directory that you created for the installation fix (for example, /tmp/image). Refer to the README provided with the downloadable fix for more information.

5. Enter the following command to deinstall DB2:

   ```
   db2_deinstall -n
   ```

6. Log out.

5.12 WebSphere Application Server - installation and configuration

Once the correct Web server has been installed and verified you can proceed with this step. WebSphere 2.0.3 requires Java 116 to be installed and this is now supplied as part of the WebSphere install package.

The first install step is to extract the WebSphere distribution archive into a directory you can install it from. We set up an install directory called `/home/was_install` (see Figure 18). Having unzipped and un-tarred the archive file into this directory, you should end up with the four *.rpm files shown. Alternately you may have received WebSphere on a CD, in which case you can mount the CD and install the WebSphere *.rpm files directly from it.

r

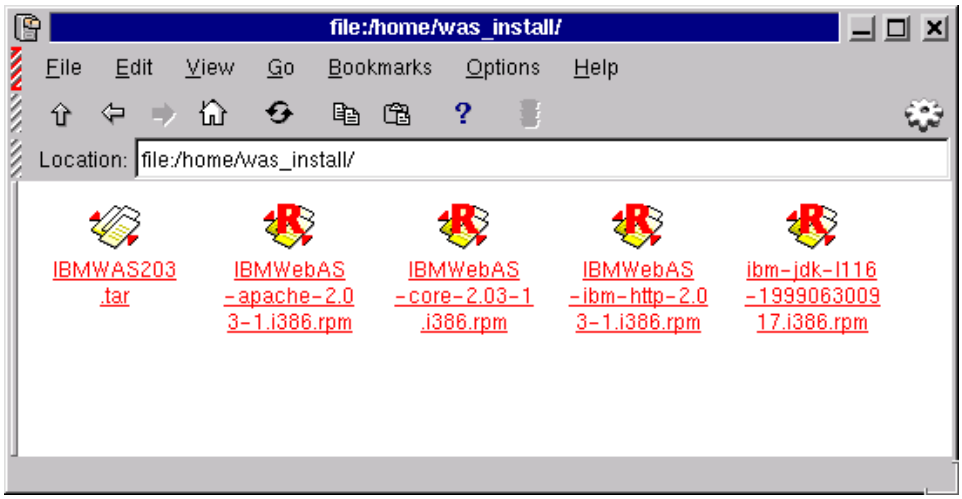


Figure 18. WebSphere 2.0.3 install package un-tarred in directory /home/was_install

Note

Do not try to install WebSphere using visual installers. These may fail to detect warning messages issued during the install process.

Open an X-term or a Linux console and run the following `rpm` commands. Pay close attention to any warning messsages. These will mostly be about prerequisite software that may be needed or will be about environment variables that need to be set for the install to proceed.

5.12.1 WebSphere install steps

```
1. rpm -i ibm-jdk-l116-199906300917.i386.rpm
2. export JAVA_HOME=/opt/ibm-jdk-l116
3. rpm -i IBMWebAS-core-2.03-1.i386.rpm
4. rpm -i IBMWebAS-ibm-httpd-2.03-1.i386.rpm
```

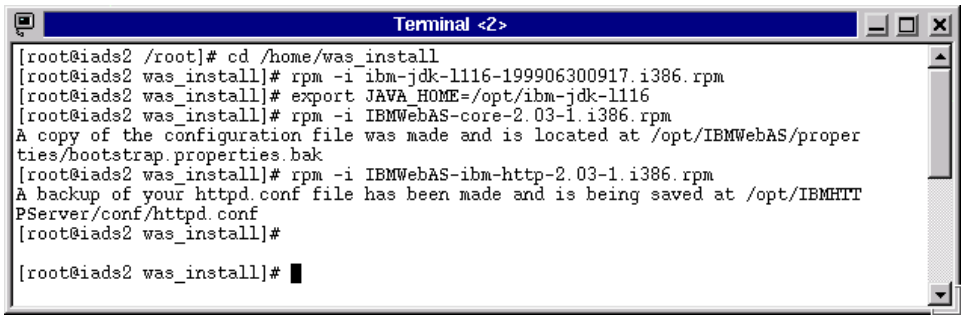


Figure 19. A standard WebSphere install

Figure 19 shows the steps. Figure 20 shows another example. The core module expects JAVA_HOME to be set. The IBMWebAS-apache module looks for file /etc/httpd/conf/httpd.conf while IBMWebAS-ibm-http looks for /opt/IBMHTTPServer/conf/httpd.conf. If either cannot be found WebSphere asks you to set an environmernt variable called HTTPD_CONF_DIR and then asks you to try again.

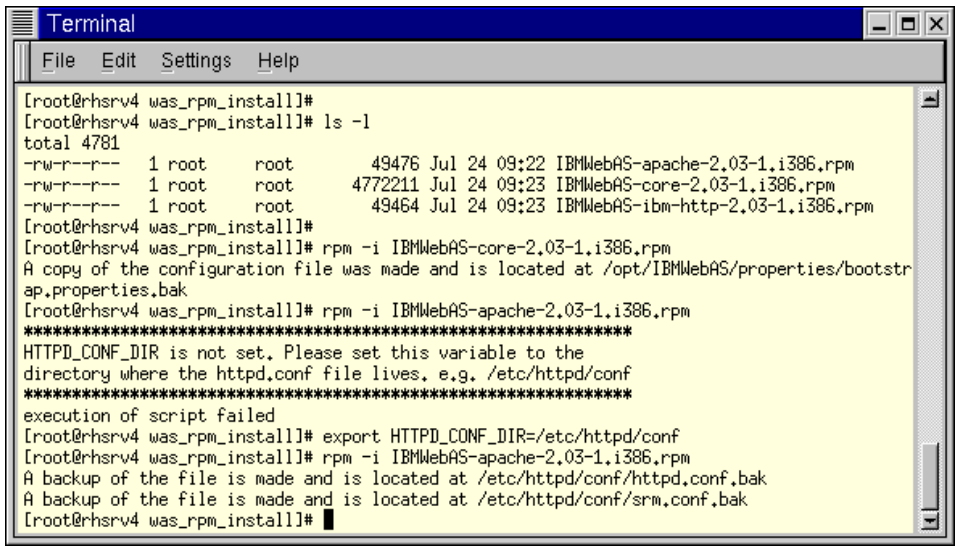


Figure 20. Another WebSphere install example

Figure 21 shows the items added to the Apache configuration file after we installed the IBMWebAS-apache configuration module. The same changes are applied to /opt/IBMHTTPServer/conf/httpd.conf if the IBMWebAS-ibm-http configuration module is installed instead of the Apache module.

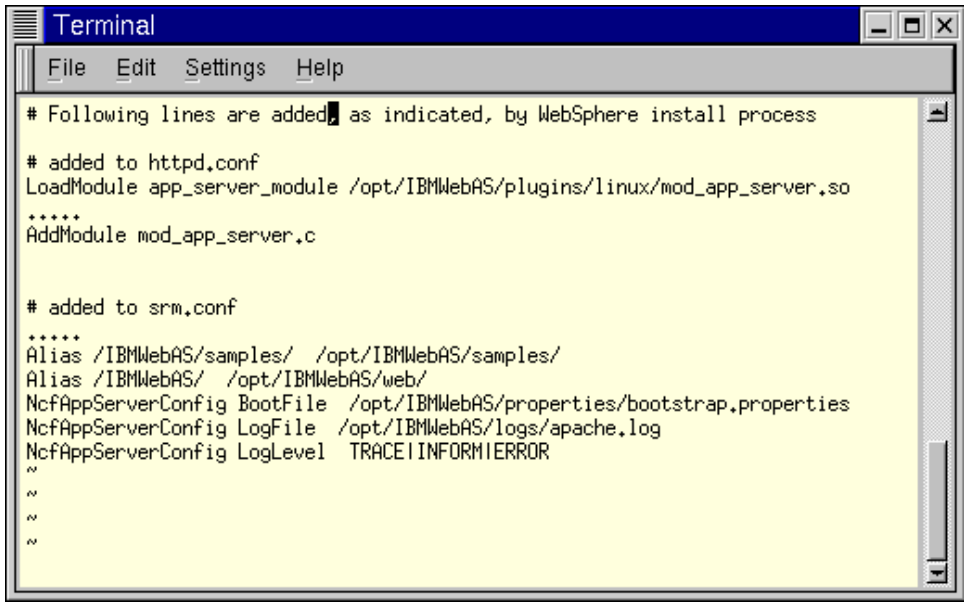


Figure 21. Changes made to Apache config files by the WebSphere install

See Figure 22 for a quick way of stopping and starting Apache.

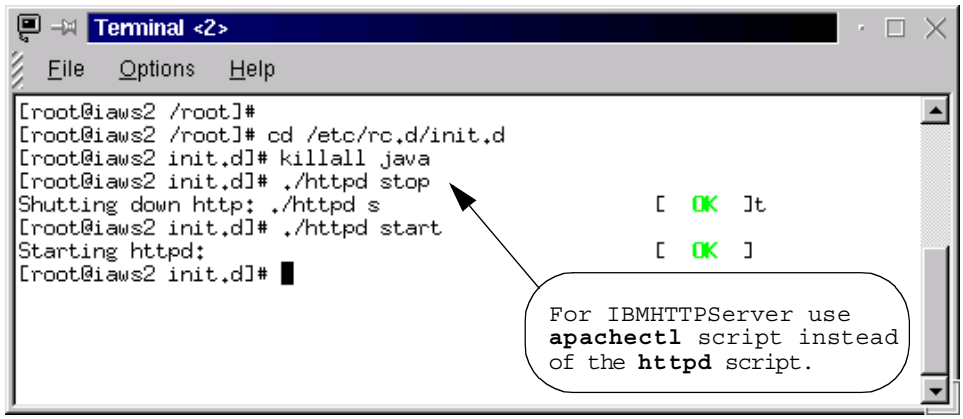


Figure 22. A quick way to stop WebSphere and Apache and start them again

Next we checked out Apache and WebSphere by accessing the following two Web sites: `http://localhost/` and `http://localhost/servlet/snoop`. Sample output from both these commands is shown in Figure 13 on page 34, Figure 14 on page 35 and Figure 25 on page 57.

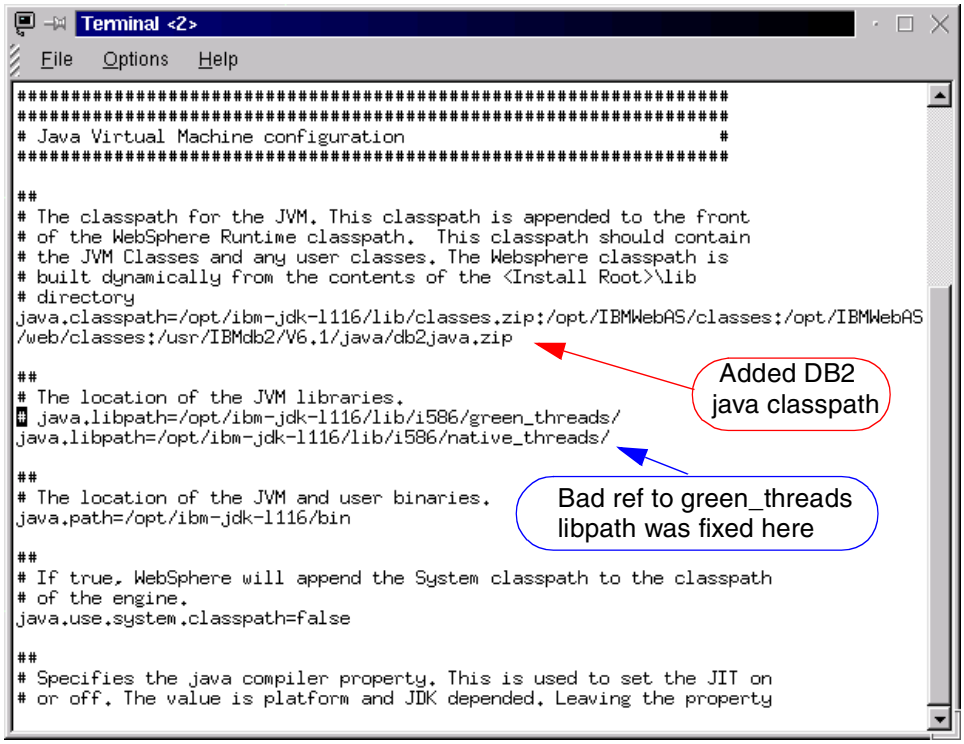


Figure 23. Part of WebSphere's bootstrap.properties file

WebSphere creates a special file in its home directory that is called /opt/IBMWebAS/properties/bootstrap.properties. During our install we noticed that WebSphere had incorrectly set up the java.libpath pointer to a wrong path. We corrected this as shown in Figure 23. You can see the error line commented out and the correct line below it. There is no green_threads sublibrary in IBM's jdk116, only a native_threads sublibrary.

The file bootstrap.properties also has parameters for setting WebSphere tracing on or off. It is worth looking through the bootstrap.properties file.

Note

We also added libpath and classpath references to DB2 libs. These are important if you plan to run DB2 on the same computer as WebSphere.

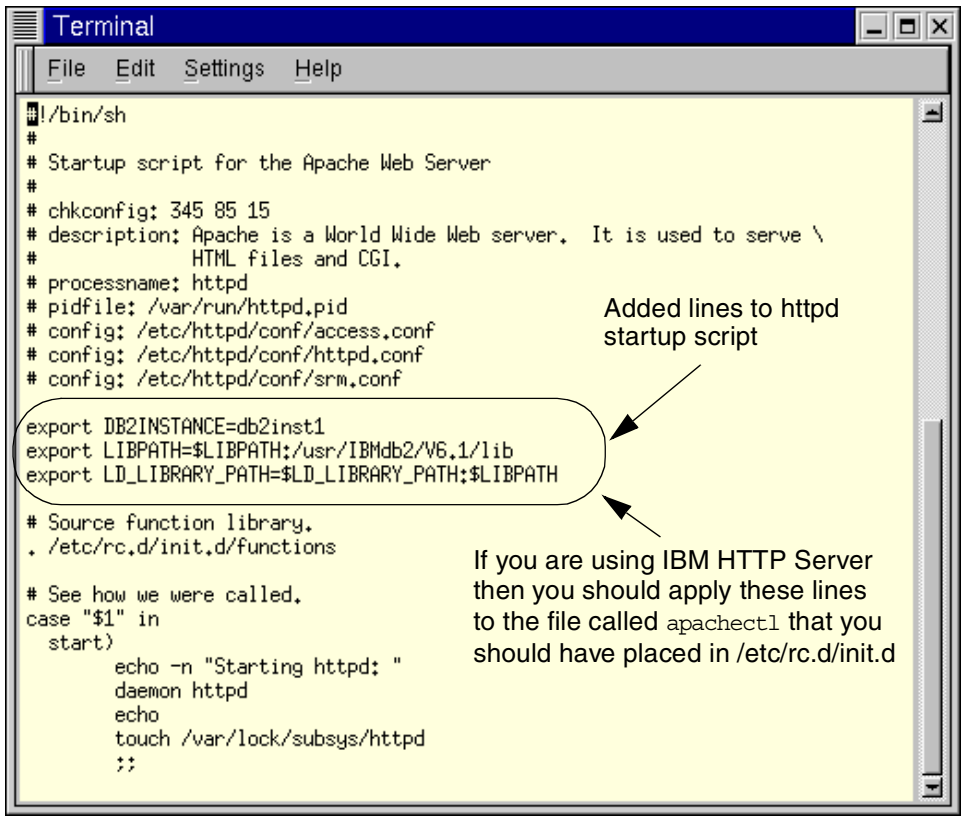


Figure 24. Lines we needed to add to get WebSphere to recognize DB2

A major problem that we encountered was to do with WebSphere starting up during the Linux boot sequence, and not picking up a DB2 environment variable identifying the DB2INSTANCE installed on the same machine.

When this occurs WebSphere will not find DB2. We solved this problem by adding the lines shown in Figure 24 to the httpd or apachectl startup script in the /etc/rc.d/init.d startup scripts directory.

Setting up apachectl script for the IBM HTTP Server was fully explained in 5.7.3, “Setting up IBM HTTP Server startup script” on page 35.

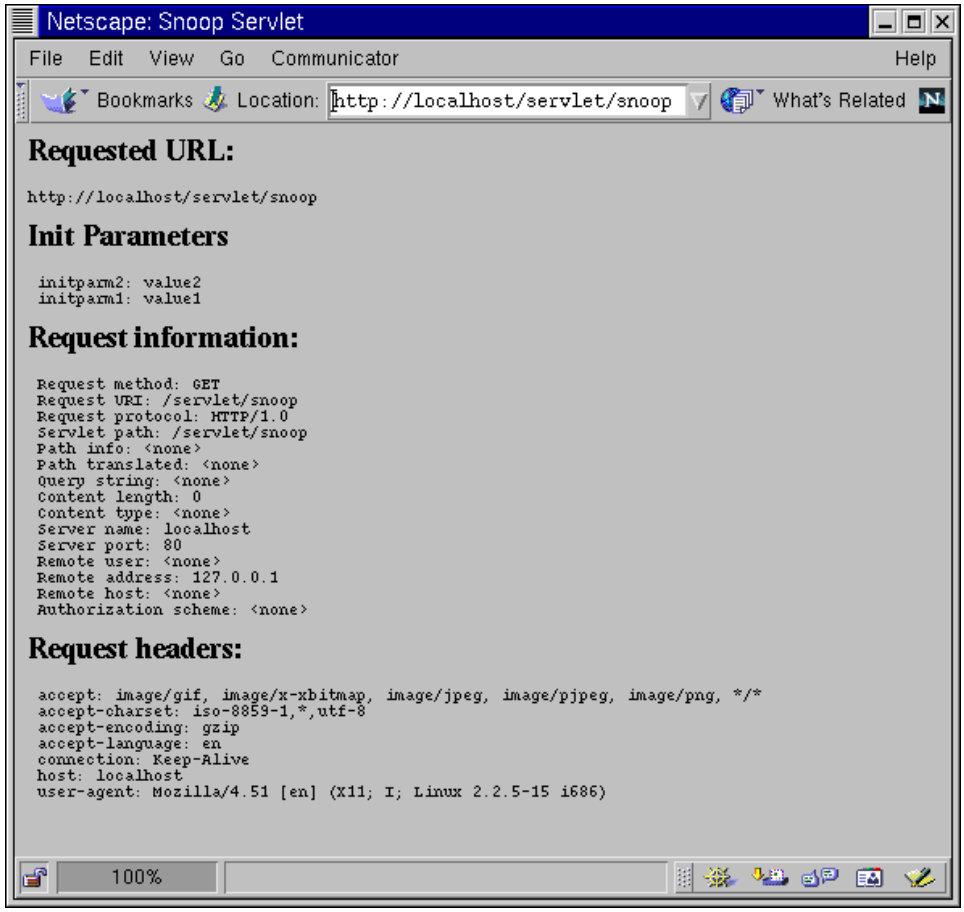


Figure 25. After WebSphere is installed, test it with the snoop servlet

Figure 25 shows the output from the snoop servlet, which is very useful for testing that WebSphere is actually running.

It returns information about the client browser's environment. The servlet code is useful for seeing how this information can be accessed and processed. The source code for the snoop servlet is provided by WebSphere in the WebSphere servlets directory.

5.13 VisualAge for Java for Linux - installation and configuration

This product was the easiest we had to install. All that is needed is to log in as yourself and copy the va4java tar file to your home directory. Note it must be your home directory. Then un-tar it, that is all that you need do. To test, you can click (KDE) or double-click (GNOME) on the vajide icon that was placed in your home directory folder. The vajide icon can also be copied to your desktop as shown in Figure 27 and Figure 28.

Note

Do not install VA for Java while logged in as root. Log in as a normal user. Also you must install it in your home directory.

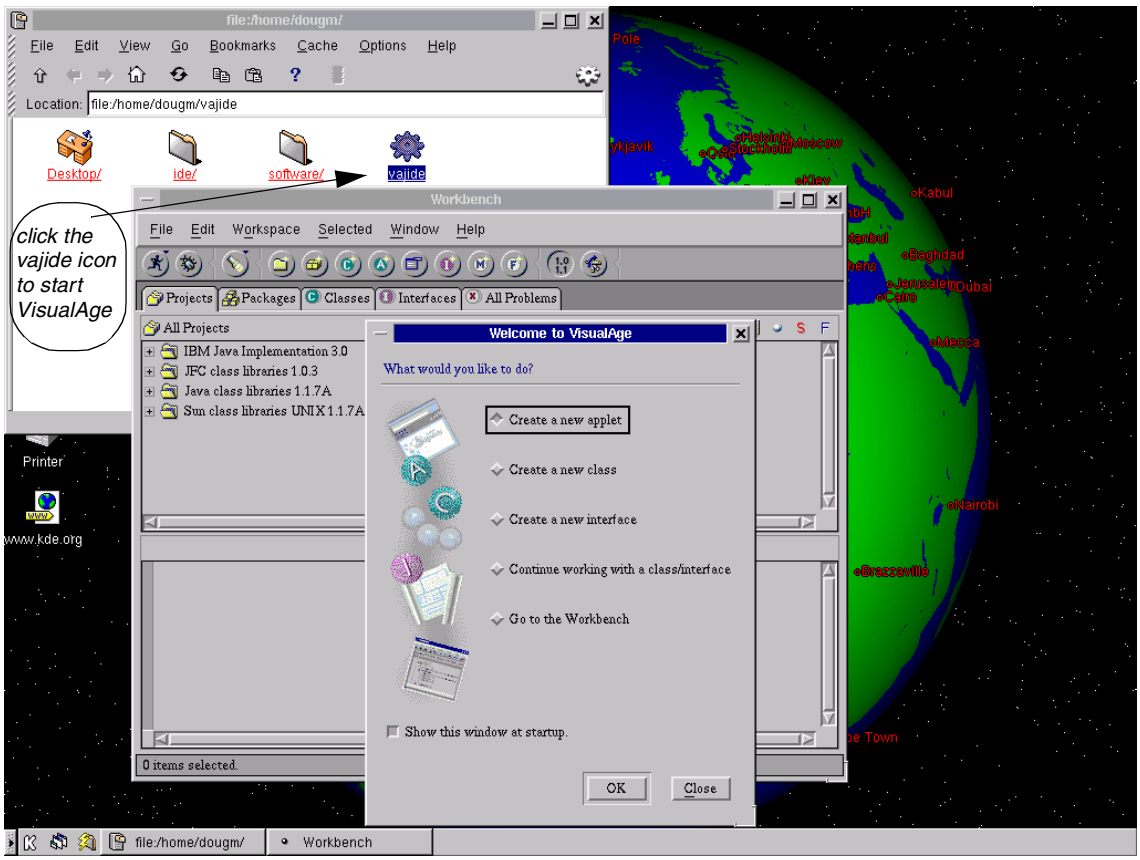


Figure 26. VisualAge shown installed and invoked on Caldera OL 2.2 using KDE desktop

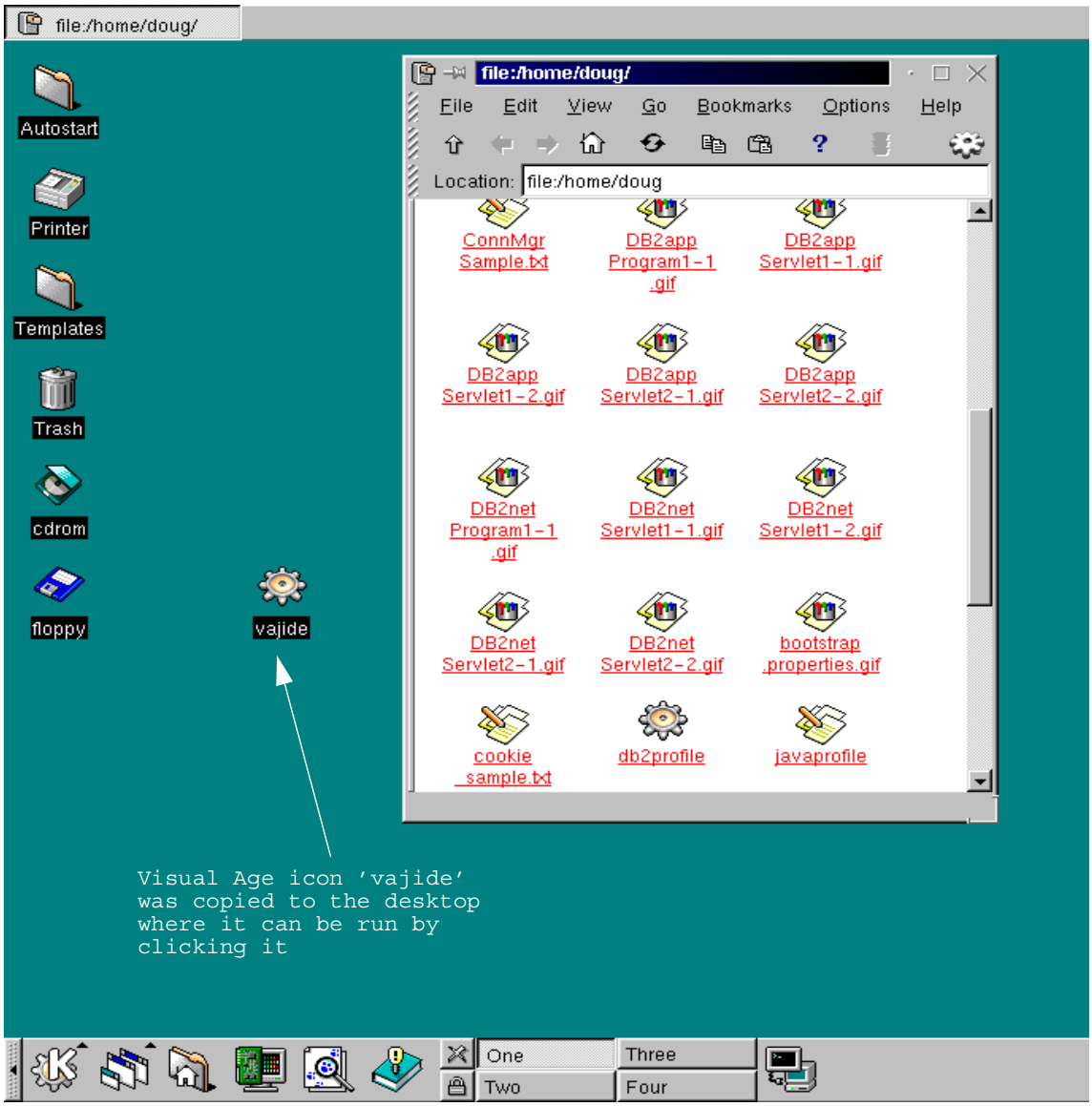


Figure 27. Red Hat KDE showing vajide as an icon that can be single-clicked to start VisualAge

Figure 27 shows where we copied the vajide icon from our home folder to the desktop where (using KDE). It can be single-clicked to start VisualAge running.

Figure 28 shows the same being done but on Red Hat's GNOME desktop. However, on GNOME you need to double-click the icon to run VisualAge.

Selection behavior can be configured to suit individual tastes, so that the number of clicks used to start an application can be tailored.

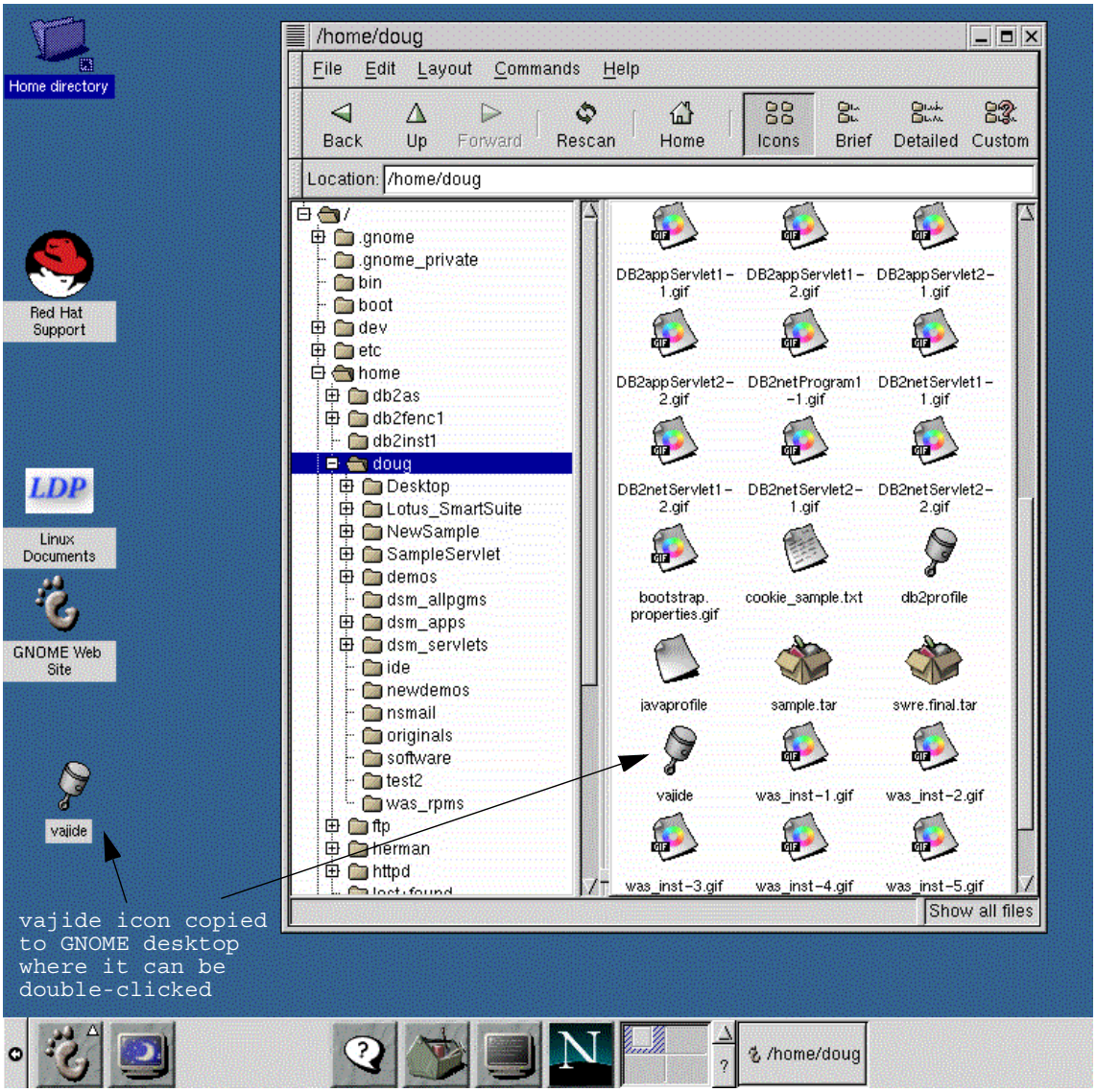


Figure 28. Red Hat GNOME desktop with vajide as an icon that can be double-clicked to run VisualAge

Part 2. Programming model

In this part we discuss the Java servlet programming model.

Chapter 6. Web programming model

This chapter discuss the following topics:

- Overview of Java servlets
- Structure of the Java servlet API
- Java Servlets Development Kit from SUN
- WebSphere Application Server Servlets API extensions
- Servlets with JSPs

6.1 Overview of Java servlets

Servlets are protocol and platform-independent server-side software components, written in Java. Servlets run on a Web server machine inside a Java-enabled server, that is, a server able to start the Java Virtual Machine (JVM) in order to support the use of Java servlets. They dynamically extend the capabilities of the server because they provide services over the Web using the request-response paradigm.

Servlets were initially supported in the Java Web Server from JavaSoft. Since then, several other Java-based Web servers have supported the standard servlet API. Servlets were introduced to interactively view and modify data and to generate dynamic Web content. From a high-level perspective, the process flow would be:

- The client sends a request to the server.
- The server sends the request information to the servlet.
- The servlet builds a response and passes it to the server. That response is dynamically built and the contents of the response usually depend on the client's request.
- The server sends the response back to the client.

The flow is shown in Figure 29 on page 64.

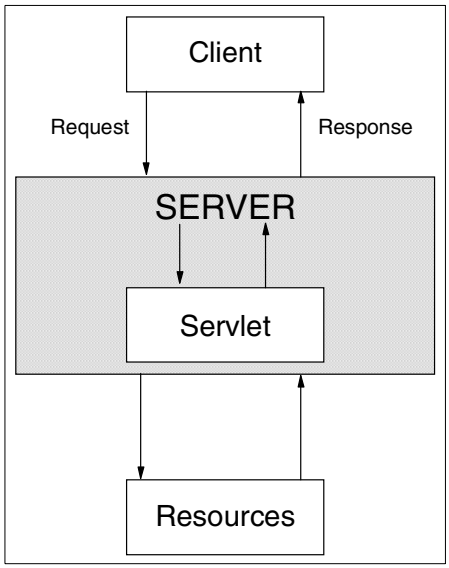


Figure 29. Process flow from a high-level Perspective

Servlets look like ordinary Java programs that begin importing some particular Java packages that belong to the Java servlet API. Since servlets are object bytecodes that can be dynamically loaded off the Net, we could say that servlets are to the server what applets are to the client. But since servlets run inside servers, they do not need a graphical user interface (GUI). In this sense servlets are also called faceless objects.

6.1.1 Advantages of servlets

Java servlets offer a lot of advantages:

- A servlet can interact with other resources (files, databases, applets, applications written in Java or in other languages) to construct the response that will be sent back to the client and, if needed, to save information about the request-response interaction.
- With a servlet approach, the server can grant full access to local facilities, such as databases, and trust that the servlet itself will control the amount and precise nature of access that is effectively afforded to external users. So, for example, the Java servlet API provides all the methods to monitor and verify the origin of all requests. Moreover, on the Internet Connection Secure Server, servlets cannot be loaded from the network. They must reside on the local system. As a consequence, if a proprietary algorithm is built into a servlet, the code never passes beyond the boundaries of the

server; only the results that it produces do. If the code is not passed to the client, it cannot be saved or disassembled.

- Servlets can be client programs of other services, for example, when they are used in distributed application systems.
- It is possible to invoke them from a local or remote disk across the network. All the examples that are described in this chapter demonstrate this.
- Servlets can be chained. This means that one servlet can call another servlet, thus becoming its client. It can also call several servlets in sequence.
- Servlets can be dynamically called from within HTML pages.
- The Servlet API is protocol-independent. It does not assume anything about the protocol used to transmit it on the Internet.
- Like all Java programs, servlets can use all the capabilities of the object-oriented Java language:
    1. They can be rapidly developed.
    2. Lack of pointers promotes robust applications (unlike C).
    3. A servlet service routine is only a thread and not an entire operating system process. That is the reason why a servlet can handle connections with multiple clients, accepting requests and downloading responses back to multiple clients. This is a more efficient mechanism than using CGI.
    4. Servlets are portable. They run on a variety of servers without needing to be rewritten.

Note

You can take the same Java source code and compile it, using the Java compiler javac, on different platforms (AIX, Solaris, DOS, Windows, OS/2, Macintosh, and more) without having to make any changes. You can even compile it on one platform and then use the same bytecode class file on different platforms.

- Memory access violations are not possible, so faulty servlets will not crash servers.

6.2 Structure of the Java servlets

Servlets are ordinary Java programs, but they use some additional packages found in the Java servlet API. When you write the code for a Java servlet, you must import at least one of the following two packages:

- javax.servlet
- javax.servlet.http

These two packages contain seven interfaces, five classes, and two exceptions. Before going on with this chapter, it is helpful to have a visual idea of the structure of the Java servlet API. The following table should help:

Table 2. Structure of the Java servlet API

	javax.servlet	**javax.servlet.http**
Interfaces	Servlet ServletConfig ServletContext ServletRequest ServletResponse	HttpServletRequest HttpServletResponse
Classes	GenericServlet ServletInputStream ServletOutputStream	HttpServlet HttpUtils
Exceptions	ServletException Unavailable Exception	

Note

Generally speaking, a Java class is composed of two things: variables and methods. In addition, all Java variables and methods must be a member of a class. Instance variables are what describes an object and typically, in pure object-oriented languages such as Java, they are encapsulated. That means that only the objects themselves can change these variables. To do this, objects use functions called methods.

As you can see, the amount of material introduced by the Java servlet API is not too big. Nevertheless, these few interfaces, classes and exceptions are able to make a Java server very powerful.

Table 2 shows 14 items. It would be good to have a general description of each of them before moving onto the next section.

Note: The description of the Java servlet API provided here is based upon the official information located at: `http://www.javasoft.com`.

6.2.1 Interface javax.servlet.Servlet

The public interface Servlet interface is used to develop servlets and it is very important. In fact, all servlets implement this interface, usually by extending either the GenericServlet class or the HttpServlet class, which is a GenericServlets descendent. The Servlet interface defines the method init() to initialize a servlet, the method service() to receive and respond to client requests, and the method destroy() to unload the servlet and its resources. These are known as life cycle methods. Other methods defined by the servlet interface are getServletConfig(), which returns a ServletConfig object containing any initialization parameters and startup values for this servlet and getServletInfo(), which returns a string containing information about the servlet.

6.2.2 Interface javax.servlet.ServletConfig

The public interface ServletConfig interface is implemented by services in order to pass configuration information to a servlet when it is first loaded. The ServletConfig interface can also be implemented by servlets. For example, the GenericServlet does this. When implemented by a servlet, the methods in the interface make getting the configuration data more convenient.

The ServletConfig interface provides two methods to handle the configuration data: getInitParameterNames() and getInitParameter(). getInitParameterNames() returns the names of the servlet's initialization parameters as an enumeration of strings or an empty enumeration if there are no initialization parameters. By passing each name to the getInitParameter() method, you can retrieve the single value for a specified parameter or null if the parameter does not exist.

Of course, if you already know the name of the servlet initialization parameters, you do not need to use the getInitParameterNames() method to retrieve those names, but you can directly call the getInitParameter() method to obtain the relative values.

The ServletConfig interface also provides the getServletContext() method, which returns a ServletContext object. A servlet can implement getServletContext() by writing:

```
public ServletContext getServletContext() {
   return getServletConfig().getServletContext();
}
```

Using the public interface ServletConfig provides access to the servlets context with a single call to the method getServletContext().

6.2.3 Interface javax.servlet.ServletContext

The public interface ServletContext gives servlets access to information about their environment. In fact it provides the following methods:

- getAttribute() returns the value of the named attribute of the network service or the value null, if the attribute does not exist.
- getMimeType() returns the mime type of the specified file or null if not known.
- getRealPath() applies alias rules to the specified virtual path and returns the corresponding real path.
- getServerInfo() returns the name and version of the network service under which the servlet is running.
- getServlet() returns the servlet of the specified name or null if not found.
- getServlets() returns an enumeration of servlet objects in this server.

The information returned by all these methods relates to the servlets environment.

The ServletContext interface also allows servlets to log significant events. Servlets writers decide what data to log using the log() method, which writes the given message string to the servlet log file.

The ServletContext interface is implemented by services and used by servlets. Servlets get the ServletContext object with the getServletContext() method of the ServletConfig object. This object is provided to the servlet at initialization and is accessible using the servlets getServletConfig() method.

6.2.4 Interface javax.servlet.ServletRequest

The public interface ServletRequest interface is used to get data from the client to the server from a service request. Network service developers implement the ServletRequest interface. Its methods are then used by the servlets when the service() method is executed. The ServletRequest object is passed as an argument to the service() method. Some of the data provided

by the ServletRequest object includes parameter names (getParameterNames()) and values (getParameterValues()), attributes (getAttribute()) and an input stream (getInputStream()). Subclasses of ServletRequest can provide additional protocol-specific data. For example, HTTP data is provided by the interface HttpServletRequest, which extends ServletRequest.

6.2.5 Interface javax.servlet.ServletResponse

The public interface ServletResponse interface is used for sending data from the servlets service() method back to the client. Network service developers implement this interface. Its methods are then used by servlets when the service() method is run, to return data to clients. The ServletResponse object is passed as an argument to the service() method. This interface provides the method getOutputStream(), which returns an output stream for writing response data.

6.2.6 Interface javax.servlet.http.HttpServletRequest

The public interface HttpServletRequest extends ServletRequest interface represents an HTTP servlet request. It gets data from the client to the servlet for use in the service() method. It allows the HTTP-protocol specified header information to be accessed from the service() method.

6.2.7 Interface javax.servlet.http.HttpServletResponse

The public interface HttpServletResponse extends ServletResponse interface represents an HTTP servlet response. It allows a service() method to manipulate HTTP protocol-specified header information and return data to its client.

This interface is often used when the Web server must send the response back to the client's browser. An example of some code that will work with this interface inside the service() method follows:

```
ServletOutputStream out = response.getOutputStream();
response.setContentType("text/html");
out.println("How to use the interface HttpServletResponse");
out.close();
```

The response is the HttpServletResponse object passed as a parameter to the service() method.

6.2.8 javax.servlet.GenericServlet

The GenericServlet class implements the servlet interface and the ServletConfig interface. Servlet developers typically subclass GenericServlet or its descendent HttpServlet.

```
public abstract class GenericServlet
   extends Object
   implements Servlet, ServletConfig
```

Note

If you are writing a servlet that interacts with HTML Web pages containing forms, you will probably create your servlet by subclassing HttpServlet. For all other servlets you will probably start by subclassing GenericServlet.

A servlet that communicates directly with an applet on the client's browser is an example of a servlet that would not use HTTP for its communications. You would create a servlet like that by subclassing GenericServlet instead of HttpServlet.

The GenericServlet provides simple versions of the life cycle methods init() and destroy() and the methods in the ServletConfig interface. The servlet developer must override only the service() method.

6.2.9 Class javax.servlet.ServletInputStream

The public abstract class ServletInputStream extends InputStream class is an abstract class that will be implemented by network services developers. An object from this class provides an input stream for reading servlet requests and it provides an efficient readLine() method. For some application protocols, such as the HTTP POST and PUT methods, servlet developers use the input stream to get data from clients. They access the input stream using the ServletRequest's getInputStream() method available from within the servlet's service() method.

6.2.10 Class javax.servlet.ServletOutputStream

The public abstract class ServletOutputStream extends OutputStream class is an abstract class that will be implemented by network services developers. An object in this class provides an output stream for writing servlet responses. Servlet writers use the output stream to return data back to

clients. They access it using the ServletResponse's getOutputStream() method available from within the servlet's service() method.

6.2.11 Class javax.servlet.http.HttpServlet

The public abstract class HttpServlet extends GenericServlet class is called the HttpServlet class. It is an abstract class that simplifies writing HTTP 1.0 servlets. It extends the GenericServlets base class. Servlet developers implement the servlet interface usually by extending either the GenericServlet class or the HttpServlet class. Because it is abstract, servlet writers must subclass it and override at least one method.

You subclass the HttpServlet class typically when your servlet must interact with HTML pages containing forms. The information entered into the form on the client Web page is passed to the servlet for processing. Two different HTML methods can be used to do this: GET and POST, depending on what is coded on the form present in the HTML page on the client.

The methods in the HttpServlet class that are normally overridden are:

- doGet() - If the information entered in the form on the client's browser is provided by the GET method.
- doPost() - If the information entered in the form on the client's browser is provided by the POST method.

It is up to the servlet developer to override the doGet() or doPost() methods in order to accept the client's request and give a response back to it.

This class implements the service() method, that in the GenericServlet class was abstract (see 6.2.8, "javax.servlet.GenericServlet" on page 70). Therefore, servlet developers can accept it if they are supporting HTTP 1.0. To support HTTP 1.1 methods (for example, OPTIONS, PUT, DELETE, and TRACE) they will probably override the service() method and handle those additional HTTP methods directly.

6.2.12 Class javax.servlet.http.HttpUtils

The class that provides a collection of static utility methods useful to HTTP servlets is public class HttpUtils extends Object. For example, getRequestURL() takes an HttpServletRequest object as an argument and reconstructs the URL used by the client to make the request. This class also provides two other utilities:

1. parsePostData() - Parses the data that is posted to the server using the HTTP POST method from a form in the client's browser.

2. parseQuesryString() - Parses the query string.

Both these methods return a hashtable object that maps keys to values.

6.2.13 Exception javax.servlet.ServletException

The exception class that is provided to indicate a servlet problem is public class ServletException extends Exception.

6.2.14 Exception javax.servlet.UnavailableException

The exception that indicates that a servlet is unavailable is public class UnavailableException extends ServletException. There are two types of this exception:

1. Permanent

   The servlet will not be able to handle client requests until some administrative action is taken to correct a servlet problem. For example, the servlet might be configured incorrectly or the state of the servlet might be corrupted.

2. Temporary

   The servlet cannot handle requests at this moment, due to a system-wide problem. For example, another server might be accessible or there may be insufficient memory or disk storage to handle requests.

Network services might safely treat both types of exceptions as permanent, but it is better to have the option to indicate a temporary outage to provide more flexibility as well as a possible means to report on the types of outages. For example, requests to the servlet might be blocked or deferred for an amount of time suggested by the servlet, rather than being rejected until the service itself restarts.

6.3 Java Servlets Development Kit from Sun

The Java Servlet Development Kit (JSDK) is a simple command-line development environment for developing and testing servlets, the javax.servlet package sources. The kit can be downloaded from `http://www.javasoft.com/products/servlet/index.html`

6.4 WebSphere Application Server Servlets API extensions

Servelet model programming offers many advantages over CGI concept. However, servlets connecting to database servers can generate many

connections to the database server and can affect the database performance. WebSphere Application Server Servlets API extensions include connection manager package that helps servlets to management connections to the database server.

This section summarizes IBM Webpshere API extension. Complete documentation can be found on IBM WebSphere Server's Documentation Center, or online at:

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

package com.ibm.servlet.connmgr
Manages pools of reusable connections to data servers, such as an IBM DB2 database. A connection management feature that caches and reuses connections to your JDBC (Java Database Connectivity)-compliant databases. When a servlet needs a database connection, it can get one from the pool of available connections, eliminating the overhead required to open a new connection for each request.

Interface: IBMConnMgrConstants, Task

Class: IBMConnMgr, IBMConnMgrAdmin, IBMConnMgrNLS, IBMConnMgrUtil, IBMConnPool, IBMConnPoolSpec, IBMConnSpec, IBMConnection, IBMJdbcConn, IBMJdbcConnPool, IBMJdbcConnSpec, IBMPoolHashtable, IBMPoolVector, TaskTimer

Exception: IBMConnMgrException

package com.ibm.servlet.personalization.sam
Register servlets with the IBM WebSphere Application Server and its Site Activity Monitor (SAM) facility. SAM lets you view Web site activity, and act or react in real time.

Class: AppPackage

package com.ibm.servlet.personalization.sessiontracking
Configures parameters for session tracking, allowing the application server to group a series of related requests into an identifiable user session.

Class: IBMSessionData, IBMSessionContextImpl, IBMHttpSessionBindingListener, IBMHttpSessionBindingEvent

package com.ibm.servlet.personalization.userprofile
Speciies the information to maintain about the people who visit the Web site.

Class: UserProfile

package com.ibm.servlet.servlets.personalization.util
Contains classes that enable Web administrators to post site-wide bulletins and Web visitors to exchange messages.

Class: CheckMessage, GetMessage, GetVariableText, SendMessage, SetVariableText

6.5 Servlets with JSPs

In the following we discuss how servlets work with JavaServer Pages (JSP).

6.5.1 JavaServer Page (JSP) Overview

IBM WebSphere Application Server supports a powerful dynamic approach to generate page content: JavaServer Pages (JSP).

JSP is a simple way to generate dynamic content using combinations of HTML, NCSA, Server-side JavaBean, <SERVLET> tag, in-line Java and JSP syntax. JSP files look like the standard HTML, XML files, but have .jsp extensions. See Figure 30.

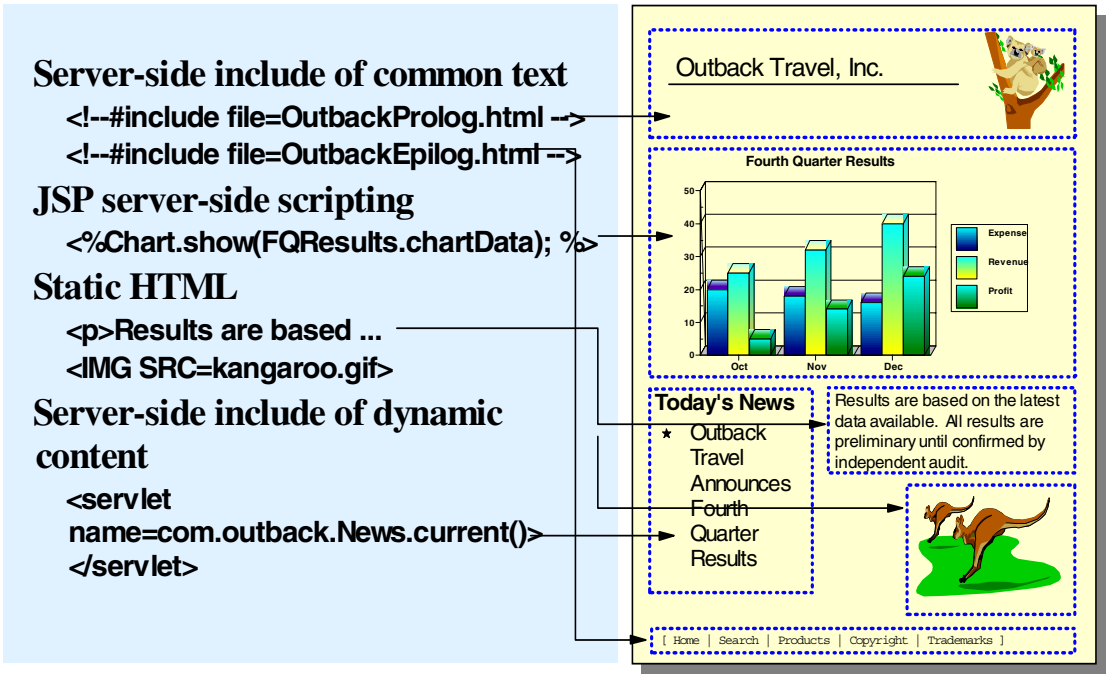


Figure 30. Use JavaServer Page (JSP) to generate dynamic presentation content

6.5.2 Advantages of JSP

An advantage of JSP is that it enables developers to separate business logic from the GUI presentation. JSP can access server-side code such as servlets, JavaBeans, inline Java code and Java-based Web applications. This accessibility allows the JSP developer to reuse components. For example, developers can share or exchange reusable components that perform common, day-to-day operations. Another advantage of JSP is it simplifies the development efforts with tag-based scripts. Because JSP is converted to Servlet on the server-side, Web designers only need to know a little about Java programming. They can concentrate their development efforts on the presentation.

6.5.3 JavaServer Page Specification

There are five categories of JavaServer Page specification: Directive, Declaration, Scriptlet, Expression and Bean tag.

6.5.3.1 Directive tag, `<% @ %>`

Use this tag to:

1. Specify the scripting language being used.
2. Specify the interfaces a servlet implements.
3. Specify the classes a servlet extends.
4. Specify the packages a servlet imports.

There are six directives: language, method, import, content_type, implements and extends.

- **language** - Specifies the scripting language used for the entire jsp file. Currently, "java" is the only valid value. Only the first language directive is evaluated.

  An example: `<%@ language ="java" %>`

- **method** - Specifies the servlet function name where the embedded Java code (see the scriptlet section) will be put in the dynamically constructed servlet (DCS). The scriptlet Java code becomes the body of the specified method name. The default method is service. Only the first method directive is evaluated.

  An example: `<%@ method ="doPost" %>`

- **import** - Specifies the package(s) imported by the DCS. A comma-separated list of Java language package names or class names DCS imports. The import directive can be specified multiple times to import different packages.

  An example: `<%@ import ="java.io.*,java.util.Hashtable" %>`

- **content_type** - Specifies the MIME type of the generated response. The default value is text/html. Only the first content_type directive is evaluated. This directive can be used to specify the character set in which the page is to be encoded.

  An example: `<%@ content_type ="text/html; charset=iso-8859-1" %>`

- **implements** - Specifies the Java interface(s) the DCS will implement. The DCS can specify multiple interfaces using a comma-separated list or multiple implement directives.

  An example: `<%@ implements ="javax.servlet.http.HttpSessionContext" %>`

- **extends** - Specifies the name of the Java language class that the DCS extends. The class must be a valid class and does not have to be a servlet class. Only the first extends directive is evaluated.

  An example: `<%@ extends ="javax.servlet.http.HttpServlet" %>`

6.5.3.2 Declaration tag, <SCRIPT> </SCRIPT>

Use this tag to declare a dynamically constructed servlet's class-wide methods and variables. The general syntax is:

```
<script runat=server>
// declaration of servlet's class-wide methods and variables
</script>
```

The attribute runat=server is needed to indicate that the tag is for server-side processing:

```
<script runat=server>
// class-wide variable
int i = 0;
String foo = "Hello";

//class-wide method
private void foo() {
// code for private method
}
</script>
```

6.5.3.3 Scriptlet tag (inline Java code), <% %>

Use this tag to add inline Java code to the DCS's specified directive method (see directive tag under method). If the directive method is not specified, the default method used to add the inline Java code is the service method.

Inline Java code can use four predefined object instances to access an essential servlet, output and input classes. These are:

1. **request** - The servlet request class as defined by javax.servlet.http.HttpServletRequest
2. **response** - The servlet response class as defined by javax.servlet.http.HttpServletResponse
3. **in** - The servlet input reader class as defined by java.io.BufferedReader
4. **out** - The servlet output writer class as defined by java.io.PrintWriter

An example:

```
<%
foo = request.getParameter("Name");
out.println(foo);
%>
```

6.5.3.4 Expression tag, <%= >

Use this tag to replace Java language expressions with the values of those expressions in the dynamically generated page. Expressions specified within these tags will first be evaluated, then the result will be converted into a string and then substitute the expression tag. For example:

```
<%= foo %>
```

will substitute the value of "foo" in place of the tag.

6.5.3.5 Bean tag[1], <bean> </bean>

Use this tag to access server-side JavaBean, and soon Enterprise JavaBeans. The syntax for the bean tag applies for beans:

1. Created from a serialized file or a class file
2. Referred to from an HTTP session
3. Passed to the page from a servlet

The bean tag syntax is:

```
<BEAN
name="<lookup name>"
varname="<alternate variable name>"
type="<class or interface name>"
introspect="{yes | no}"
beanName="<file name>"
create="{yes | no}"
scope="{request | session}"
> </BEAN>
```

Note

Between <BEAN> and </BEAN> tag, an optional list of <PARAM> tags can be specified. The syntax of the PARAM tag is:

```
<PARAM {beanPropertyName="name" beanPropertyValue="value"}>
```

See the example Bean tag on the next page.

name: Attribute used as a key to identify this bean.

varname: This is an optional attribute used to identify the variable name that will refer to this bean in the current JSP file. If it is not specified, the name of the variable defaults to the name of the bean as specified in the name attribute above.

[1] The <BEAN> tag from the JSP 0.91 specifiication has been renamed <USEBEAN> in the JSP 1.0 specification.

type: Attribute defining the bean's class or interface. This name is used for declaring the bean instance. When not specified the value defaults to the type object.

introspect: When it is not specified, the default value of this attribute is *yes* which will examine all request properties. The appropriate setter methods are called corresponding to the matching properties.

scope: This is an optional attribute which defaults to *request* when it is not specified.

- *request*: This bean is retrieved from the request context. The bean is set as a context in the request by a servlet invoking this dynamic page using JavaServer Page API. If the bean is not part of the request context, then the bean is created and stored in the request context unless the create attribute is no.
- *session*: This bean is reused from the current session if present. If not present and the create attribute is yes, it is created and stored as part of the session.

create: When it is not specified, the default value of this attribute is *yes* which will create the specified bean if not found in the scope. If no is specified and the bean is not found, an error is returned to the client browser.

beanName: This attribute is the name of the serialized file or class file that contains the bean. It is used only when the bean is not present in the scope of the BEAN tag and the value of create is *yes*.

An example BEAN tag:

```
<BEAN name="foobar" type="FooClass" scope="request">
<PARAM fooProperty="fooValue" barProperty="1">
</BEAN>
```

6.5.4 HTML template syntax for variable data

The Application Server HTML template syntax enables you to put variable fields on your HTML page and have your servlets and JavaBeans dynamically replace the variables with values from a database when the page is returned to the browser.

Note

This capability is an IBM extension of JSP to make it easier to reference variable data. The syntax can only be used in JSP files.

The HTML template syntax consists of two tags:

`<INSERT>` tags for embedding variables in an HTML page

`<REPEAT>` tags for repeating a block of HTML tagging that contains the `<INSERT>` tags and the HTML tags for formatting content

These tags are designed to pass intact through HTML authoring tools. Each tag has a corresponding end tag. Each tag is case-insensitive, but some of its attributes are case-sensitive.

The IBM WebSphere Studio makes it easy to develop JSP files that contain the HTML template syntax. Please see `http://www.software.ibm.com` for more information on this product.

6.5.4.1 The basic HTML template sytax

The `<INSERT>` tag is the base tag for specifying variable fields. The general syntax is:

```
<insert requestparm=pvalue requestattr=avalue bean=name
property=property_name(optional_index).subproperty_name(optional_index)
default=value_when_null></insert>
```

Where:

requestparm: The parameter to access within the request object. This attribute is case-sensitive and cannot be used with the Bean and property attributes.

requestattr: The attribute to access within the request object. The attribute would have been set using the setAttribute method. This attribute is case-sensitive and cannot be used with the Bean and property attributes.

bean: The name of the JavaBean declared by a <BEAN> tag within the JSP file.

The value of this attribute is case-sensitive.

When the Bean attribute is specified but the property attribute is not specified, the entire Bean is used in the substitution. For example, if the

Bean is type String and the property is not specified, the value of the string is substituted.

property: The property of the Bean to access for substitution. The value of the attribute is case-sensitive and is the locale-independent name of the property. This attribute cannot be used with the requestparm and requestattr attributes.

default: An optional string to display when the value of the Bean property is null. If the string contains more than one word, the string must be enclosed within a pair of double quotes (for example "HelpDesk number"). The value of this attribute is case-sensitive. If a value is not specified, an empty string is substituted when the value of the property is null.

Some examples of the basic syntax are:

```
<insert bean=userProfile property=username></insert>
<insert requestparm=company default="IBM Corporation"></insert>
<insert requestattr=ceo default="Company CEO"></insert>
<insert bean=userProfile property=lastconnectiondate.month></insert>
```

In most cases, the value of the property attribute will be just the property name. However, you access a property of a property (subproperty) by specifying the full form of the property attribute. The full form also gives you the option to specify an index for indexed properties. The optional index can be a constant (for example 2) or an index. Some examples of using the full form of the property tag:

```
<insert bean=staffQuery property=address(currentAddressIndex)></insert>
<insert bean=shoppingCart property=items(4).price></insert>
<insert bean=fooBean property=foo(2).bat(3).boo.far></insert>
```

6.5.4.2 The alternate HTML template syntax

The HTML standard does not permit embedding HTML tags within HTML tags. Consequently, you cannot embed the `<INSERT>` tag within another.

The HTML tag, for example, is the anchor tag (`<A>`). Instead, use the HTML template alternate syntax.

To use the alternate syntax:

1. Use the `<INSERT>` and `</INSERT>` to enclose the HTML tag in which the substitution is to take place.
2. Specify the Bean and property attributes:

   To specify the Bean and property attributes, use the form:

   ```
   $(bean=b property=p default=d)
   ```

Where *b*, *p*, and *d* are values as described in the basic syntax.

To specify the requestparm attribute, use the form:

```
$(requestparm=r default=d)
```

Where r and d are values as described in the basic syntax.

To specify the requestattr attribute, use the form:

```
$(requestattr=r default=d)
```

Where r and d are values as described in the basic syntax.

Some examples of the alternate HTML template syntax are:

```
<insert>
  <img src=$(bean=productAds property=sale default=default.gif)>
</insert>
<insert>
  <a
href="http://www.myserver.com/map/showmap.cgi?country=$(requestparms=co
untry default=usa)
  &city$(requestparm=city default="Research Triangle Park")&email=
     $(bean=userInfo property=email)>Show map of city</a>
</insert>
```

The syntax of the <REPEAT> tag is:

```
<repeat index=name start=starting_index end=ending_index>
</repeat>
```

Where:

index: An optional name used to identify the index of this repeat block. The value is case-sensitive.

start: An optional starting index value for this repeat block. The default is 0.

end: An optional ending index value for this repeat block. The maximum value is 2,147,483,647. If the value of the end attribute is less than the value of the start attribute, the end attribute is ignored.

The following show how to use the <REPEAT> tag. The examples produce the same output if all indexed properties have 300 or fewer elements. If there are more than 300 elements, Examples 1 and 2 will display all elements, while Example 3 will show only the first 300 elements.

Example 1 shows implicit indexing with the default start and default end index. The bean with the smallest number of indexed properties restricts the number of times the loop will repeat:

```
<table>
<repeat>
  <tr><td><insert bean=serviceLocationsQuery
property=city></insert></tr></td>
  <tr><td><insert bean=serviceLocationsQuery
property=address></insert></tr></td>
  <tr><td><insert bean=serviceLocationsQuery
property=telephone></insert></tr></td>
</repeat>
</table>
```

Example 2 shows indexing, the starting index and ending index:

```
<table>
<repeat index=myIndex start=0 end=2147483647>
  <tr><td><insert bean=serviceLocationsQuery
property=city(myIndex)></insert></tr></td>
  <tr><td><insert bean=serviceLocationsQuery
property=address(myIndex)></insert></tr></td>
  <tr><td><insert bean=serviceLocationsQuery
property=telephone(myIndex)></insert></tr></td>
</repeat>
</table>
```

Example 3 shows explicit indexing and ending index with an implicit starting index. Although the index attribute is specified, the indexed property city can still be implicitly indexed because the (i) is not required:

```
<table>
<repeat index=myIndex end=299>
  <tr><td><insert bean=serviceLocationsQuery
property=city></insert></tr></td>
  <tr><td><insert bean=serviceLocationsQuery
property=address(myIndex)></insert></tr></td>
  <tr><td><insert bean=serviceLocationsQuery
property=telephone(myIndex)></insert></tr></td>
</repeat>
</table>
```

You can nest <REPEAT> blocks. Each block is separately indexed. This capability is useful for interleaving properties on two beans, or properties that have subproperties. In the example, two <REPEAT> blocks are nested to display a list of songs on each compact disc in a user's shopping cart:

```
<repeat index=cdindex>
  <h1><insert bean=shoppingCart property=cds.title></insert></h1>
  <table>
  <repeat>
```

```
    <tr><td><insert bean=shoppingCart
property=cds(cdindex).playlist></insert>
    </td></tr>
  </table>
  </repeat>
</repeat>
```

6.5.5 JavaServer Page API

Two interfaces support the JSP technology. These APIs provide a way to separate business logic from a GUI presentation (for example, an HTML Web page design). The interfaces that support JSP are:

- **com.sun.server.http.HttpServiceRequest**

  This class implements the javax.servlet.http.HttpServletRequest interface and a setAttribute() method to set attributes defined by name.

- **com.sun.server.http.HttpServiceResponse**

  This class implements the javax.servlet.http.HttpServletResponse interface and adds a callPage() method enabling servlets to call JSP files and optionally pass a context.

6.5.5.1 callPage() method

Use the callPage() method to serve a JSP from within a servlet. The served page (a JSP file) is returned as the response to the browser. The calling servlet can also pass some context using the request object. You should code the header of the served page to include a directive to tell the browser not to cache the file.

The syntax of the `callPage()` method is:

```
public void callPage(String fileName, HttpServletRequest req) throws
ServletException, IOException
```

Where:

`fileName` is the name of the URL that identifies the file that will be used to generate the output and present the content. If the file name begins with a slash (/), the file location is assumed to be relative to the document root. If the file name does not begin with a slash, the location is assumed to be relative to the URL with which the current request was invoked.

The `callPage()` method does not support calling pages with the file extension .html. If you need to invoke HTML pages using the `callPage()` method, you must first rename the HTML files to have the file extension .jsp.

req is the HttpServletRequest object of the servlet that invoked this method. Most often, the content is passed as a Bean in the context of the request object.

To use the `callPage()` method, you must cast the response object to a special Sun object: `com.sun.server.http.HttpServiceResponse`.

Using the `setAttribute()` method to store an attribute in the request context, the syntax is:

```
public void setAttribute(String key, Object o)
```

Where:

key - Is the name of the attribute to be stored

o - Is the context object stored with the key.

To use the `setAttribute()` method, you must cast the request object to a special Sun object: `com.sun.server.http.HttpServiceRequest`. See the sample PopulateBeanServlet for an example of the syntax.

6.5.6 Preventing Web page caching

To prevent Web browsers and proxy servers from caching dynamically generated Web pages (meaning dynamic output that results from processing JSP files, SHTML files, and servlets), use the following code to set headers in the HTTP response:

```
<%
response.setHeader("Pragma", "No-cache");
response.setDateHeader("Expires", 0);
response.setHeader("Cache-Control", "no-cache");
%>
```

Setting the HTTP headers is a more effective method of controlling browser caching than using the `<META>` tag equivalents. For example, `<META HTTP-EQUIV="Pragma" CONTENT="No-cache">` is the equivalent of the first HTTP header setting. Setting the HTTP headers is the recommended method because of one of the following reasons:

Some browsers do not treat the `<META>` tags in the same way as the equivalent HTTP header settings.

On some browsers, the `<META>` tag equivalents do not work when the `callPage()` method is used to load a JSP file that contains the `<META>` tags.

There may be instances when you want to permit a page to be cached, but you do not want the proxy server to permit multiple users to have access to the cached page. For example, suppose your servlet does not use session tracking and it generates a Web page that contains user input. To maintain that personalization, you would want to prevent other users from having access to the cached page. To prevent the proxy server from sharing cached pages, use the following code:

```
<%
response.setHeader("Cache-Control", "private");
%>
```

This header can be combined with the three recommended headers for preventing caching.

6.5.6.1 PopulateBeanServlet.java sample

```
import java.io.*;
import java.beans.Beans;
import javax.servlet.*;
import javax.servlet.http.*;
import DataBean;
/**********************************************************************
* PopulateBeanServlet - This servlet creates an instance of a Bean
* (DataBean), sets several of its parameters, sets the Bean instance
* as an attribute in the request object, and invokes a JSP file to
* format and display the Bean data.
**********************************************************************/
public class PopulateBeanServlet extends HttpServlet
{
   public void Service(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
  {
DataBean dataBean;
// Create an instance of DataBean
try
     {
         dataBean = (DataBean)
Beans.instantiate(this.getClass().getClassLoader(), "DataBean");
      }
      catch (Exception ex)
      {
         throw new ServletException("Can't create BEAN of class DataBean: "
                                    + ex.getMessage());
      }

      // Set some Bean properties (content generation)
```

```
      dataBean.setProp1("Value1");
      dataBean.setProp2("Value2");
      dataBean.setProp3("Value3");

      // To send the Bean to a JSP file for content formatting and display
      // 1) Set the Bean as an attribute in the current request object
      ((com.sun.server.http.HttpServiceRequest)
req).setAttribute("dataBean", dataBean);

     // 2) Use callPage to invoke the JSP file and pass the current request
object
      ((com.sun.server.http.HttpServiceResponse)
res).callPage("/DisplayData.jsp", req);

   }

} /* end of class PopulateBeanServlet */
```

Chapter 7. Servlet programming model

In this chapter we discuss the following topics:

- Issues with CGI scripts and Web server API extensions
- CGI scripts, Web server API extensions and servlets - life cycles
- Summmary of a servlet and its life cycle
- Environment variables in CGI versus servlets
- Servlet threading - reentrancy of servlets
- Servlet programming under a microscope
- Migrating from a CGI base to servlets
- Programming WebSphere's servlet API extensions

7.1 Issues with CGI scripts and Web server API extension

Common Gateway Interface (CGI), has been traditionally exploited in two main ways: CGI scripts and Web server API extention. CGI scripts are interpreted scripts that reside in a /cgi-bin/ directory pointed to by the Web server. Web server API extensions are implemented as a binary executable, usually as a DLL or Shared Object module that gets loaded at Web server startup time and is accessed as a Web server service and configured in the Web server's configuration files. Examples of API extensions include Netscape NSAPI, IBM ICAPI, Apache API, and Microsoft ISAPI.

All CGI applications are in the broadest sense, complex to develop. This is because there is a need for indepth technical knowledge on how to work with parameter passing and in using specific scripting languages. In the case of the API extensions, there are the problems of having to program and compile a server extension module plug-in, using a nonportable language with a different compiled binary for each supported operating system and each supported Web server on each operating system platform. The skills needed to do this are not common. The support quickly becomes a burden and a limiting factor on how portable the solution actually is.

Most CGI Scripts are not portable. A CGI application written for a specific platform will usually only be able to run in that platform environment. A normal CGI Script is brought to life in a memory process that is activated by a client browser request and is destroyed after the client has been served. This causes high startup, memory, and CPU costs and also, multiple clients cannot be served by that same single process when running.

Although API extension modules can be loaded as shared libraries, the parameter passing is still by way of environment variables and there is still an

overhead in servicing each request. A programmer is fully responsible for analyzing and parsing the variables' data stream passed to the API extension module. These modules generally run as part of the Web server process and if badly written and buggy, can take the entire Web server down.

On the other hand, servlets offer all the advantages of Java programs; they are portable and isolated applications and they are comparatively easy to develop. The variables (both system and parameters) are already parsed and handed to the servlet as a tag/data list (see 7.3, "Environment variables in CGI versus Servlets" on page 95) for easy access and processing. Servlets also allow you to generate dynamic portions of HTML pages embedded in static HTML pages using the servlet tag. Servlets provide easy access to a wide range of middleware JavaBeans designed to access all manner of data sources. The range of available middleware JavaBeans is growing constantly and includes interfaces and support for most leading database products.

A further advantage of servlets over CGI script processes is that a servlet is activated by the first client that sends it a request and remains active and ready to service other requests. Servlets can also be set to load and initialize at Server startup time before the first request has been received. There is also the feature called remote servlets, which is when a servlet is requested at one server and this server downloads and activates the servlet dynamically from a remote server, as a secure or nonsecure file.

Once loaded and activated, a servlet continues waiting in the background for further requests and each request generates a new service thread, not an entire process. Because the service process of a servlet is by nature reentrant, multiple clients may be served simultaneously inside the same servlet and typically the servlet process dies only when the Web server is shut down.

Servlets are also conceptually and functionally very similar to Fast CGI. Below is a link to a reference paper that covers the CGI API:

CGI Scripts: `http://hoohoo.ncsa.uiuc.edu/cgi/interface.html`

7.2 CGI scripts, API extensions and servlets - life cycles

The following sections look at CGI scripts, Web server API extensions and servlets in terms of how they are developed and their life cycles.

7.2.1 CGI scripts - life cycle

CGI scripts are written as interpretive code in languages such as perl, tcl, rexx or even vanilla UNIX shell scripts. When the CGI request arrives at the Web server, it forks off a new process to run the script and sets up the environment variables for this process. Included in the variables is any form input as well as information about the requestor such as the browser or its level. Any response from the CGI script is processed by the Web server according to its content. Typically the response will be an HTML stream that will then be sent back to the client and appear in the client's browser window.

The sequence of events for a CGI script are as follows:

1. A client at a browser clicks a link that contains the /bin-cgi/ phrase (positioned immediately after the host name portion of the URL).
2. The Web server intercepts the request and looks in its CGI-BIN directory for the script named in the URL portion following the /cgi-bin/ keyword.
3. A process is forked off to run the interpreter that will process the script.
4. The process is passed a set of environment variables that include both system information and any parameters that were attached to the name of the cgi-bin script command name.
5. The script is interpreted (runs) and in the process of executing, usually will generate an HTML stream mixed with data to return to the client. The HTML stream is constructed on the fly per the design and function of the script.
6. The process terminates and resources are freed.
7. The client sees the resulting HTML stream as yet another Web page or item on a Web page. Page counters are a typical small CGI script item.

7.2.2 API extension - life cycle

API extensions are where the CGI functionallity is built into a binary module that is then added as an extension of a particular Web server on a particular operating system. Being an extension means it can be loaded into memory (on Windows - as a Dynamic Load Library (DLL); on Linux as an SO Shared Object) module, but because it is an extension of the Web server each request runs in the server's address space. The parameter passing technique available is by way of environment variables and the onus is on the API extension programmer to process and parse system variables and to further parse the parameters sent by a client as part of GET/POST requests.

A software company offering a package that uses the API extensions interface has to deal with the logistics of distributing multiple binaries for all the Web servers and operating system combinations they want to support.

With WebSphere and Java, a servlet is the exact same code for all the supported Web servers and operating system platforms. IBM as a middleware provider carries the overhead of integrating WebSphere with the many combinations of Web server and operating system. This leaves customers and Web developers to focus on running solutions and writing business logic.

The sequence of events for an API extension call are as follows:

1. A client at a browser clicks a link that contains a special trigger phrase; an example is /abtwsi/ (IBM's Smalltalk WebConnection API interface).
2. The Web server intercepts the request when it sees the trigger phrase that like the /cgi-bin/ phrase is the first phrase after the host name in a URL. In API extensions the data following the trigger phrase is passed to the API extension service module which interprets it based on what the extension is designed to perform. The process is passed to the environment variables (both system info and GET/POST parameters).
3. Control is passed to the entry point in the API extension module.
4. The extension module executes according to its design and will normally return an HTML stream to the Web server.
5. The process terminates and resources are freed.
6. The client gets the resulting HTML stream and data.

7.2.3 Summary of a servlet

Similar to the way applets run on a browser and extend the browser's capabilities, servlets run on a Java-enabled Web server and extend the server's capabilities.

Servlets are Java programs that use the Java servlet application programming interface (API) and the associated classes and methods. In addition to the Java servlet API, servlets can use Java class packages that extend and add to the API.

Servlets extend server capabilities by creating a framework for providing request and response services over the Web. When a client sends a request to the server, the server can send the request information to a servlet and have the servlet construct the response that the server sends back to the client.

A servlet can be loaded automatically when the Web server is started, or it can be loaded the first time a client requests its services. After loading, a servlet continues to run, waiting for additional client requests.

Servlets perform a wide range of functions. For example, a servlet can:

- Create and return an entire HTML Web page containing dynamic content based on the nature of the client request.
- Create a portion of an HTML Web page (an HTML fragment) that can be embedded in an existing HTML page.
- Communicate with other server resources, including databases and Java-based applications.
- Handle connections with multiple clients, accepting input from and broadcasting results to the multiple clients. A servlet can be a multi-player game server, for example.
- Open a new connection from the server to an applet on the browser and keep the connection open, allowing many data transfers on the single connection. The applet can also initiate a connection between the client browser and the server, allowing the client and server to easily and efficiently carry on a conversation. The communication can be through a custom protocol or through a standard such as IIOP.
- Filter data by MIME type for special processing, such as image conversion and server-side includes (SSI).
- Provide customized processing to any of the server's standard routines. For example, a servlet can modify how a user is authenticated.

7.2.4 Servlet life cycle

The life cycle of a servlet begins when it is loaded into Web server memory and ends when the servlet is terminated or reloaded.

The diagram represented in the following figure is a graphical representation of the servlet life cycle.

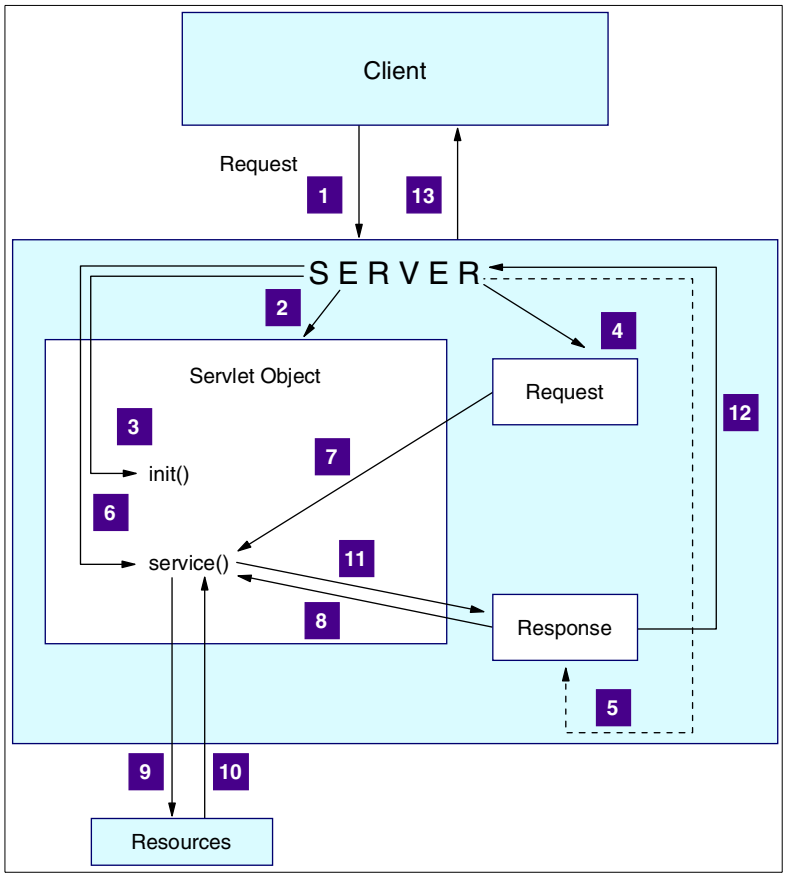


Figure 31. Servlet life cycle

The servlet life cycle can be summarized in the following way:

- The servlet is loaded. This operation is typically performed dynamically, that is, when the first client has access to the servlet 1. However, servers will usually provide an administrative option to force loading and initializing particular servlets when the server starts up.
- The server creates an instance to the servlet 2.
- The server calls the servlet init() method 3.
- A client request arrives at the server 1. A client request is already at the server if the client request initiated the servlet load.
- The server creates a request object 4.
- The server creates a response object 5.

- The server invokes the servlet service() method 6, passing the request 7 and response 8 objects as parameters.
- The service() method gets information about the request object and processes the request accessing the other resources 9 and getting the necessary information 10.
- The service() method uses methods of the response object to pass the response 11 back to the server 12 and then to the client 13. The service() method may invoke other methods to process the request, such as doGet() or doPost() or new methods that the servlet developer wrote.
- For additional client requests, the server creates new request and response objects, again invokes the service() method of the servlet and passes those two objects as parameters to it. Therefore, this loop is repeated, but without the need to recall the init() method. The servlet, in general, is initialized only once.
- When the server no longer needs the servlets (typically when the server is shut down), the server invokes the servlet destroy() method.

7.3 Environment variables in CGI versus Servlets

In a CGI script, the user is is able to issue commands to examine the environment variables passed over as part of the process that was forked to service the CGI request. These variables are documented as part of the CGI-BIN specification put out by the W3 Consortium. A published list of these variables includes:

DOCUMENT_NAME	The complete local directory path of the current document.
DOCUMENT_URI	The local path of the current document referenced to the base directory of the Web space.
QUERY_STRING_ UNESCAPED	The unescaped query string sent by the client browser, all shell-special characters escaped with \.
DATE_LOCAL	The current local date and time.
DATE_GMT	The current Greenwich Mean date and time.
LAST_MODIFIED	The date and time of the last modification of the current document.
REMOTE_ADDR	The IP address of the remote client browser.
QUERY_STRING	The raw query string sent from the remote browser.

SERVER_SOFTWARE	The name of the HTTP server software.
SERVER_NAME	The local computer name of the HTTP server.
GATEWAY_INTERFACE	The name/version of the Common Gateway Interface served on this HTTP server.
SERVER_PROTOCOL	The name/version of HTTP served on this HTTP server.
SERVER_PORT	The IP port om which the HTTP server is answering.
REQUEST_METHOD	The method by which the current document was requested.
PATH_INFO	The extra path info that is sent. This information is regarded as virtual (the path is relative to the base directory of the HTTP server).
PATH_TRANSLATED	The PATH_INFO variable translated from virtual to local (physical) disk location.
SCRIPT_NAME	The virtual path of the script being executed.
REMOTE_HOST	The host name of the remote client.
AUTH_TYPE	The authentication method used to validate the remote client.
REMOTE_USER	The user name used to validate authentication from the remote client. Great for use in password protected sites.
REMOTE_IDENT	The remote user name if supporting RFC 931 identification.
CONTENT_TYPE	The content type of the attached information in the case of a POST or PUT.
CONTENT_LENGTH	The length of the attached information in the case of a POST or PUT.
HTTP_ACCEPT	A comma separated list of mime types that are accepted by the remote browser.
HTTP_USER_AGENT	The name of the remote client browser software.
REFERER	The URL of the HTML document that referred the remote client to this document.
FROM	The name (most likely the mail address) of the remote client user. Unlikely to be set.

FORWARDED	The name of the proxy server through which this document is being processed.
ACCEPT_LANGUAGE	The human languages that are acceptable to the remote client.
HTTP_COOKIE	The cookie sent by the remote client.

The CGI programmer has to write or obtain a set of parsing routines that will extract needed environment variables and in the case of the variable called QUERY_STRING have to further parse the fields within.

All these same variables are passed to a servlet but *not* using environment variables. They are passed as lists of tag/data pairs that a servlet programmer can request by name from the servlet's RequestObject using a simple method call. This saves the programmer a significant effort in dealing with these variables and the QUERY_STRING parameter data.

7.4 Servlet threading - reentrancy of servlets

Servlets servicing HTTP client requests are inherently reentran, having been designed to serve multiple clients concurrently. A servlet programmer can still, using a poor programming technique, corrupt this reentrancy by interacting with a resource such as a data field by altering it while other instances of the same servlet or other servlets can do the same. Typically in such a scenario two instances of the same servlet could be trying to alter the same data at the same time.

If you have a requirement to run only a single instance of a servlet service because it accesses a shared resource that you want to protect, you can program your servlet or call a servlet that handles only one client request at a time. (An alternate technique is to synchronize access to the resource using the synchronize of the threads API.)

To get a servlet to handle only one client service request at a time, have your servlet implement the SingleThreadModel interface in addition to extending the HttpServlet class.

Implementing the SingleThreadModel interface does not involve writing any extra methods. You merely declare that the servlet implements the interface, and the server makes sure that your servlet runs only one service method at a time.

For example, a StockUpdate servlet that alters the available quantity of a stock item, should be held in storage in a way that forces the servlet to control

access to the quantity field. By implementing the SingleThreadModel you will be able to have requests change this field filtered through just the one instance of the servlet. As mentioned earlier, an alternate method is to use the synchronization of the threads API to do the same. So the servlet can either synchronize access to that resource, or it can implement the SingleThreadModel interface. If the servlet implements the interface, the only change in the code is the line below shown in bold:

```
  public class ReceiptServlet extends HttpServlet
               implements SingleThreadModel {

      public void doPost(HttpServletRequest request,
                  HttpServletResponse response)
                  throws ServletException, IOException {
                  ...
       }
       ...
}
```

A servlet using the above can thus be used to control access to any process that should only be done by one active task at any given point in time.

Examples of activities that this could cover include:

- Updating warehouse stock inventory levels
- Altering product price and availability information
- Obtaining an order reference number from a control file

7.5 Programming WebSphere's servlet API extensions

WebSphere comes with a number of very useful extensions to the servlet API. These are designed to further ease the work in programming servlets for specific uses and to help minimize the highoverhead processing associated with such tasks as opening and closing connections to a relational database.

Following this page is code showing how two of the more important APIs are actually used. Before covering them, though, we will point you to the following excellent resource material for getting yourself started on developing e-commerce applications.

WebSphere includes several very detailed examples showing ways you can use these API extensions. If you wish to research the topic in depth and you have installed WebSphere and have it running. Access the following URL:

```
http://<your_server_name>/IBMWebAS/samples/index.html
```

Figure 32 on page 100 shows the samples applications page. This page has links explaining the steps needed to set up WebSphere and to load a DB2 database for the applications to run.

Additional information, above and beyond that in the above samples page, can be found by invoking:

```
http://<your_websphere_server>:9527/
```

Once Admin has started, select the **WebSphere Documentation Center**, then select **Resources**, then select **Samples**. The source code for a number of samples is available. Also there are detailed descriptions for using the APIs and functions.

Both the above resources are highly recommended as starting points.

Figure 32. The WebSphere sample servlets page

7.5.1 DB connection pooling

WebSphere comes with a database connection pooling facility that can be used by programmers to manage connections to a database. This is an API that contains methods establishing a pool of open connections to a given database and for requesting a connection from the managed pool plus for returning them to the pool. The pool of connections for a given database is managed through the WebSphere admin facility.

The reason that WebSphere offers connection pooling is that there is a high overhead to opening and closing a connection to the database. So by adding a connection manager to WebSphere, the programmer only has to call the method that returns an open connection, then when the servlet is finished the still open connection is returned to the connection pool.

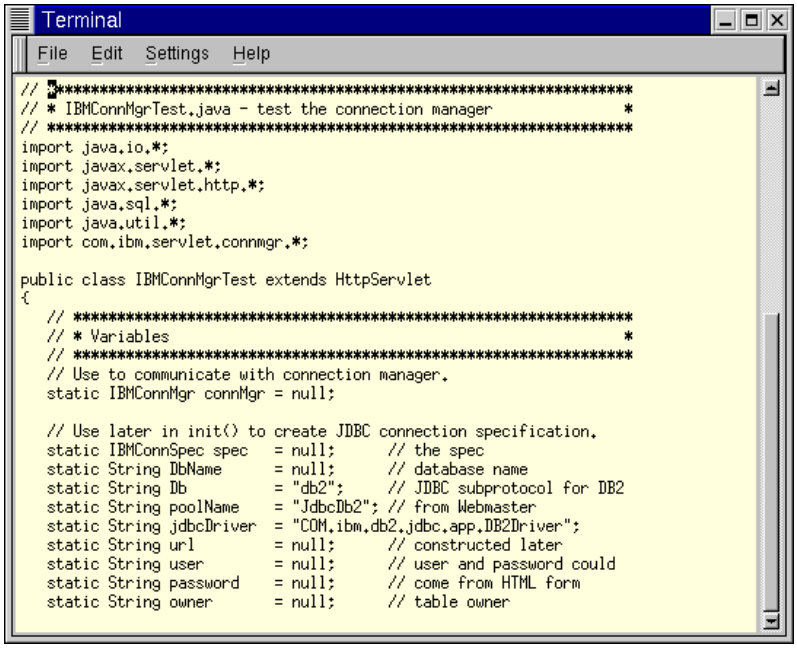


Figure 33. Declarations used with IBMConnMgr class

Figure 33 shows the code used in a servlet that will set up an IBMConnMgr object to manage a pool of threads. The declarations are all static because these are class variables shared by all instances of this servlet. The variables are only used in the init() section of the servlet, but the service methods are able to make requests to the connection manager object that gets instantiated in the init() code.

Figure 34 shows the code in init() that creates the connection pool which can only be done if the servlet already knows which database it is going to use. A servlet that does not know this information cannot really take advantage of the connection pool manager because a pool will normally only be of use if established during the init() part of a servlet's life.

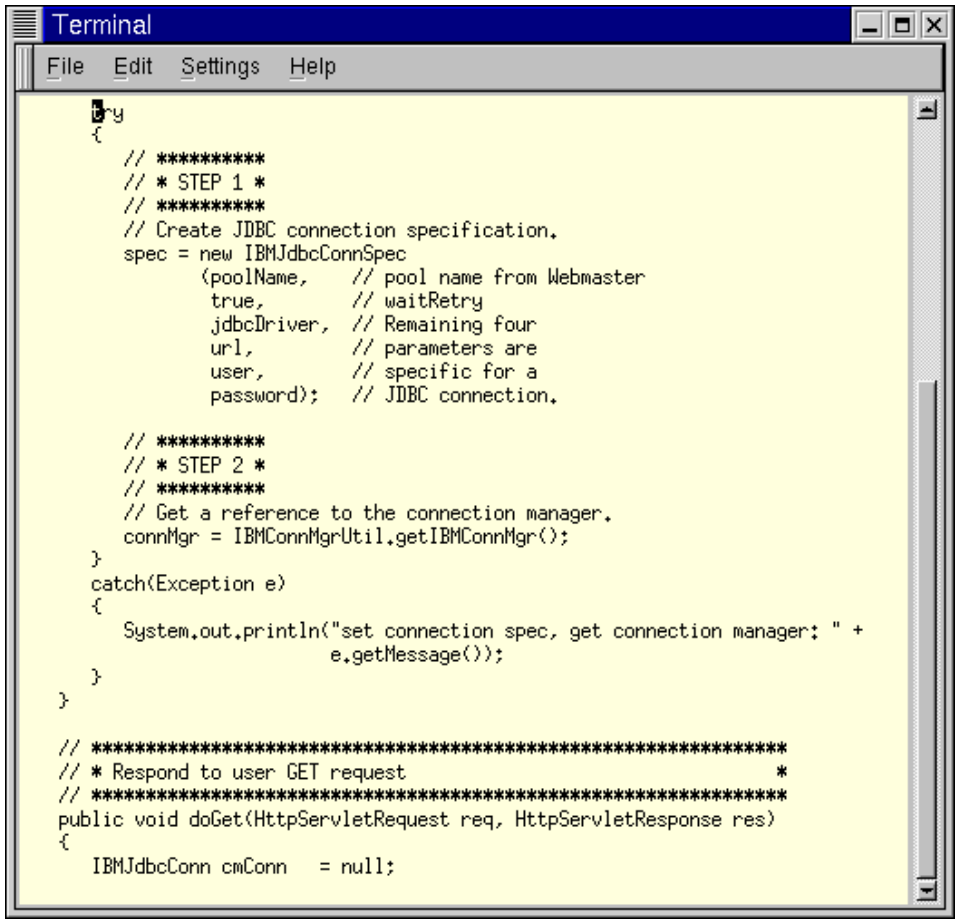


Figure 34. The init() code used to create the connection pool

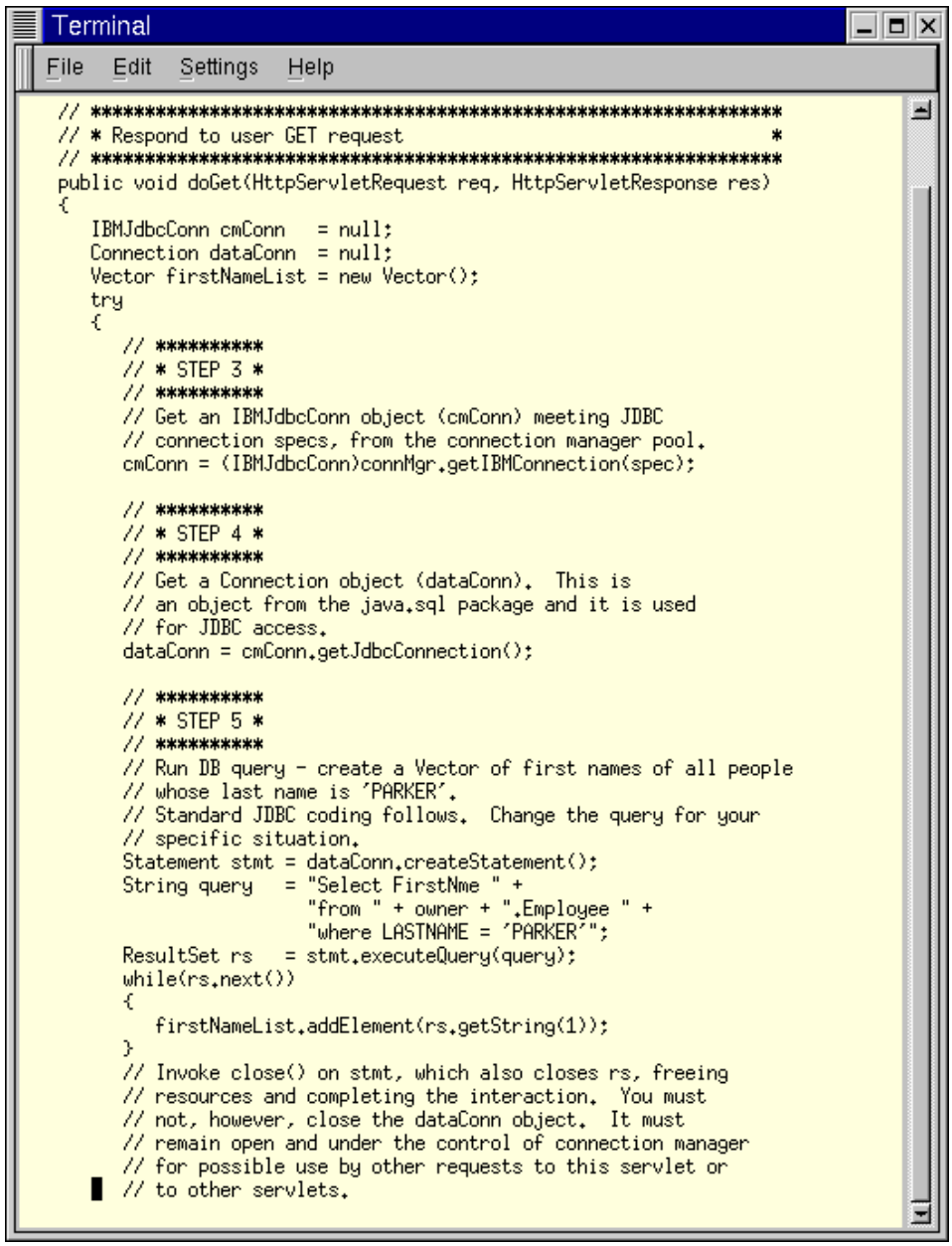


Figure 35. Hhow the servlet service (doGet() requests a connection from the pool

Figure 35 shows the service cycle of the servlet where the doGet() method is called. A connection is requested from the connection pool, and an SQL

statement is opened using the connection. A result set gets returned and once the data is processed the service is completed and the connection returns by default to the connection pool. The connection manager does not expect the servlet to notify it that it has finished with the connection. This is deduced by the fact that the instance of the servlet has ended.

Figure 36 shows the code in the *finally* method that gets invoked when either the servlet is unloaded, or replaced, or WebSphere is about to terminate.

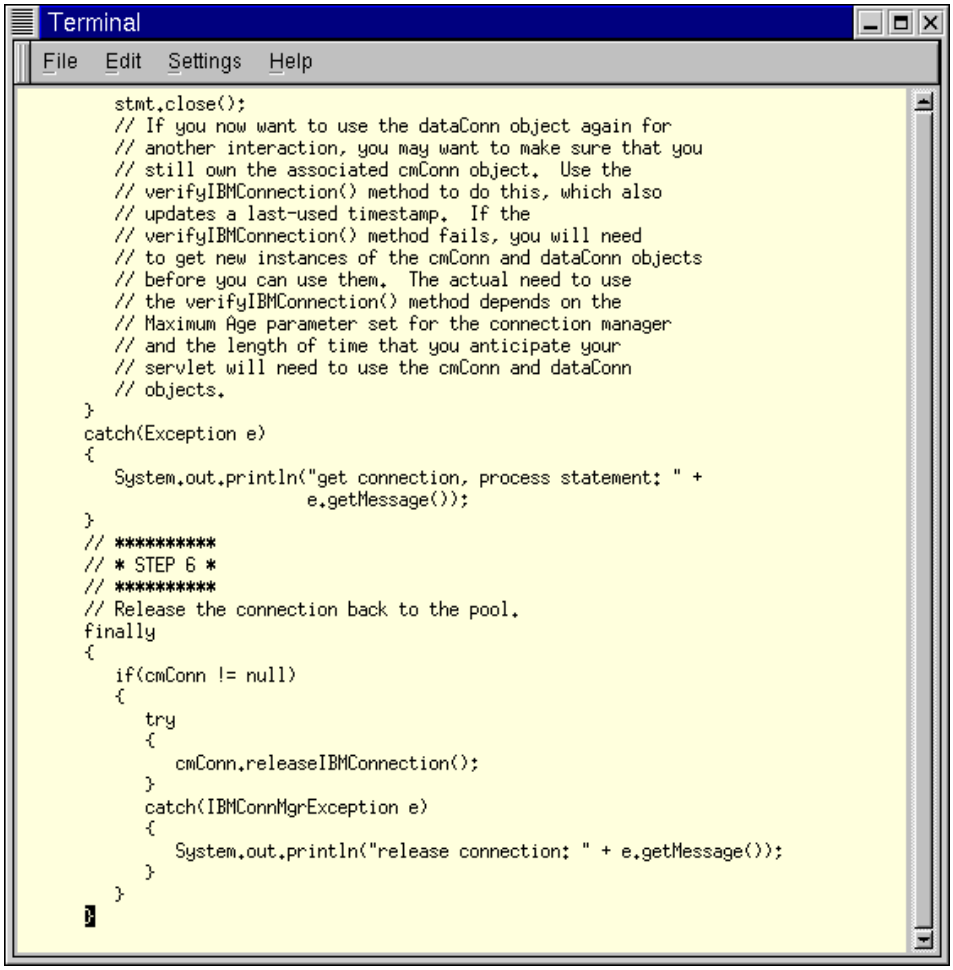


Figure 36. The code that terminates the connection pool

7.5.2 Session management

Figure 37 is sample code from WebSphere online documentation showing a simple session tracking example:

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public
class SessionSample  extends HttpServlet {
public void doGet (HttpServletRequest request, HttpServletResponse
response)
       throws ServletException, IOException
  {
    // Step 1: Get the Session object
      boolean create = true;
      HttpSession session = request.getSession(create);
   // Step 2: Get the session data value
     Integer ival = (Integer)
       session.getValue ("sessiontest.counter");
     if (ival == null) ival = new Integer (1);
     else ival = new Integer (ival.intValue () + 1);
     session.putValue ("sessiontest.counter", ival);
   // Step 3: Output the page
     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     out.println("<html>");
     out.println("<head><title>Session Tracking
Test</title></head>");
     out.println("<body>");
     out.println("<h1>Session Tracking Test</h1>");
     out.println ("You have hit this page " + ival + " times" +
"<br>");
     out.println ("Your " + request.getHeader("Cookie"));
     out.println("</body></html>");
  }
}
```

Figure 37. Sample session code

The example picks up the counter cookie for a client session and increments it. What information you want to keep and how you use it are very much up to how you want your application to work.

There are times when it is useful to save session information in the servlet rather than in cookies on the client's computer. Some people are almost paranoid about cookies being placed on their computers. One approach to keeping session information in the servlet is by declaring a class variable Vector (as static) in the servlet class declarations area, and also declaring a class instance vairable Hashtable (not static), both set to null.

Example code follows (this is only sample code and does not include all the normal includes and other instructions):

```
import java.util.*;  // includes Vector, Hashtable, Date etc:

/*********************************
 *
 * SampleSessionSavingServlet
 * Doug Marker - Aug 1999
 */.

import java.util.*   // include Vectors & Hashtables in code

public class SampleSessionSavingServlet {

    private static Vector savedSessionsStack = null;

    private Hashtable savedSessionValues = null;
    private HttpSession session = null;

}  // end of SampleSessionSavingServlet
```

Then in the init() code area, instantiate the Vector by creating an instance of one. Doing the above makes it a shared resource among all the instances of this particular servlet.

```
public void init(ServletConfig swervletConfig) {
    super.init(servletConfig);

    savedSessionStack = new Vector;   // now create the Vector

    <add your code to make any DB connection or create a pool>

}    // end of init().
```

If is important to remember that while it is okay for each instance of a servlet to read the data from a shared resource such as the Vector just declared, it is critical to use synchronization when adding objects (such as the session

information in the Hashtable) to a shared resource like the Vector. This is to avoid it becoming corrupted. See the following code segments.

```
public void doGet(HttpServletRequest req,
                  HttpServletResponse res)
       throws ServletException IOException {

    String sessionId = locateSessionObject(req);


}  // end of doGet().

/********************
 *
  */
public String locateSessionId(HttpServletRequest req) {

    session = req.getSession(true); // instance variable
    return session.getId();
}
```

The above routines are coded as separate methods. This can be done this way as the instance variables are coded in the class definition and thus can be referred to across methods. If they were not declared that way, each routine would have to pass the variables in its method signature. In fact the routine below does exactly this because the variables sessionId, aPassedObj1 and aPassedObj2 were declared in the doGet() method and thus have to be passed in the method call signature:

```
private synchronized  void addEntryToSavedSessionStack(String
    sessionId, Object1 aPassedObj1, Object2 aPassedObj2)
{
    savedSessionData = new Hashtable(); // create Hashtabble.
    savedSessionData.put("id", sessionId),
    savedSessionData.put("obj1", aPassedObj1);
    savedSessionData.put("obj2", aPassedObj2);
    savedSessionData.put("date", new java.util.Date());
    savedSessionValues.addElement(savedSessionData);
}
```

7.6 Servlet programmming under a microscope

The following section examines in more detail the various classes available to write servlets.

7.6.1 Using GenericServlet class versus HttpServlet class

One point that needs understanding early in your learning cycle is the difference between GenericServlet class and HttpServlet class.

Put very simply, GenericServlet class allows a programmer to write servlets that do not take HTML input or send HTML output. There are functions that a programmer may want to write in a servlet that access other forms of data or get passed objects to be worked on. The servlet programmer only has to write a service() method that gets invoked when a GenericServlet is called.

HTMLServlet is an extension of GenericServlet and replaces the service() method with the doGet() and doPut() methods. The HTMLServlet adds method calls to assist the servlet programmer in accessing HTML input and in sending HTML output. So it is clear that one type of servlet is for non-HTML related processing while the other is specialized for HTML processing.

7.6.2 GET/POST processing in servlets

With a servlet, the same servlet can handle both GET and POST methods when receiving parameters from a client's browser request, but if you want to handle both GET and POST methods by submitting a form and using normal CGI scripts, you would need two different scripts or two different code branches with some conditional logic. With servlets you can switch between GET and POST methods while requesting pages without altering any code in the servlet that handles and responds to the request.

Figure 38 and Figure 39 show how a servlet can discover information about the server, the client environment, and all the data sent by the client. They also describe loops of a typical structure in servlets.

```
/**************************************************************************
 *
 *    FormInfo Servlet
 *
 */
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
&rbl.
public class FormInfo extends HttpServlet
{
   public void service(HttpServletRequest request, HttpServletResponse r
      throws ServletException, IOException
   {
      ServletOutputStream out = response.getOutputStream();
      response.setContentType("text/html");

      out.println("<html><head><title>Form Information</title></head>");
      out.println("<body><h1>Form Information</h1>");
      out.println("HTML Method: " + request.getMethod() + "<br>");
      out.println("URI: " + request.getRequestURI() + "<br>");
      out.println("Protocol: " + request.getProtocol() + "<br>");
      out.println("Servlet Path: " + request.getServletPath() + "<br>");
      out.println("Path Info: " + request.getPathInfo() + "<br>");
      out.println("Path Translated: "+request.getPathTranslated()+"<br>");
      out.println("Query String: " + request.getQueryString() + "<br>");
      out.println("Content Length: "+request.getContentLength()+"<br>");
      out.println("Content Type: " + request.getContentType() + "<br>");
      out.println("Server name: " + request.getServerName() + "<br>");
      out.println("Server port: " + request.getServerPort() + "<br>");
      out.println("Remote User: " + request.getRemoteUser() + "<br>");
      out.println("Remote Address: "+request.getRemoteAddr()+"<br>");
      out.println("Remote Host: " + request.getRemoteHost() + "<br>");
      out.println("Authentication Scheme: "+request.getAuthType()+"<br>");
      out.println("<p>");

      Enumeration names = request.getHeaderNames();
      while (names.hasMoreElements())
      {
         String header = (String) names.nextElement();
         out.println(header + ": " + request.getHeader(header) + "<br>")
      }
      out.println("<p>");
```

Figure 38. Sample form servlet (part 1 of 2)

```
        names = request.getParameterNames();
        while (names.hasMoreElements())
        {
           String key, value;
           key = (String)names.nextElement();
           value = request.getParameter(key);
           out.println("KEY: " + key + " VALUE: " + value + "<br>");
        }
        out.println("<p>");
        while (names.hasMoreElements())
        {
           String key = (String)names.nextElement();
           String [] values = (String []) request.getParameterValues(key);
           if (values != null)
           {
              out.println("KEY: " + key + "VALUES: ");
              for (int i = 0; i < values.length; i++)
                 out.println(values[i] + "     ");
              out.println("<br>");
           }
        }
      out.close();
     }
  }
```

Figure 39. Sample form servlet (part 2 of 2)

This example demonstrates how you can handle very complex data in a servlet. This servlet parses an arbitrary form request and echoes the data back in an HTML page, so it can be very useful for testing forms. In addition, it uses several HttpServletRequest methods to get information:

- getMethod() returns the method used to submit the request.
- getRequestURI() returns the URI that was requested.
- getProtocol() returns the protocol and the version of the request.
- getServletPath() returns the part of the request URI that refers to the servlet being invoked.
- getPathInfo() returns optional extra path information following the servlet path, but immediately preceding the query string. It returns the value null if not specified.
- getPathTranslated() returns extra path information translated into a real path and returns null if no extra path information was specified.
- getQueryString() returns the query string part of the servlet URI or null if none.
- getContentLength() returns the size of the request entity data or -1 if not known.

- getContentType() returns the Internet Media Type of the request entity data or null if not known.
- getServerName() returns the host name of the server that received the request.
- getServerPort() returns the port number of the port on which the request was received.
- getRemoteUser() returns the name of the user making the request or null if not known.
- getRemoteAddr() returns the IP address of the agent that sent the request.
- getRemoteHost() returns the fully qualified host name of the agent that sent the request.
- getAuthType() returns the authentication scheme of the request or null if not known.
- getHeaderNames() returns an enumeration of strings representing the header names for this request. By passing each name to the getHeader() method, you can retrieve the value of the corresponding header or null if not known.
- getParameterNames() returns the parameter names for this request as an enumeration of string or an empty enumeration if there are no parameters or the input stream is empty. By passing each name to the getParameter() method, you can retrieve the lone value of the specified parameter or null if the parameter does not exist. For example, it is the getParameter() method that retrieves the value of the age parameter. If the parameter has or could have more values, the getParameterValues() method returns the values of the specified parameter or null if the named parameter does not exist. For example, it is the getParameterValues() method that retrieves the values of the VisitedCity parameter.

There is one more piece of code not yet explained. This piece of code was not necessary in the example in Figure 38 on page 109 because we knew all of the names of the parameters we were expecting to send. The example we describe now is more general, because it can be used to test forms that have names of parameters that we don't know in advance. That is the reason why we parse all the parameters with an enumeration:

```
Enumeration names = request.getParameterNames();
```

We can now use the hasMoreElements() method to test for the end of the parameter list:

```
while (names.hasMoreElements())
   {
      ...
   }
```

Inside the while cycle, we can retrieve the next parameter name or key from the nextElement() method of enumeration:

```
String key, value;
key = (String)names.nextElement();
```

Given the key string, we can look up the parameter value from the response argument to the servlet with the getParameter() method:

```
value = request.getParameter(key);
```

Now we can do whatever processing or output we like with key and value:

```
out.println("KEY: " + key + " VALUE: " + value + "<br>");
```

We see frequent loops of this structure in servlets:

```
while (names.hasMoreElements())
{
   String key, value;
   key = (String)names.nextElement();
   value = request.getParameter(key);
   out.println("KEY: " + key + " VALUE: " + value + "<br>");
}
```

Figure 40. Loops of a typical structure in servlets

7.6.3 The init(), service(), and destroy() methods

Notice that both the classes GenericServlet and HttpServlet are abstract, that is, they contain methods that have not been completed. It is up to the subclasses of the abstract class to override at least one method. You can recognize a typical Java application because of the presence of the main() method. It is automatically executed when the application is loaded. Java applets are event-driven. This is done with a series of methods created by inheritance from the applet classes:

- init()

- start()
- stop()
- destroy()
- paint()
- repaint()

Servlets look like ordinary Java programs. How do you recognize them? First of all they must implement the servlet interface, usually by extending either the GenericServlet class or the HttpServlet class. Both the GenericServlet and the HttpServlet classes contain three methods, which they take by inheritance from the servlet interface:

1. init()
2. service()
3. destroy()

These methods are used by the servlet to communicate with the server. As has been said, these three methods are called life cycle methods. You will work with these three methods in a slightly different way, depending on whether you are extending the GenericServlet class or the HttpServlet class.

However, the simplest possible servlet defines the single service() life cycle method. In other words, the service() method is required and the others are optional.

```
import java.io.*
import javax.servlet.*;

public class SimplestServlet extends GenericServlet
{
   public void service(ServletRequest request, ServletResponse response)
      throws ServletException, IOException
   {
      PrintStream out = new PrintStream(res.getOutputStream());
      out.println("Hello World!");
   }

   public String getServletInfo()
   {
      return "This servlet defines the single service() life cycle method";
   }
}
```

Figure 41. The simplest possible servlet contains a single life cycle method, service()

It would be helpful to have a general description of each of them before moving on to the next section. Notice that this servlet also presents the getServletInfo() method for the servlet interface. This method is not really required, which means that this example could be made even simpler. We just want to show the use of this method in order to return general information about a servlet, for example, its author.

The init() and the destroy() methods have the same properties for the GenericServlet class or the HttpServlet class. Descriptions of these methods follow:

- The init() method is run only once when the server loads the servlet and the servlet is started. It is guaranteed to finish before any service() requests are accepted. The servlet can be activated when the server starts or when the first client accesses the servlet. The biggest advantage is that the init() method is called only once, without considering how many clients access the servlet. The default init() method logs the servlet initialization and it is possible to configure it in order to save other information.

  The default init() method can usually be accepted as it is, without the need to override it, because it is not abstract. Servlet developers may, if they want, provide their own implementation of this method, overriding it and creating a custom init(). A custom init() is typically used to perform setup of servlet-wide resources only once, rather than once per request. For example, you might want to write a custom init() to load GIF images one time only, where the servlet returns the images multiple times in response to multiple client requests to the servlet. Further examples may be initializing sessions with other network services or getting access to their persistent data (stored in a database or in a file).

Note

When you write your own code overriding

```
public void init(ServletConfig config) throws ServletException
```

you should always invoke `super.init(config)`. In fact, the init() method is called automatically, by default, by the network service each time it loads the servlet. The most important function the init() method provides when it is called is to return the config object for use to the servlet. If you override it without calling the super.init() method, this function will no longer be provided and the config object will no longer be initialized by the server.

- The destroy() method is run only once when the server stops the servlet and unloads it. The servlet-log file gives information about all the initialized and destroyed methods and this is a way to see if, for example, a servlet is unloaded. Moreover, the servlet-log file offers a good way to debug.

  Usually, the servlets are unloaded when the server is shut down. The default destroy() method also can be accepted as is, without the need to override it, because it is not abstract like the init() method. Servlet writers may, if they wish, override the destroy call, providing their own custom destroy() method. A custom destroy() method is often used in the management of servlet-wide resources. For example, the server might accumulate data when it is running and you might want to save this data to a file when the servlet is stopped.

The service() method is the heart of the servlet. In fact, as we said, the simplest possible servlet defines only the service() method. Unlike the init() and destroy() methods, it is called for each client request, and not only one time in the life cycle of the servlet. Moreover, it must be handled differently when it is based on the GenericServlet class or HttpServlet class.

If the servlet is based on the GenericServlet class, the service method is abstract, so you *must* override it. The service() method obtains information about the client request, prepares the response, and returns this response to the client. You should also consider that multiple clients might have access to the service() method at the same time, so you also have to include threads and synchronized code.

If the servlet is based on the HttpServlet class, the service method is not abstract. Therefore, you can accept it if you are supporting the HTTP 1.0 protocol. The service() method determines whether the information entered onto the form on the client's browser is provided by GET or POST. If the answer is GET, the service() method calls the doGet() method of the HttpServlet class. If the answer is POST, the service() method calls the doPost() method of the HttpServlet class. It is up to the servlet developer to override the doGet() or doPost() methods in order to accept the client's request and give a response back to it, as appropriate. If you are supporting the HTTP 1.1 protocol, then you will probably override the service() method to support HTML 1.1 extensions (OPTIONS, PUT, DELETE, and TRACE). Calling super.service() from within the overridden service method provides the default handling on the other methods (such as GET and POST).

Note

When writing your servlets, you can write your own code overriding the following servlet class methods:

- init()
- service()
- destroy()
- getServletInfo()
- doGet()
- doPost()

All other Java servlet API methods are implemented by the WebSphere for processing. You should make sure that you do not override any other Java servlet API methods or you will be overriding the server's code.

In addition you should never call the java.lang.System.exit() method, because it terminates the JVM currently running and servlets will no longer run until your Java-enabled Web server is restarted.

7.6.4 Parameters passed by the server

The service() method, as we said, is the heart of a servlet. It is through the service() method that the server and servlet can exchange data. In fact, when the server invokes the servlet service() method, it also passes two objects as parameters.

- If the servlet is based on the GenericServlet class, the two objects are instances of:
  - ServletRequest
  - ServletResponse
- If the servlet is based on the HttpServlet class, the two objects are instances of:
  - HttpServletRequest
  - HttpServletResponse

These objects, let us call them request and response for convenience, encapsulate the data sent by the client, providing access to parameters and allowing the servlets to report status including errors if they occurred. You can decide to write to System.out and System.err using the println() method and you can decide to write in the servlet-log file using the log() method.

The server creates an instance for the request and response objects and passes them to the servlet. Both these objects are used by the server to exchange data with the servlet.

The servlet invokes methods from the request object in order to discover information about the client environment, the server environment, and all the information provided by the client; for example, all the data entered on a form on the client's browser and set by the GET and POST methods. The specific methods of the request object that the servlet uses to retrieve information from the client are:

- getParameterNames()
- getParameter()
- getParameterValues()

The servlet invokes methods for the response object to send the response that it has prepared, back to the client. The primary method of the response object that the servlet uses to send the response back to the client is the getOutputStream() method. This method returns a ServletOutputStream object and you may use the print() and println() methods of this object for writing the servlet response back to the client.

The input stream that the servlet gets using the getInputStream() method and the output stream that the servlet gets using the getOutputStream() method may be used with data in whatever format is appropriate. For example, HTML and several image formats may be valid data formats.

7.7 Migrating from a CGI base to servlets

There are many reasons why people want to move off CGI scripts and Fast-CGI applications. These boil down to portability, scalability, ease of support, and skill.

While there is always likely to be a place for CGI scripts, they are not likely to be the basis for any serious scalable e-commerce applications, even though API extensions do scale acceptably.

7.7.1 Migration - decisions criteria

The decision-making issues to consider in migrating from CGI to servlets, include:

1. How important is it to keep improving your CGI functions?
2. Migrating a specific CGI function:

- How many operating system platforms do you have to or want to support?
- How many different Web servers do you have to or want to support?
- Do you care about being locked into minimal Server/OS platforms?

3. Are your company CGI skills concentrated in more than one or two people?
4. How scalable is your existing CGI application?
5. Is the cost of scaling your existing CGI proving a burden?
6. Is the ability to balance client request loads important?
7. Is robust OOT important to your future software direction?

7.7.2 Migration - an approach

In order to migrate CGI applications to servlets, the existing CGI funtionality needs to be documented and then rewritten in Java using the Java servlet API. To do this successfully means reviewing the current Java servlet API for the services it offers (allowing that the API is still expanding), then mapping the old CGI functions to the available Java servlet services. The next step is coding your servlets and testing them.

One of the best tools available for developing your Java servlets is IBM's VisualAge for Java. The new Version 3.0 Professional edition now includes both DB2 access JavaBean generation facilities and servlet generation and testing facilities. These were previously only available in the Enterprise Edition of VisualAge for Java.

Part 3. WebSphere and design patterns for e-commerce

In this part we discuss WebSphere Application Server technology and servlet design patterns for e-commerce.

Chapter 8. WebSphere Application Server technology

In this chapter we illustrate servlet security through administering users, groups, access control list, and resources under the WebSphere Application Server administration. Then, we discuss extending the servlet's functionality using Enterprise JavaBean (EJB) and Extended Markup Language technology.

8.1 WebSphere Application Server security

In the following we will give a comprehensive treatment of IBM WebSphere Application Server security.

IBM WebSphere Application Server consists of a Java-based servlet engine that is independent on both the Web server on which it is installed and the underlying operating system. This way the goal, write once, run everywhere becomes available also for servlet development.

In addition to a servlet engine and plug-ins, WebSphere Application Server provides the following components:

- Implementations of the JavaSoft Java servlet API, plus extensions of and additions to the API.
- Sample applications that demonstrate how to use the basic classes and the extensions.
- The application server manager, a graphical user interface (GUI) that makes it easy to set options for loading local and remote servlets, sets initialization parameters, specifies servlet aliases, creates servlet chains and filters, monitors resources used by the application server, monitor loaded servlets and active servlet sessions, logs servlet messages, and performs other servlet management tasks. This feature will be used several times in this chapter.
- A connection management feature that caches and reuses connections to your Java Database Connectivity (JDBC)-compliant databases. When a servlet needs database connections, it can go to the pool of available connections. This eliminates the overhead required to open a new connection each time.
- Additional Java classes, coded to the JavaBeans specifications, allow programmers to access JDBC-compliant databases. These data access beans provide enhanced function while hiding the complexity of dealing

with relational databases. They can be used in a visual manner in an integrated development environment.

- Support for a new technology for dynamic page content called JavaServer Pages (JSP). JSP files can include any combination of HTML tags, <SERVLET> tags, <INSERT> tags, <BEAN> tags and NCSA tags (special tags that were the first method of implementing the server-side includes).
- CORBA Support: An Object Request Broker (ORB) and a set of services that are compliant with the Common Object Request Broker Architecture (CORBA).

To see all the steps we followed to install and configure WebSphere Application Server correctly, refer to WebSphere Application Server installation and configuration in 5.12, “WebSphere Application Server - installation and configuration” on page 51.

8.1.1 WebSphere Application Server security management

In this section, we consider the security features that WebSphere Application Server has to offer.

Basically, from a security point of view, WebSphere Application Server permits the administrator to restrict access to some or all of the resources that have been installed on the Web server and that have been registered in WebSphere Application Server. Access can be allowed to some of the users based on certificates or passwords. The administrator can create users and groups within realms, and add users to one or more groups. Positive permissions (permit) can be set to users and groups, in the sense that, with the permission model implemented by WebSphere Application Server, the WebSphere Application Server administrator can specify who can access a given resource. As we show later there are certain simple rules as to how conflicting permissions between users and groups are handled.

Figure 42. Login Page for WebSphere Application Server.

We went to the WebSphere Application Server manager page at `http://servername:9090`, typed in the admin password, and logged into the system.

We got the screen shown in Figure 42.

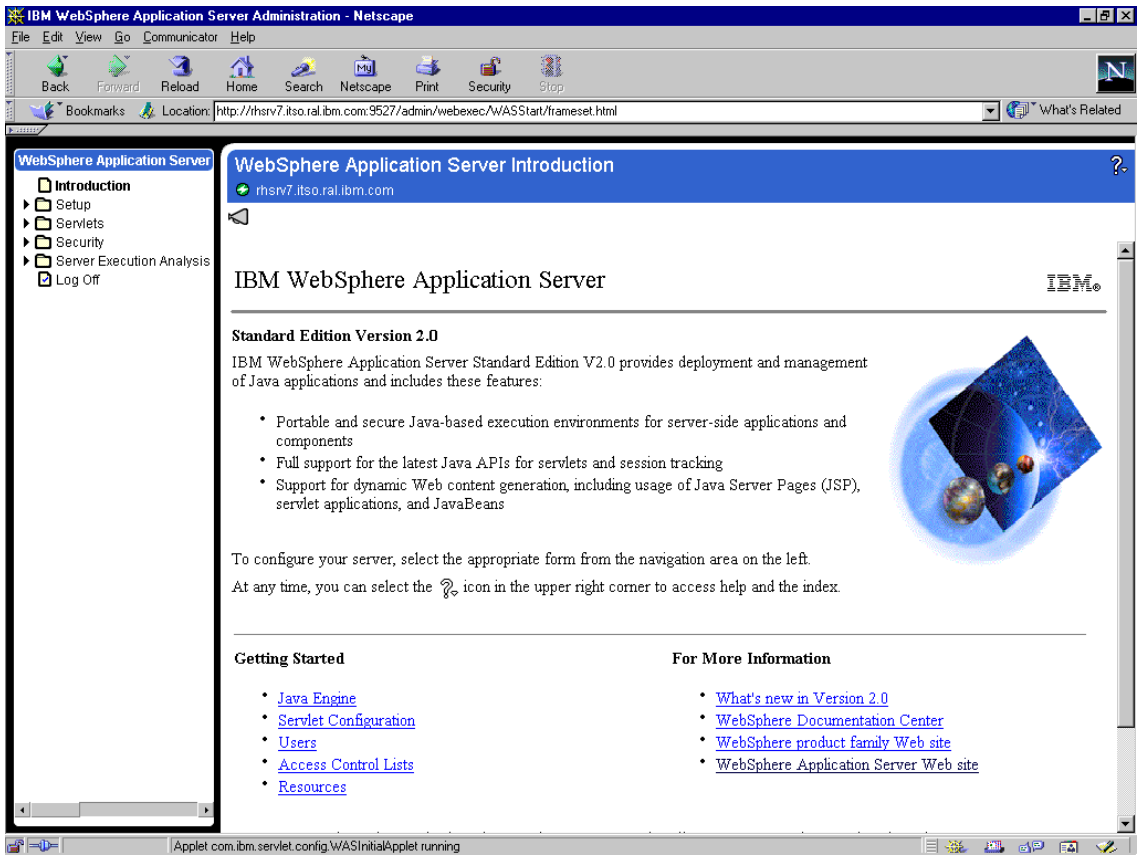


Figure 43. Login page for WebSphere Application Server starts with the Introduction page

Then we clicked the **Security** folder on the left navigation frame.

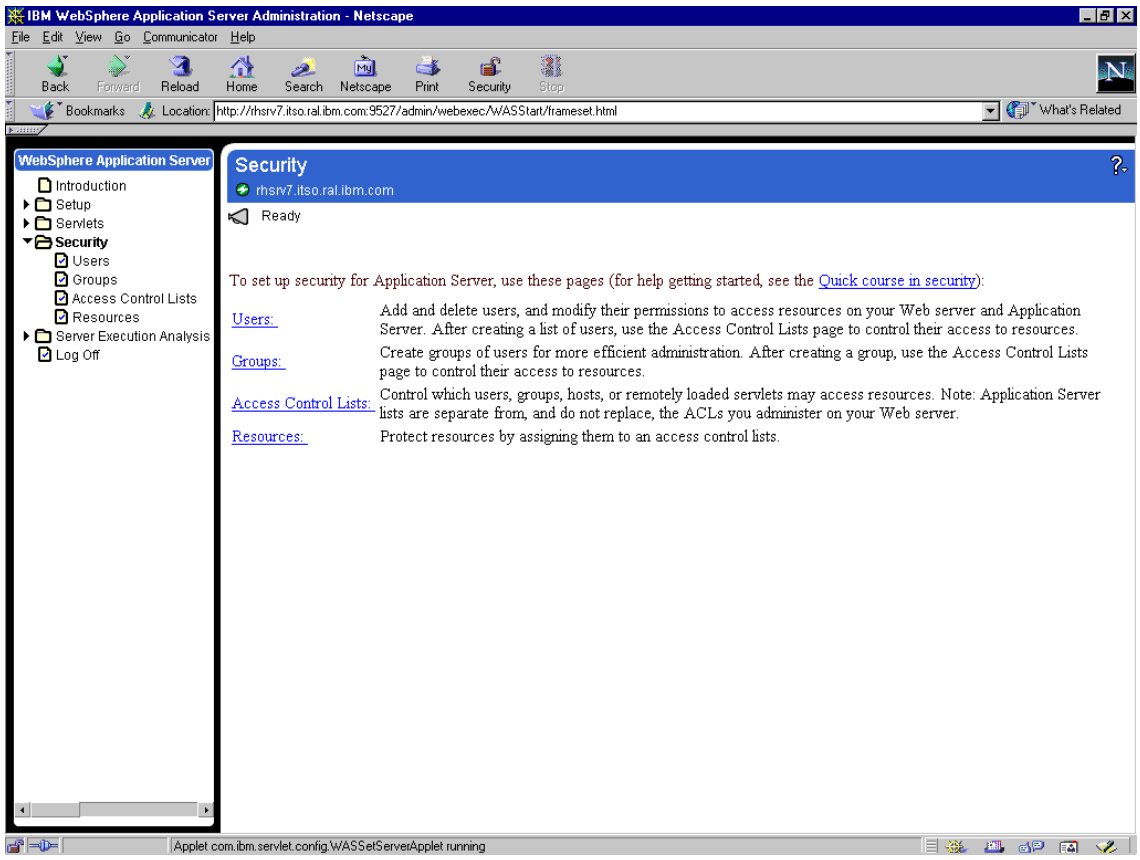


Figure 44. Administering WebSphere Application Server

We have now logged in, and reached the Security page. Let us now understand how the system really functions.

8.1.2 Realms

Realms are used to organize users, groups, and ACLs in a structured way to protect Web resources. In the context of WebSphere Application Server, realms are in particular used for two different purposes: to authenticate a client and to decide which remote servlets to trust. In this section, we will familiarize you with how these are accomplished using the concept of realm.

The system has three realms built into it:

1. defaultRealm
2. servletMgrRealm

3. UNIX

Here is what we can do with these realms:

1. defaultRealm

   The users registered in this realm are given permissions to execute certain servlets. Typically, the system administrator would include, in this realm, the *users* who are expected and permitted to access the servlets on the system, and the *resources* that are to be protected. The administrator can also add groups, and ACLs to facilitate the handling of these.

2. servletMgrRealm

   This realm contains a list of servlet-signers. The system administrator would add into this realm certificates of those signers whose signed servlets would be trusted to run on the server.

3. UNIX realm

   This realm is used to give the users of the system who already have IDs on the server access to the servlets through the Web. The system administrator cannot add or delete any users from this realm. The users in this realm will gain access to the servlets pretty much in the same way as those in defaultRealm, with keying in similar user IDs and passwords. This has been added to save the system administrator the trouble of creating duplicate IDs, and the other to run servlets. The system administrator would add to this realm those resources that the users who have a login into the server can access. Apart from the fact that the system administrator cannot modify the user list, this realm functions pretty much in the same way as the defaultRealm.

We went browsing through the directories to get some specific information on where data is stored about realms. Our WebSphere Application Server was installed in <as_root directory>, and we went to <as_root>/realms, where there was one file corresponding to each realm. We found these files to be text files, capable of being opened by notepad, containing information about the Java class name and the directory associated with this realm. We also saw that the defaultRealm, the servletMgrRealm and the UNIX realms, which we will be working with extensively, have different classes associated with them.

The file name and content for the defaultRealm are given below.

```
# @(#)defaultRealm      1.4 97/09/10
#
# Configuration for the "default" realm, a low-security shared-password
# realm used for demo purposes.
#

classname=com.sun.server.realm.sharedpassword.SharedPasswordRealm
directory=realms/data/defaultRealm
```

Figure 45. D:\WebSphere Application Server\realms\defaultRealm

The defaultRealm is implemented using the class com.sun.server.realm.sharedpassword.SharedPasswordRealm, which according to Sun's documentation, implements a very simple authentication database that stores user passwords in a text file. We confirmed this by going to the data/defaultRealm subdirectory as indicated in the file and opening the file named keyfile. We saw all the users we had created in this realm, and their passwords.

```
Narayan::c3cxNTA0cg==
Tintin::c3cxNTA0cg==
jeeves::amVldmVz
Wooster::c3cxNTA0cg==
Popye::c3cxNTA0cg==
admin::YWRtaW4=
Asterix::c3cxNTA0cg==
pistoia::c3cxNTA0cg==
```

The entries on the left are the user names created in the defaultRealm and the entries on the right are their passwords, encoded, not encrypted, in base64 format. Since the passwords can be easily decoded by any base64-to-binary converter, access to this file should be restricted using the options available in the operating system. Also, we tried hacking into the system, by adding a user name and a base64-encoded password into this file, and then accessing the system as the newly created user. It worked. Hence, this file ought to be protected at all costs by any means available in the operating system. The consolation, however, is that this file is in no way accessible from the Web, and any hacker must not find a way to access this file from the intranet, by logging on to the machine as a user.

The file name and content for the servletMgrRealm are given below.

```
# @(#)servletMgrRealm   1.5 97/09/10
#
# Configuration for the "servletMgr" realm, used to control the privileges
# assigned through the server sandbox.
#

classname=com.sun.server.realm.certificate.CertificateRealm
certclassname=sun.security.x509.X509Cert
directory=realms/data/servletMgrRealm
```

Figure 46. <as_root/realms/servletMgrRealm

The servletMgrRealm is implemented using the class com.sun.server.realm.certificate.CertificateRealm, with the certificate class name sun.security.x509.X509Cert. This will provide access to users based on their certificate, to be enrolled with it. Hence, users in this realm will be identified by their certificates, and can thus access their resources. *Users*, in this case, really means servlet-signers.

The file name and content for the UNIX realm are given below:

```
# @(#)UNIX      1.4 97/09/10
#
# Configuration for the "UNIX" realm, providing access to user accounts
# available through the UNIX getpwent family of calls.  Uses a local
# directory to store ACLs; doesn't support UNIX groups, and currently
# requires some POSIX-standard native code.
#

classname=com.sun.server.realm.unix.UNIXRealm
```

Figure 47. <as_root>/realms/UNIX

The UNIX realms contain, as mentioned before, users that are already present in the system; in other words, those who have logins in the UNIX machine in question. These same logins and passwords can be used to access the services offered by WebSphere Application Server remotely.

Also, if you open the realms directory, you would see yet another realm, called the adminRealm, although this does not show up on the GUI. This is used to store information pertaining to the administrator's user ID and password. We opened the file <as_root>/realms/data/adminRealm/keyfile and we found only one line in this file:

```
admin::YWRtaW4=
```

This line represents the administrator's user ID, admin, followed by the password encoded in base64 format.

8.1.3 Users

The WebSphere Application Server administrator, named admin, can create any number of users in the system. These users are to be created under defaultRealm or servletMgrRealm, but not under UNIX, for the reason mentioned earlier. It is not possible, using the WebSphere Application Server Manager GUI, to add a new user under the adminRealm, so no new administrators can be created.

If the WebSphere Application Server administrator is spoofed, by adding another user ID and password in the file *WebSphere Application Server_root*/realms/data/adminRealm/keyfile, the newly added user can also create a user. We made this experiment: we created another administrator by manually adding a new user name in the keyfile below the adminRealm directory, followed by a password that we encoded in base64 format using again the binary-to-base64 converter. When we logged in the WebSphere Application Server Manager as the new administrator, we were really able to add new users under the defaultRealm and the servletMgrRealm. However, the WebSphere Application Server development team strictly warned us against accessing or modifying these files directly, since these might have unpredictable effects on the functioning of the system.

The following is how to create a new user in the defaultRealm. If a user is created under the defaultRealm, the administrator sets the password for the user, which cannot be changed by the user. This could have some security implications, since it is preferable to change passwords frequently. However, the only way the users can be permitted to change their passwords directly is by having a servlet do the password changing, and this will mean that the file storing the passwords is accessible from the Net. If you so desire, you could write a servlet that takes the user ID and password, takes the new password, encodes it in base64 format, and replaces the old password with the new password in the file *WebSphere Application Server_root*/realms/data/defaultRealm/keyfile. This would obviate the necessity for an administrator's intervention to change the password for those using basic authentication.

Let us now go through the process of creating a user, the way we did it. From the Security options, choose **Users** and from the drop-down Realm list box, choose **defaultRealm**.

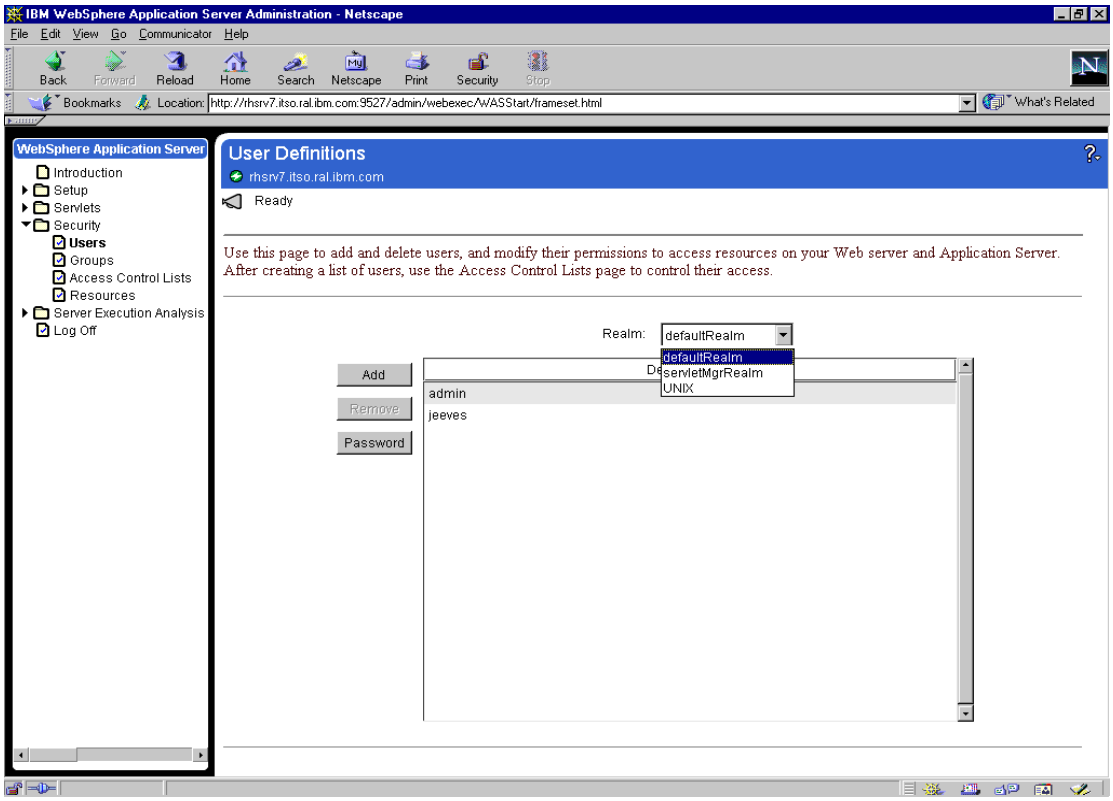


Figure 48. Users

You can see that we have already created an interesting group of users. The way we did it was to simply click the **Add** button at the bottom, fill in the form that is automatically brought up, and click **OK**.

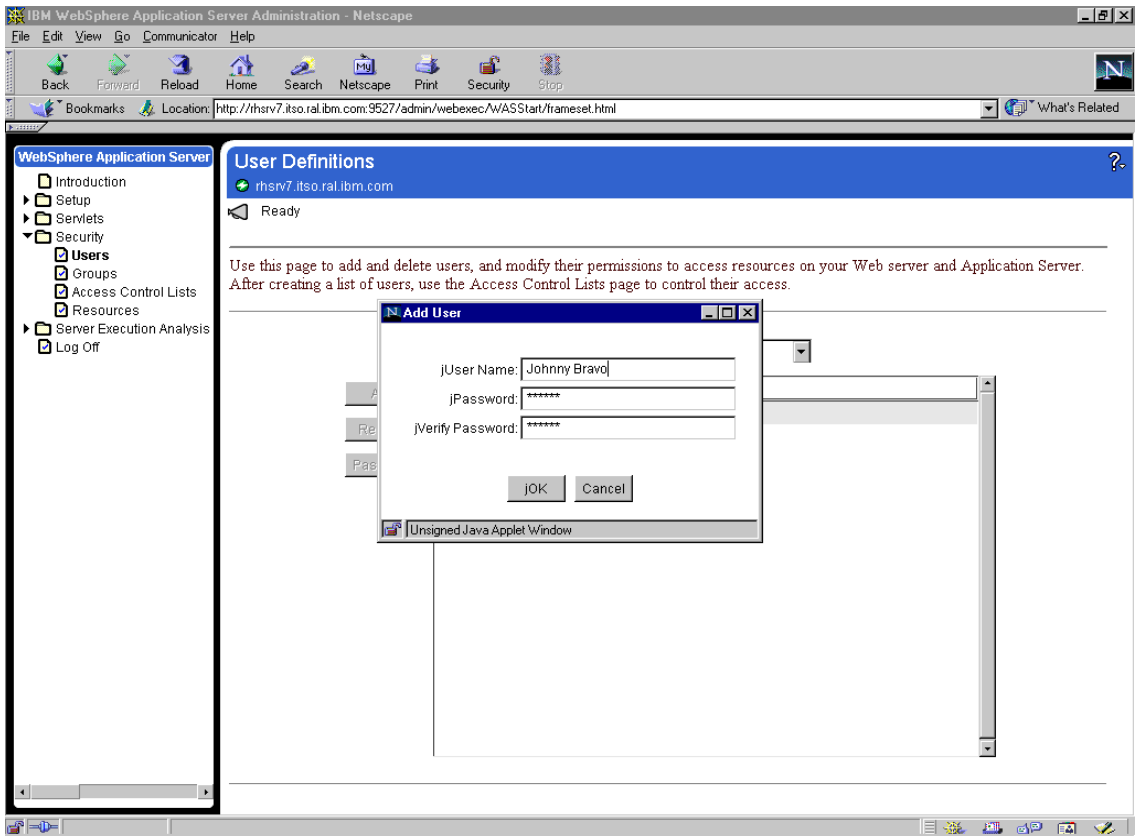


Figure 49. Create User

Creating a user in the servletMgrRealm implies creating a user whose signature on servlets is valid. In later sections, we will explain how servlets can be loaded remotely, and there you will see that it is possible to control access permissions to remotely loaded servlets based on who signed it. For this feature, it is important to enter all the signers in one place first. The users in servletMgrRealm are those whose signatures on the servlet WebSphere Application Server will recognize.

Based on who signed the JAR file containing the servlet, the servlet will be given permissions to perform various actions, such as reading from a file, writing to a file or opening a remote socket. The good thing about this is that the permissions can be controlled finely, in the sense that the administrator can decide the servlets signed by which servletMgrRealm user to be given permissions to read system files, and which ones are to be permitted to write to them, and which ones to be permitted to open remote sockets, etc. Note

however, that all servlets signed by a particular user will have the same permissions.

Servlet-Signer

Throughout this chapter, we use the terms servlet-signer and user in servletMgrRealm interchangeably. These two refer to the same thing, really, since a user in the servletMgrRealm is actually a signer of servlets, as we have explained earlier.

To create a user in the servletMgrRealm, here is what has to be done. To say to WebSphere Application Server to trust all servlets signed by so-and-so to such an extent, we first have to give WebSphere Application Server the public key of the entity to verify the signatures. To register a servlet-signer, you would first have to get the signer to sign a JAR file, put it on a Web server, and give the URL of that JAR file at the time of registering the user. Notice that we did not say that the signed JAR file must contain a servlet class. In fact we were also able to add a new user under the servletMgrRealm by presenting to WebSphere Application Server a signed JAR file containing a simple text file.

Now we will show you how to create a new servlet-signer, step by step. First, we went to the Security page under servletMgrRealm.

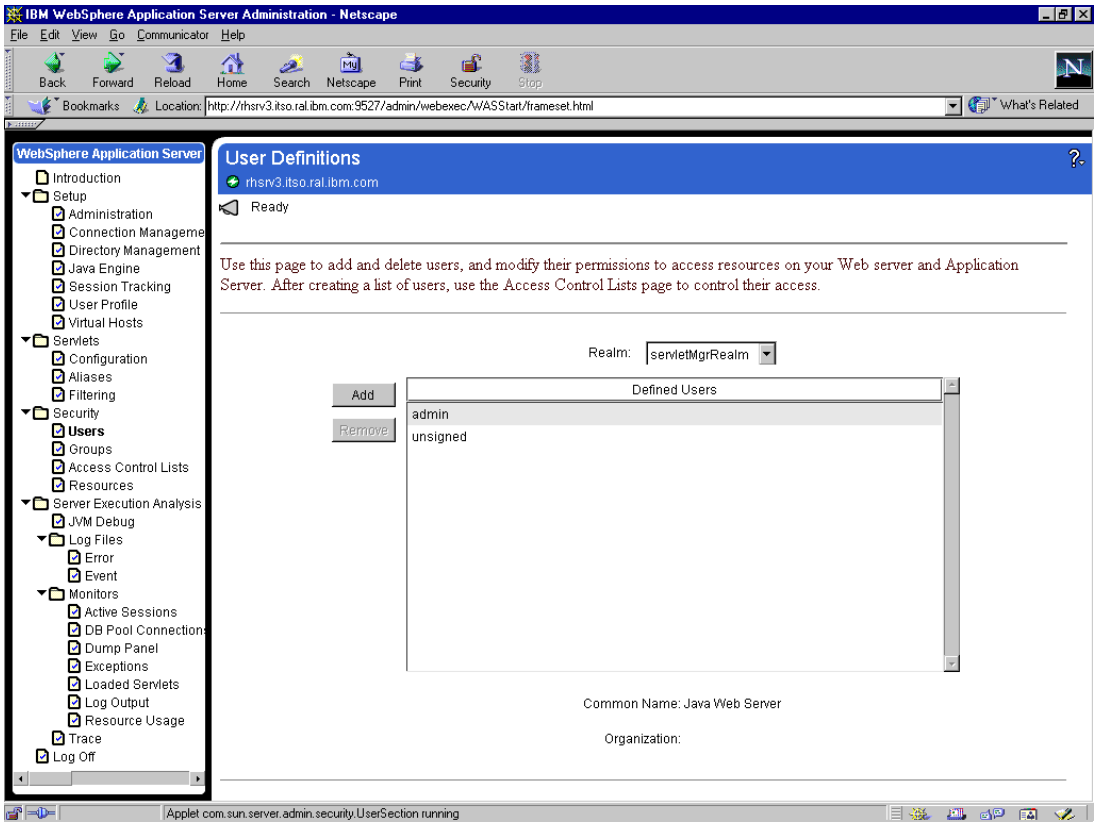


Figure 50. Adding a User in the servletMgrRealm

We clicked **Add**. Then we filled in the particulars as shown.

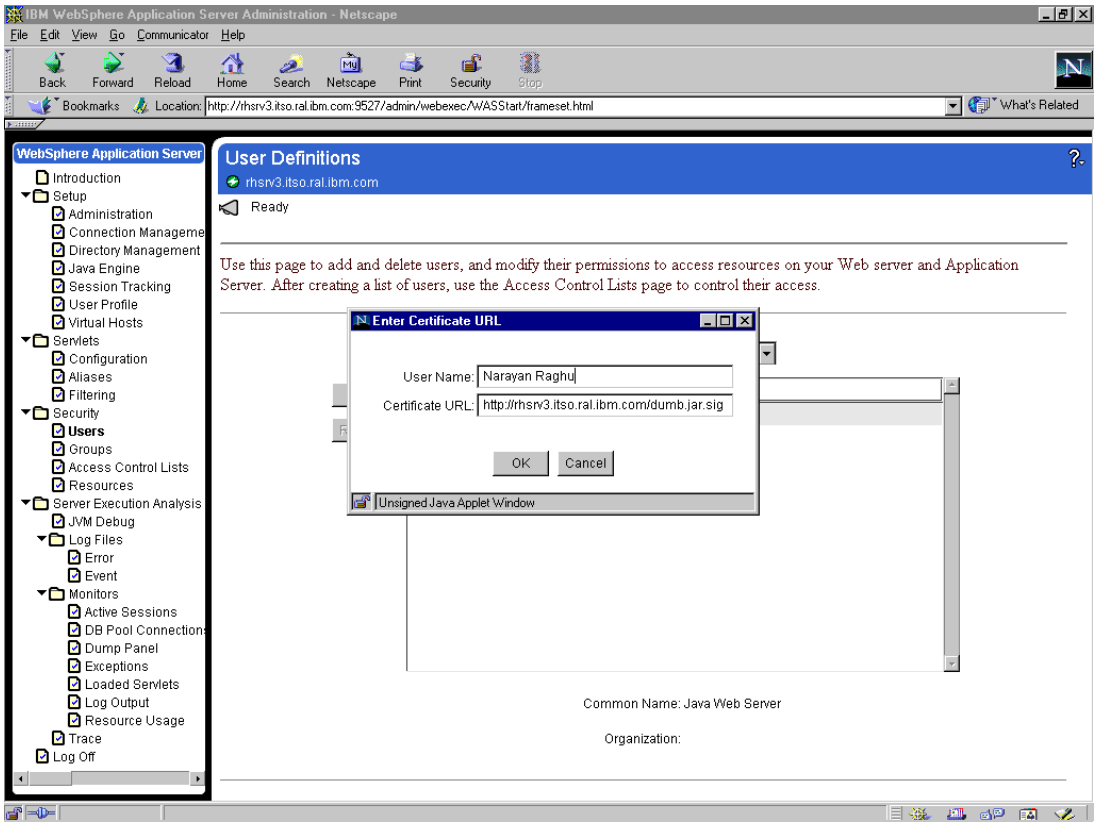


Figure 51. Users

The URL `http://rhsrv3.itso.ral.ibm.com/dumb.jar.sig` that we entered pointed to the location (<http server root>/htdocs/) containing the JAR file that had been signed by the particular signer we wanted registered. Three important points must be considered:

1. When you enter the User Name in the above panel, you do not have to type the name that is registered in the certificate. The User Name that you enter here is simply the name of the servlet-signer that WebSphere Application Server identifies as the owner of the certificate.
2. Even if the above panel has a field named Certificate URL, what WebSphere Application Server really expects you to type there is the URL of a JAR file signed by the trusted user you want to add.

3. The signed JAR file is not supposed to contain a servlet class. Actually, you could jar whatever file, even a text file, sign it, and use that file to add a user under the servletMgrRealm.

This is enough to add a user under the servletMgrRealm, but now we want to show you how we created the signed JAR file dumb.jar.sig to which the Certificate URL points.

To produce the JAR file dumb.jar from the class file DumbServlet.class, we used the command:

```
jar cvf dumb.jar DumbServlet.class
```

JAR files and signed JAR files

A discussion on JAR files in JDK 1.x is found at `http://java.sun.com/products/`

Notice, however, that even if the Java support that our Web server used was still 1.1, no differences have been noticed between the jar utility of JDK 1.1 and the jar utility of JDK 1.2. You can therefore apply the same considerations.

However, the technique to sign a JAR file changes in JDK 1.2 from what it was in JDK 1.1. When IBM HTTP Webserver and WebSphere Application Server support JDK 1.2, you will have to use the new security features and tools provided by JDK 1.2.

In order to add a servlet-signer to the servletMgrRealm, we needed to use the `javakey` commandline tool. The javakey.exe file comes with the JDK in the bin directory below JAVA_HOME. It is a commandline utility that can create entities, designate them trusted or untrusted, create DSA key pairs, sign certificates, etc. Again, type `javakey` to get all the possible options. We recommend the JavaSoft Web site `http://www.javasoft.com/security/usingJavakey.html`, which deals with how `javakey` is to be used. However, you can see how we did it now.

First, we created the entity narry through the following command:

```
javakey -cs "narry" true
```

We then generated a key pair for the entity:

```
javakey -gk "narry" DSA 512 narry_pub narry_priv
```

Then we created a certificate directive file as shown, and stored it in the file `cert.direc`:

```
issuer.name=narry
issuer.cert=1
subject.name=narry
subject.real.name=Narayan Raghu
subject.org.unit=ITSO
subject.org=IBM
subject.country=US
start.date=02 Jun 1999
end.date=01 Jun 2000
serial.number=1500
out.file=narry.x509
```

Then we created a certificate using:

```
javakey -gc cert.direct
```

We had already created a JAR file, named dumb.jar from the class file DumbServlet.class. We then created a signature directive file like the one shown, and stored it in sign.direc:

```
signer=narry
cert=1
chain=0
signature.file=narrySig
```

Then, finally, we signed the JAR file dumb.jar, using:

```
javakey -gs sign.direc dumb.jar
```

to obtain the file dumb.jar.sig. Notice that a signed JAR file in JDK1.1 carries the double extension .jar.sig, but in JDK1.2 JAR files keep the extension .jar even when they are signed. Keep this difference in mind when IBM HTTP Webserver and WebSphere will have a full JDK 1.2 support.

Note that we used the DSA algorithm, which ships free with JDK. The common standard for X.509 certificates is, however, RSA (though this is not required by the standard).

In this way we have just shown you the full process necessary to create a new user, or servlet-signer, under the servletMgrRealm. As you can see, if you want to add a user to servletMgrRealm, you have no other choice than to generate a signed JAR file and then point to its URL, even if it is not required

that the signed JAR file contain a servlet class. As we said, a signed JAR file containing a text file works fine anyway.

What about new users in the UNIX realm? This realm is used to give access to users who already have an ID on the UNIX machine, to the servlet resources on the server. You will not be able to create users here, since WebSphere picks up information about the UNIX users from the operating system it is running on. We will talk more about this when we talk about ACLs.

At this stage, the only form of authentication of users supported is the basic authentication, with user ID and password, and the only realm that can be used to store user profiles, such as which servlets the user is permitted to run, and which of these servlets can access system resources, is the defaultRealm, unless you would not mind programming on WebSphere using the JDK APIs.

The user that you can create in the servletMgrRealm is used only for signing servlets that are to be remotely loaded. In more unambiguous terms, the user in the defaultRealm is not the same as, or even similar in functionality to the user in the servletMgrRealm. The former is a client accessing the resources the server and the servlets have to offer, and the latter is a person whose signatures on JAR files containing servlets the WebSphere has been instructed to trust. At this stage, we would like to mention that for obvious reasons, it is not possible to create users in the UNIX realm. These users are read off the underlying operating system.

8.1.4 Groups

Let us now see how to create groups. To begin with, let us stick to defaultRealm. Click the **Groups** link, and it will take you to the groups page, as shown in Figure 52 on page 138.

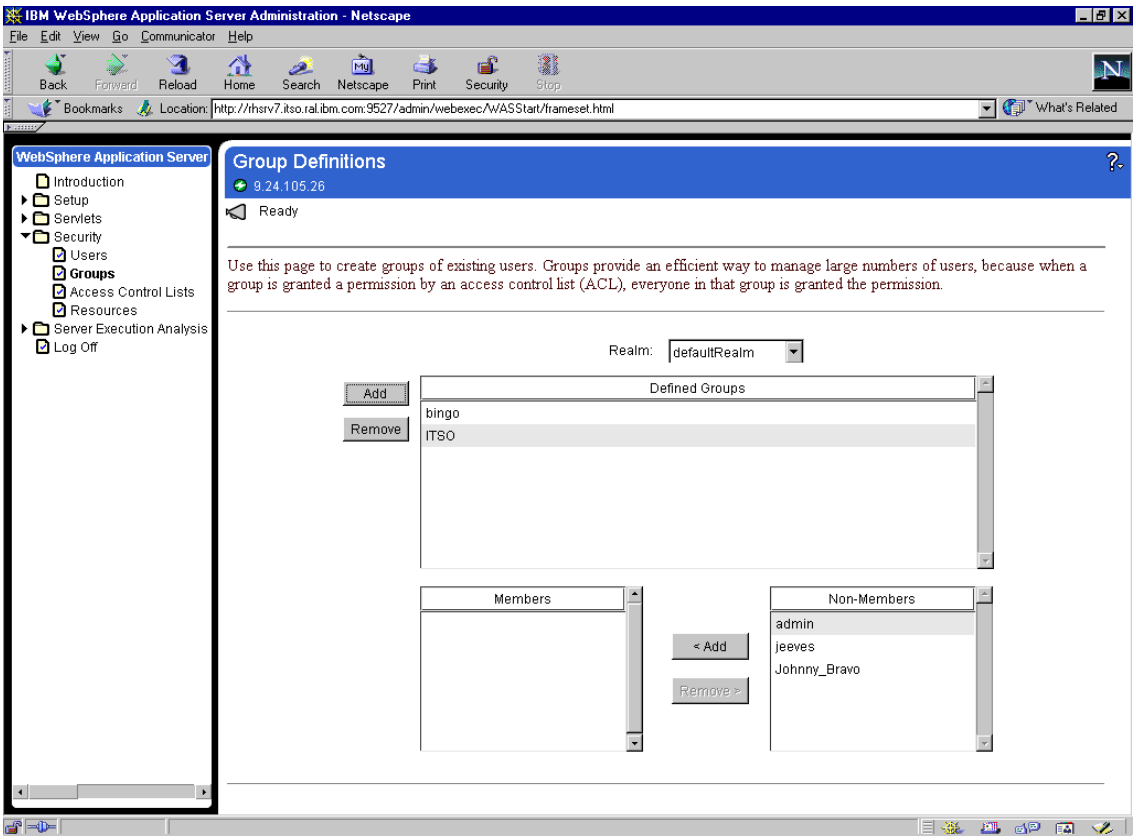


Figure 52. Groups

As you can see, two groups have been created. You can create another one by clicking the **Add Group** button. You can enter the name in the message box, and click **Add**, as shown in Figure 53 on page 139.

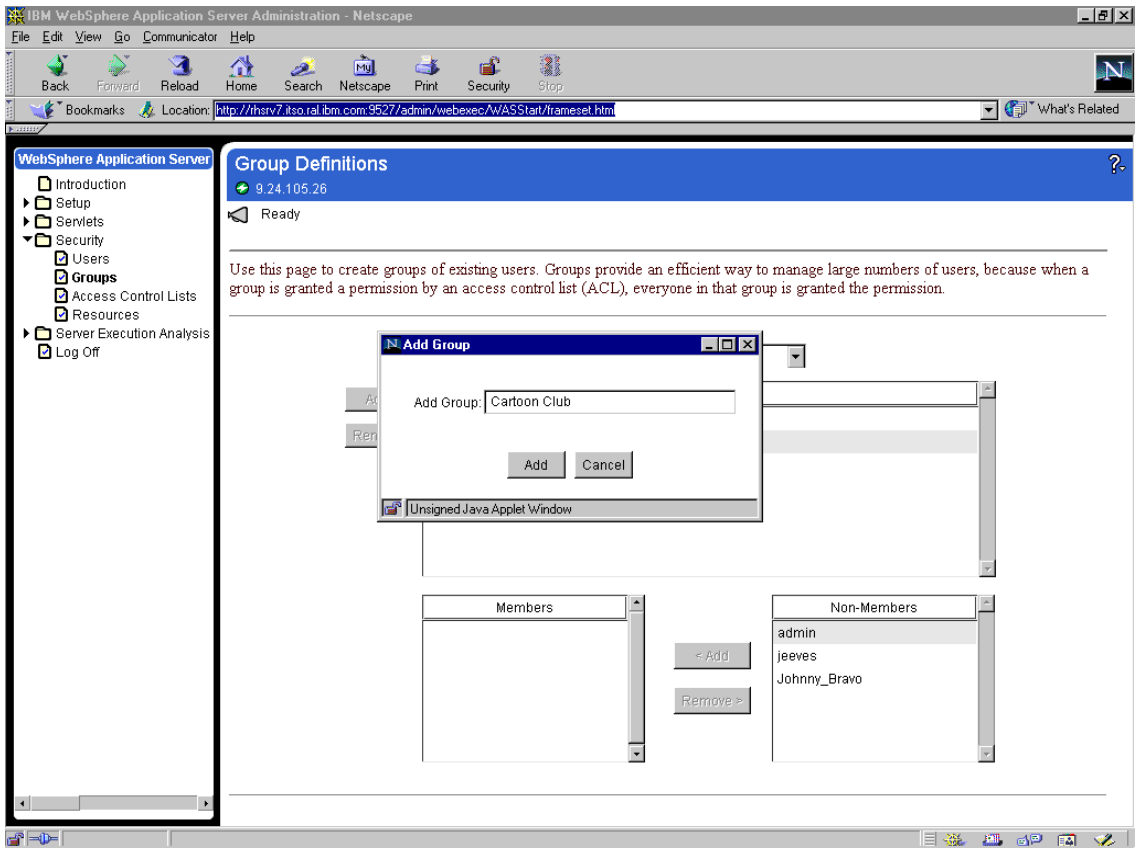


Figure 53. Add Groups

Groups are not really entities in themselves. They are used to make it easier for the administrator to give or revoke permissions to a number of people at the same time. Hence, we now have to add users into groups. Note that a user can belong to more than one group, and a group can have any number of users. In the beginning, all the users within the realm are non-members of a newly created group, meaning that they appear in the Non-Members list. The admin can choose to add any number into the group by clicking the **Add** button, and remove any number by clicking the **Remove** button. Note that the union of the users in the Members list and the users in the Non-Members list is the list of users in the realm. Also note that no user can be a member and a non-member at the same time; in other words, the intersection of the two sets is always a null set.

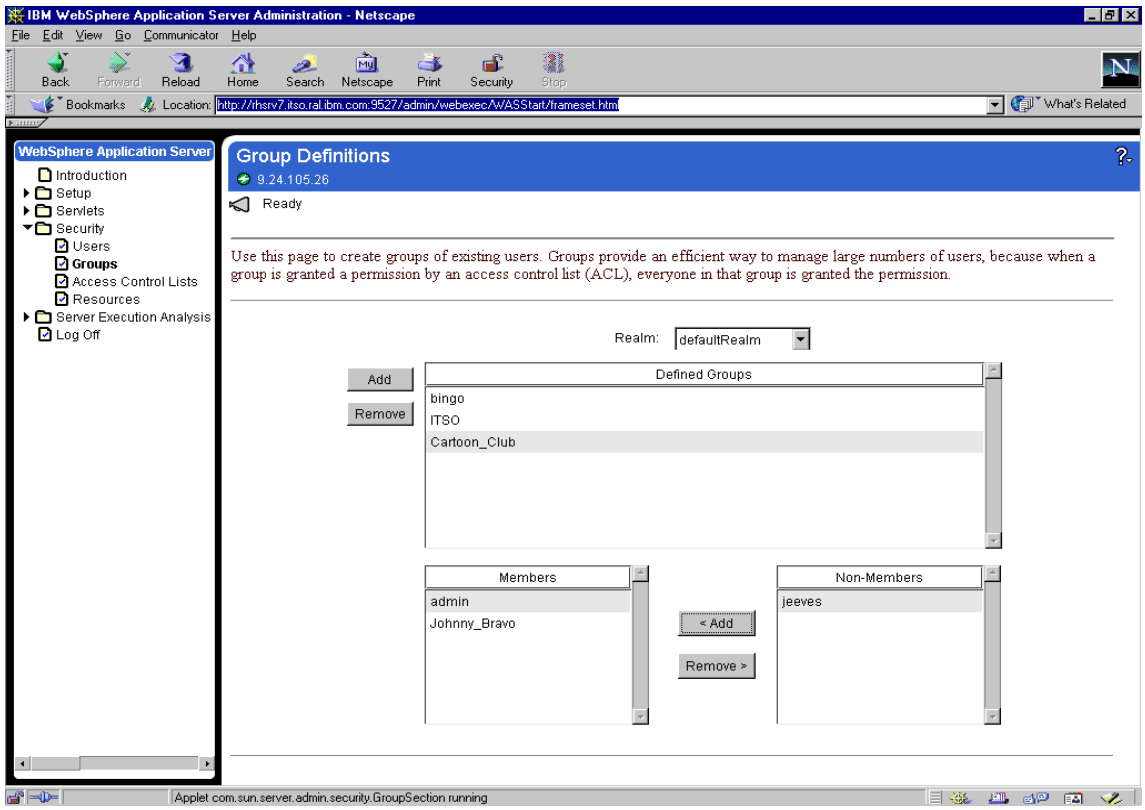


Figure 54. Add to and remove users from groups

Now, if you are as inquisitive as we were and want to go to the file and check how this information is stored, go to the directory *WebSphere Application Server_root*/realm/data/defaultRealm and type out the file with the same name as the newly created group. We typed the file /opt/IBMWebAS/realms/data/defaultRealm/Cartoon_Club.grp and we got a listing of the user names we just added into the group.

You will notice that a new file is created for each group, and each new user is added to the file named keyfile in the appropriate directory. You could try to edit these files yourself and try crashing the system, but we did not do that and so cannot predict the result. Note that groups cannot be created in the servletMrgRealm.

8.1.5 Access control lists

Now, let us tackle one of the more important parts of the WebSphere Application Server setup - the ACLs. The administrator can add ACLs or delete existing ACLs in a given realm. The addition of an ACL is pretty similar to the addition of users and groups. We will show one just for completeness. Select **Access Control List** from the Security tree and then click **Add ACL:**

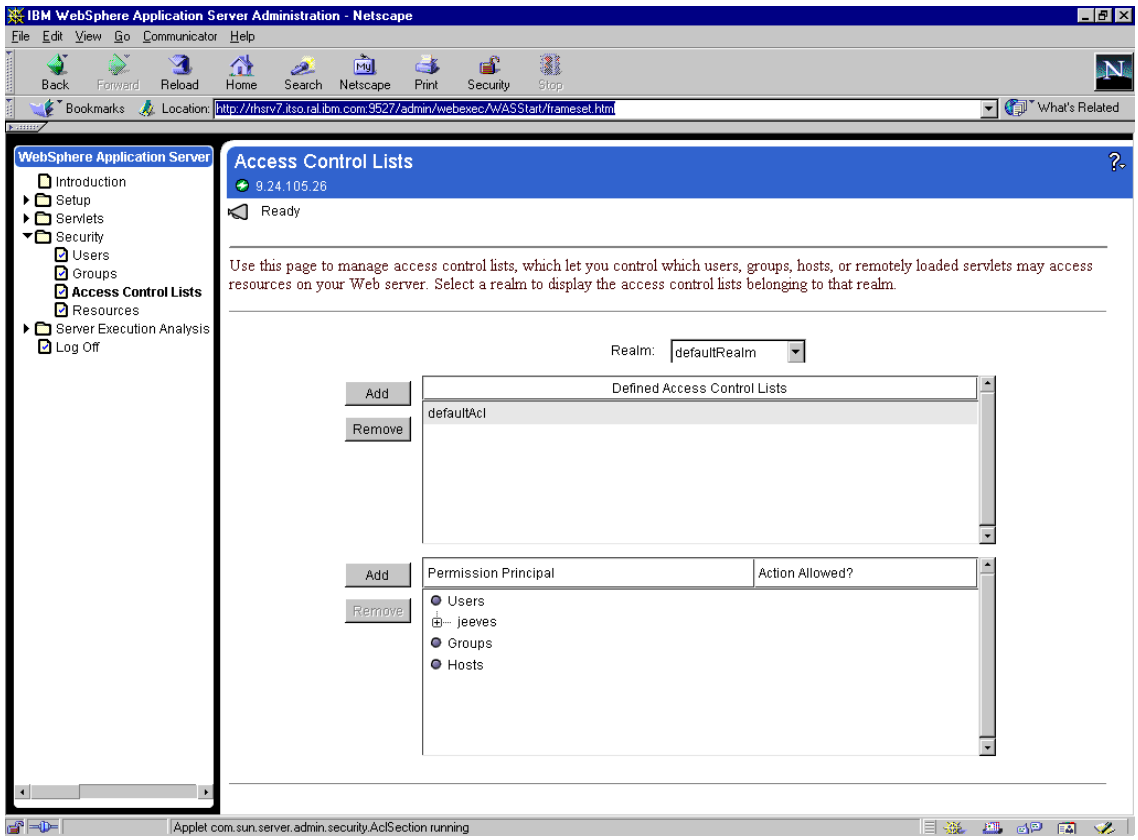


Figure 55. Security - Access Control Lists page

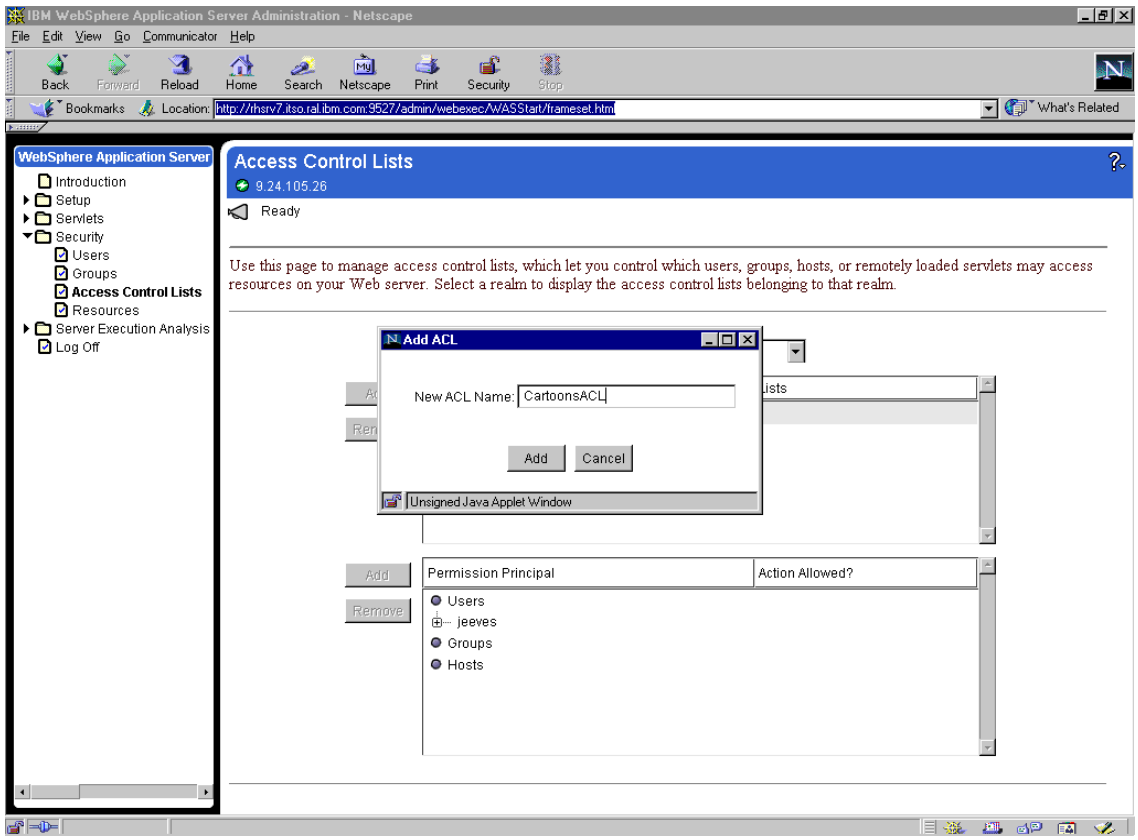


Figure 56. Addition of ACLs

You can type the name in the message box and then click **Add**.

Now, what do you do with an ACL? You simply add or remove permissions for certain users and groups for certain resources in the ACL. Note that all this happens only within the realm, and it is not possible to add a user registered under a different realm to an ACL in this one. Let us now try to add some permissions. Click **Add**. You should see one of the following three screens, depending on whether you have marked **User**, **Group** or **Computer**:

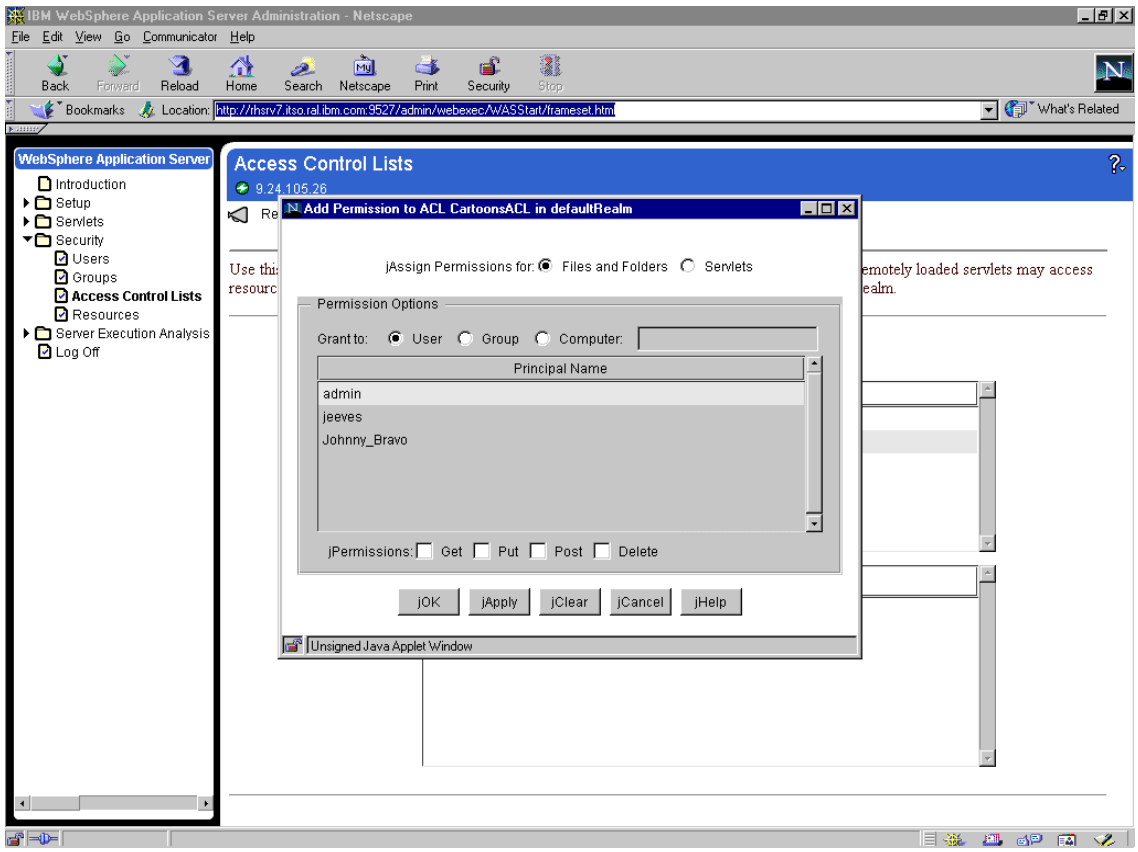


Figure 57. Adding file access permissions to users

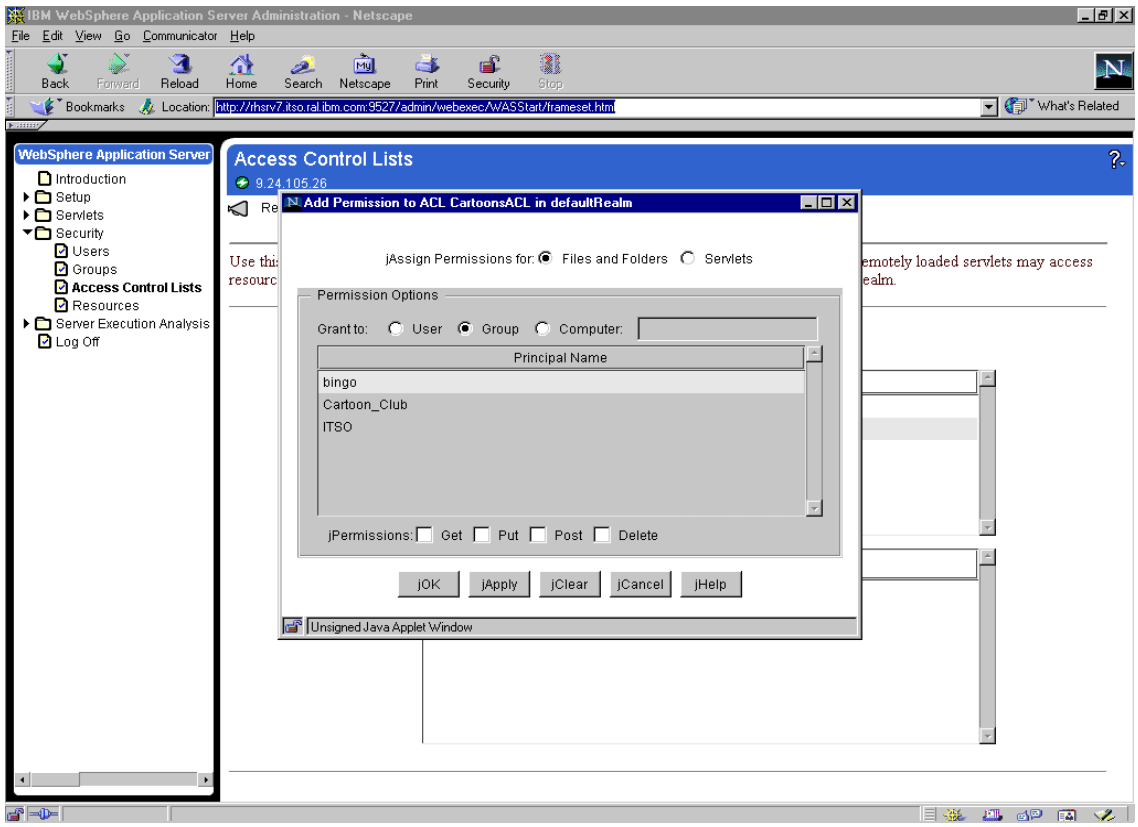


Figure 58. Adding file access permissions to groups

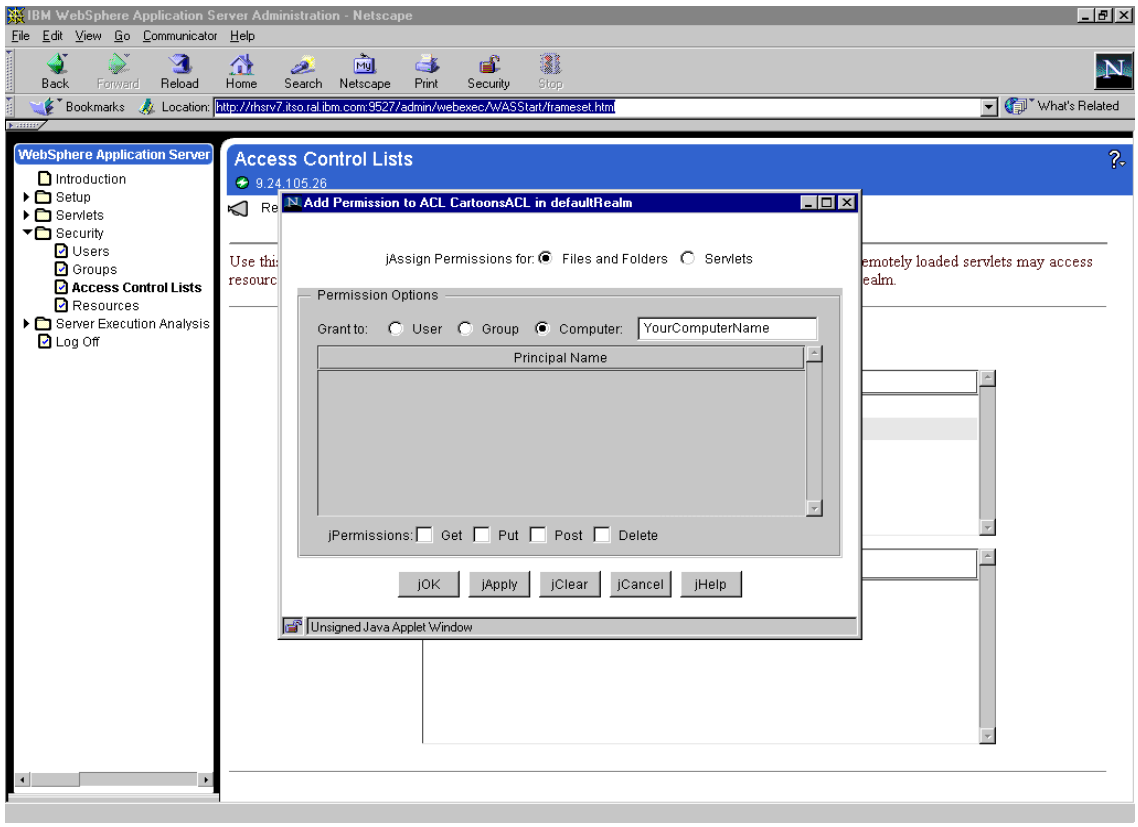


Figure 59. Adding file access permissions a specific computer

Note the file access permissions you can give to users, groups, and specified computers on the network. The file permissions are the same as the HTTP methods of the form handling protocol:

1. GET

   This method is used for sending form data. The data is sent to the server from the client by appending the data in the URL field. The same stream is used for the requested URL and the form data.

2. PUT

   This is used to put files on the server. You might not have access to this method from a normal browser. However, most HTML authoring tools use this method to put an edited HTML page onto the server.

3. POST

   As the submitted form data got larger, it was found that the URL field was inadequate to handle this data. Further, it is really not very graceful to clutter up the URL window. The POST method is used to send data from the client to the server. It is different from the GET method in that another stream is used to send the data.

4. DELETE

   This option, like the PUT method, might not be available from your normal client. Moreover, though specified in the HTTP protocol (see `http://www.w3.org/Protocols/`), most servers do not grant permission for this action. This is used to delete a file from a server. This method might be supported by some Web authoring clients, but as mentioned before, might not be supported by your server.

Let us now add some permissions both to users and groups, and see if they work. But before we verify if they work, we have to associate a resource with them. So, let us go ahead with granting permissions first. The following screens are those of us granting permissions to users, groups, and computers. Since these are pretty much self explanatory, you probably would not like redundant sentences between them. However, in the rest of the chapter you will see how these permissions really work.

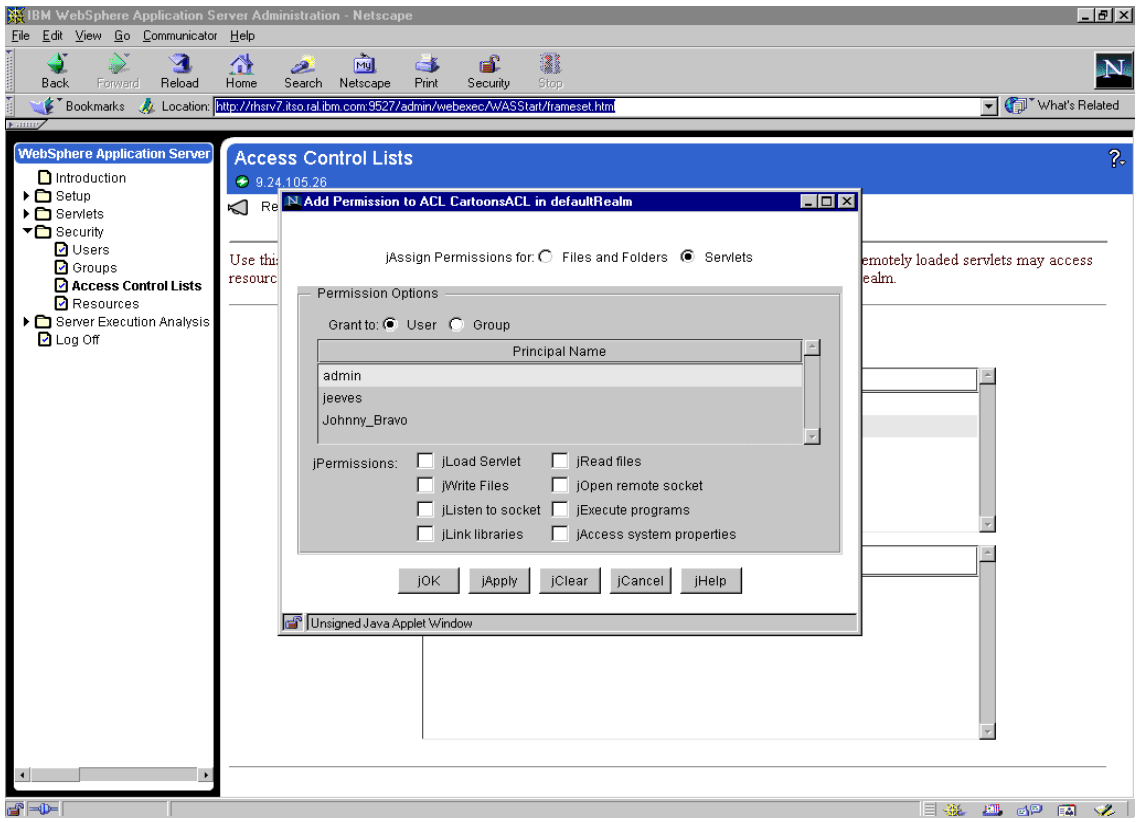


Figure 60. Adding no servlet permission to user admin

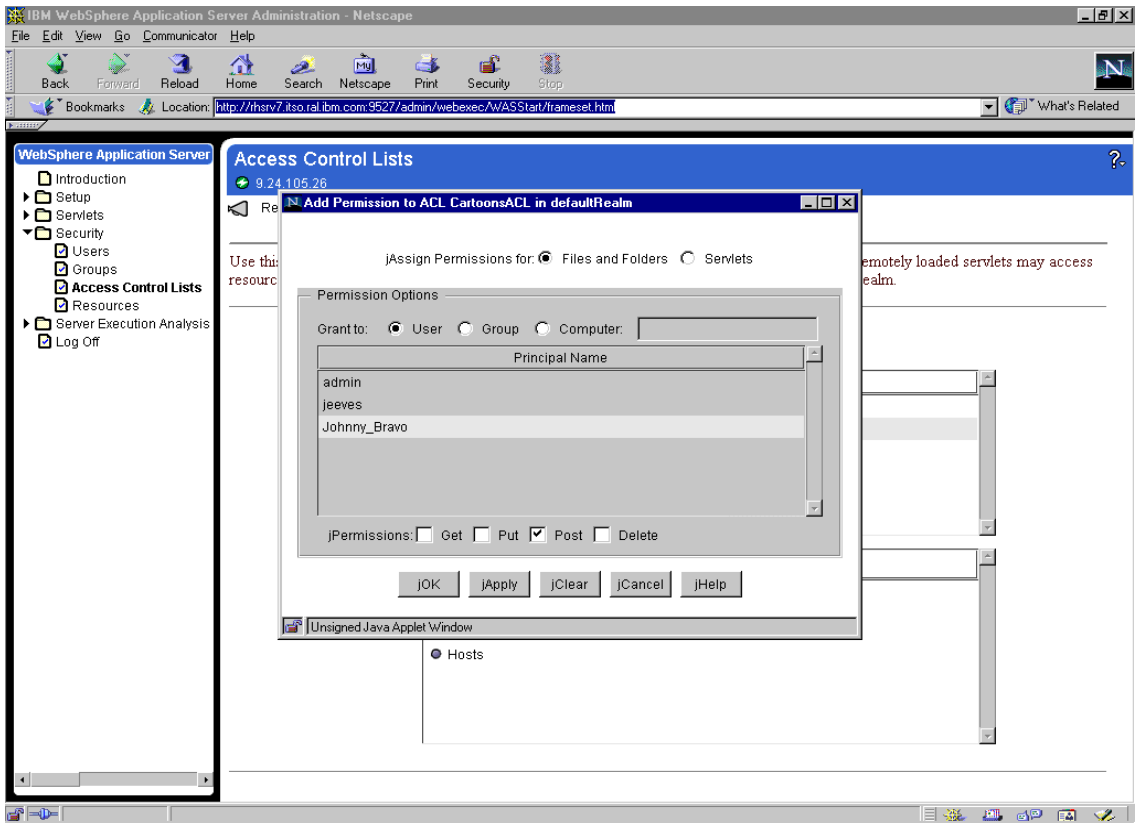


Figure 61. Adding file post permission to user Johnny_Bravo

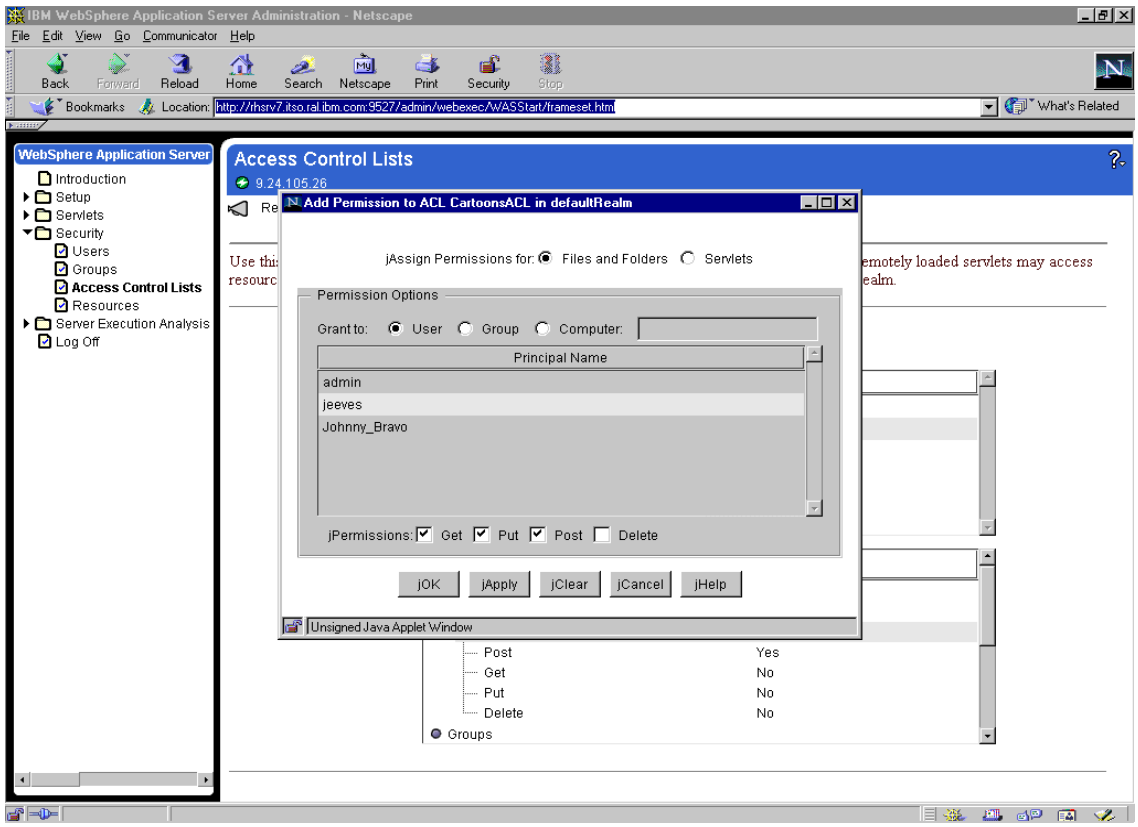


Figure 62. Adding file permissions Get, Put and a Post to user jeeves

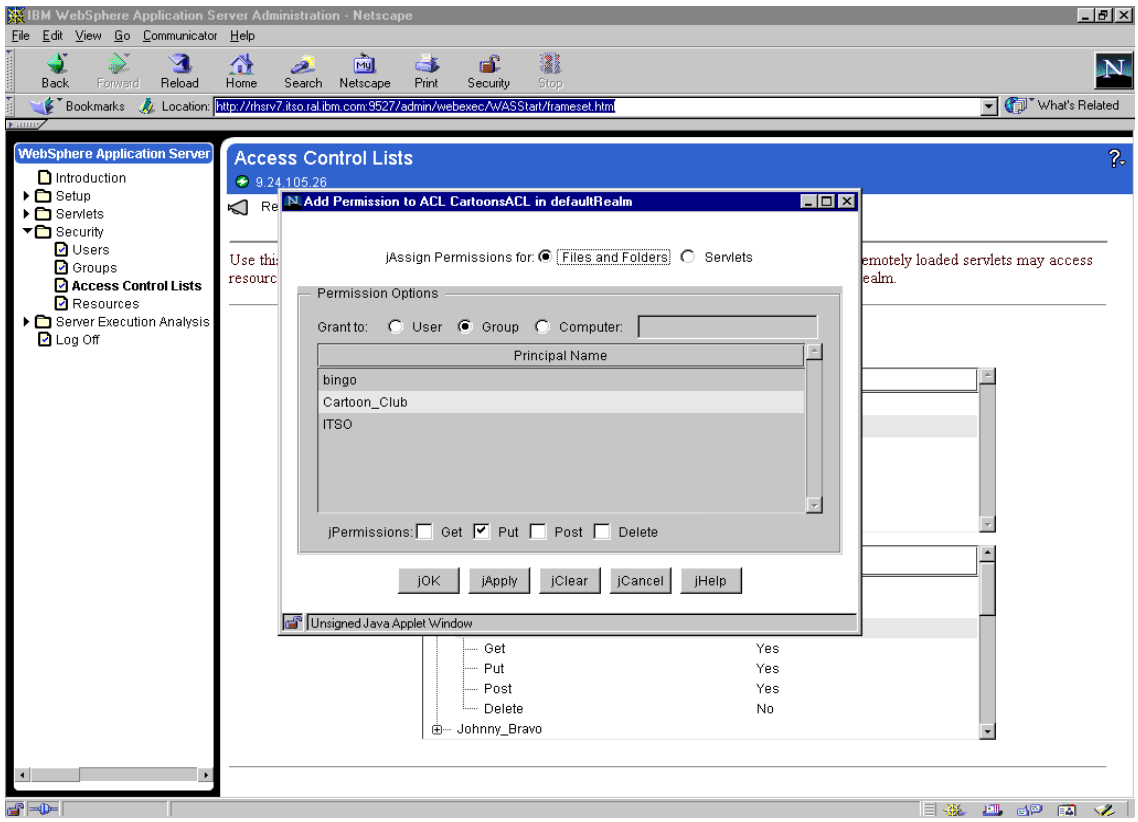


Figure 63. Adding file Put permission to group Cartoon_Club

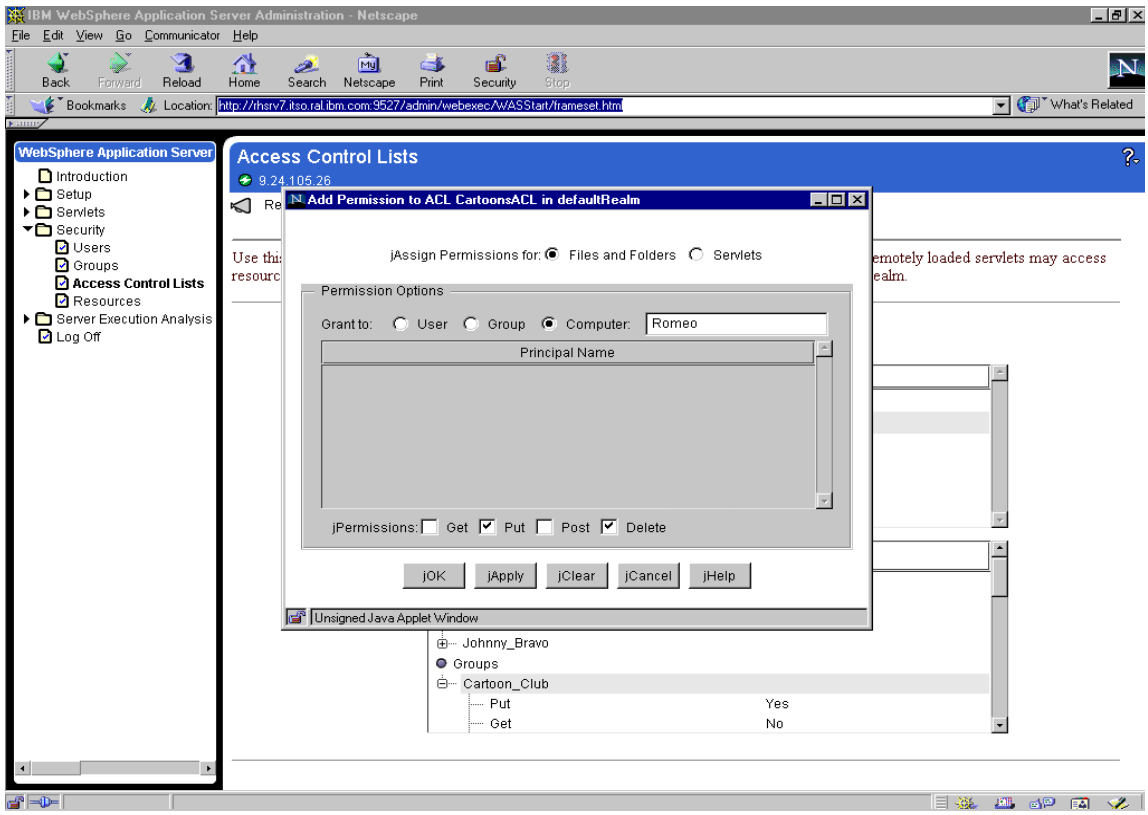


Figure 64. Adding file permissions Put and Delete to computer Romeo

Note that all the permissions were added in one particular ACL, the CartoonsACL that we had defined. Now, if you come over to the main screen and look at the bottom window, you can get complete information about how the access controls are set. Users, groups, and computers for which permissions were set appear in the Principal tree.

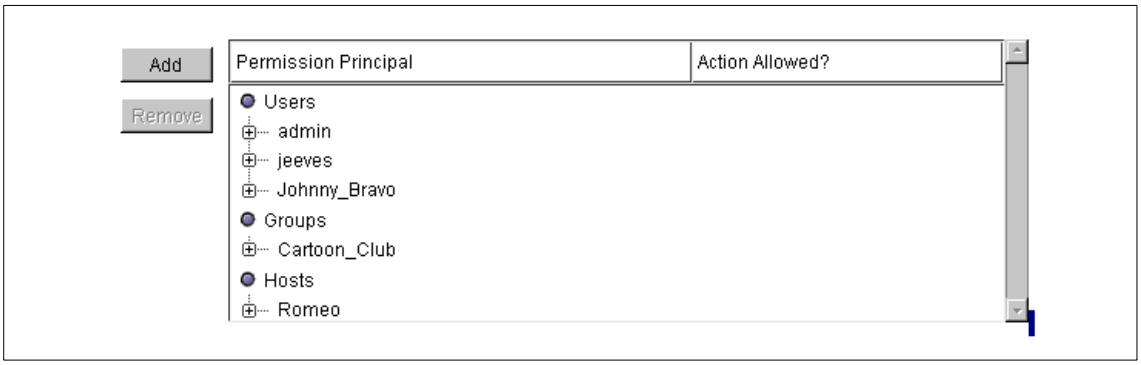


Figure 65. Principal tree

You must click the plus sign (**+**) beside each item on the list to see the specific permissions.

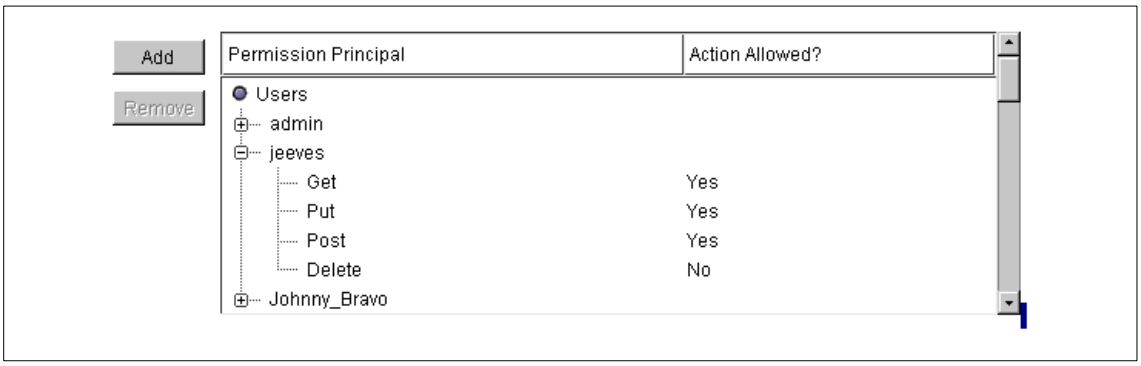


Figure 66. Checking specific permissions

Notice that Figure 64 shows the particular case when you want to allow access only from specific computers. In the Computer field, you can enter the name of the host either as a host name or as an IP address. You can use the wildcard character * when entering a host name, for example *.com. Requests that originate from hosts other than the specified one will be denied.

Note that you cannot add ACLs in the servletMgrRealm. However, you can add permissions to either servlets or files in the servletACL. The servletACL is the only ACL that is available in the servletMgrRealm. This is created automatically, and cannot be deleted. Here is where you would have to indicate what permissions you would like to give to servlets signed by a

particular user in the servletMgrRealm. We will deal more with this in later sections.

The UNIX realm, however, behaves pretty much like the defaultRealm, except for the fact that the user list cannot be modified. Notice that ACLs can be added to the UNIX realm.

8.1.6 Resources

A resource, in this context, is essentially a service the Web server has to offer, which needs to be protected. HTML pages and servlets that need not be protected are not considered resources in this context. In general, to restrict access to static HTML pages, you would have to use the facilities provided by IBM HTTP Web server, or any other Web server you might be using, and you would use WebSphere Application Server to control access to servlets. However, WebSphere Application Server, using the servletMgrRealm, allows the administrator to protect also the resources that servlets can access, such as files and socket connections. This could create a conflict with the underlying Web server, in that access to a Web page could be denied by the Web server and allowed to a servlet by WebSphere Application Server. In general, the relationship between the Web server and WebSphere Application Server about conflicting permissions is as follows. If the Web server first denies access to a resource, then access is denied to WebSphere Application Server too. If the Web server permits the access, then it is up to WebSphere Application Server to protect the resource. In other words, it is the Web server that takes precedence.

Put briefly, access control in the WebSphere Application Server security model is all about how WebSphere Application Server gives users and groups access to several of the resources using predefined ACLs.

The administrator of WebSphere Application Server can add new resources, in the sense that he or she can declare more of the existing resources to be protected. We tried this out, and to do so, we first went to the Resources screen, which for the defaultRealm looked like the following figure, and clicked **Add**.

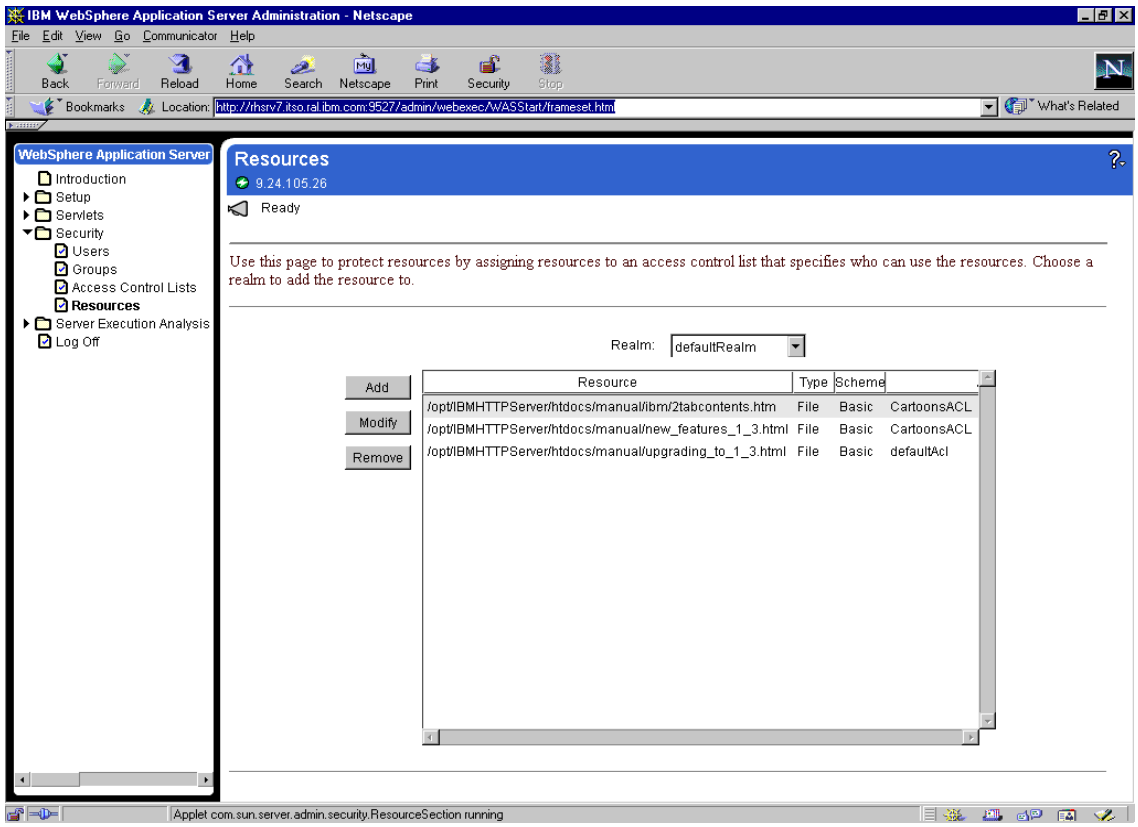


Figure 67. WebSphere resources

The second screen (Figure 68) shows the kind of resource you can add to an ACL.

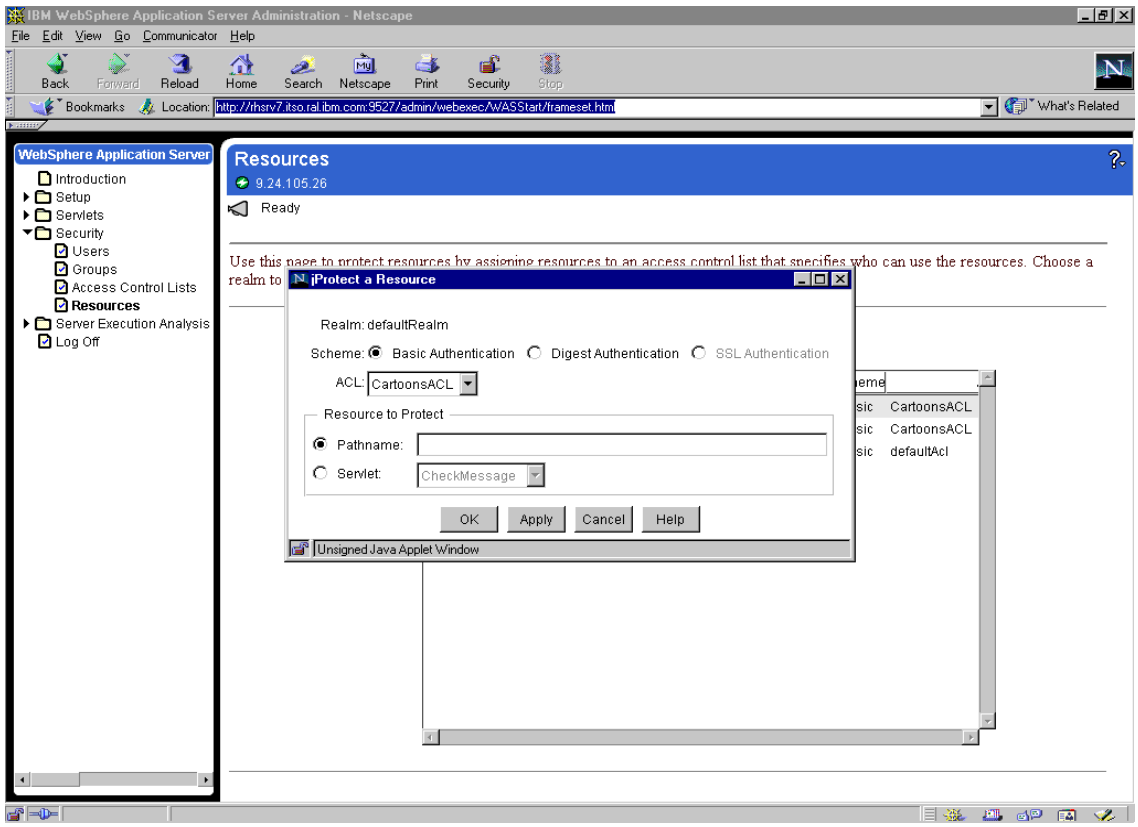


Figure 68. Adding resources to ACLs

The screen begins by confirming that we are still in the defaultRealm. The next line shows the type of authentication. Basic Authentication will ask the client to key in a user ID and password, which will be sent to the server over the network. Digest Authentication will do the same, but the information will be sent to the server in an encrypted form over the Net. SSL Authentication, which we expect that you will find disabled on your screen, will authenticate the user through their client certificate. However, this is not supported in the simple installation of WebSphere Application Server. You could gain this functionality by writing your own realm, using the JDK APIs.

Next is a drop-down list box containing a list of all the ACLs in the realm. WebSphere Application Server permits us to add a particular resource to only one ACL, since if the same resource is in more than one ACL, there might be conflicts in permissions. We tried adding a particular resource that already belonged to one ACL, to yet another ACL, and we found that the resource

now belonged to the new ACL, but not to the older one. In other words, status of a resource can be overwritten.

Next is the option to choose the resource to protect. The administrator can choose between a servlet and a file. If the resource is a file or a directory containing HTML pages, the administrator is to key in the complete path name.

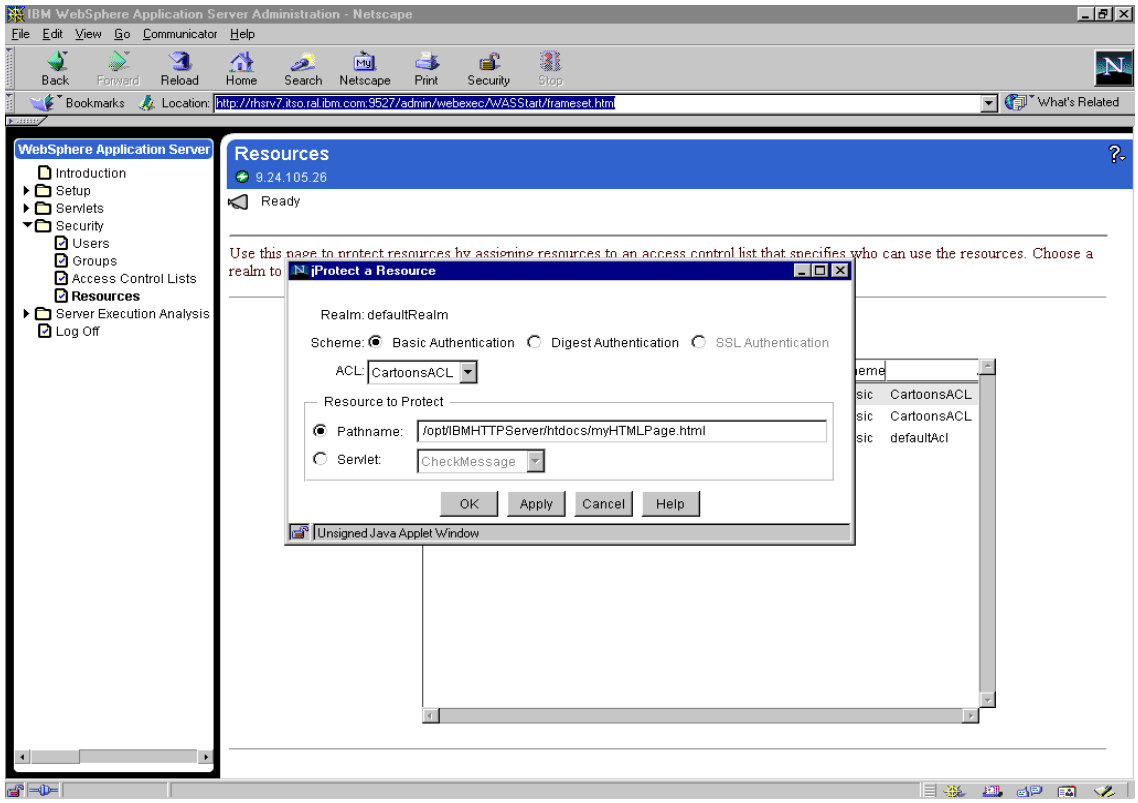


Figure 69. Adding a file resource

If the resource in question is a servlet, it can be selected from the drop down list box.

8.1.7 Examples of Security Using HTTP and SSL

In this section we will illustrate the differences between the two HTTP methods, GET and POST, that are commonly used by forms to handle the information sent to the server. We will also see the security implications of how GET and POST perform over plain HTTP and SSL.

To do this, we wrote a very simple servlet that welcomes the user, with the entered user name. This servlet is invoked from a simple HTML page, alternatively using the GET and POST methods. The HTML code is as shown:

```
<HTML>
<HEAD>
<TITLE> The Get Method </TITLE>
</HEAD>
<BODY BGCOLOR="white">
<BR>
<CENTER><h2> Please enter your particulars </h2></CENTER>
<FORM Action="/servlet/EchoServlet" Method="Get">
<PRE>
User Name : <INPUT Type="TEXT" Name="userid">
Password : <INPUT Type="password" Name="passwd">
<INPUT Type="SUBMIT">
</PRE>
</FORM>
</BODY>
</HTML>
```

Figure 70. The id_get.html file

When the GET method is used, the filled-in form inputs variable names and their values are sent to the server by simply appending them to the URL of the next request. We show here the source code of the servlet that receives the form values from the HTML page:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class EchoServlet extends HttpServlet
{
    public void init(ServletConfig conf) throws ServletException
    {
       super.init(conf);
       log("Echo Server Initialized");
    }

    public void service(HttpServletRequest req, HttpServletResponse res)
    throws IOException, ServletException
    {
       String userid = req.getParameter("userid");
       ServletOutputStream ops = res.getOutputStream();
       res.setContentType("text/html");

       ops.println("<HTML><HEAD><TITLE>Generated by a
       Servlet</TITLE></HEAD>");
       ops.println("<BODY>");
       ops.println("<CENTER><H1> Hello </H1>");
       ops.println("Welcome, " + userid + " how are you today");
       ops.println("<BR>");
       ops.println("</BODY></HTML>");
       ops.close();
    }
}
```

Figure 71. The EchoServlet source code

Note

Save the above code as EchoServlet.java, and compile it using jdk117_v3. You will need to set your CLASSPATH variable to include the javax package. At a bash shell command line, type:

```
export CLASSPATH=<jdk_root>/lib/classes.zip:<was_root>/lib/jsdk.ar
```

where <jdk_root> is the directory leading to jdk117_v3; in our case, it is /usr/local/jdk117_v3.

Where, <was_root> is the directory leading to IBM WebSphere; In our case, it is /opt/IBMWebAS/lib.

The following figure shows the HTML page in a browser, after entering the user name and password:

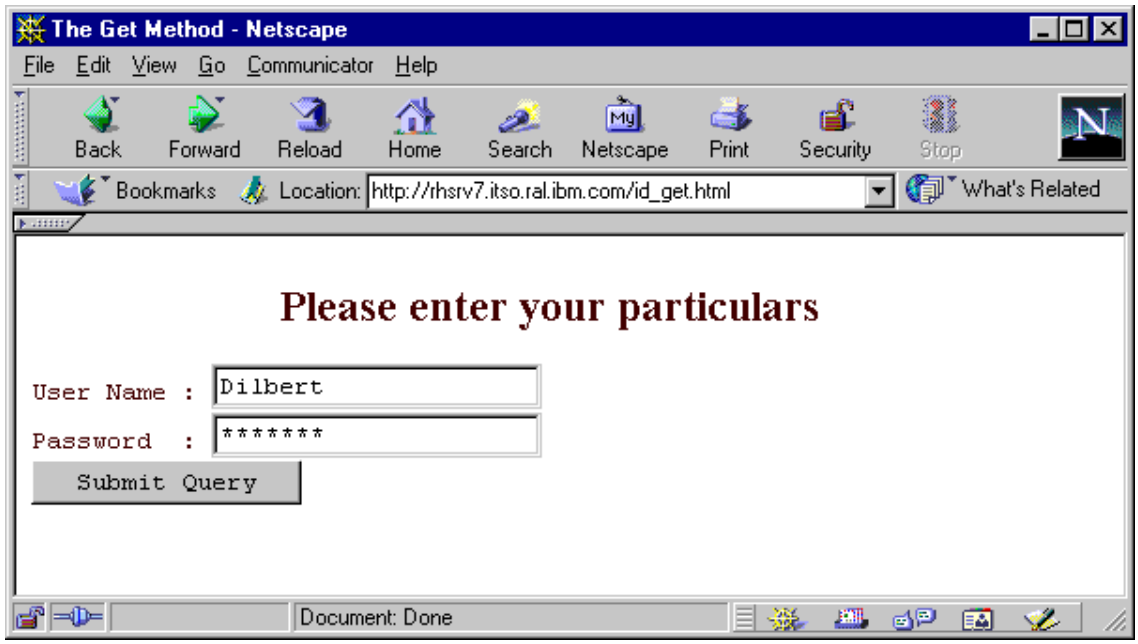


Figure 72. User ID and password

Note that what you type in the password field appears hidden by a sequence of asterisks, rather than the characters you typed in. This happens to grant privacy and security while you type your password, so that people sitting near you cannot read what you enter. Actually, the asterisks grant only the appearance of privacy and security, as we are going to demonstrate.

If you read through the servlet code, you see that the servlet will simply welcome the person using the user name entered, and will do nothing with the password. We added the password field here to show how confidential information flows between client and server when a form uses GET or POST through HTTP or SSL. The two fields, the User Name and Password, have been used to represent public data (that you would not mind people reading) and private data (confidential data, which you would not like any third-party members reading, such as your passwords or credit card number). We have just shown you how, on the screen of the client machine, confidential information is treated differently from public data.

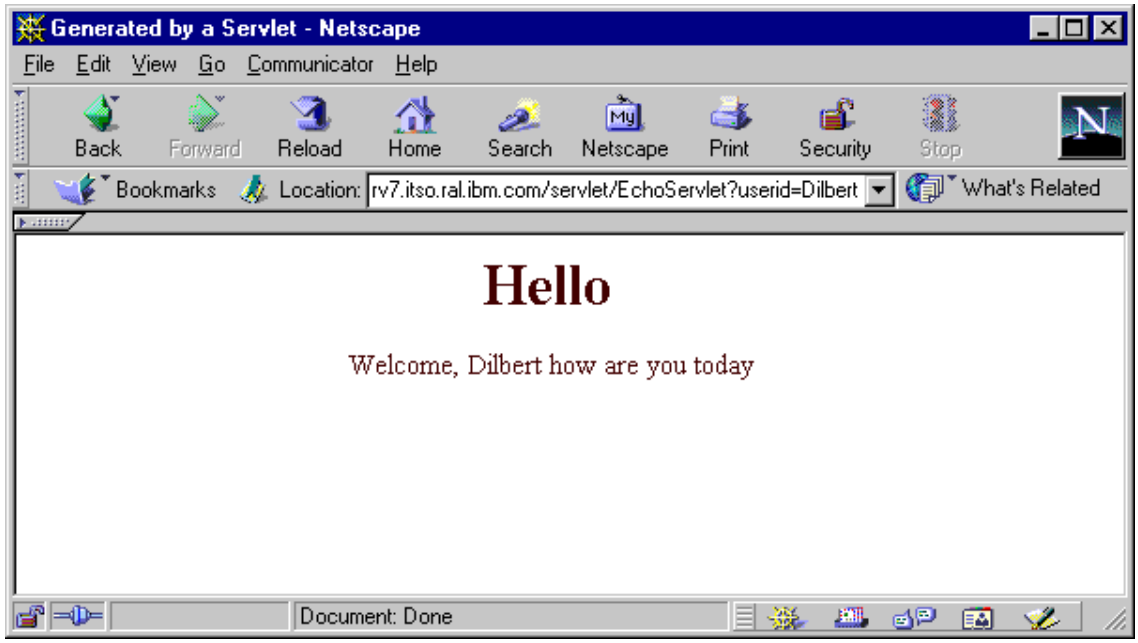


Figure 73. Confidential information displayed on the URL field

Now, let us try doing the same with the method POST. The servlet code remains the same. In the HTML page, let us simply change the method to POST. Note that this would not be possible with simple CGI; you would have to make some modifications in the CGI script and recompile, unless you have taken care to put an if condition in the script to check what method is used.

```
<HTML>
<HEAD>
<TITLE> The POST Method </TITLE>
</HEAD>
<BODY BGCOLOR="white">
<BR>
<CENTER><h2> Please enter your particulars </h2></CENTER>
<FORM Action="/servlet/EchoServlet" Method="POST">
<PRE>
User Name : <INPUT Type="TEXT" Name="userid">
Password : <INPUT Type="password" Name="passwd">
<INPUT Type="SUBMIT">
</PRE>
</FORM>
</BODY>
</HTML>
```

Figure 74. The id_post.html file

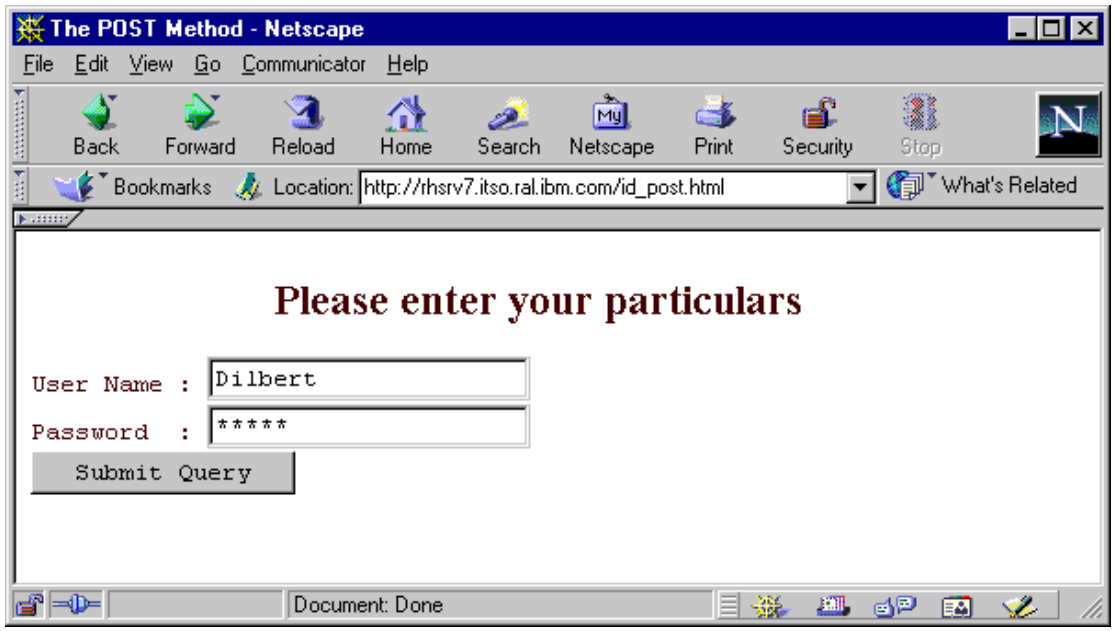


Figure 75. Using the POST method

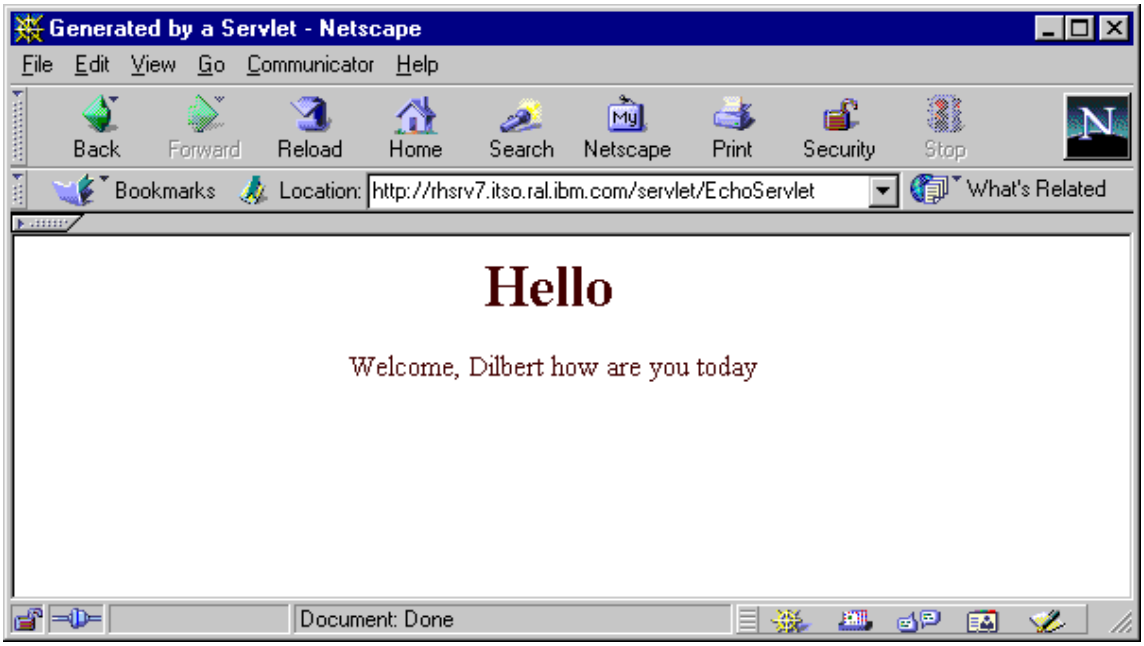


Figure 76. After using the POST method

Notice how the passed fields are not visible, this time, as a part of the URL (or anything else). This happens with the POST method because the form variable names and values are sent in the HTTP request body. This implies that they are not shown by the browser as part of the URL, which instead, is sent in the HTTP request header. If the GET method isuses, the URL and the data both go in the HTTP request header. This demonstrates that at least from this point of view the POST method grants more privacy and security.

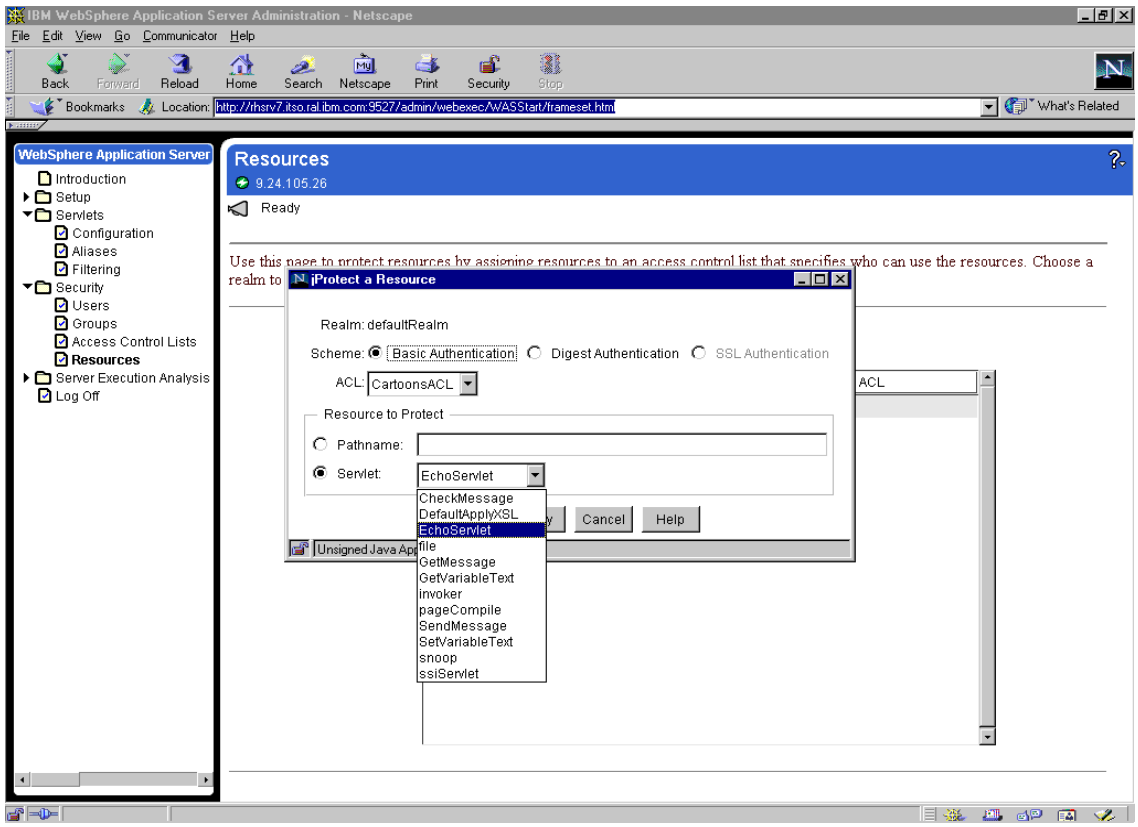


Figure 77. Adding a servlet resource

The help link from this section is very meaningful. You can read it by clicking the **Help** button.

Now let us try out the real stuff. We set an ACL called the CartoonACL, and registered EchoServlet as a resource in the CartoonACL, as shown in Figure 77. The Resources window looks somewhat like Figure 78 on page 164.

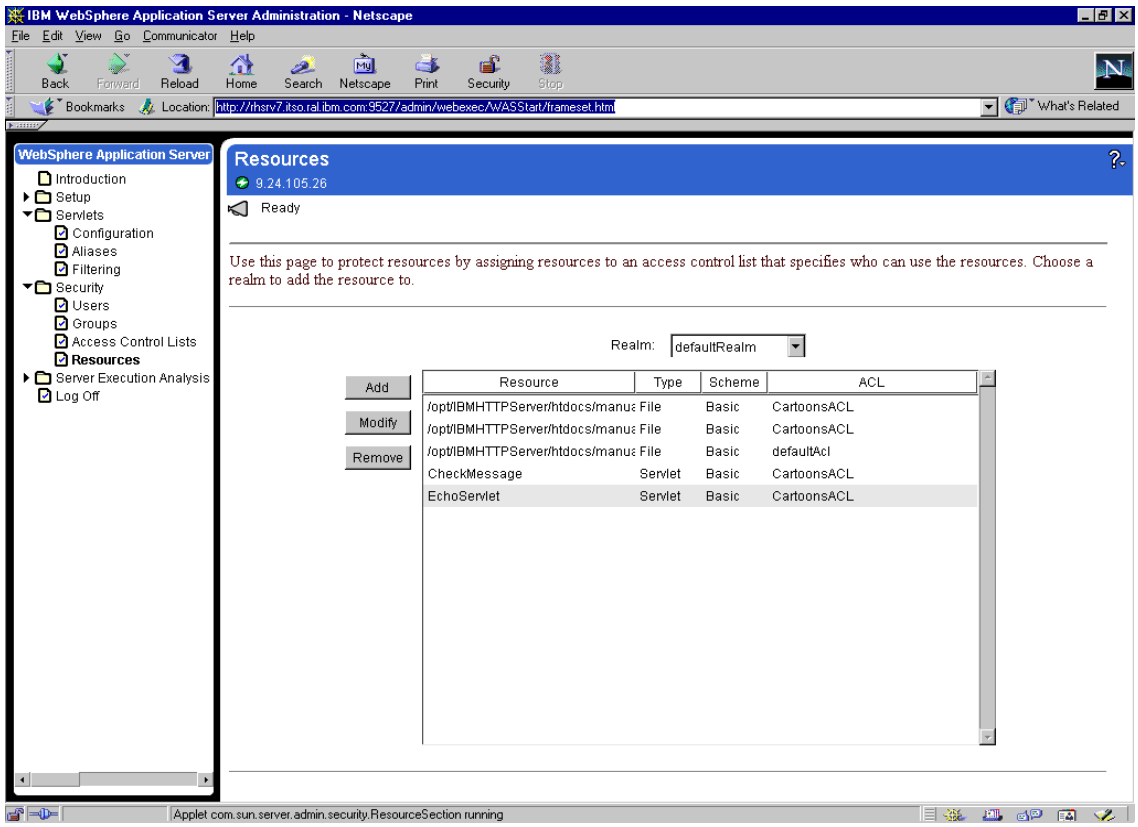


Figure 78. The resources window

Next, we add two more users under Security->Users: Asterix and Popye. Further, within the CartoonACL, we give the user Asterix permission *only* to POST, and the user Popye permission *only* to GET. At this stage, the ACL window looks somewhat like Figure 79 on page 165

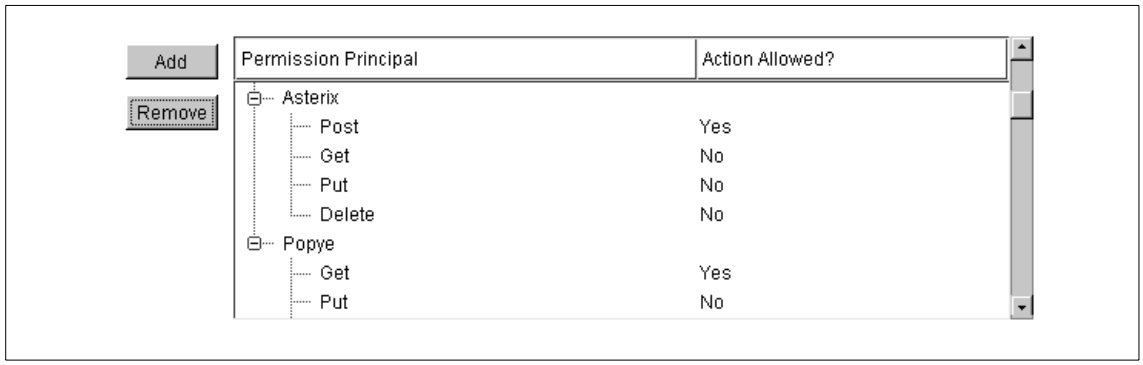


Figure 79. The ACL Window

You would do well at this point to refer back to the source code of EchoServlet, which we wrote earlier in this chapter to test the POST and GET methods (see Figure 70 on page 157, Figure 74 on page 161 and Figure 71 on page 158).

Let us now try to access the EchoServlet through the pages id_post.html and id_get.html, which use respectively, the POST and GET methods to communicate with the Web server. We opened up the Netscape window to access the HTML pages, and not surprisingly, there was no trouble getting to the pages themselves.

When we clicked **Submit** from the page id_post.html, a window popped up asking for a user name and password. Note that the page containing the same fields (User Name and Password) is purely coincidental and has nothing to do with access restrictions.

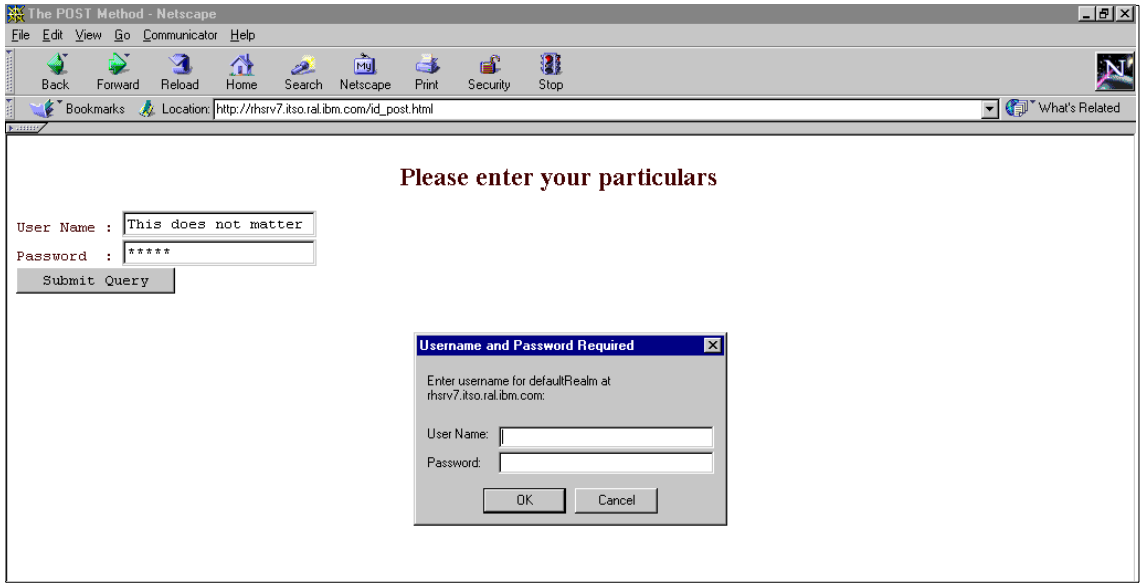


Figure 80. Asking for authentication

We entered the user ID `Popye` and his password `spinach`. The password appeared hidden by a sequence of asterisks. Remember that Popye had permission to GET and the id_post.html page uses the POST method.

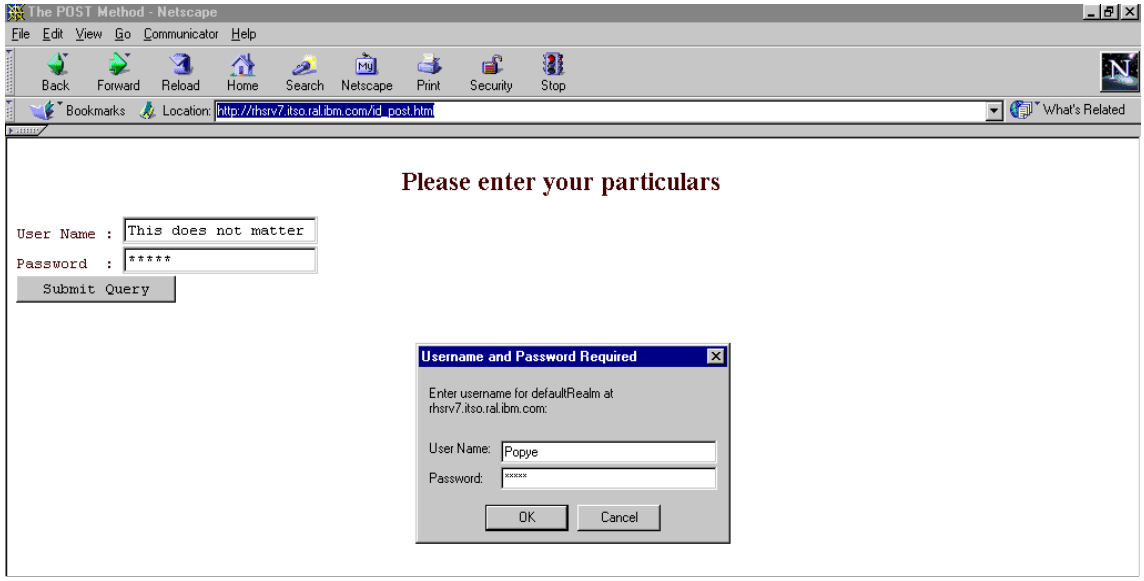


Figure 81. Entering unauthorized user ID and password

And we got the following screen:

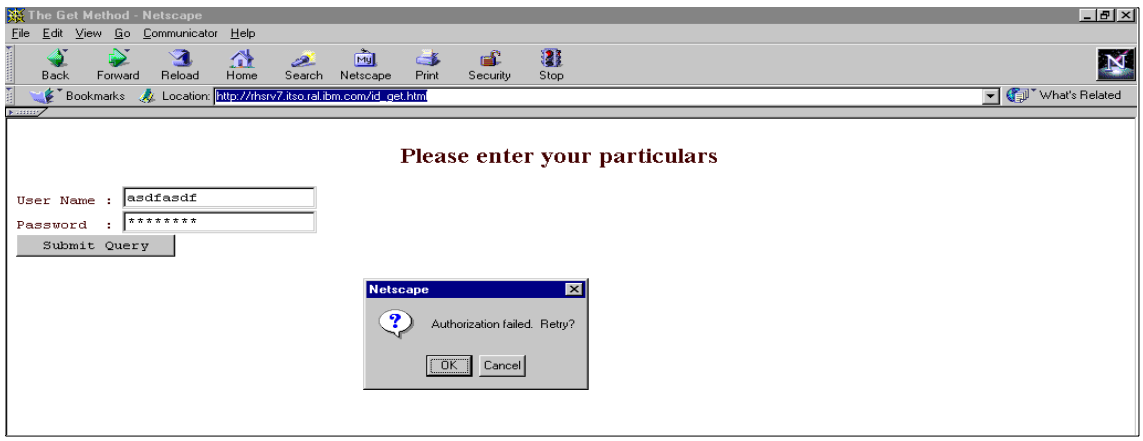


Figure 82. Authorization failed

So we understand now that the user Popye, who has no permission to POST, cannot invoke a servlet protected by WebSphere Application Server using the POST method. Then we tried clicking on the **Cancel** button, and it gave us this result:

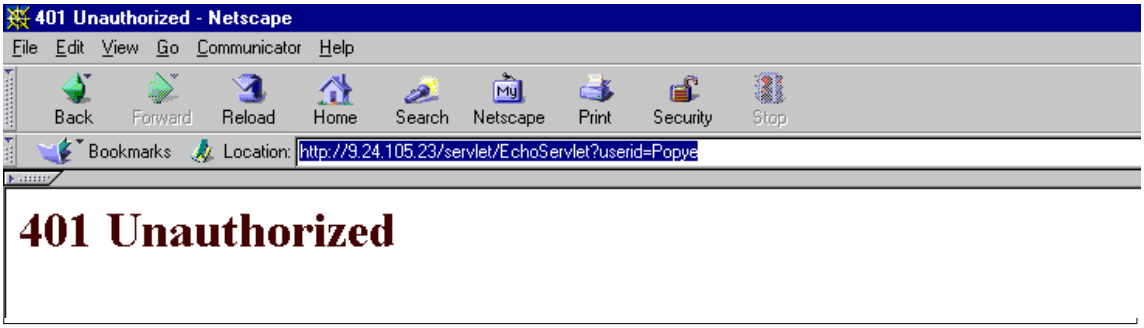


Figure 83. No Access to the servlet for an unauthorized user ID

After that, we reaccessed the id_post.html page and tried posting the information, but this time we gave the user ID Asterix and the corresponding password.

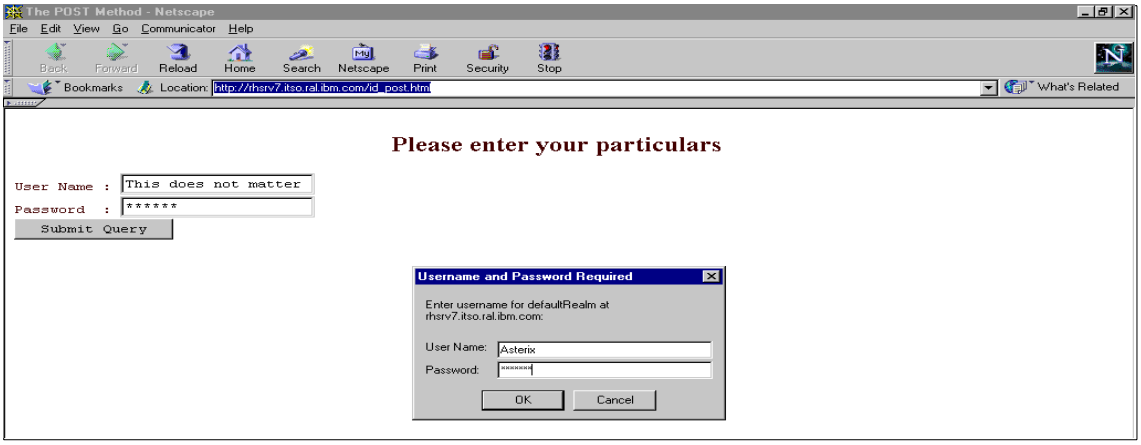


Figure 84. Accessing the servlet with the proper user ID

And since Asterix was authorized to POST, we got access to the output of the servlet as expected.

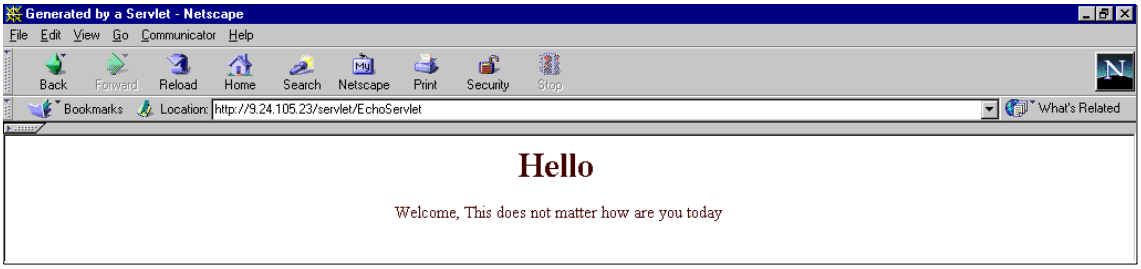


Figure 85. Got access to the servlet

With that confirmed, we tried to use the GET method. We accessed the page id_get.html, and tried submitting the information. Here is what we got:

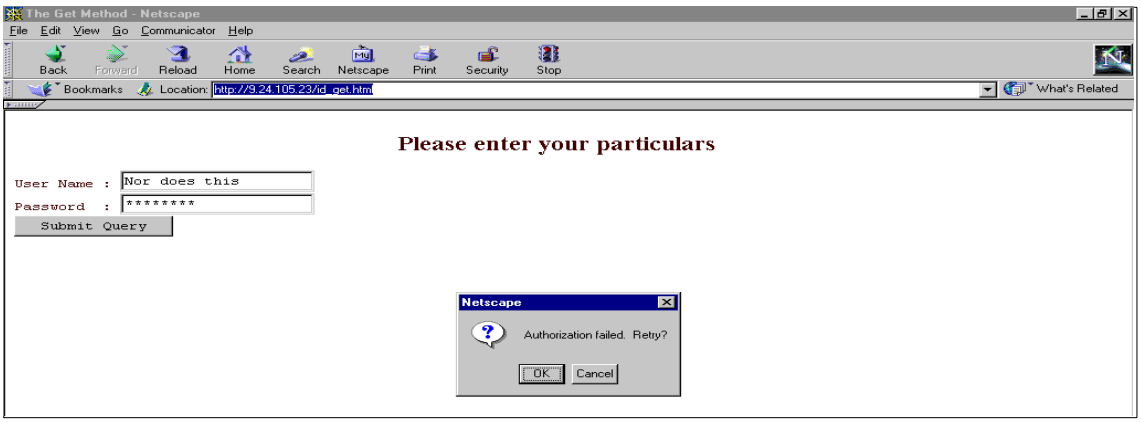


Figure 86. Accessing using the GET method

Why was that? The browser maintains the user ID and password mapped with a specific URL. This makes it convenient for the user, by not requiring him or her to key in the user ID and password with each request. The browser, however, after getting the user ID information from the user for the first time, stores it and sends it to the server each time. Hence, the browser sends the same user ID information to the server over and over again, until the server declines to accept it for some reason, upon which it pops up the window in Figure 82 on page 167.

Note here that there is no way of telling the browser that you would like to change the user ID, unless you would consider restarting the browser, upon which the mapping the browser maintains between the user ID - password pair and the URL is reset. Getting back to using our EchoServlet, we clicked

OK, entered `Popye` and his password and got regular access to the output of the servlet.

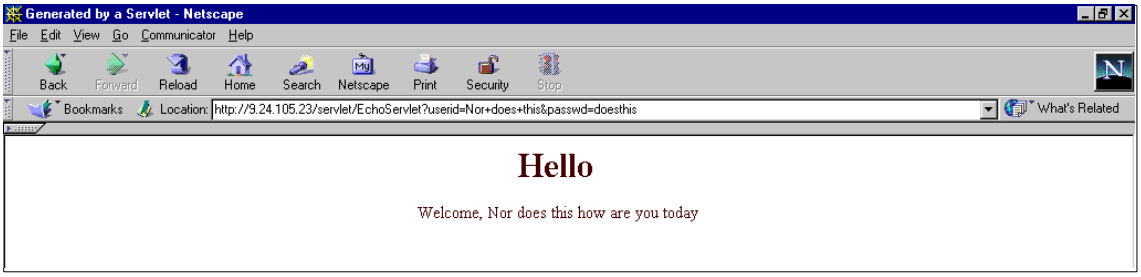


Figure 87. Accessing using the GET method with proper user ID

Everything worked fine with the user Popye, since we had granted Popye permission to GET and the HTML page id_get.html uses the GET method to access the servlet.

Once again, notice that the dynamic information with the GET method is always appended to the URL in the HTTP request header, and with the POST method it is sent through the HTTP request body (see Figure 85 on page 169).

Now let us consider an interesting case. Suppose we want to run the servlet without the HTML page - perhaps by giving the values at the URL (as in the GET method), or maybe we have a servlet that takes no input at all. One example for this could be a servlet that extracts information from a client certificate and performs a certain action such as getting some information from a database or simply welcoming the person by name. We will try to keep our example simple. So let us write a rather simple servlet, that just says `Hello` without taking any inputs. We admit that we could just as well have written a static HTML page for this functionality, but we have mentioned the use of a no GET or POST input servlet. Here is the code of a servlet that really does nothing but give out static information. The name we gave to this servlet was StaticServlet.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class StaticServlet extends HttpServlet
{
  public void init(ServletConfig conf) throws ServletException
  {
    super.init(conf);
    log(" Static Server Initialized ");
  }

  public void service(HttpServletRequest req, HttpServletResponse res)
    throws IOException, ServletException
  {
    ServletOutputStream ops = res.getOutputStream();
    res.setContentType("text/html");
    ops.println("<HTML><HEAD><TITLE>Generated by a Servlet</TITLE></HEAD>");
    ops.println("<BODY>");
    ops.println("<CENTER><H1> Hello </H1>");
    ops.println(" Welcome, partner. How are you today");
    ops.println("<BR>");
    ops.println("</BODY> </HTML>");
    ops.close();
  }
}
```

Figure 88. StaticServlet.java

What we now need to do is to restrict access to this servlet. So we add this as a resource in the CartoonsACL. For details, refer to 8.1.5, “Access control lists” on page 141.

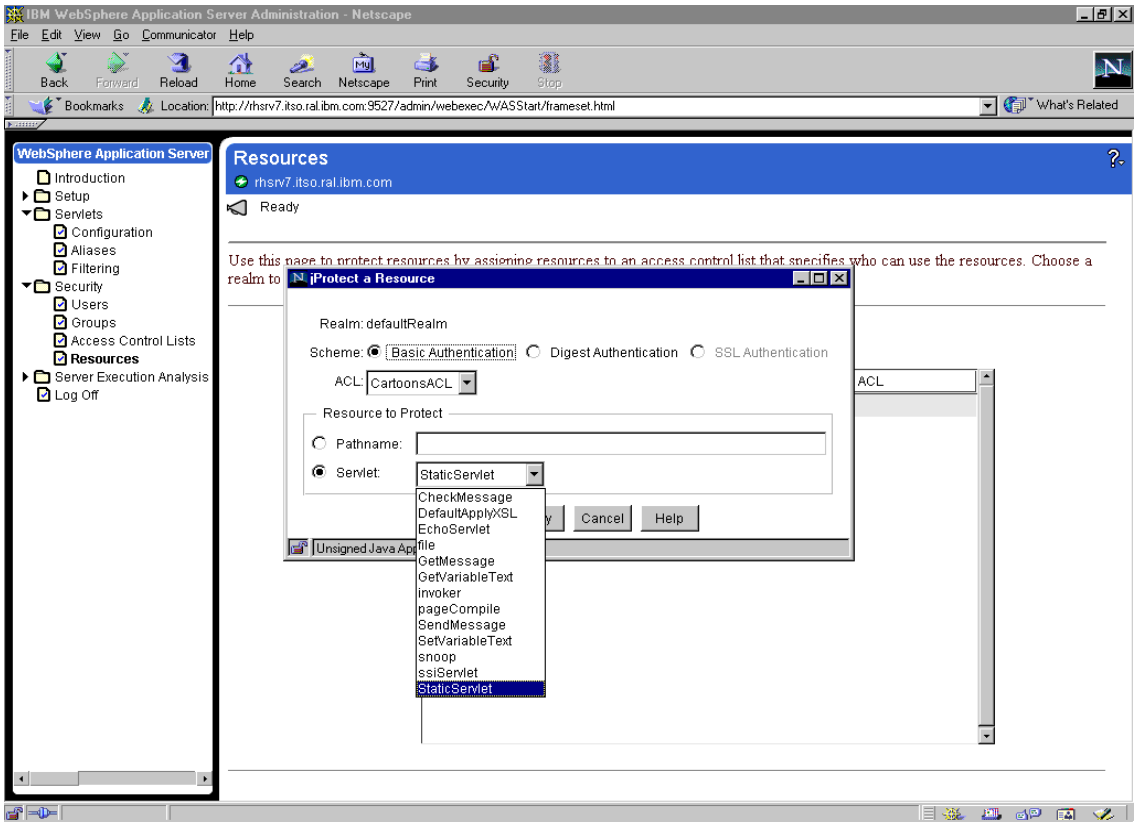


Figure 89. Addition of StaticServlet resource to the list

Note that we have not changed the permissions of either users or groups within the ACL. We have just added another resource in the ACL. Asterix still has permission *only* to POST, and Popye still has permission *only* to GET. Now which of these do you think will be able to access the servlet? If you guessed Popye, with the GET permission, you were right. As we have mentioned before, the GET method uses the same stream for the form data as well as the URL. By giving someone permission to do a GET, you are really giving permission to connect using the URL stream. Obviously, if the URL stream is not read, the particular file cannot be delivered at all. So, a GET permission is required to read a file. Further, since GET uses the same stream for data and URL, our guess is that the user that has permission to do a GET method can run a servlet from the URL. We verified this, and here are the relevant screens:

After adding StaticServlet as a resource in the CartoonACL, using a browser, we went to the servlet directly pointing to its URL `http://rhsrv7.itso.ral.ibm.com/servlet/StaticServlet`. On keying in the URL, and pressing Enter, we were prompted to give the user name and password:

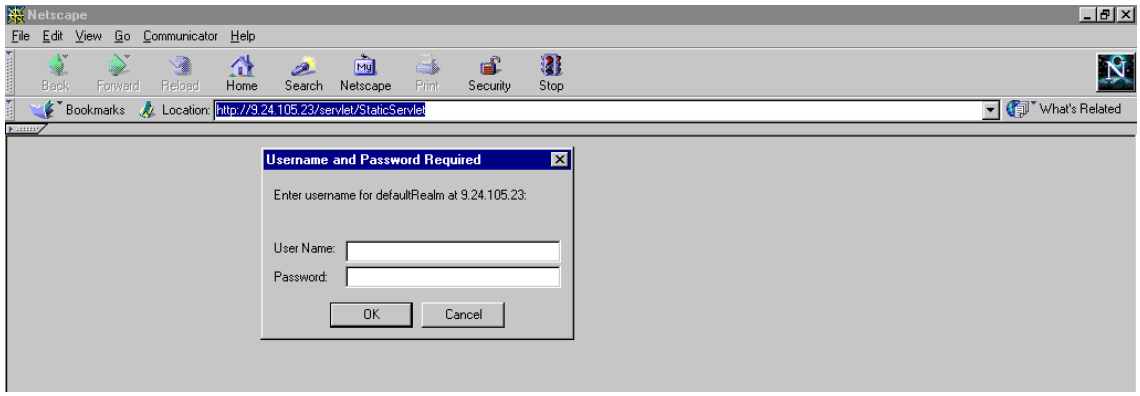


Figure 90. Prompt for user name and password to run a servlet

The system behaved pretty much as expected when we clicked **Cancel**, by giving us the panel as in Figure 83 on page 168. Also, when we tried entering the user ID as `Asterix`, since Asterix has the POST permission only, it gave us a message saying `Authorization failed` (as in Figure 86 on page 169). However, when we keyed in the user ID as `Popye` and the correct password, we got the output of the servlet.

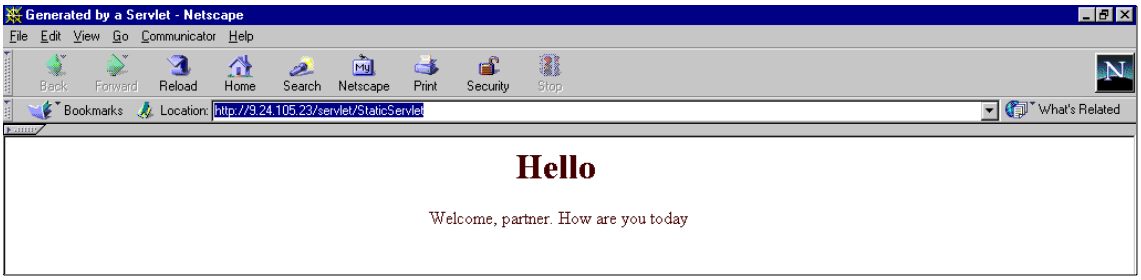


Figure 91. The output of the StaticServlet

Note that all that we found to be applicable to defaultRealm, we also found to be applicable to the UNIX realm, except for adding users and groups. We could add resources under the UNIX realm, create ACLs, add permissions for the users created in the UNIX system, and get them to use these resources by logging on through the browser.

8.2 Enterprise JavaBeans

Enterprise JavaBeans (EJBs) is a Java programming specification (currently Version 1.0 and draft Version 1.1 of the specification is available), which defines the EJB classes and the interfaces between the Enterprise JavaBeans technology-enabled server and the component.

EJB were introduced to simplify the process of extending enterprise data to the Web without having to code all of the middleware. EJB allows programmers to write large programs in reusable chunks, or JavaBeans, that can be reused, updated, and recombined to form new programs. Such component-based programming makes it possible to develop, test and maintain much more complicated programs —which may be distributed across hundreds or thousands of servers — more quickly than ever before.

> EJB: The convergence of these three concepts -- server-side behaviors written in the Java programming language, connectors to enable access to existing enterprise systems, and modular, easy-to-deploy components

Essentially, EJBs generate custom JavaBean components that leverage the "plumbing" provided by middleware, instead of forcing developers to manually code to the various APIs encountered in an enterprise. JavaBean components are Java classes that conforms with a set of rules. These rules defines a universal access method so the classes can be reused and modularized. The rules are:

1. Have a default (no-arg) constructor
2. Follow naming conventions (get/set etc)
3. Impolement serializable
4. Be in a JAR with a manifest file that specifies JavaBean
5. Optional: throw/catch property change events
6. Optional: provide a BeanInfo class to enhance design time

EJB and JavaBeans components are conceptually related, but EJB is an elaboration of the JavaBean software component specification. The EJB specification defines a container model, a definition for each of the services that this container provides to an EJB, and the management capabilities of the container.

Based on the self-contained, manageable components of Enterprise JavaBeans, this specification provides dynamic binding of new business logic to the underlying data storage and transaction services, as well as to clients, existing data and applications, the network infrastructure, and the server platform. The benefits of this dynamic adaptibility are realized in greater flexibility for deploying, managing and reusing business logic.

8.2.1 EJB Structure

The structure of an EJB is simple: several server-side Java files are archived in a JAR file, and includes a MANIFEST descriptor file that describes the JAR file's contents.

8.2.1.1 EJB classes

The developer creates EJB classes. The EJB classes are the Java classes that represent the business-logic components. Two types of EJBs have been specified: Session Beans, which perform session management to Java clients, and Entity Beans, which take on the role of mapping data sources to Java classes and therefore making these data sources transparent to users. See Figure 92 on page 175.

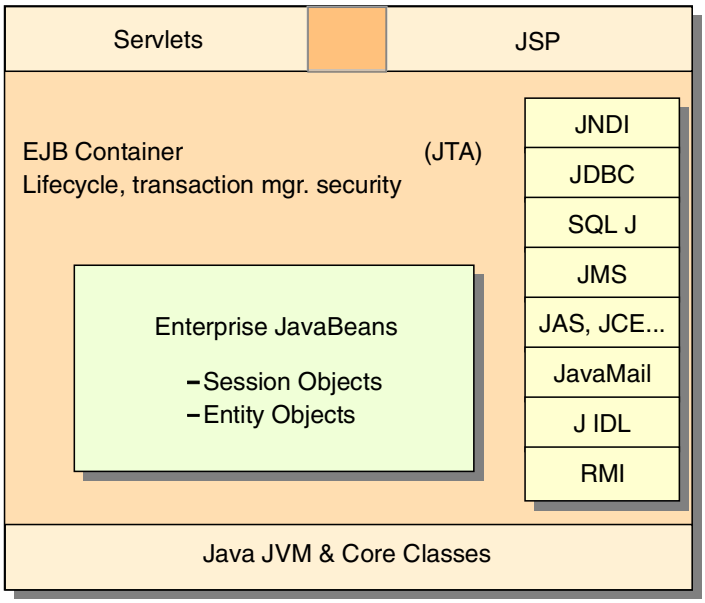


Figure 92. EJB Container

Session beans - session beans represent a process that will be performed on the server. Each client will request a service from a session bean, the client will have its own instance of the bean; instances of session beans cannot be shared among multiple clients. Session beans can be separated into two types: stateless and stateful:

Stateless beans do not store any information relating to a client between calls. For example, a bean that determines whether a part number is valid will be stateless, as it simply processes a number passed as an argument each time it is called.

Stateful beans store information between calls from a client. For example, a bean that provides the next available value in a sequence will be stateful, as it will need to remember the last value returned to a client. For the developer, it's important to know that session beans do not survive if a server is restarted; in that event, the client must reestablish a new session object

Entity beans - Entity beans map a Java class to a data source. The source could be a single row in a database, an entire table, or some form of legacy data not represented in a database. Each entity bean has a primary key associated with it that identifies the data within. It would be difficult to control changes to multiple copies of the same data, so only one instance of an entity bean exists for any given primary key in a system (even in a distributed system).

Entity Beans can be separated into two types: beanmanaged and container managed. These types refer to the way the data held in the bean is transferred to the underlying persistent storage:

1. Container-managed entity beans rely on the container in which they run to provide all database access calls. This is the simplest form for developers to use, because the developer does not need to worry about how the data is moved to and from the persistent store.
2. Bean-managed entity beans provide all the database access calls within the bean itself. The disadvantage of this type of bean is that it ties it more closely to the underlying architecture. But in the event of a server being restarted, this will be transparent to clients of entity beans.

8.2.1.2 EJB Interfaces

Each EJB component has two interfaces associated with it: EJBHome and EJBObject. The classes that represent these interfaces are created when the EJB is loaded into the container.

EJBHome - Both session and entity beans have create methods that will return an instance of the EJBObject class. There may be a number of these methods, each taking different parameters depending upon what data is required to initialize the EJB. In the case of entity beans, there will also be "finder methods," which take a primary key as a parameter and return a reference to a collection of EJBObjects matching that key (this may be one or more objects).

EJBObject - An instance of the EJBObject class is used by the client to access the methods provided by the EJB component. The EJBObject class

acts as a proxy, putting the necessary infrastructure-specific code between the client and the EJB component.

Home and Remote interfaces and classes, the role of which are to control access to EJB classes, are generated as part of the development process.

8.2.1.3 EJB deployment

A deployment descriptor and the MANIFEST are generated at deployment time.

JAR files reside in an EJB container which provide essential services (naming, state management, transactions, security, and others) to the EJBs. The run time environment, including system services and resource management, is provided by an EJB server.

8.3 Extensible Markup Language (XML)

XML is a standard data format for exchanging structured documents on the Web. XML is a World Wide Web Consortium (`www.w3c.org`) standard that lets you create your own tags. This standard extends the traditional HTML tags by providing a structured way of defining data over the Web. The data is defined using tags that describe what each piece of data is.

For example, following is a mail address formatted in HTML and XML documents:

HTML	XML
<HTML> <BODY> <P><B>IBM- ITSO</B> Dept ABC, Building 123 1001 Winstead Drive Cary, NC 27513</P> </BODY>	<ADDRESS> <COMPANY>IBM- ITSO</COMPANY> <ADDR1>Dept ABC, Building 123</ADDR1> <ADDR2>1001 Winstead Drive</ADDR2> <CITY>Cary</CITY> <STATE>NC</STATE> <ZIP>27513</ZIP> </ADDRESS>

The address stored in an HTML document provides no additional information about the data itself. The address stored in an XML document provides data attribute information. With XML, we can separate the content and presentation of information on the Web. The information content is structured and can be easily referenced by programs. Programs such as Web agents that use XML parsers will be able to extract the information from documents.

Such Web agents will be very useful for Web search engines to find documents on the Web more intelligently.

For more thorough discussions on XML, please refer to the IBM XML resource page:

http://www.ibm.com/developer/xml/

8.3.1 XML Parser

XML parsers are programs that read, modify, and write XML documents. IBM XML Parser for Java is a Java API package that lets an application developer use Java objects that represent XML and HTML documents.

8.3.2 Document Object Model (DOM)

DOM is application programming interface (API) that can be used to represent an existing XML document or generate an XML document. Using DOM, application programs can share and modify an XML document. The DOM object is created and stored in JVM's computer memory and represented as a tree data structure. This tree enables application programs to:

- Perform tree operations such as modifying the XML document strucutre or searching the XML document
- Share the document in computer memory with other programs

To create DOM objects, you must use a DOM-based XML parser such as IBM's XML Parser for Java (XML4J). Using XML4J, the DOM object is created and stored in JVM's computer memory. DOM-tree parsing requires that the entire document tree be retained in Java virtual memory (JVM). However, in the latest IBM XML4J version (V2.0.13, as of July 1999, is available at http://www.alphaworks.ibm.com), XML4J allows DOM objects to be serialized.

A simple XML document, below, will generate DOM in Figure 93 on page 179.

```
<?xml version="1.0" ?>
<state stateid="MN">
<city cityid="mn12">
<name>Johnson></name>
<population numid="5000"/>
</city>
<city cityid="mn15">
<name>Pineville</name>
<population numid="60000"/>
</city>
<city cityid="mn20">
```

```
<name>Lake Bell</name>
<population numid="20"/>
</city>
</state>
```

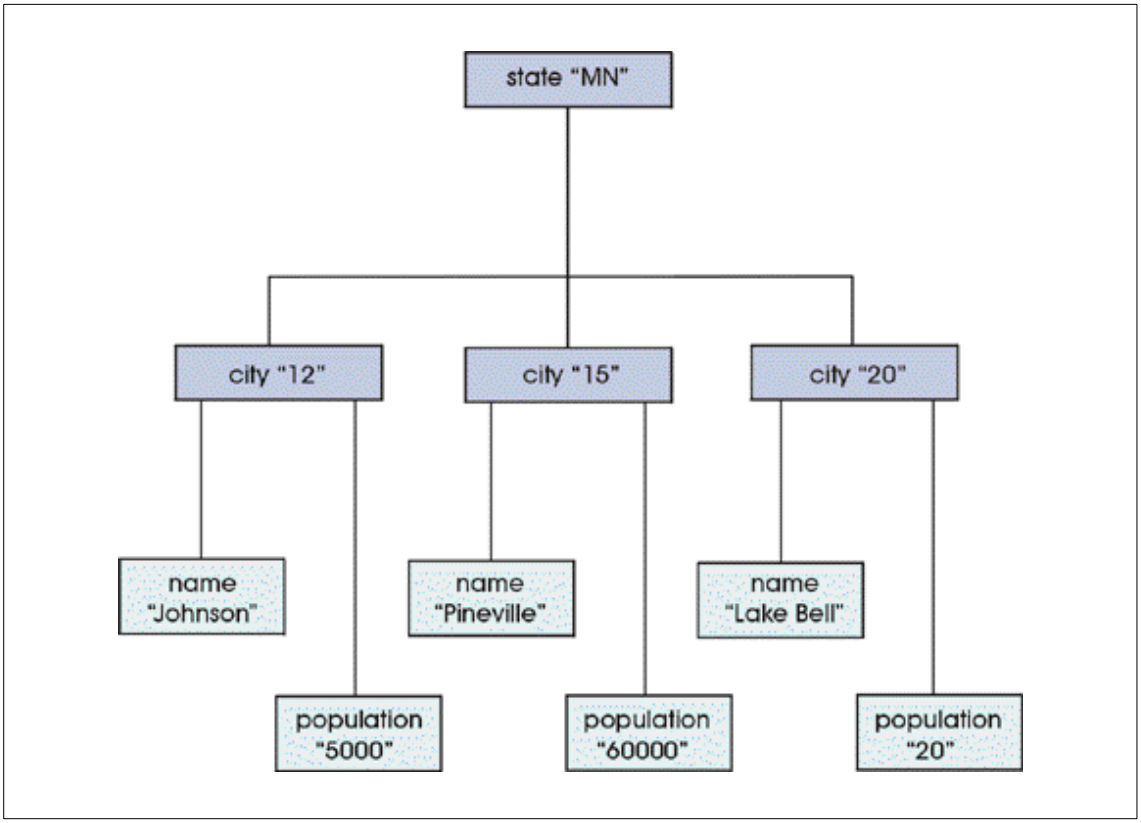


Figure 93. DOM for the sample XML state document

DOM API packages information can be found on the WebSphere Application Server directory at:

com.ibm.xml.parser package

```
http://<hostname>/IBMWebAS/doc/apidocs/Package-com.ibm.xml.parser.html
```

org.w3c.dom package

```
http://<hostname>/IBMWebAS/doc/apidocs/Package-org.w3c.dom.html
```

Where `<hostname>` is the domain name or IP address of the WebSphere Application Server.

XML4J also provides Simple API for XML (SAX) and ElementHandler APIs.

8.3.3 Simple API for XML (SAX)

SAX is an event-driven parser, unlike DOM, it does not create a tree data structure. It parses an XML document and generates events such as the start of an element or the end of an element which trigger application tasks such as reading in element names. In our mail address definition, a sequence of SAX events are:

```
startElement: ADDRESS
startElement: COMPANY
characters: IBM- ITSO
endElement: COMPANY
startElement: ADDR1
characters: Dept ABC, Building 123
endElement: ADDR1
startElement: ADDR2
characters: 1001 Winstead Drive
endElement: ADDR2
startElement: CITY
characters: Cary
endElement: CITY
startElement: STATE
characters: NC
endElement:STATE
startElement: ZIP
characters: 27513
endElement: ZIP
endElement: ADDRESS
```

Application tasks that implements event handlers defined in the packages are:

- com.ibm.xml.parser.SAXDriver
- org.xml.sax

Complete API packages can be found in the WebSphere Application Server directory under:

`http://<hostname>/IBMWebAS/doc/apidocs/com.ibm.xml.parser.SAXDriver.html`
and
`http://<hostname>/IBMWebAS/doc/apidocs/Package-org.xml.sax.html`

Where `<hostname>` is the domain name or IP address of the WebSphere Application Server.

8.3.3.1 ElementHandler

ElementHandler supports event-driven parsing of a DOM tree. Similar to SAX, an application using ElementHandler can receive events. However, the ElementHandler creates a DOM tree, while SAX does not.

ElementHandler supports the following API packages and can be found online in the WebSphere Application directory:

com.ibm.xml.parser.Parser
`http://<hostname>/IBMWebAS/doc/apidocs/com.ibm.xml.parser.Parser.html`

org.w3c.dom.Document
`http://<hostname>/IBMWebAS/doc/apidocs/org.w3c.dom.Document.html`

com.ibm.xml.parser.TXDocument
`http://<hostname>/IBMWebAS/doc/apidocs/com.ibm.xml.parser.TXDocument.html`

com.ibm.xml.parser.TXElement
`http://<hostname>/IBMWebAS/doc/apidocs/com.ibm.xml.parser.TXElement.html`

com.ibm.xml.parser.ElementHandler
`http://<hostname>/IBMWebAS/doc/apidocs/com.ibm.xml.parser.ElementHandler.html`

where `<hostname>` is the domain name or IP address of the WebSphere Application Server.

Chapter 9. Servlet design patterns for e-commerce

In this chapter we discuss different servlet design patterns for e-commerce.

9.1 Guiding principles

- e-commerce systems are very dynamic. Because the medium is changing so rapidly, and the bar is constantly being raised as to what features and implementations are state of the art, e-commerce sites evolve much more quickly than traditional online systems.
- e-commerce system development is a multidisciplinary task that requires the collaboration of a variety of disparate roles with differing skill sets, concerns, and viewpoints. Tight time-to-market requirements for e-commerce require teamwork. In addition, the integration of different media and technologies requires these teams to be heterogeneously composed.

Successful e-commerce systems will therefore be constructed with flexibility and scalability in mind. Successful e-commerce development methodologies will partition the development workload along well-defined competency lines and facilitate communication between heterogeneous development teams.

9.2 High-level design patterns

In the following we discuss single function per servlet, tiered topology, and the separation of processing and display responsibilities.

9.2.1 Single function servlets

In CGI-based applications, it is common practice to have a singular, monolithic script that receives a parameter such as action to determine the type of processing to perform. This results in brittle systems that resist change. In contrast to this, the servlet API encourages the creation of a number of processing components that each has specific functionality and responsibilities. Quality servlet applications are designed in such a manner that objects in the system be given clear and distinct responsibilities.

For example, rather than creating a monolithic servlet called Shop.class that handles all aspects of an e-commerce application, the processing involved with shopping would be broken into a number of servlets: AddItem.class, Recalculate.class, Checkout.class, and FinalizeOrder.class. This allows functionality to be redefined or added on the fly as servlets are reloaded or

added to the system. Classes with related functionality can be grouped into the same JAR file for distribution if some packaging is required.

9.2.2 Tiered topology

Constructing Web applications using a tiered structure provides a number of benefits to e-commerce development projects. Most notably, tiered structures allow for parallelized development and provide for systems that are flexible and extensible. HttpServlets should be used as a gateway to the underlying shopping application by bridging the gap between HTTP requests and their actual meaning for the business system being developed. Wherever possible, servlets should defer business policy decisions to the handling provided by the application's business components.

There are a number of different roles found in e-commerce development projects. Each role has specific responsibilities and is filled by individuals with a given skill set and perspective on the overall project. The same individual may fill more than one role over the course of the project, but as projects scale and additional staff are added, a separation between these roles develops naturally.

Graphic designers develop the look and feel for the Web site, selecting colors, fonts and page layouts that are attractive and facilitate a simple user experience. These individuals generally have an understanding of the capabilities and limitations of HTML and related client-side technologies such as CSS and JavaScript.

The work of the graphic designers is passed to HTML production staff. HTML production staff menbers translate the layouts of the graphic designers into HTML. To accomplish this they have an understanding of the inner workings of HTML and some knowledge of the HTTP protocol so they can pass information to the server using HTML <FORM> and <INPUT> tags. They should also have an understanding of JavaServer Pages (JSP), and the nodding familiarity with the Java programming language JSP development requires.

These inputs are received by code written by high-level servlet developers. High level developers take requests from online users by way of the browser and use them to manipulate server-side business objects. High-level developers are well versed in the Java language and the servlet API.

Low-level Java developers handle the inner workings of these server-side business objects. These individuals handle object synchronization and

persistence, and have familiarity with technologies and APIs such as JDBC, RMI, and CORBA.

Dividing e-commerce applications into tiers allows for parallelized development. After the capabilities of the Web application have been determined, graphic designers can begin to develop the look and feel while low level developers begin architecting business logic plumbing. As the project progresses, the graphic designers hand their work off to the HTML production staff while high-level developers begin to write servlets to manipulate business objects according to user requests.

The goal in structuring these tiered frameworks is to provide a simple interface for personnel operating at each level to communicate with the level below in terms they understand. HTML production can be given a specification outlining which servlets perform various functions, what parameters should be sent, and what values are considered acceptable. High-level developers should have a detailed Javadoc spec outlining the public methods available from service servlets and documentation about the proper use and manipulation of their business objects. Based on this they can manipulate business objects in the session and know what behaviors to expect. Low-level developers generally have their APIs already set out for them as they leverage the existing persistence and synchronization facilities while translating them to the application's business logic.

The servlet API has a number of features lending themselves to tier-based development. By deliberately constructing the class inheritance structure, different types of servlets can be created, each fulfilling roles within the e-commerce application. Interservlet communication is to be used to pass messages between these separate conceptual layers.

Servlets that extend HttpServlet can be accessed using a Web browser by making POST and GET requests. These processing servlets are used to handle interaction between end users and the business objects of the e-commerce system. Users click on links and submit forms to add items to their cart, recalculate cart quantities, submit order information, and finalize their orders.

Web applications can also define servlets that extend GenericServlet (or even implement Servlet directly). These servlets can be constructed as service servlets, providing access to a set of facilities for use within the Web application. These services can be used either by other services or by processing servlets. One specific use for these services is to mediate the interactions between the processing servlets and business object persistence mechanisms.

For example, a ProductFactory service could perform product searches and return a Vector of Product objects to the caller. This ProductFactory could be used both by a Search processing servlet that was performing searches and displaying the results to HTML through a JSP page. An XMLProductGateway processing servlet that was used to send out product information in structured fashion could also use this ProductFactory. The XMLProductGateway servlet would receive a request for XML product structures, make a request to the ProductFactory, receive the product business objects and then reformat them into XML format to be returned to the caller. In this way the conversion of database tables and rows into the business objects of the online application is centralized and leveraged across multiple client components.

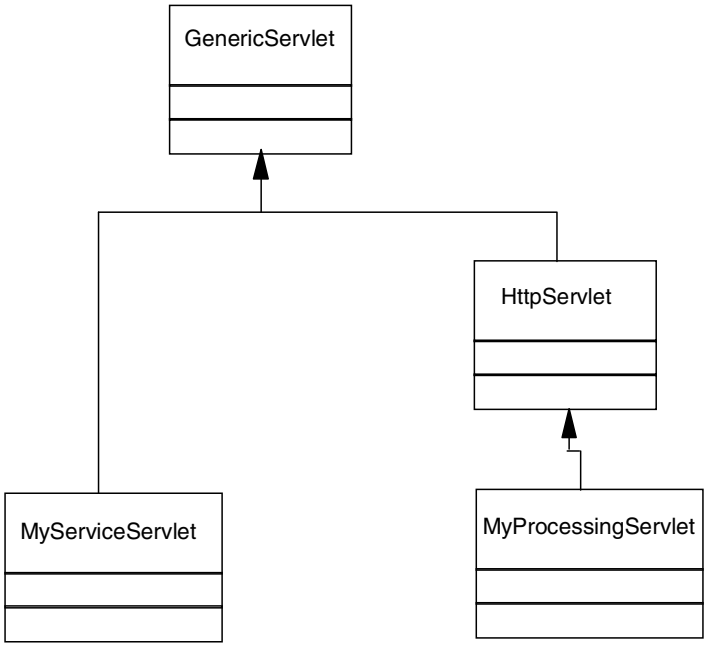


Figure 94. Tiered architecture topology

Service servlets act as intermediaries between the Web application and other systems, such as relational databases. They can also encapsulate shared business logic processing (see Figure 94).

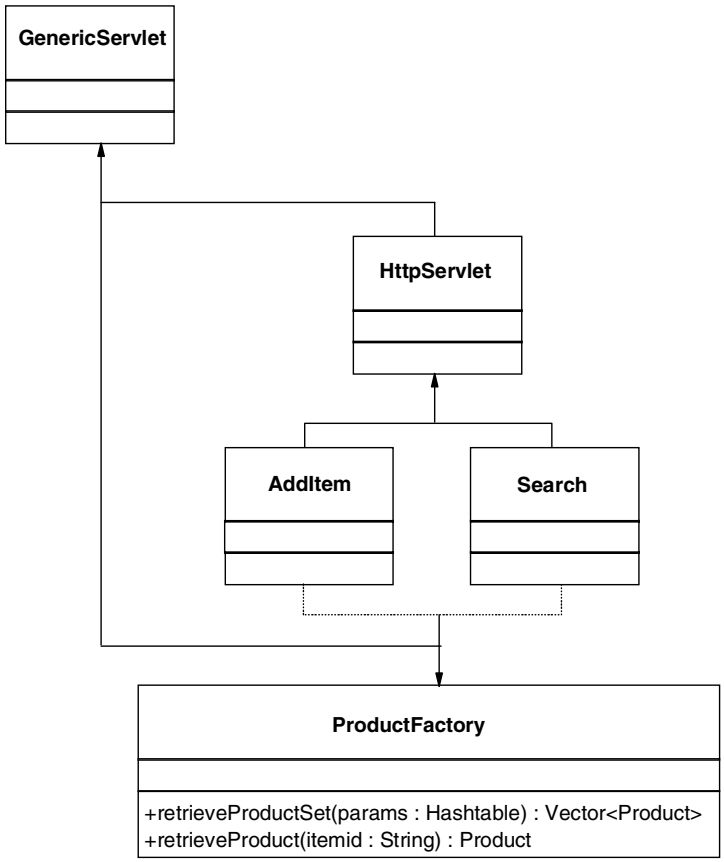


Figure 95. Example of the tiered architecture

ProductFactory is a GenericServlet subclass that handles database to Product object mappings, and the business logic associated with retrieving products. AddItem and Search are both HttpServlet subclasses that act as clients to ProductFactory, leveraging its Product generation infrastructure (see Figure 95).

In addition to providing reuse, these service servlets help componentize development, resulting in flexible and extensible systems. For example, the FinalizeOrder processing servlet needs to perform a number of tasks; it must calculate taxation and shipping information, verify that the resulting order object is valid, and persist it to a data store. If all of this functionality is tied up within the FinalizeOrder processing servlet, it will be rather brittle and subject to repeated changes as the application evolves. However, if its functionality is broken out into a group of services, these components can be updated and

replaced separately. Creating TaxCalculator and ShippingCalculator servlets and an OrderManager servlet both componentizes this functionality and makes it available across the application using the ServletContext.getServlet(String) component lookup method. The tax and shipping calculations can be reused in the shopping portion of the application to display initial values, and the order persisting service can be connected to an XML order gateway or other remote order generation facilities.

9.2.3 Separation of processing and display responsibilities

Server-side scripting systems such as ASP tend to place all code into the same location; processing code and display code are placed together. This can be convenient for small development projects, but for larger efforts where maintainability and extensibility are key factors this model can be cumbersome. Using the servlet API in conjunction with JavaServer Pages allows for the construction of Web-based applications that separate processing and display. Servlets accept HTTP requests and manipulate the business objects of the system and then defer display of the results to a separate body of code. The system business objects are the common currency used in both areas.

JSP pages should generally not have side effects. They should be used to show, not to do. If more sophisticated handling for objects is required, encapsulate the manipulation in a servlet and defer the display to a JSP page.

9.3 Specialized applications

We now discuss specialized applications.

9.3.1 Personalization

After a user session has been identified, servlets can be used to serve client-specific logos and preferences. By creating resource bundles containing information such as small logo image, large logo image, company name, and background color, a central servlet could then be used to serve out client-specific resources based on the request. Because servlets are not restricted to only serving out HTML these resources can make up a fairly significant portion of the display generation. This can be accomplished using Java Server-Side Includes to generate an <IMG> tag pointing to the appropriate image by placing <SERVLET> tags into HTML files:

```
<SERVLET CODE=com.atension.servlet.personalization.SmallImage>
```

```
</SERVLET>
```

or using a servlet reference in the HTML file to serve out the image data itself:

```
<IMG src="/servlet/com.atension.personalization.SmallImage">
```

Requests to this servlet will serve out content of the type image/gif or image/jpeg. Information in the user's session can be used to determine which images should be served in given cases.

9.3.2 Asynchronous event processing using threads

Generally, all events during an e-commerce user's shopping experience are handled synchronously, completely finishing one task before moving on to another. However, there are a number of situations where asynchronous processing might offer certain advantages. An excellent example is that of sites providing real-time credit card validation. Credit card validation requires a number of steps that can work across a number of hosts on the Internet and other networks. Because of the communication requirements, real-time credit card validation can be a slow process. A number of sites post disclaimers around links to order finalization routines warning the user that the transaction could last for quite some time and that repeatedly clicking on buttons or links can cause credit cards to be charged multiple times. Though having these disclaimers is probably better than leaving the unsuspecting user without warning, the overall quality of the application suffers due to its inability to work in what the shopper will perceive as real time.

Another approach would be to use Java's integrated threading mechanisms to cause time-consuming events to be handled asynchronously. A credit card processing servlet could maintain a pool of threads that run authorizations for credit card transactions. When a shopper attempts to finalize an order requiring credit card authorization, this authorization is queued and left to a persistent thread to handle. The user's shopping experience continues as if everything has succeeded (because as a general rule this process will be successful). If the transaction should happen to fail, the e-commerce application would be notified of the failure. If the user were still online, a message could be delivered during the session. If the shopper has already moved on, the order could be held back from fulfillment, and the user could be notified using e-mail. Though this will undoubtedly cause some confusion for customers who were declined, this will most likely be the case for all customers who have transactions fail, regardless of whether they were processed synchronously or asynchronously. The performance increases should be significant.

9.3.3 Utilizing an e-commerce event model

The servlet API has some limited event model support already built in. The SessionBindingEvent and SessionUnbindingEvent callback facilities provide the developer with the capability to execute code based on system activities. These event handlers can be especially useful for e-commerce applications. SessionBindingEvents can be used to associate sessions with global, persistent repeat user data. SessionUnbindingEvents can be used to release inventory associated with a user's cart back into the available pool.

In addition, extending this event model to handle more business-logic specific functions can greatly increase the usability and extensibility of e-commerce applications. InventoryEvents could be fired by servlets that add items to the user's cart or recalculate the quantities. These could be handled by facilities that would maintain a count of the available product levels. OrderFinalized events could be used to trigger customer-service e-mails and other back-end integration utilities. CheckoutFailure events could be used to log and analyze site usability, identifying bottlenecks.

One rather useful application of this technique can be found in logging and application tracking. There are a number approaches logging for e-commerce applications, and the specific policy applied depends on a number of factors. During development, it is important to log the status of the system and metrics on system performance to help the team building the site verify its progress. After the site has been taken live, the majority of this debugging logging activity should be removed to enhance performance. However, additional logging can be added to the system that will track user activities and provide valuable demographic and usage data. If a user comes to the site by way of a specialized link, a SessionReferred event could be fired, and that logged and correlated with subsequent AddItem and OrderFinalized events. This type of analysis is invaluable for companies deploying e-commerce solutions because it provides metrics for the evaluation of site performance and chart future development goals.

Another option is to embed plug-ins in all servlets where extensible functionality is required. When a servlet gets to a given point in processing, it scrolls through a Vector of these handlers that would seemingly provide the same functionality as event handlers. However, these plug-ins would be configured as part of the servlet, extending its functionality. Event handlers are configured separately from the servlets generating the events, making them more flexible and leveragable. Hybrid cases might also be useful.

9.3.4 Leveraging the HTTP protocol in servlet-based applications

The servlet API's access to server and HTTP protocol information allows Web applications access to a robust set of primitives in addition to servlet-specific facilities. Servlets can access information such as IP addresses and HTTP headers. While it is possible to create new login facilities and other services within the context of the e-commerce system, leveraging this existing infrastructure for online applications has the potential to decrease development time and increase maintainability. Utilizing HTTP logins also allows for better interoperability with legacy CGI applications executing on the same server and existing resource protection done using .htaccess files.

One potential use for integrating servlet applications with the HTTP protocol would be in enhancing security. A user's session for an online broker could be associated both with an HTTP authorization name and the accessing IP address. This significantly decreases the chances that malicious third parties could hijack sessions. Moving this inspection across sessions allows for the analysis of usage patterns: where does a particular customer access this Web application?, how often does her or she travel?, etc. This allows not just for demographic analysis, but also for fraud-detection for paid content providers and other account-based e-business systems.

9.3.5 Structuring parameter names and values

The naming and usage of request parameters can be used to create a number of different access methods. Servlet requests allow for two different access methods: direct requests based on a key, and a request for available keys. The most common access method is by key; if a developer wants to know what color a user specified, HttpRequest.getParameter (String) is used to query the available parameters. However, the HttpServletRequest.getParameterNames (String) method also allows the developer to work through an array of all available parameters.

When a user is attempting to update the quantities in the shopping cart and each item in the cart can be referenced by a unique key, using a prefix such as qty_ for the names of all parameters indicating new quantities for a given key allows the developer to search through all available parameters, find those whose name starts with the string qty_, and update the cart's quantities based on the value of that parameter. A similar technique could be used for items that should be removed outright from the cart, perhaps prefixing those parameter names with del_. This allows both operations to occur concurrently through the same request while decreasing the possibility of name collisions or problems if additional functionality is added at a later date. Similar

techniques could be used for workflow applications where a number of similar items are submitted and must be maintained.

9.3.6 Non-cookie-based state maintenance

As a general rule, state maintenance for Web applications is accomplished through the use of browser cookies. Servlet chaining can be used to provide additional state maintenance facilities on top of the standard servlet engine behavior. Most servlet engines handle state maintenance by automatically using browser cookies or requiring manual rewriting of URLs output by servlets using the HttpServletResponse.encodeUrl (String) method. This can be cumbersome for servlet developers who would rather use either HTML object libraries or JSP to generate dynamic HTML. In addition, the user state would be lost when browsing through .html or .htm files if cookie-based session tracking were disallowed due to browser incompatibility or by passing through firewalls that strip such headers. Rather than depend upon cookies for these applications, it is also possible to maintain a state in a browser and firewall-independent manner using URL rewriting.

A state maintaining parser servlet can be set to handle requests for all text/html MIME type output before ultimately being returned to the client browser. This parser servlet would copy all HTTP headers from its request (the response of the previous servlet or the HTML page being returned) and parse through the HTML being returned to the user, rewriting all appropriate tags such as <A>, <AREA>, <FRAME>, and inserting hidden <FORM> variables.

9.3.7 Servlet-based cron facility

Another useful facility stemming from the ability of Java servlets to spawn off independent threads of execution is the ability to set up scheduled events within the Java VM. Similar to the UNIX cron facility, a system timer thread could be used to perform a variety of polling, execution, and analysis functions in the background.

This immediately raises the question, "Why not use the existing operating system scheduling facilities?" The primary reason for moving this responsibility in-process for Web-based applications is the ability to centralize the administration of Web application maintenance tasks and maintain control over event synchronization.

Having the system event scheduling inside the Java VM allows for simple Web-based administration tools for event scheduling that is integrated with the rest of the application's administration tools and accessible from

throughout the Web application. While it is possible to create gateways to the VM's processing using RMI or servlet gateways (as in the XML data transfer example), there is value in the integration.

Also, holding system maintenance events within the Java VM allows developers to leverage Java's built-in synchronization primitives. Web applications tend to have a variety of internal caches and object stores that are directly accessible from inside the VM, and maintenance tasks tend to affect these objects. Having administration inprocess allows easy access and fine-grained control so that performance and correctness can be maintained. For example, if an e-commerce site caches Product objects in an internal data structure, using the database only for searching, and the database needs massive uploads and changes on a nightly basis, if these maintenance tasks are handled from within the Java process the object cache can be kept in sync with changes to the database as they are made, without creating any gateway facilities that allow for out of process communication. Also, if an e-commerce site communicated with backend fulfillment systems on a batch basis and maintains a single batch file of orders for a day's transactions, all access to the contents of the file can be controlled through the VM without relying on operating system-dependent file locking mechanisms. New orders can be added and batches can be rolled over without fear of race conditions or inconsistent states.

9.3.8 Dynamically generated images

It is possible to use Java's imaging capabilities and third-party libraries to dynamically generate images on the server side. These images are retrieved by way of an image-serving servlet. Examples of Java-based image manipulation can be seen on the Range Rover Web site (`http://www.landrover.com`), where an applet-based configuration tool allows the user to select truck body styles and colors, wheels, and accessories. This does an excellent job of showcasing their product, but it is rather slow and makes significant requirements on client browsers to support the application. The same functionality could be encapsulated in a server-side application that would use request parameters and cookies to configure a GIF or JPEG image served back to the user:

```
<IMG SRC=http://www.whatever.com/servlets/carimage?body=truck&color=red>
```

This architecture allows for very exciting e-commerce shopping applications. A page or series of pages could be created allowing a shopper to view different styles for shirts, pants, and shoes. These could be mixed in any combination. Traditionally, these types of applications have been done using extensive client-side scripting and programming, placing the requirement of

compatibility upon the shopper client rather than the server. Server-side images that are dynamically generated remove these responsibilities from the comparatively fragile client environment and place them onto the more predictable, stable server.

This approach is most useful for cases where there is a large enough quantity of images requiring the same dynamic modification before being served back to the user. With small quantities and/or fixed modifications that must be made, it makes sense to cache all the available permutations with actual files in the file system. However, if there is a large quantity of images that must be altered, and if the alterations either change over time or are specific to the individual viewer, dynamic generation (with perhaps some object or file-based caching) can greatly increase the simplicity and maintainability of the system.

9.3.9 HTML components to aid in JSP processing

JSP leverages the power of Java to create a flexible server-side scripting language for generating dynamic HTML. However, there are certain repetitive tasks when generating HTML that can be cumbersome and result in cluttered JSP pages. By using HTML generation component objects some of these repetitive tasks can be encapsulated, resulting in cleaner JSP code that is easier to understand and maintain.

An excellent example of the use of this technique can be found when trying to output HTML <SELECT> boxes that should be filled in with the current month value. Although there may be other approaches, writing pure JSP code to handle this problem could look something like this:

```
<SELECT name="yearbox" value="year">

<OPTION value="2000"<% if("2000".equals(str) out.print(" SELECTED");%>>2000

<OPTION value="2001"<% if("2001".equals(str) out.print(" SELECTED");%>>2001

<OPTION value="2002"<% if("2002".equals(str) out.print(" SELECTED");%>>2002

<OPTION value="2003"<% if("2003".equals(str) out.print(" SELECTED");%>>2003

<OPTION value="2004"<% if("2004".equals(str) out.print(" SELECTED");%>>2004

<OPTION value="2005"<% if("2005".equals(str) out.print(" SELECTED");%>>2005

<OPTION value="2006"<% if("2006".equals(str) out.print(" SELECTED");%>>2006

<OPTION value="2007"<% if("2007".equals(str) out.print(" SELECTED");%>>2007

<OPTION value="2008"<% if("2008".equals(str) out.print(" SELECTED");%>>2008
```

```
<OPTION value="2009"<% if("2009".equals(str) out.print(" SELECTED");%>>2009

</SELECT>
```

However, it is possible to create a component that would encapsulate all of this decision logic. Usage of such a component could look like this:

```
<%

YearBox yb = new YearBox (2000, 2009);

yb.setSelected (str);

yb.streamTo (out);

%>
```

The code utilizing the component is much cleaner and simpler. These components could also potentially build up code to increase performance by reducing string comparisons or other operations associated with the pure-JSP implementation. In addition, it would be possible to generalize the behavior of the <SELECT> box into a base class to be extended for specific cases such as this. This is an excellent scenario because <SELECT> boxes are fairly standard to display. Trying to extend this technique past simple examples can tend to generate Java code that is just as complicated as the HTML to be generated, but the benefits gained from simple applications are significant. There are a number of freely available toolkits that can be used for generating HTML, or as in this case it is fairly simple to create a robust toolkit of reusable components.

9.3.10 Summary

E-commerce applications present a number of challenges to architects and developers. By applying several simple architectural patterns, and exploiting the power and flexibility of the servlet API developers can construct robust, scalable, and flexible Web applications. Also, the power of the Java language and associated APIs provide a rich base for extending Web application functionality.

Part 4. DB2 Universal Database

Part four discusses different methods of accessing a DB2 database.

Chapter 10. Accessing DB2 data

As well as providing a relational database to store your data, DB2 lets you issue requests to administer, query, update, insert, or delete data (among other functions) using local or remote client applications. An application can access DB2 data over the Web or over a LAN connection. This data can reside on host, nonhost, or AS/400 machines that are running DB2 or a select number of the world's most popular database management systems.

The focus of this book is accessing DB2 data over the Web. However, this chapter will briefly look at the different database access methodologies that DB2 offers and the most common implementations. This chapter will first lay down the foundation of database access and then build on these examples to create an e-commerce solution over the Web.

DB2 clients provide a run-time environment that enables client applications to access one or more remote databases. With a DB2 Administration Client, you can remotely administer a DB2 server. Local applications, and all Java applications (either local or remote), access a DB2 database through a DB2 client. All remote applications that are not Java applets must have a DB2 client installed on the machine before they can access a DB2 database. A DB2 client is part of any DB2 server installation. For more information on DB2 clients, refer to the DB2 documentation.

DB2 Version 6.1 clients are supported on the following platforms:

- Windows NT, Windows 95, and Windows 98
- UNIX (AIX, HP-UX, Linux, SGI IRIX, and Solaris
- OS/2

Pre-Version 6.1 DB2 Clients for the following platforms are available for download from the Web:

- DOS
- Macintosh
- SCO Open Server and SCO UnixWare 7
- SINIX
- Windows 3.x

To obtain these client, connect to the IBM DB2 Clients Web site at:
`www.software.ibm.com/data/db2/db2tech/clientpak.html`.

10.1 Accessing DB2 data from remote clients over a LAN connection

The following diagram shows a server that is being accessed by local and remote applications. Remote applications must have the appropriate DB2 client installed to access data on the remote servers:

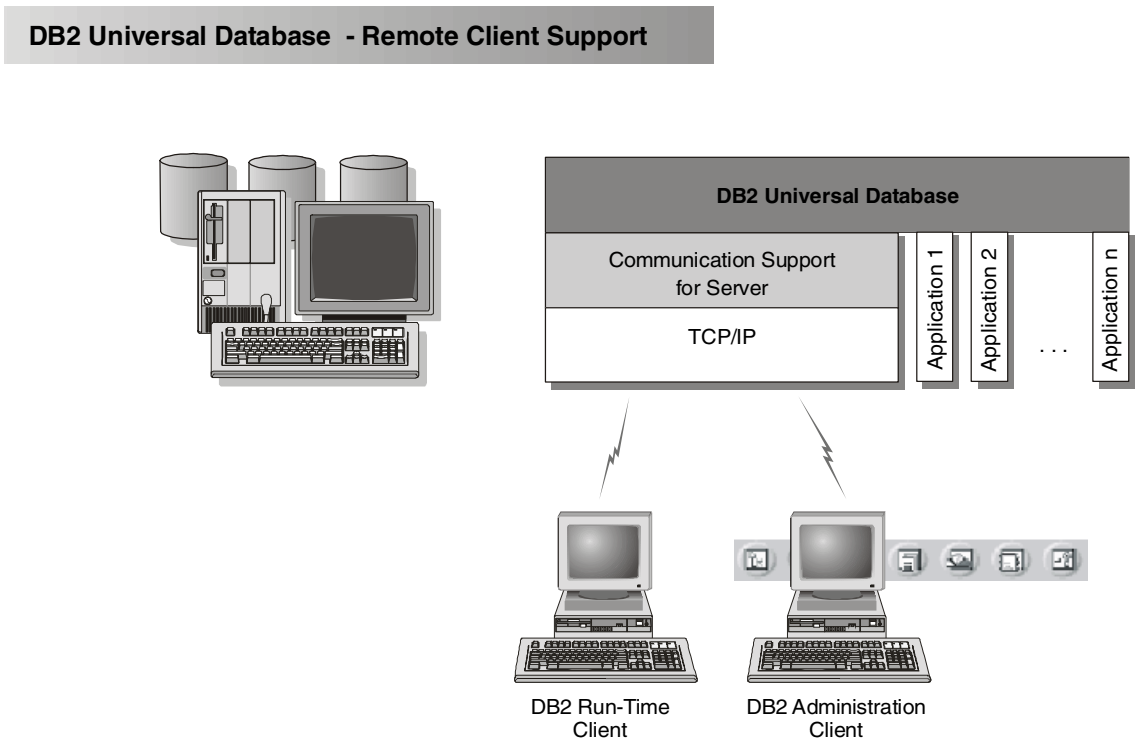


Figure 96. Remote client control

In this example, database access is not via JDBC or a Web connection. This example illustrates your typical connection scheme in an office. This method of access would not be suitable for e-commerce, since all of your customers would have to have a DB2 Client.

10.2 Accessing host or AS/400 DB2 data over a LAN connection

A DB2 server with the DB2 Connect Server Support feature installed enables DB2 clients on a LAN access to data that is stored on host or AS/400 systems. The DB2 Connect Server Support feature is part of DB2 Enterprise Edition.

Applications are provided with transparent access to host or AS/400 data through a standard architecture for managing distributed data. This standard is known as Distributed Relational Database Architecture (DRDA). Use of DRDA allows your applications to establish a fast connection to host and AS/400 databases without expensive host components or proprietary gateways.

A great deal of the data in many large organizations is managed by DB2 for AS/400, DB2 for MVS, DB2 for OS/390, or DB2 for VM. Applications that run on any of the supported platforms can work with this data transparently, as if a local database server managed it.

In addition, you can use a wide range of off-the-shelf or custom-developed database applications with DB2 and its associated tools. For example, you can use DB2 products with:

- Spreadsheets, such as Lotus 1-2-3 and Microsoft Excel, to analyze real-time data without having the cost and complexity of data extract and import procedures.
- Decision support tools, such as Business Objects, Cognos, and Crystal Reports, to provide real-time information.
- Database products, such as Lotus Approach and Microsoft Access.
- Development tools, such as PowerSoft PowerBuilder, Microsoft Visual Basic, and Borland Delphi, to create client/server solutions.

Figure 97 on page 202 shows a server that is being accessed by local and remote applications. These applications are accessing DB2 data from host and non-host DB2 servers.

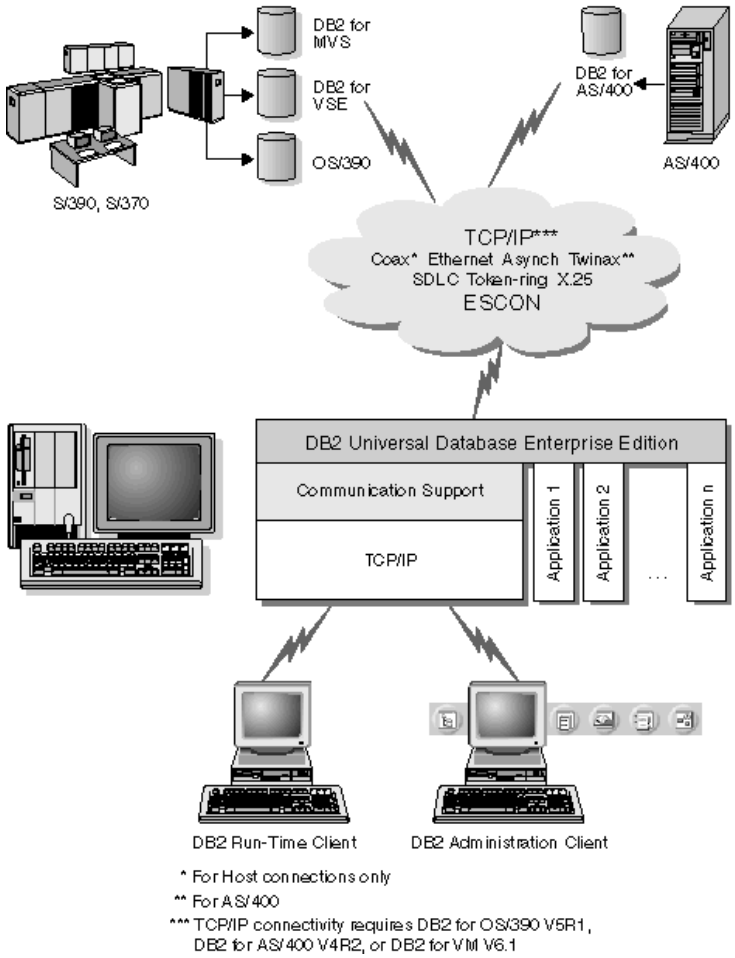


Figure 97. Accessing DB2 data from host and non-host DB2 servers

In this example, database access is not via JDBC or a Web connection. This example illustrates your typical connection scheme in an office that needs access to data residing on host or AS/400 machines. This builds upon the previous example in the sense that the DB2 data sources are also on host or AS/400 machines. Though the access method did not change, the breadth of data access did. This is a typical scenario for many large organizations. This method gives us access to a broader range of DB2 data. However, it would not be suitable for e-commerce since all of your customers would have to have a DB2 Client.

This example could be widened to include some of the world's leading database management systems, for example, Oracle. You can access heterogeneous data sources using DataJoiner. For more information, refer to your DB2 documentation.

10.3 Accessing DB2 data from the Web using java

Java Database Connectivity (JDBC) and Embedded SQL for Java (SQLJ) are provided with DB2 to allow you to create applications that access data in DB2 databases from the Web.

Programming languages containing embedded SQL are called host languages. Java differs from the traditional host languages C, COBOL, and FORTRAN, in ways that significantly affect how it embeds SQL:

- SQLJ and JDBC are open standards, enabling you to easily port SQLJ or JDBC applications from other standards-compliant database systems to DB2.
- All Java types representing composite data, and data of varying sizes, have a distinguished value, *null*, which can be used to represent the SQL NULL state. This gives Java programs an alternative to NULL indicators that are a fixture of other host languages.
- Java is designed to support programs that, by nature, are heterogeneously portable (also called "super portable" or simply "downloadable"). Along with Java's type system of classes and interfaces, this feature enables component software. In particular, an SQLJ translator written in Java can call components that are specialized by database vendors in order to leverage existing database functions such as authorization, schema checking, type checking, transactional, and recovery capabilities, and to generate code optimized for specific databases.
- Java is designed for binary portability in heterogeneous networks, which promises to enable binary portability for database applications that use static SQL. You can run JDBC applets inside a Web page on any system with a Java-enabled browser, regardless of the platform of your client. Your client system requires no additional software beyond this browser. The client and the server share the processing of JDBC and SQLJ applets and applications.

The JDBC server and the DB2 client must reside on the same machine as the Web server; remember, DB2 server is installed with a DB2 client. The JDBC server calls the DB2 client to connect to a local, remote, host, or AS/400

databases. When the applet requests a connection to a DB2 database, the JDBC client opens a TCP/IP connection to the JDBC server on the machine where the WebSphere is running.

Figure 98 is an example of a Java-enabled browser accessing data from remote DB2 databases. This scenario is considered a two-tier configuration because the DB2 server and WebSphere reside on the same machine. In this scenario, the bundled DB2 client on the DB2 server is being used.

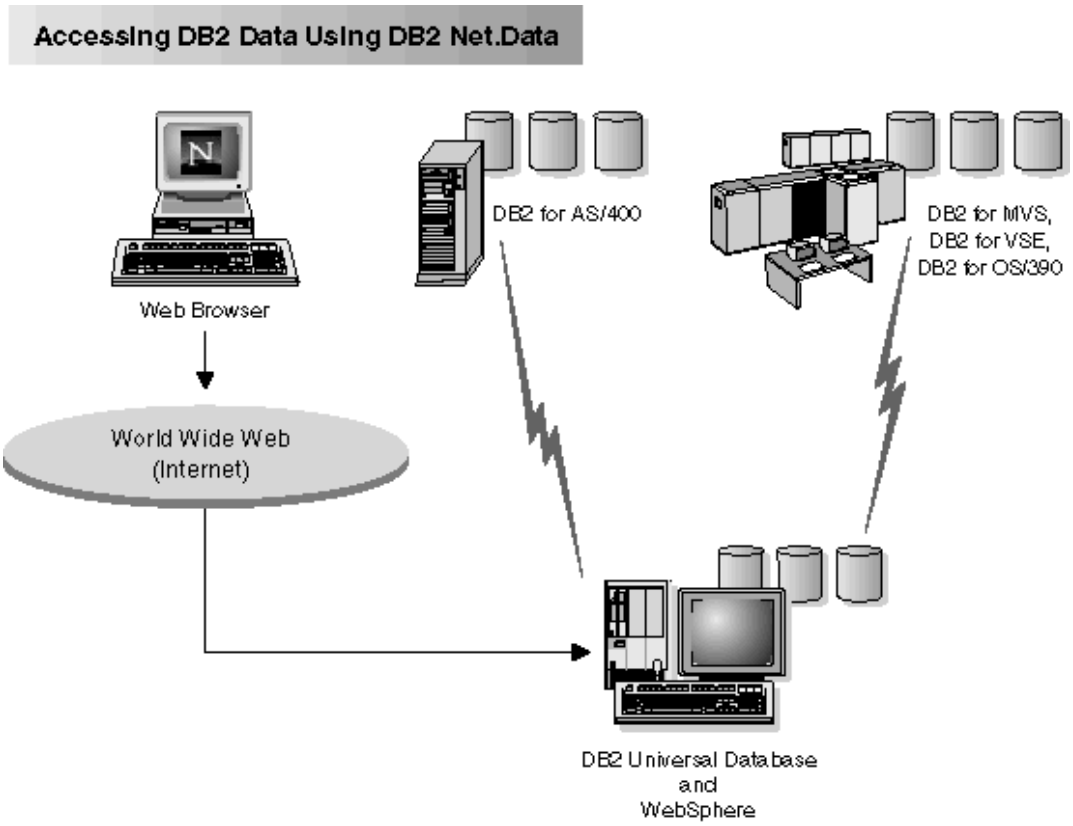


Figure 98. Accessing a two-tier configuration

In this scenario, the DB2 Server and the WebSphere server must reside on the same machine.

This same connection could also be accomplished using a three-tier configuration as in Figure 99 on page 205.

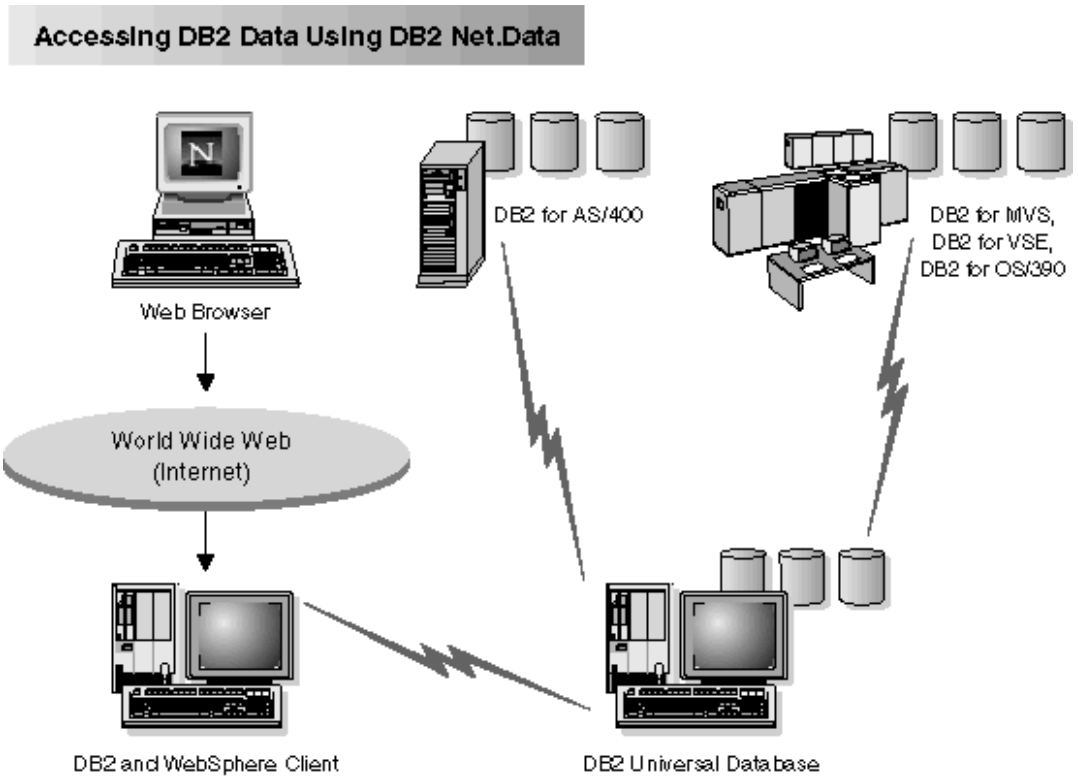


Figure 99. Accessing a three-tier configuration

In this scenario, the DB2 client and the WebSphere server must reside on the same machine, but the DB2 server resides on another tier. To install a DB2 client, refer to your DB2 documentation.

You can use JDBC to create applications that are stored on a WebSphere server and viewable from any Web browser, despite the operating system on which it is running. While viewing HTML documents, users can select automated queries or define new ones that retrieve specified information directly from a DB2 database.

Automated queries do not require input; they are links in an HTML document and, when selected, trigger existing SQL queries and return the results from a DB2 database. These links can be triggered repeatedly to access current DB2 data. Customized queries require user input. Users define the search characteristics on the Web page by selecting options from a list or by entering values in fields. They submit the search by clicking on a push button and the

dynamic SQL statement is generated and sent to the DB2 server for processing.

These are the most typical configurations for e-commerce solutions. Users from around the world can access your applications over the Web. The user could fill out forms that would then be converted to SQL and used to search a product database. If the product is in stock, the user could order it over the Web and the vendor's inventories would be update automatically.

The sample scenarios in this book focus on a two-tier or three-tier JDBC configuration. For more information on these samples, see "Sample Java JDBC programs" on page 209, "Sample servlets without ConnMgr" on page 225 and "Sample servlets using ConnMgr" on page 245.

For more information on Java enablement, refer to the DB2 Java Enablement Web page at `http://www.software.ibm.com/data/db2/java/`. For more information on the JDBC API, point your browser to `http://splash.javasoft.com/`.

Part 5. Sample Scenarios

This part provides a series of samples that will help you to better understand the applications we are working with in this book.

Chapter 11. Sample Java JDBC programs

This chapter contains two sample stand-alone Java JDBC programs that will assist you in testing both local and remote JDBC access to a DB2 instance. The programs are largely self-explanatory. Also included is a copy of the “javaprofile” file that we used to establish access to the needed Java classes for compiling both the stand-alone programs and our Java servlets. You can compile and run both these programs. The full source code is listed here.

The output from running the first program, DB2appProgram1.java, is shown in Figure 100 (in this sample there is no typed input as the pre-programmed defaults were taken on each input line). The program asks the user to enter a database name followed by the user ID then the password. If a connection can be made using the input information provided, the program then responds with a prompt >. The user can then type in an SQL query such as:

```
select * from org
```

To execute a typed-in query, the user types in `go` at the next prompt and the query is then executed. The programs will continue to run until the user types in the keyword `quit`. The advantage of these stand-alone programs is that you can validate both local and net access to your DB2 instance in advance of testing a connection through servlets and WebSphere.

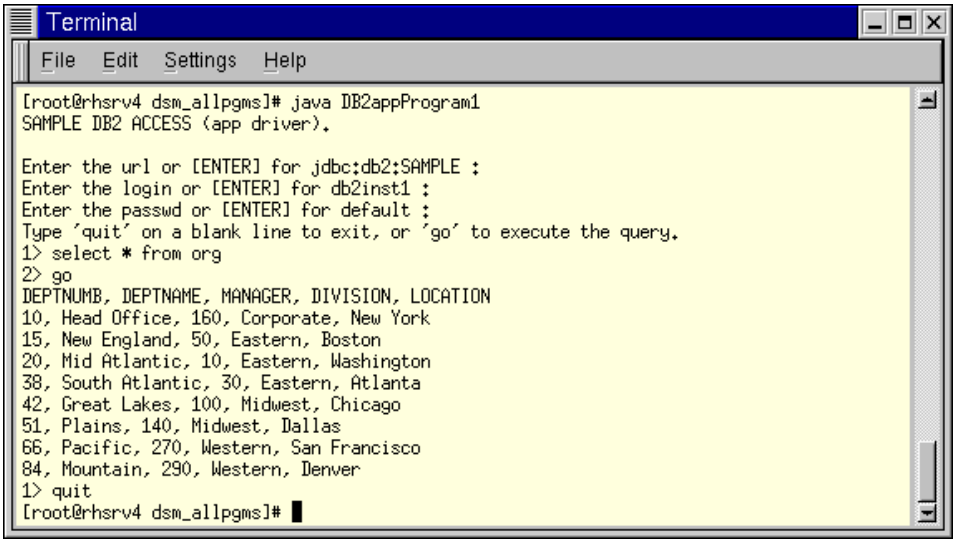


Figure 100. Sample run of program DB2appProgram1.java

This second program DB2netProgram1.java (Figure 101) runs just like the first program but uses the net driver instead, which allows access to remote databases connected via TCP/IP. The only typed-in difference is the URL, which in this program must include the type of connection, the subtype, the host computer, and the port that the `db2jstrt` command was started with.

For example, let's assume you have started DB2 on your server by logging into your db2instance and running `db2start`, and then you started the TCP/IP jdbc interface using the `db2jstrt <port>` command, as follows:

```
db2jstrt 8083
```

You should then have DB2 running and waiting for remote connections on port 8083. If our server had an Internet IP address of dbserver1.ibm.com and we wanted to query a table in the sample database, we would enter the URL:

```
jdbc:db2://dbserver1.ibm.com:8083/sample
```

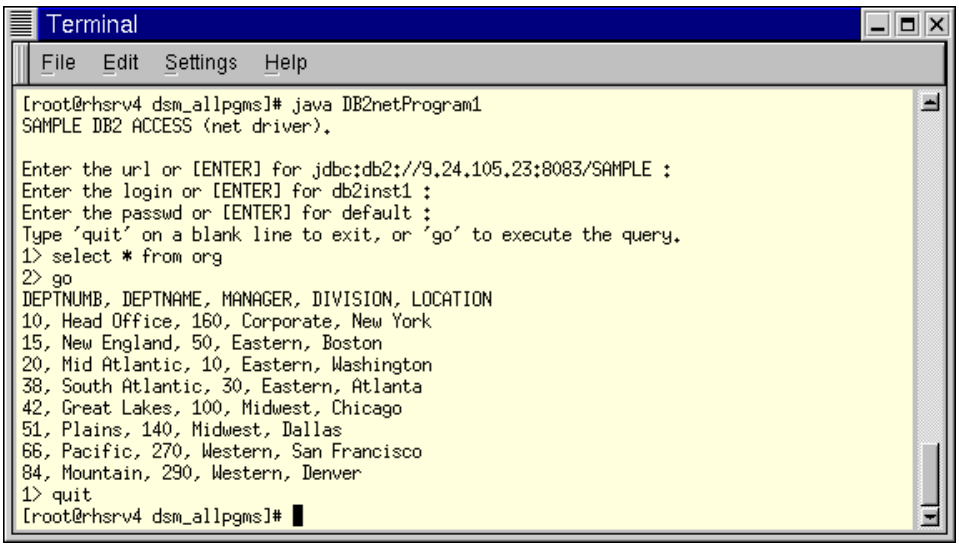


Figure 101. Sample run of program DB2netProgram1.java

We have also included the db2profile script we used when we wanted to run the programs. The DB2 profile was copied from our db2inst1 account after we had installed DB2.

In setting up your javaprofile and db2profile, you will need to create your own profiles with your own directory path names. Ours are just a guide. The profiles should be in your home directory and run using the `source` command as follows:

```
source javaprofile

source db2profile
```

Once the profiles are set you can compile Java programs and servlets as well as run the Java programs.

11.1 Script of javaprofile

```
DB2INSTANCE=db2inst1
JAVA_HOME=/usr/lib/jdk116
PATH=$PATH:$JAVA_HOME/bin

CLASSPATH=$CLASSPATH:.
CLASSPATH=$CLASSPATH:/opt/IBMWebAS/lib/ibmjndi.jar
CLASSPATH=$CLASSPATH:/opt/IBMWebAS/lib/jst.jar
CLASSPATH=$CLASSPATH:/opt/IBMWebAS/lib/xml4j.jar
CLASSPATH=$CLASSPATH:/opt/IBMWebAS/lib/jsdk.jar
CLASSPATH=$CLASSPATH:/opt/IBMWebAS/lib/ejsclientruntime.jar
CLASSPATH=$CLASSPATH:/opt/IBMWebAS/lib/x509v1.jar
CLASSPATH=$CLASSPATH:/opt/IBMWebAS/lib/ibmwebas.jar
CLASSPATH=$CLASSPATH:/opt/IBMWebAS/lib/databeans.jar
CLASSPATH=$CLASSPATH:/opt/IBMWebAS/lib/lotusxsl.jar
CLASSPATH=$CLASSPATH:/opt/IBMWebAS/lib/jndi.jar

CLASSPATH=$CLASSPATH:/home/doug/dsm_apps:/home/doug/dsm_servlets
CLASSPATH=$CLASSPATH:/usr/IBMdb2/V6.1/java/db2java.zip

export CLASSPATH JAVA_HOME PATH DB2INSTANCE
```

Figure 102. Sample javaprofile file used in our project

A javaprofile is needed so you can establish access to the various Java zip and jar files needed to compile both Java programs and Java servlets.

As you can see in Figure 102, we have included a CLASSPATH pointer to the DB2 db2java.zip file that is needed for loading the jdbc app or net driver, depending on which program you run.

There are many ways you can establish access to your Java classes. We think that running the above script using the `source` command is a simple and tidy way of keeping track of CLASSPATH variables within your own login account.

11.2 Script of db2profile

```
#############################################################################
#
# NAME:     db2profile
#
# FUNCTION: This script sets up a default database environment for
#           Bourne shell or Korn shell users.
#
# USAGE:    . db2profile
#           This script can either be invoked directly as above or
#           it can be added to the user's .profile file so that the
#           database environment is established during login.
#           A user may also copy this script into their directory
#           structure and customize it.
#
#############################################################################

# Default DB2 product directory
DB2DIR="/usr/IBMdb2/V6.1"

#----------------------------------------------------------------------
# DB2INSTANCE [Default null, values: Any valid instance name]
# Specifies the instance that is active by default.
#----------------------------------------------------------------------
DB2INSTANCE=db2inst1
export DB2INSTANCE

INSTHOME=/home/db2inst1

#----------------------------------------------------------------------
# First remove any sqllib entries from the user's path.
# Add the directories:
#   INSTHOME/sqllib/bin  - database executables
#   INSTHOME/sqllib/adm  - sysadm executables
#   INSTHOME/sqllib/misc - miscellaneous utilities
# to the user's PATH.
#----------------------------------------------------------------------

PATH=`echo ${PATH}: | sed \
         -e "s/:[^:]*sqllib\/bin[^:]*:/:/"   \
         -e "s/:[^:]*sqllib\/adm[^:]*:/:/"   \
         -e "s/:[^:]*sqllib\/misc[^:]*:/:/"  \
         -e "s/\(.*\):$/\1/"`

PATH=${PATH}:${INSTHOME}/sqllib/bin:${INSTHOME}/sqllib/adm
PATH=${PATH}:${INSTHOME}/sqllib/misc
export PATH
```

Figure 103. Sample db2profile as used in our project (part 1 of 2)

```
#------------------------------------------------------------------------
# UDB Extender initialization
#------------------------------------------------------------------------
if [ -f ${INSTHOME}/dmb/dmbprofile ]; then
    . ${INSTHOME}/dmb/dmbprofile
fi

#------------------------------------------------------------------------
# The following variables are used for JDBC support
#------------------------------------------------------------------------
#------------------------------------------------------------------------
# Only add sqlj.zip if SDK is installed
#------------------------------------------------------------------------
CLASSPATH=${CLASSPATH:-""}

if [ -f ${INSTHOME}/sqllib/java/sqlj.zip ]; then
    CLASSPATH=$CLASSPATH:${INSTHOME}/sqllib/java/sqlj.zip
fi

CLASSPATH=$CLASSPATH:${INSTHOME}/sqllib/java/db2java.zip
CLASSPATH=$CLASSPATH:${INSTHOME}/sqllib/java/runtime.zip:.
export CLASSPATH

LD_LIBRARY_PATH=${LD_LIBRARY_PATH:-""}
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${INSTHOME}/sqllib/lib
export LD_LIBRARY_PATH
```

Figure 104. Sample db2profile as used in our project (part 2 of 2)

The db2profile you use should be the same as the one generated by DB2 and placed in the home directory of your db2instance at the time you did the DB2 install.

If you try to run program DB2appProgram1.java without first setting the needed DB2 environment variables, you will get error messages indicating the program was not able to find a suitable driver.

You may also notice that we have doubled up in setting some of the environment variables as shown in both the javaprofile and db2profile scripts. This is because there are times when you may only need one or the other script such as if you are only running DB2 and not doing any Java compiles.

Note

If you do not have the DB2 environment variable DB2INSTANCE set, you will get errors running programs that access DB2 on the same computer.

11.3 Java program DB2appProgram1.java

The following figures show the source code for DB2appProgram1.java. The code has been broken up but grouped into windows to keep an orderly and readable flow to the logic.

```
/**************************************************************************************
 * @(#)DB2appProgram1.java        5.0 99/07/01
 *
 */
import java.sql.*;
import java.io.*;
import java.util.*;

public class DB2appProgram1 {

    static DataInputStream kbd = new DataInputStream(System.in);

    static String url = "jdbc:db2:SAMPLE";
    static String driver = "COM.ibm.db2.jdbc.app.DB2Driver";
    static String login = "db2inst1";
    static String passwd = "linux0";

    static Connection curConn = null;

// *************************************************************************************
// * Main Method
// *************************************************************************************
    public static void main(String argv[]) throws IOException
    {
        String temp = "";

        System.out.println("SAMPLE DB2 ACCESS (app driver).\n");
        System.out.print("Enter the url or [ENTER] for " + url + " : ");
        System.out.flush();
        temp = kbd.readLine();
        if (!temp.equals("")) url = temp;
        System.out.print("Enter the login or [ENTER] for " + login + " : ");
        System.out.flush();
        temp = kbd.readLine();
        if (!temp.equals("")) login = temp;
        System.out.print("Enter the passwd or [ENTER] for default : ");
        System.out.flush();
        temp = kbd.readLine();
        if (!temp.equals("")) passwd = temp;

        DB2appProgram1 session = new DB2appProgram1();
    }
```

Figure 105. Sample program DB2appProgram1.java (part 1 of 5)

```
// ********************************************************************************************
// * DB2appProgram1 Method
// ********************************************************************************************
    public DB2appProgram1() throws IOException
    {
        try {
            Class.forName(driver);
            curConn = DriverManager.getConnection(url, login, passwd);
            checkForWarnings(curConn.getWarnings ());
        }
        catch(java.lang.Exception ex) {
            System.out.println("url    : " + url);
            System.out.println("login  : " + login);
            System.out.println("passwd : " + passwd);
            ex.printStackTrace();
            return;
        }
        prepareQueries();
        finalize();
    }

// ********************************************************************************************
// * finalize Method
// ********************************************************************************************
    protected void finalize()
    {
        try {
            curConn.close();
        }
        catch (SQLException ex) { }
    }
```

Figure 106. Sample Program DB2appProgram1.java (part 2 of 5).

```
// ********************************************************************************************
// * prepareQueries Method
// ********************************************************************************************
    private void prepareQueries() throws IOException
    {
        int i = 1;
        String temp = "";
        String query = "";
        String results = "";

        System.out.println("Type 'quit' on a blank line to exit, or 'go' to execute the query.");
        do {
            System.out.print(i + "> ");
            System.out.flush();
            temp = kbd.readLine();
            if (temp.equals("quit"))
                break;
            if (temp.equals("go")) {
                performQuery(query);
                i = 1;
                query = "";
            }
            else {
                query = query + " " + temp;
                i++;
            }
        } while (0 == 0);
    }
```

Figure 107. Sample program DB2appProgram1.java (part 3 of 5)

```
// ************************************************************************************
// * performQuery Method
// ************************************************************************************
    private void performQuery(String sqlText) {
        boolean resultSetIsAvailable;
        boolean moreResultsAvailable;
        int i = 0;
        int res=0;
        try {
            Statement curStmt = curConn.createStatement();
            resultSetIsAvailable = curStmt.execute(sqlText);
            ResultSet rs = null;
             for (moreResultsAvailable = true; moreResultsAvailable; ) {
                checkForWarnings(curConn.getWarnings());
                 if (resultSetIsAvailable) {
                    if ((rs = curStmt.getResultSet()) != null) {
                        // we have a resultset
                        checkForWarnings(curStmt.getWarnings());
                        ResultSetMetaData rsmd = rs.getMetaData();
                        int numCols = rsmd.getColumnCount();
                        // display column headers
                        for (i = 1; i <= numCols; i++) {
                            if (i > 1) System.out.print(", ");
                            System.out.print(rsmd.getColumnLabel(i));
                        }
                        System.out.println("");
                        // step through the rows
                        while (rs.next()) {
                            // process the columns
                            for (i = 1; i <= numCols; i++) {
                                if (i > 1) System.out.print(", ");
                                System.out.print(rs.getString(i));
                            }
                            System.out.println("");
                        }
                    }
                }
                else {
                    if ((res = curStmt.getUpdateCount()) != -1) {
                        // we have an updatecount
                        System.out.println(res + " row(s) affected.");
                    }
                    // else no more results
                    else {
                        moreResultsAvailable = false;
                    }
                 }
                if (moreResultsAvailable) {
                    resultSetIsAvailable = curStmt.getMoreResults();
                }
            }
            if (rs != null) rs.close();
            curStmt.close();
        }
```

Figure 108. Sample program DB2appProgram1.java (part 4 of 5)

```
 catch (SQLException ex) {
            // Unexpected SQL exception.
            ex.printStackTrace ();
        }
        catch (java.lang.Exception ex) {
            // Got some other type of exception.  Dump it.
            ex.printStackTrace ();
        }
    }

// ********************************************************************************************
// * checkForWarnings Method
// ********************************************************************************************
    private static void checkForWarnings (SQLWarning warn)
            throws SQLException
    {
        while (warn != null) {
            System.out.println(warn);
            warn = warn.getNextWarning();
        }
    }
}
```

Figure 109. Sample program DB2appProgram1.java (part 5 of 5)

11.4 Java program DB2netProgram1.java

The next set of figures show the source code for DB2netProgram1.java. It differs from DB2appProgram1.java only in that it uses the net driver for TCP/IP access, instead of the app driver that uses direct access.

```
/*****************************************************************************************
 * @(#)DB2netProgram1.java        5.0 99/07/01
 *
 */
import java.sql.*;
import java.io.*;
import java.util.*;

public class DB2netProgram1 {

        static DataInputStream kbd = new DataInputStream(System.in);

    static String url = "jdbc:db2://9.24.105.23:8083/SAMPLE";
    static String driver = "COM.ibm.db2.jdbc.net.DB2Driver";
    static String login = "db2inst1";
    static String passwd = "linux0";

    static Connection curConn = null;

// *****************************************************************************************
// * Main Method
// *****************************************************************************************
    public static void main(String argv[]) throws IOException
    {
        String temp = "";

        System.out.println("SAMPLE DB2 ACCESS (net driver).\n");
        System.out.print("Enter the url or [ENTER] for " + url + " : ");
        System.out.flush();
        temp = kbd.readLine();
        if (!temp.equals("")) url = temp;
        System.out.print("Enter the login or [ENTER] for " + login + " : ");
        System.out.flush();
        temp = kbd.readLine();
        if (!temp.equals("")) login = temp;
        System.out.print("Enter the passwd or [ENTER] for default : ");
        System.out.flush();
        temp = kbd.readLine();
        if (!temp.equals("")) passwd = temp;

        DB2netProgram1 session = new DB2netProgram1();
    }
```

Figure 110. Sample program DB2netProgram1.java (part 1 of 5)

```
// ********************************************************************************************
// * DB2netProgram1 Method
// ********************************************************************************************
    public DB2netProgram1() throws IOException
    {
        try {
            Class.forName(driver);
            curConn = DriverManager.getConnection(url, login, passwd);
            checkForWarnings(curConn.getWarnings ());
        }
        catch(java.lang.Exception ex) {
            System.out.println("url    : " + url);
            System.out.println("login  : " + login);
            System.out.println("passwd : " + passwd);
            ex.printStackTrace();
            return;
        }
        prepareQueries();
        finalize();
    }

// ********************************************************************************************
// * finalize Method
// ********************************************************************************************
    protected void finalize()
    {
        try {
            curConn.close();
        }
        catch (SQLException ex) { }
    }
```

Figure 111. Sample program DB2netProgram1.java (part 2 of 5)

```
// ********************************************************************************************
// * prepareQueries Method
// ********************************************************************************************
    private void prepareQueries() throws IOException
    {
        int i = 1;
        String temp = "";
        String query = "";
        String results = "";

        System.out.println("Type 'quit' on a blank line to exit, or 'go' to execute the query.");
        do {
            System.out.print(i + "> ");
            System.out.flush();
            temp = kbd.readLine();
            if (temp.equals("quit"))
                break;
            if (temp.equals("go")) {
                performQuery(query);
                i = 1;
                query = "";
            }
            else {
                query = query + " " + temp;
                i++;
            }
        } while (0 == 0);
    }
```

Figure 112. Sample program DB2netProgram1.java (part 3 of 5)

```
// ************************************************************************************
// * performQuery Method
// ************************************************************************************
    private void performQuery(String sqlText) {
        boolean resultSetIsAvailable;
        boolean moreResultsAvailable;
        int i = 0;
        int res=0;
        try {
            Statement curStmt = curConn.createStatement();
            resultSetIsAvailable = curStmt.execute(sqlText);
            ResultSet rs = null;
             for (moreResultsAvailable = true; moreResultsAvailable; ) {
                checkForWarnings(curConn.getWarnings());
                 if (resultSetIsAvailable) {
                    if ((rs = curStmt.getResultSet()) != null) {
                        // we have a resultset
                        checkForWarnings(curStmt.getWarnings());
                        ResultSetMetaData rsmd = rs.getMetaData();
                        int numCols = rsmd.getColumnCount();
                        // display column headers
                        for (i = 1; i <= numCols; i++) {
                            if (i > 1) System.out.print(", ");
                            System.out.print(rsmd.getColumnLabel(i));
                        }
                        System.out.println("");
                        // step through the rows
                        while (rs.next()) {
                            // process the columns
                            for (i = 1; i <= numCols; i++)
                            {
                                if (i > 1) System.out.print(", ");
                                System.out.print(rs.getString(i));
                            }
                            System.out.println("");
                        }
                    }
                }
                else {
                    if ((res = curStmt.getUpdateCount()) != -1) {
                        // we have an updatecount
                        System.out.println(res + " row(s) affected.");
                    }
                    // else no more results
                    else {
                        moreResultsAvailable = false;
                    }
                 }
                if (moreResultsAvailable) {
                    resultSetIsAvailable = curStmt.getMoreResults();
                }
            }
            if (rs != null) rs.close();
            curStmt.close();
        }
```

Figure 113. Sample Program DB2netProgram1.java (part 4 of 5).

```
 catch (SQLException ex) {
            // Unexpected SQL exception.
            ex.printStackTrace ();
        }
        catch (java.lang.Exception ex) {
            // Got some other type of exception.  Dump it.
            ex.printStackTrace ();
        }
    }

// ********************************************************************************************
// * checkForWarnings Method
// ********************************************************************************************
    private static void checkForWarnings (SQLWarning warn)
            throws SQLException
    {
        while (warn != null) {
            System.out.println(warn);
            warn = warn.getNextWarning();
        }
    }
}
```

Figure 114. Sample program DB2netProgram1.java (part 5 of 5)

Chapter 12. Sample servlets without ConnMgr

This chapter lists two JDBC servlet programs that will show you two different ways of accessing DB2. Also, these programs do not use the Connection Manager API but show how to connect to DB2 using database information supplied by the client or from elsewhere. The first program uses the app driver that makes direct connections to DB2. The driver's full name is:

```
COM.ibm.jdbc.db2.app.DB2Driver
```

The above driver uses combined Java and native code. The second program in this chapter uses an all-Java driver called:

```
COM.ibm.jdbc.db2.net.DB2Driver
```

This driver supports connections to remote DB2 databases using TCP/IP. These are usually referred to as the app and net drivers.

These servlets make a new connection to DB2 each time the doGet() service method is called. This technique is useful when you do not know which database you want to connect to until information is supplied by the client. If you are writing a servlet whose sole purpose is to access one database and to access only one table or view in that database, then you should use the Connection Manager API to manage your connections to the database. The sample programs called DB2appServlet2.java and DB2netServlet2.java show how to do this.

Sample output is shown for each of the programs. They also include metadata extracted from the result set. They are helpful examples of how you can extract and process detailed information about the table or view you are querying.

The output also lists details of the drivers used and their code level.

When the servlets are invoked, they check to see if any data was entered in the form and if there is none, the servlet only displays the form and none of the other data.

Figure 115. DB2appServlet with a query filled in and ready to run

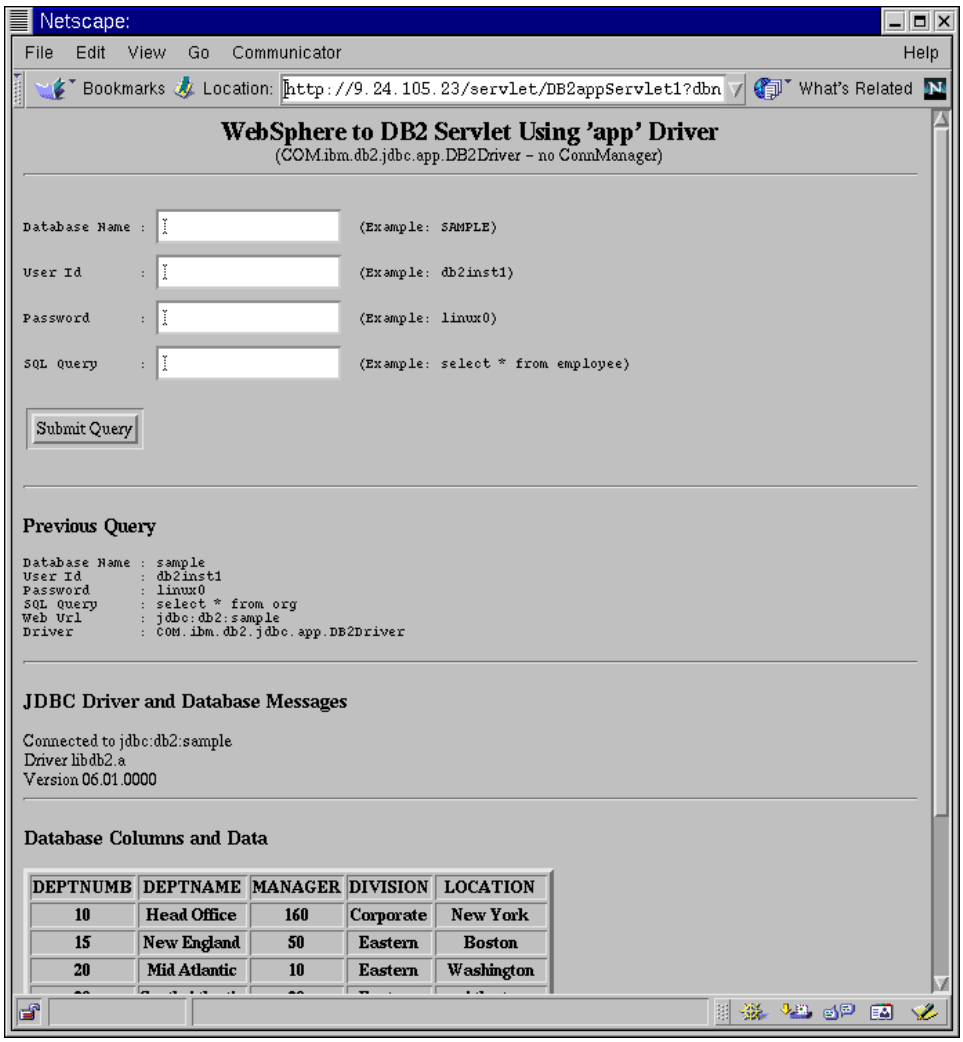


Figure 116. DB2appServlet after the query was run

12.1 App DB2 servlet - DB2appServlet1.java

```
/*****************************************************************************************
 * @(#)DB2appServlet1.java        5.0 99/07/01
 *
 */
import java.sql.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import sun.misc.*;
import java.util.*;
import java.net.*;

/*****************************************************************************************
 * DB2appServlet1
 *
 * Access SAMPLE database created in DB2
 * Program (DB2appServlet1.java) opens DB2 connections without using the Connection Manager
 * Program (DB2appServlet2.java) opens DB2 connections using the Connection Manager
 *
 */

public class DB2appServlet1 extends HttpServlet
{
  public void init (ServletConfig config) throws ServletException {
        super.init(config);
  }
```

Figure 117. DB2appServlet1.java (part 1 of 8)

The code in this section is standard Java servlet code but includes the java.util.* import in order to use Vector, hashtable, and date classes.

```
// ********************************************************************************************
// * Service Method
// ********************************************************************************************

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {

        String dbname, username, password, query, url, driver;
        Connection con = null;
        Statement stmt = null;
        ResultSet rs   = null;
        ServletOutputStream out = res.getOutputStream();

        // fetch the parameters
        dbname = req.getParameter("dbname");
        username = req.getParameter("username");
        password = req.getParameter("password");
        query = req.getParameter("query");

        res.setContentType("text/html");
        printPageHeader(out);

        // if no parameters, just print the form
        if (dbname == null || username == null || password == null ||
            query == null) {
            printPageFooter(out);
            return;
        }

        url    = "jdbc:db2:" + dbname;
        driver = "COM.ibm.db2.jdbc.app.DB2Driver";

        out.println("<hr><h3>Previous Query</h3>");
        out.println("<pre>");
        out.println("Database Name : "+dbname);
        out.println("User Id       : "+username);
        out.println("Password      : "+password);
        out.println("SQL Query     : "+query);
        out.println("Web Url       : "+url);
        out.println("Driver        : "+driver);
        out.println("</pre>");
```

Figure 118. DB2appServlet1.java (part 2 of 8)

The service method is the standard doGet(), since we are extending HttpServlet. This program establishes direct connections to the DB2 database and does not use the Connection Manager to manage the connections. Program DB2appServlet2.java is essentially the same program but does use the Connection Manager.

```
try {
            // find the jdbc dbname
        Class.forName(driver);
            // get the connection to the database
            con = DriverManager.getConnection(url, username, password);
            out.println("<hr>");
            out.println("<h3>JDBC Driver and Database Messages</h3>");
            checkForWarning(con.getWarnings(), out);
            DatabaseMetaData dma = con.getMetaData();
            out.println("Connected to " + dma.getURL() + "<br>");
            out.println("Driver       " + dma.getDriverName() + "<br>");
            out.println("Version      " + dma.getDriverVersion() + "<br>");

            // create and execute the query
            stmt = con.createStatement();
            rs = stmt.executeQuery(query);
            // print out the result
            dispResultSet(rs, out);
            rs.close();
            stmt.close();
            con.close();
            out.println("<hr>");
        }
        catch (SQLException ex) {
            out.println("<hr>*** SQLException caught ***<p>");
            while (ex != null) {
              out.println("SQLState: " + ex.getSQLState() + "<br>");
              out.println("Message:  " + ex.getMessage() + "<br>");
              out.println("Vendor:   " + ex.getErrorCode() + "<br>");
              ex = ex.getNextException();
            }
        }
        catch (java.lang.Exception ex) {
            ex.printStackTrace();
        }
    finally {
        try
        {
            if(stmt != null)
            {
              stmt.close();
              con.close();
            }
        }
        catch(SQLException e1){};
        }
        printPageFooter(out);
    }
```

Figure 119. DB2appServlet1.java (part 3 of 8)

```
/*****************************************************************************
    * Return servlet info
    */
   public String getServletInfo() {
      return "A simple WebSphere servlet to connect to DB2 SAMPLE or other, Database";
   }


   /*****************************************************************************
    * Check if the database server has anything to say
    */
   private void checkForWarning(SQLWarning warn, ServletOutputStream out)
         throws SQLException, IOException
   {
      boolean rc = false;

      if (warn != null) {
         out.println("<hr>*** Warning ***<p>");
         rc = true;
         while (warn != null) {
            out.println("SQLState: " + warn.getSQLState() + "<br>");
            out.println("Message:  " + warn.getMessage() + "<br>");
            out.println("Vendor:   " + warn.getErrorCode() + "<br>");
            warn = warn.getNextWarning();
         }
      }
   }
```

Figure 120. DB2appServlet1.java (part 4 of 8)

```
/*********************************************************************************
 *  Print one element
 */
private void dispElement(ResultSet rs, int dataType,
                         ServletOutputStream out, int col)
      throws SQLException, IOException
{

    // ask for data depending on the datatype
    switch(dataType) {
    case Types.DATE:
      java.sql.Date date = rs.getDate(col);
      out.println("<th>" + date.toString() + "</th>");
      break;
    case Types.TIME:
      java.sql.Time time = rs.getTime(col);
      out.println("<th>" + time.toString() + "</th>");
      break;
    case Types.TIMESTAMP:
      java.sql.Timestamp timestamp = rs.getTimestamp(col);
      out.println("<th>" + timestamp.toString() + "</th>");
      break;
    case Types.CHAR:
    case Types.VARCHAR:
    case Types.LONGVARCHAR:
      String str = rs.getString(col);
      out.println("<th>" + str + "</th>");
      break;
    case Types.NUMERIC:
    case Types.DECIMAL:
      java.math.BigDecimal numeric = rs.getBigDecimal(col, 10);
      out.println("<th>" + numeric.toString() + "</th>");
      break;
    case Types.BIT:
      boolean bit = rs.getBoolean(col);
      out.println("<th>" + new Boolean(bit) + "</th>");
      break;
    case Types.TINYINT:
      byte tinyint = rs.getByte(col);
      out.println("<th>" + new Integer(tinyint) + "</th>");
      break;
    case Types.SMALLINT:
      short smallint = rs.getShort(col);
      out.println("<th>" + new Integer(smallint) + "</th>");
      break;
    case Types.INTEGER:
      int integer = rs.getInt(col);
      out.println("<th>" + new Integer(integer) + "</th>");
      break;
    case Types.BIGINT:
      long bigint = rs.getLong(col);
      out.println("<th>" + new Long(bigint) + "</th>");
      break;
```

Figure 121. DB2appServlet1.java (part 5 of 8)

```
  case Types.REAL:
          float real = rs.getFloat(col);
          out.println("<th>" + new Float(real) + "</th>");
          break;
        case Types.FLOAT:
        case Types.DOUBLE:
          double longreal = rs.getDouble(col);
          out.println("<th>" + new Double(longreal) + "</th>");
          break;
        case Types.BINARY:
        case Types.VARBINARY:
        case Types.LONGVARBINARY:
          byte[] binary = rs.getBytes(col);
          out.println("<th>" + new String(binary, 0) + "</th>");
          break;
        }
    }
```

Figure 122. DB2appServlet1.java (part 6 of 8)

```
/*******************************************************************************
     *  Print header
     */
    private void printPageHeader(ServletOutputStream out)
                throws IOException
    {
        out.println("<html>");
        out.println("<head>");
        out.println("<tltle>WebSphere DB2 Servlet Test (app driver)</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<center><font size=5>" +
                    "<b>WebSphere to DB2 Servlet Using 'app' Driver</b><br>" +
                    "</font>" +
                    "(COM.ibm.db2.jdbc.app.DB2Driver - no ConnManager)" +
                    "</center>");
        out.println("<hr>");
        out.println("<form action=\"/servlet/DB2appServlet1\" method=\"get\">");
        out.println("<pre>");
        out.println("Database Name : <input type=textarea name=dbname>  (Example: SAMPLE)");
        out.println("User Id       : <input type=textarea name=username>  (Example: db2inst1)");
        out.println("Password      : <input type=textarea name=password>  (Example: linux0)");
        out.println("SQL Query     : <input type=textarea name=query>  (Example: select * from
employee)");
        out.println("</pre>");
        out.println("<input type=submit>");
        out.println("</form>");
    }
```

Figure 123. DB2appServlet1.java (part 7 of 8)

```
    /*********************************************************************************************
     *  Print footer
     */
    private void printPageFooter(ServletOutputStream out)
                throws IOException
    {
        out.println("<br>End of Servlet Page<br>");
        out.println("</body>");
        out.println("</html>");
        out.flush();
    }
}
```

Figure 124. DB2appServlet1.java (part 8 of 8)

12.2 Net DB2 servlet - DB2netServlet1.java

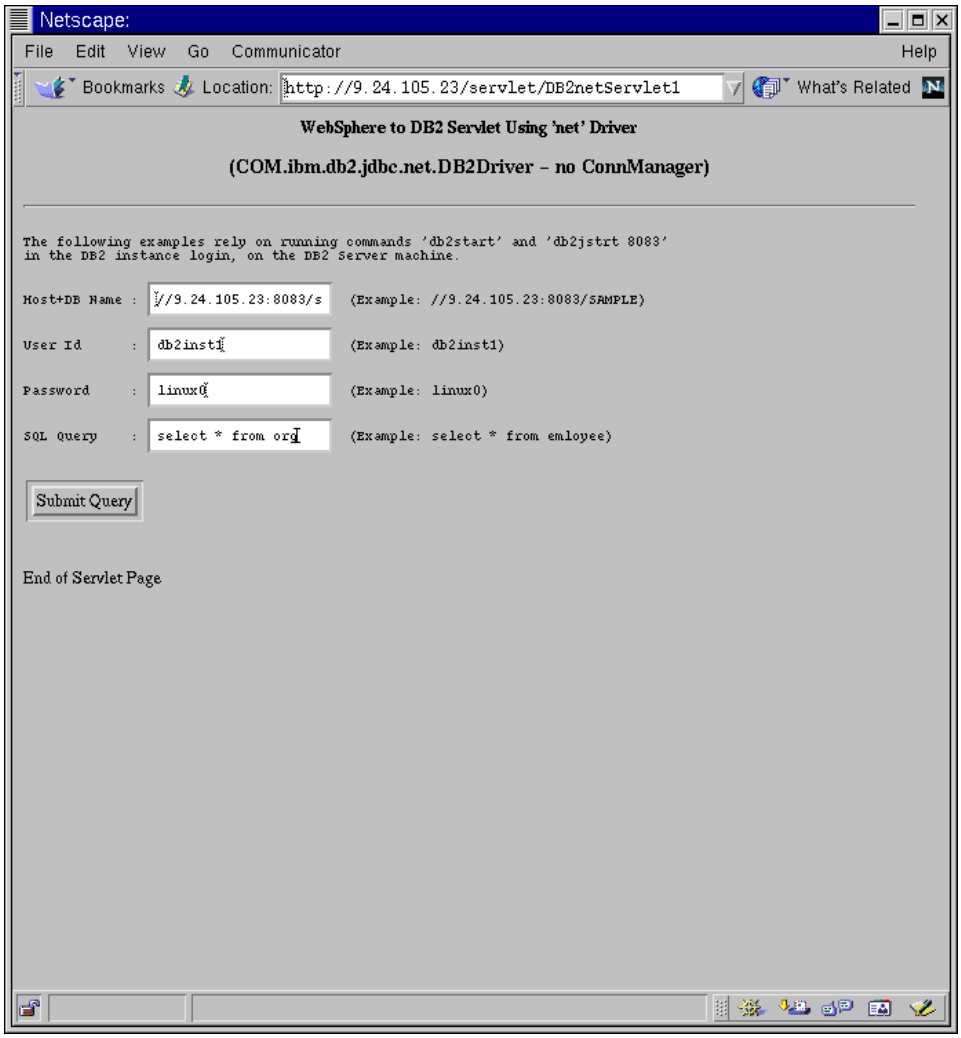


Figure 125. DB2netServlet with a query entered and ready to run

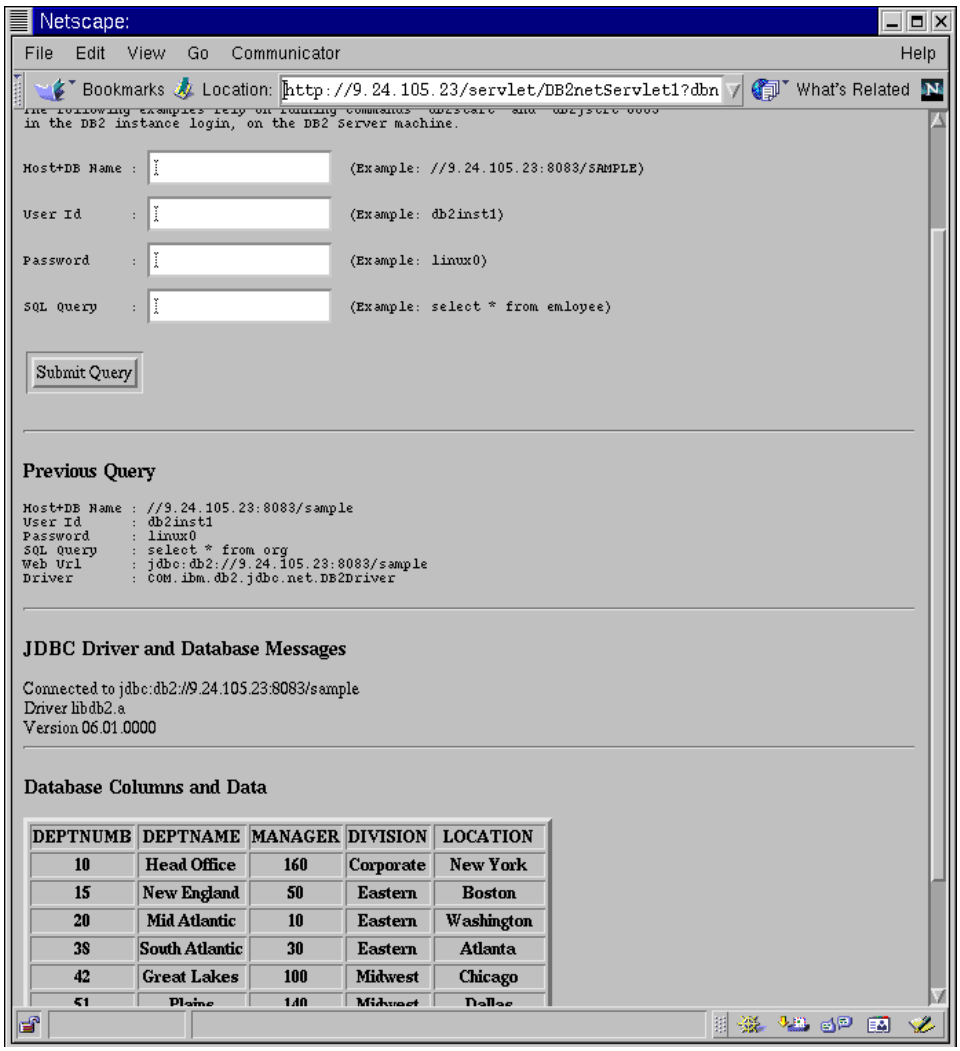


Figure 126. DB2netServlet after the query was run

```
/*****************************************************************************************
 * @(#)DB2netServlet1.java        5.0 99/07/01
 *
 */
import java.sql.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import sun.misc.*;
import java.util.*;
import java.net.*;

/*****************************************************************************************
 * DB2netServlet1
 *
 * Access SAMPLE database created by DB2
 * Program (DB2netServlet1.java) opens DB2 connections without using the Connection Manager
 * Program (DB2netServlet2.java) opens DB2 connections using the Connection Manager
 *
 */

public class DB2netServlet1 extends HttpServlet
{
// *****************************************************************************************
// * Init Method
// *****************************************************************************************
  public void init (ServletConfig config) throws ServletException {
        super.init(config);
  }
```

Figure 127. DB2netServlet1.java (part 1 of 8)

```
// ********************************************************************************************
// * Service Method
// ********************************************************************************************
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        String dbname, username, password, query, url, driver;
        Connection con = null;
        Statement stmt = null;
        ResultSet rs   = null;
        ServletOutputStream out = res.getOutputStream();

        // fetch the parameters
        dbname = req.getParameter("dbname");
        username = req.getParameter("username");
        password = req.getParameter("password");
        query = req.getParameter("query");

        res.setContentType("text/html");
        printPageHeader(out);

        // if no parameters, just print the form
        if (dbname == null || username == null || password == null ||
            query == null) {
                printPageFooter(out);
                return;
        }

        url    = "jdbc:db2:" + dbname;
        driver = "COM.ibm.db2.jdbc.net.DB2Driver";

        out.println("<hr><h3>Previous Query</h3>");
        out.println("<pre>");
        out.println("Host+DB Name : "+dbname);
        out.println("User Id      : "+username);
        out.println("Password     : "+password);
        out.println("SQL Query    : "+query);
        out.println("Web Url      : "+url);
        out.println("Driver       : "+driver);
        out.println("</pre>");
```

Figure 128. DB2netServlet1.java (part 2 of 8)

```
        try {
            // find the jdbc dbname
            Class.forName(driver);

            // get the connection to the database
            con = DriverManager.getConnection(url, username, password);
            out.println("<hr>");
            out.println("<h3>JDBC Driver and Database Messages</h3>");
            checkForWarning(con.getWarnings(), out);
            DatabaseMetaData dma = con.getMetaData();
            out.println("Connected to " + dma.getURL() + "<br>");
            out.println("Driver       " + dma.getDriverName() + "<br>");
            out.println("Version      " + dma.getDriverVersion() + "<br>");

            // create and execute the query
            stmt = con.createStatement();
            rs = stmt.executeQuery(query);

            // print out the result
            dispResultSet(rs, out);
            rs.close();
            stmt.close();
            con.close();
            out.println("<hr>");
        }
        catch (SQLException ex) {
            out.println("<hr>*** SQLException caught ***<p>");
            while (ex != null) {
                out.println("SQLState: " + ex.getSQLState() + "<br>");
                out.println("Message:  " + ex.getMessage() + "<br>");
                out.println("Vendor:   " + ex.getErrorCode() + "<br>");
                ex = ex.getNextException();
            }
        }
        catch (java.lang.Exception ex) {
            ex.printStackTrace();
        }
        finally {
            try {
                if(stmt != null) {
                    stmt.close();
                    con.close();
                }
            }
            catch(SQLException e1){};
            }
        printPageFooter(out);
    }
```

Figure 129. DB2netServlet1.java (part 3 of 8)

```
/*****************************************************************************
    * Return servlet info
    */
   public String getServletInfo() {
      return "A simple WebSphere servlet to connect to DB2 SAMPLE or other, Database";
   }


   /*****************************************************************************
    * Check if the database server has anything to say
    */
   private void checkForWarning(SQLWarning warn, ServletOutputStream out)
         throws SQLException, IOException
   {
      boolean rc = false;

      if (warn != null) {
         out.println("<hr>*** Warning ***<p>");
         rc = true;
         while (warn != null) {
            out.println("SQLState: " + warn.getSQLState() + "<br>");
            out.println("Message:  " + warn.getMessage() + "<br>");
            out.println("Vendor:   " + warn.getErrorCode() + "<br>");
            warn = warn.getNextWarning();
         }
      }
   }
```

Figure 130. DB2netServlet1.java (part 4 of 8)

```
/*****************************************************************************
    * Display results in html table format
    */
   private void dispResultSet(ResultSet rs, ServletOutputStream out)
      throws SQLException, IOException
   {
      int i;

      // metadata can supply information about the schema
      ResultSetMetaData rsmd = rs.getMetaData();
      int numCols = rsmd.getColumnCount();
      out.println("<hr>");
      out.println("<h3>Database Columns and Data</h3>");
      out.println("<table border=3>");
      out.println("<tr>");
      for (i=1; i<=numCols; i++) {
         out.println("<th>" + rsmd.getColumnLabel(i) + "</th>");
      }
      out.println("</tr>");

      // for entire data
      while (rs.next()) {
         out.println("<tr>");

         // for one row
         for (i=1; i<=numCols; i++) {
            dispElement(rs, rsmd.getColumnType(i), out, i);
         }
         out.println("</tr>");
      }
      out.println("</table>");
   }
```

Figure 131. DB2netServlet1.java (part 5 of 8)

```
/*******************************************************************************
    *  Print one element
    */
   private void dispElement(ResultSet rs, int dataType,
                            ServletOutputStream out, int col)
         throws SQLException, IOException
   {

       // ask for data depending on the datatype
       switch(dataType) {
       case Types.DATE:
         java.sql.Date date = rs.getDate(col);
         out.println("<th>" + date.toString() + "</th>");
         break;
       case Types.TIME:
         java.sql.Time time = rs.getTime(col);
         out.println("<th>" + time.toString() + "</th>");
         break;
       case Types.TIMESTAMP:
         java.sql.Timestamp timestamp = rs.getTimestamp(col);
         out.println("<th>" + timestamp.toString() + "</th>");
         break;
       case Types.CHAR:
       case Types.VARCHAR:
       case Types.LONGVARCHAR:
         String str = rs.getString(col);
         out.println("<th>" + str + "</th>");
         break;
       case Types.NUMERIC:
       case Types.DECIMAL:
         java.math.BigDecimal numeric = rs.getBigDecimal(col, 10);
         out.println("<th>" + numeric.toString() + "</th>");
         break;
       case Types.BIT:
         boolean bit = rs.getBoolean(col);
         out.println("<th>" + new Boolean(bit) + "</th>");
         break;
       case Types.TINYINT:
         byte tinyint = rs.getByte(col);
         out.println("<th>" + new Integer(tinyint) + "</th>");
         break;
       case Types.SMALLINT:
         short smallint = rs.getShort(col);
         out.println("<th>" + new Integer(smallint) + "</th>");
         break;
       case Types.INTEGER:
         int integer = rs.getInt(col);
         out.println("<th>" + new Integer(integer) + "</th>");
         break;
       case Types.BIGINT:
         long bigint = rs.getLong(col);
         out.println("<th>" + new Long(bigint) + "</th>");
         break;
```

Figure 132. DB2netServlet1.java (part 6 of 8)

```
case Types.REAL:
         float real = rs.getFloat(col);
         out.println("<th>" + new Float(real) + "</th>");
         break;
       case Types.FLOAT:
       case Types.DOUBLE:
         double longreal = rs.getDouble(col);
         out.println("<th>" + new Double(longreal) + "</th>");
         break;
       case Types.BINARY:
       case Types.VARBINARY:
       case Types.LONGVARBINARY:
         byte[] binary = rs.getBytes(col);
         out.println("<th>" + new String(binary, 0) + "</th>");
         break;
       }
   }
```

Figure 133. DB2netServlet1.java (part 7 of 8)

```
/*********************************************************************************
    *  Print header
    */
   private void printPageHeader(ServletOutputStream out)
               throws IOException
   {
       out.println("<html>");
       out.println("<head>");
       out.println("<tltle>WebSphere DB2 Servlet Test (net driver)</title>");
       out.println("</head>");
       out.println("<body>");
       out.println("<center><fonr size=5>" +
                   "<b>WebSphere to DB2 Servlet Using 'net' Driver</b><br>" +
                   "</font>" +
                   "<h3>(COM.ibm.db2.jdbc.net.DB2Driver - no ConnManager)</h3>" +
                   "</center>");
       out.println("<hr>");
       out.println("<form action=\"/servlet/DB2netServlet1\" method=\"get\">");
       out.println("<pre>");

       out.println("The following examples rely on running commands 'db2start' and 'db2jstrt
8083'");
       out.println("in the DB2 instance login, on the DB2 Server machine.");
       out.println("<p>");
       out.println("Host+DB Name : <input type=textarea name=dbname col=80>  (Example:
//9.24.105.23:8083/SAMPLE)");
       out.println("User Id      : <input type=textarea name=username col=25>  (Example:
db2inst1)");
       out.println("Password     : <input type=textarea name=password col=25>  (Example: linux0)");
       out.println("SQL Query    : <input type=textarea name=query col=25>  (Example: select * from
emloyee)");
       out.println("</pre>");
       out.println("<input type=submit>");
       out.println("</form>");
   }

/*********************************************************************************
    *  Print footer
    */
   private void printPageFooter(ServletOutputStream out)
               throws IOException
   {
       out.println("<br>End of Servlet Page<br>");
       out.println("</body>");
       out.println("</html>");
       out.flush();
   }
```

Figure 134. DB2netServlet1.java (part 8 of 8)

Chapter 13. Sample servlets using ConnMgr

The programs shown here (DB2appServlet2.java and DB2netServlet2.java) are the same as DB2appServlet1.java and DB2netServlet1.java except that instead of establishing a new connection each time the doGet() method is run, the programs were changed to make use of the Connection Manager API supplied as part of WebSphere. A connection pool is established at init() time to a known database and each invocation of doGet() requests an already open connection to this database from the Connection Manager.

The reason why it is preferable to use the Connection Manager for managing connections to an RDBMS is to reduce the latency and considerable overhead of having to open and close a connection each time the servlet is called. The disadvantage, however, is that the servlet must know exactly to which database it is connecting, at the time the servlet is first activated.

The information identifying a particular database and tables or view can be hardcoded into the servlet program (as it is in these examples) or be placed in a bundle or set in a properties file, as it is in the WebSphere sample program called IBMConnMgrTest.java (see "Samples" in the WebSphere online Documentation Center).

Most servlets are designed to perform a specific task, such as accessing data from one database and one table or view. Having one servlet perform access to multiple databases or views is not a very efficient use of WebSphere and servlets. However, there may be servlet applications that do not know which database they will connect to at any given invocation and thus may require the client/user to enter information that identifies a particular database, table, or view. In this case the servlet needs the information from the client before it can open a connection. By looking at these modified sample programs you should be able to see the differences between using either technique.

The Connection Manager can be configured for a number of parameters at the time it is first called in the init() method. These are covered in the Connection Manager API. The two most important parameters are the maximum and minimum number of open connections to be managed and how long a connection can live.

The next two figures show servlet DB2appServlet2 being used. The full source code then follows. Servlet DB2netServlet2 is shown later in this section.

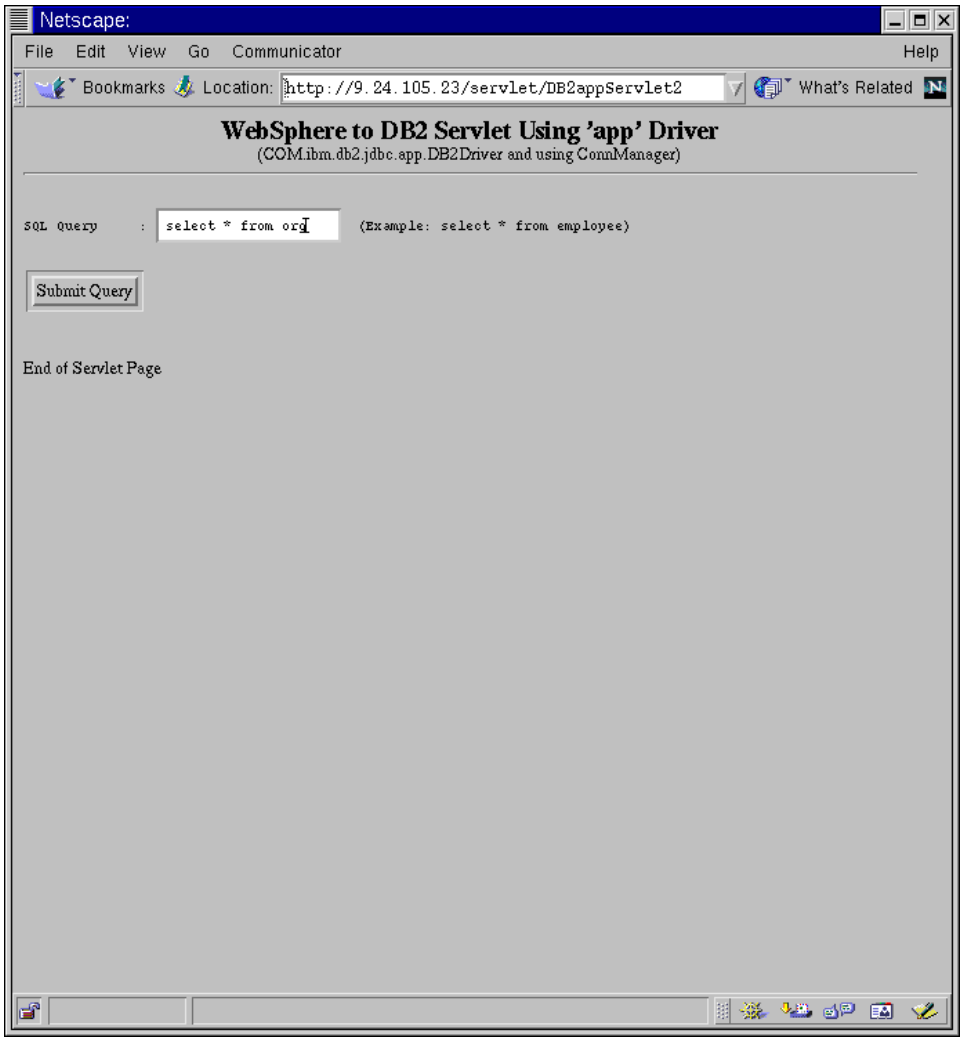


Figure 135. DB2appServlet2 with a query entered and ready to run

When called, the servlet checks to see if the query is null. If it is, the servlet knows to respond only with the initial form.

In this servlet the user types in a query and submits the form. The servlet requests an open connection from the pool manager and executes the query.

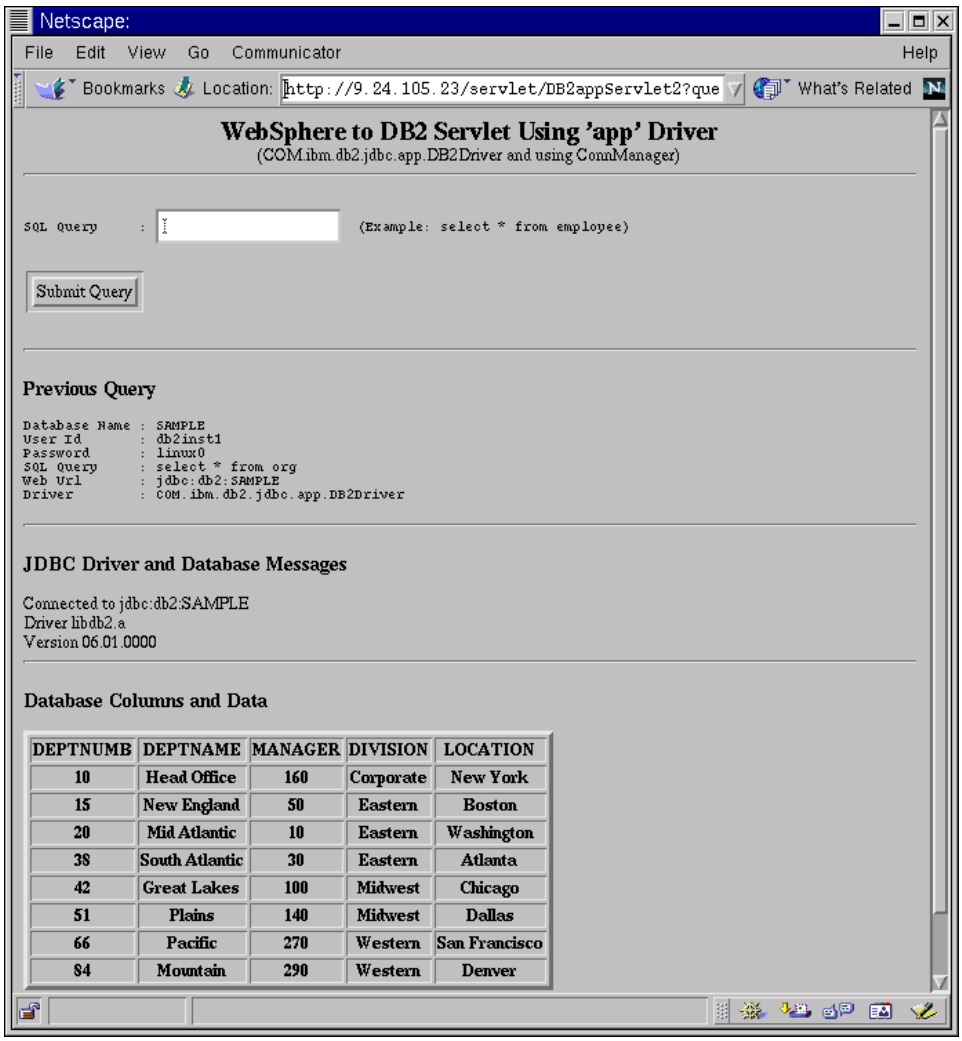


Figure 136. DB2appServlet2 after the query was run

As with the earlier samples these servlets display metadata from the database result set.

13.1 App DB2 servlet - DB2appServlet2.java

```
/*****************************************************************************************
 * @(#)DB2appServlet2.java        5.0 99/07/01
 *
 * This program is similar to DB2appServlet1.java but because it uses the
 * Connection Manager, we have to shift some of the code from the service
 * method doGet() into init(). This is because to use the Connection Manager
 * we have to hard code (or use a resource bundle) to identify the database
 * information needed to create a connection pool.
 *
 * The Connection manager portions of this program came from IBMConnMgrTest.java
 *
 */
import java.sql.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import sun.misc.*;
import java.util.*;
import java.net.*;
import com.ibm.servlet.connmgr.*;    // added import for Connection Manager

/*****************************************************************************************
 * DB2appServlet2
 *
 * Access SAMPLE database created in DB2
 * Program (DB2appServlet1.java) opens DB2 connections without using the Connection Manager
 * Program (DB2appServlet2.java) opens DB2 connections using the Connection Manager
 *
 */

public class DB2appServlet2 extends HttpServlet
{

// ***************************************************************************************
// * Variables
// ***************************************************************************************
  // A class variable Connection Manager - note it is initialized to null.
  static IBMConnMgr connMgr = null;

  // More class variables for Use later in init() to create JDBC connection specification.
  static IBMConnSpec spec   = null;      // the spec
  static String dbName      = null;      // database name
  static String db          = "db2";     // JDBC subprotocol for DB2
  static String poolName    = "JdbcDb2"; // from Webmaster
  static String jdbcDriver  = "COM.ibm.db2.jdbc.app.DB2Driver";
  static String url         = null;      // constructed later
  static String user        = null;      // user and password could
  static String password    = null;      // come from HTML form
  static String owner       = null;      // table owner
```

Figure 137. DB2appServlet2.java, (part 1 of 7)

```
// ********************************************************************************************
// * Init Method
// ********************************************************************************************
  public void init (ServletConfig config) throws ServletException {
        super.init(config);

      // The following data would normally be kept in a resource bundle and loaded at init time

      dbName   = "SAMPLE";        // code or import your database name here
      url      = "jdbc:" + db + ":" + dbName;
      user     = "db2inst1";        // code or import your userid here
      password = "linux0";        // code or import your password here
      owner    = "db2inst1";        // code or import your owner here

      try                        // Here we establish the Connection pool for this database
      {
         // Create JDBC connection specification.
         spec = new IBMJdbcConnSpec
                (poolName,    // pool name from Webmaster
                 true,        // waitRetry
                 jdbcDriver,  // Remaining four
                 url,         // parameters are
                 user,        // specific for a
                 password);   // JDBC connection.

         // Here we obtain a reference to the connection manager just created.

         connMgr = IBMConnMgrUtil.getIBMConnMgr();
      }
      catch(Exception ex)
      {
         System.out.println("set connection spec, get connection manager: " +
                            ex.getMessage());
      }
  }
```

Figure 138. DB2appServlet2.java, (part 2 of 7)

```
// ********************************************************************************************
// * Service Method
// ********************************************************************************************
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        IBMJdbcConn cmConn   = null;
        Connection dataConn  = null;
        Statement stmt       = null;
        ResultSet rs         = null;
        String query         = null;
        ServletOutputStream out = res.getOutputStream();
        // fetch the database query string enterd by the user
        query = req.getParameter("query");
        res.setContentType("text/html");
        printPageHeader(out);

        // if no parameters, just print the form
        if (query == null) {
            printPageFooter(out);
            return;
        }
        out.println("<hr><h3>Previous Query</h3>");
        out.println("<pre>");
        out.println("Database Name : "+dbName);
        out.println("User Id       : "+user);
        out.println("Password      : "+password);
        out.println("SQL Query     : "+query);
        out.println("Web Url       : "+url);
        out.println("Driver        : "+jdbcDriver);
        out.println("</pre>");

        try {
           // Get an IBMJdbcConn object (cmConn) meeting JDBC
           // connection specs, from the connection manager pool.

           cmConn = (IBMJdbcConn)connMgr.getIBMConnection(spec);

           // Get a Connection object (dataConn).  This is
           // an object from the java.sql package and it is used
           // for JDBC access.

           dataConn = cmConn.getJdbcConnection();

           // We now have an open connection
        }
        catch(Exception ex) {
           System.out.println("get connection, process statement: " +
                              ex.getMessage());
        }
        out.println("<hr>");
        out.println("<h3>JDBC Driver and Database Messages</h3>");
```

Figure 139. DB2appServlet2.java, (part 3 of 7)

```
        try {
            DatabaseMetaData dma = dataConn.getMetaData();
            out.println("Connected to " + dma.getURL() + "<br>");
            out.println("Driver       " + dma.getDriverName() + "<br>");
            out.println("Version      " + dma.getDriverVersion() + "<br>");
        }
        catch (SQLException ex) {
            out.println("<hr>*** SQLException caught ***<p>");
            while (ex != null) {
               out.println("SQLState: " + ex.getSQLState() + "<br>");
               out.println("Message:  " + ex.getMessage() + "<br>");
               out.println("Vendor:   " + ex.getErrorCode() + "<br>");
               ex = ex.getNextException();
            }
        }
        // create and execute the query
        try {
            stmt = dataConn.createStatement();
            rs = stmt.executeQuery(query);
            dispResultSet(rs, out);               // print out the result
            rs.close();
            stmt.close();
            out.println("<hr>");
        }
        catch (SQLException ex) {
            out.println("<hr>*** SQLException caught ***<p>");
            while (ex != null) {
               out.println("SQLState: " + ex.getSQLState() + "<br>");
               out.println("Message:  " + ex.getMessage() + "<br>");
               out.println("Vendor:   " + ex.getErrorCode() + "<br>");
               ex = ex.getNextException();
            }
        }
        catch (java.lang.Exception ex) {
            ex.printStackTrace();
        }
        finally {
            try {
                if(stmt != null) {
                   stmt.close();
                }
            }
            catch(SQLException ex){};
            if(cmConn != null) {
                try {
                   cmConn.releaseIBMConnection();
                }
                catch(IBMConnMgrException ex)
                {
                   System.out.println("release connection: " + ex.getMessage());
                }
            }
            printPageFooter(out);
        }
    }
```

Figure 140. DB2appServlet2.java, (part 4 of 7)

```
/*********************************************************************************************
 * Return servlet info
 */
public String getServletInfo() {
   return "A simple WebSphere servlet to connect to DB2 SAMPLE or other, Database";
}


/*********************************************************************************************
 * Display results in html table format
 */
private void dispResultSet(ResultSet rs, ServletOutputStream out)
   throws SQLException, IOException
{
   int i;

   // metadata can supply information about the schema
   ResultSetMetaData rsmd = rs.getMetaData();
   int numCols = rsmd.getColumnCount();
   out.println("<hr>");
   out.println("<h3>Database Columns and Data</h3>");
   out.println("<table border=3>");
   out.println("<tr>");
   for (i=1; i<=numCols; i++) {
      out.println("<th>" + rsmd.getColumnLabel(i) + "</th>");
   }
   out.println("</tr>");

   // for entire data
   while (rs.next()) {
      out.println("<tr>");

      // for one row
      for (i=1; i<=numCols; i++) {
         dispElement(rs, rsmd.getColumnType(i), out, i);
      }
      out.println("</tr>");
   }
   out.println("</table>");
}
```

Figure 141. DB2appServlet2.java, (part 5 of 7)

```
/*********************************************************************************
 * Print one element
 */

private void dispElement(ResultSet rs, int dataType,
                         ServletOutputStream out, int col)
      throws SQLException, IOException
{

    // ask for data depending on the datatype
    switch(dataType) {
    case Types.DATE:
      java.sql.Date date = rs.getDate(col);
      out.println("<th>" + date.toString() + "</th>");
      break;
    case Types.TIME:
      java.sql.Time time = rs.getTime(col);
      out.println("<th>" + time.toString() + "</th>");
      break;
    case Types.TIMESTAMP:
      java.sql.Timestamp timestamp = rs.getTimestamp(col);
      out.println("<th>" + timestamp.toString() + "</th>");
      break;
    case Types.CHAR:
    case Types.VARCHAR:
    case Types.LONGVARCHAR:
      String str = rs.getString(col);
      out.println("<th>" + str + "</th>");
      break;
    case Types.NUMERIC:
    case Types.DECIMAL:
      java.math.BigDecimal numeric = rs.getBigDecimal(col, 10);
      out.println("<th>" + numeric.toString() + "</th>");
      break;
    case Types.BIT:
      boolean bit = rs.getBoolean(col);
      out.println("<th>" + new Boolean(bit) + "</th>");
      break;
    case Types.TINYINT:
      byte tinyint = rs.getByte(col);
      out.println("<th>" + new Integer(tinyint) + "</th>");
      break;
    case Types.SMALLINT:
      short smallint = rs.getShort(col);
      out.println("<th>" + new Integer(smallint) + "</th>");
      break;
    case Types.INTEGER:
      int integer = rs.getInt(col);
      out.println("<th>" + new Integer(integer) + "</th>");
      break;
    case Types.BIGINT:
      long bigint = rs.getLong(col);
      out.println("<th>" + new Long(bigint) + "</th>");
      break;
```

Figure 142. DB2appServlet2.java, (part 6 of 7)

```
 case Types.REAL:
          float real = rs.getFloat(col);
          out.println("<th>" + new Float(real) + "</th>");
          break;
        case Types.FLOAT:
        case Types.DOUBLE:
          double longreal = rs.getDouble(col);
          out.println("<th>" + new Double(longreal) + "</th>");
          break;
        case Types.BINARY:
        case Types.VARBINARY:
        case Types.LONGVARBINARY:
          byte[] binary = rs.getBytes(col);
          out.println("<th>" + new String(binary, 0) + "</th>");
          break;
        }
    }

/*******************************************************************************
     *  Print header
     */
    private void printPageHeader(ServletOutputStream out)
                throws IOException {
        out.println("<html>");
        out.println("<head>");
         out.println("<tltle>WebSphere DB2 Servlet Test (app driver)</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<center><font size=5>" +
                    "<b>WebSphere to DB2 Servlet Using 'app' Driver</b><br>" +
                    "</font>" +
                    "(COM.ibm.db2.jdbc.app.DB2Driver and using ConnManager)" +
                    "</center>");
        out.println("<hr>");
        out.println("<form action=\"/servlet/DB2appServlet2\" method=\"get\">");
        out.println("<pre>");
        out.println("SQL Query     : <input type=textarea name=query>  (Example: select * from
employee)");
        out.println("</pre>");
        out.println("<input type=submit>");
        out.println("</form>");
    }

    /*******************************************************************************
     *  Print footer
     */
    private void printPageFooter(ServletOutputStream out)
                throws IOException
    {
        out.println("<br>End of Servlet Page<br>");
        out.println("</body>");
        out.println("</html>");
        out.flush();
    }
}
```

Figure 143. DB2appServlet2.java, (part 7 of 7)

13.2 Net DB2 servlet - DB2netServlet2.java

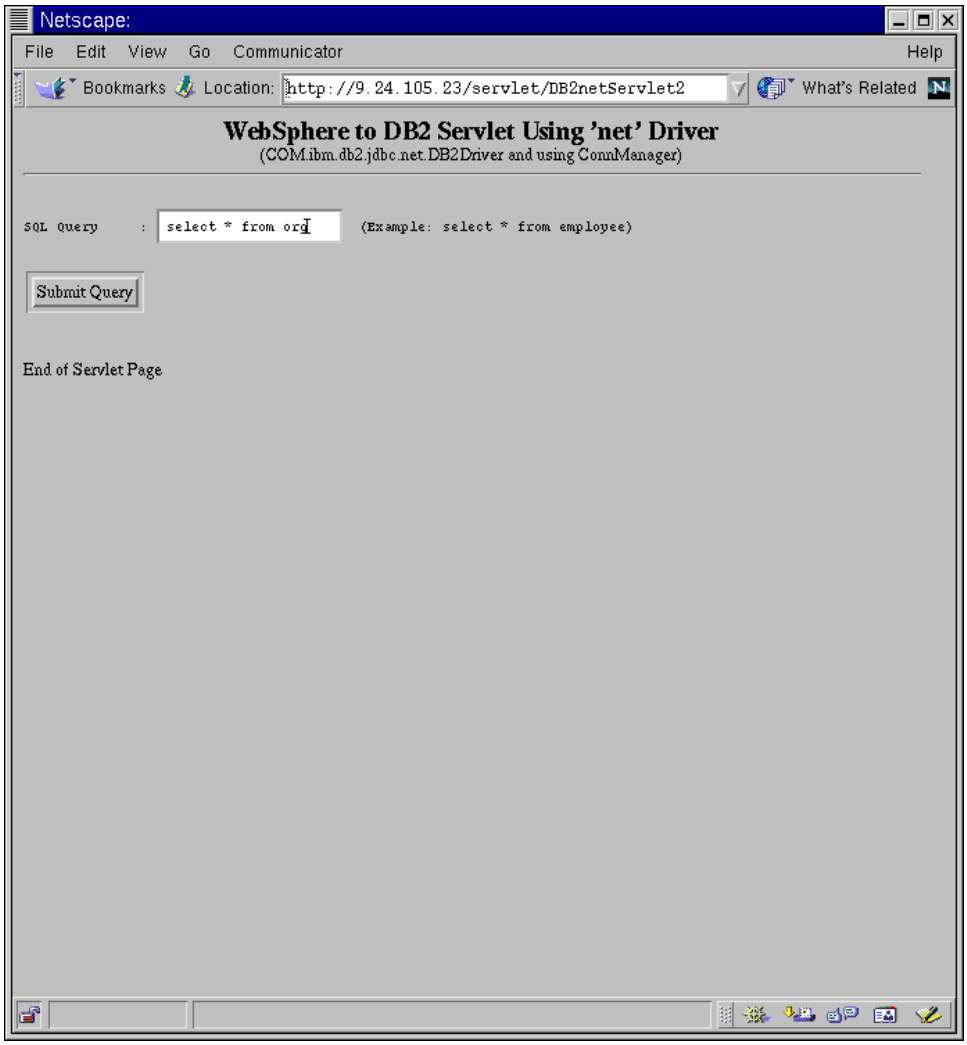


Figure 144. DB2netServlet2 with a query entered and ready to run

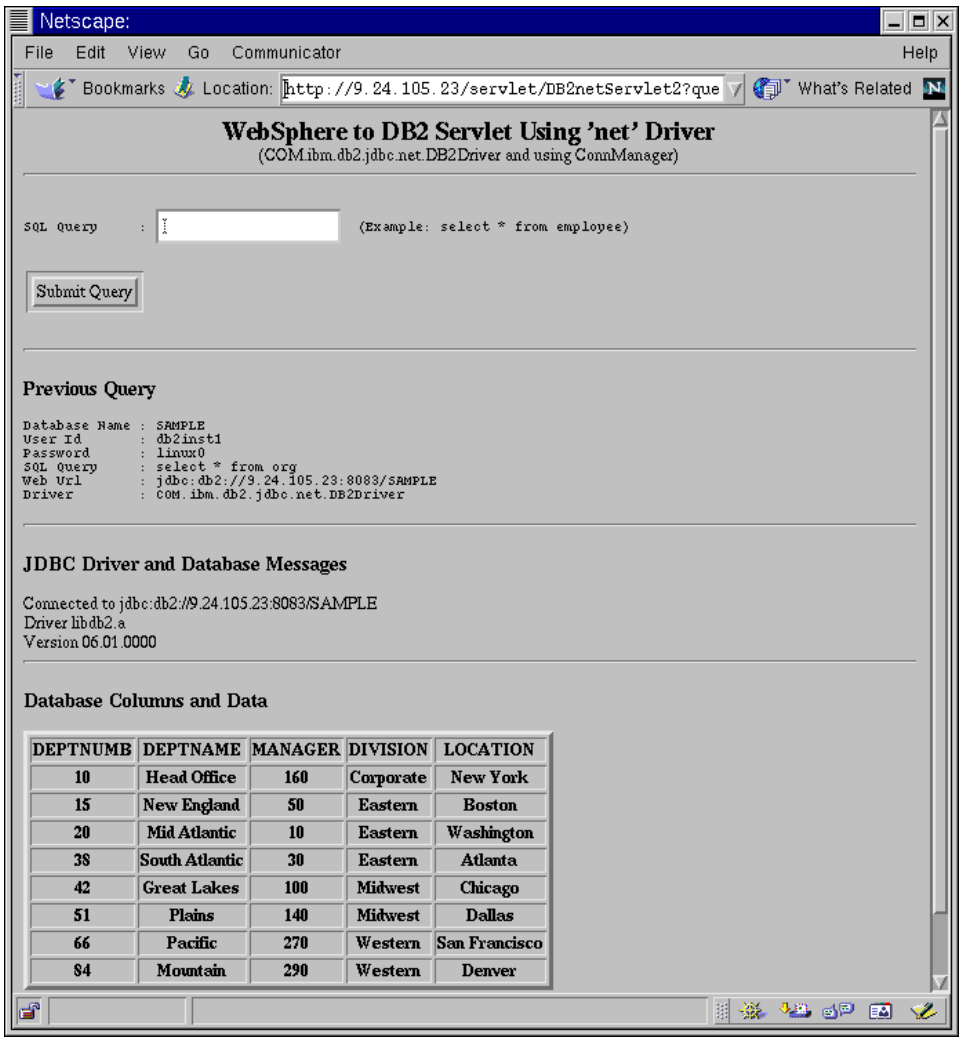


Figure 145. DB2netServlet2 after the query was run

```
/*********************************************************************************************
 * @(#)DB2netServlet2.java        5.0 99/07/01
 *
 * This program is similar to DB2netServlet1.java but because it uses the
 * Connection Manager, we have to shift some of the code from the service
 * method doGet() into init(). This is because to use the Connection Manager
 * we have to hard code (or use a resource bundle) to identify the database
 * information needed to create a connection pool.
 *
 * The Connection manager portions of this program came from IBMConnMgrTest.java
 *
 */
import java.sql.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import sun.misc.*;
import java.util.*;
import java.net.*;
import com.ibm.servlet.connmgr.*;    // added import for Connection Manager

/*********************************************************************************************
 * DB2netServlet2
 *
 * Access SAMPLE database created in DB2
 * Program (DB2netServlet1.java) opens DB2 connections without using the Connection Manager
 * Program (DB2netServlet2.java) opens DB2 connections using the Connection Manager
 */

public class DB2netServlet2 extends HttpServlet {

// *********************************************************************************************
// * Variables
// *********************************************************************************************
  // A class variable Connection Manager - note it is initialized to null.
  static IBMConnMgr connMgr = null;

  // More class variables for Use later in init() to create JDBC connection specification.
  static IBMConnSpec spec   = null;      // the spec
  static String dbName      = null;      // database name
  static String hostName    = null;      // hostname
  static String portNum     = null;      // port num
  static String db          = "db2";     // JDBC subprotocol for DB2
  static String poolName    = "JdbcDb2"; // from Webmaster
  static String jdbcDriver  = "COM.ibm.db2.jdbc.net.DB2Driver";
  static String url         = null;      // constructed later
  static String user        = null;      // user and password could
  static String password    = null;      // come from HTML form
  static String owner       = null;      // table owner
```

Figure 146. DB2netServlet2.java, (part 1 of 7)

```
// ********************************************************************************************
// * Init Method
// ********************************************************************************************
  public void init (ServletConfig config) throws ServletException {
        super.init(config);

      // The following data would normally be kept in a resource bundle and loaded at init time

      dbName   = "SAMPLE";        // code or import your database name here
      hostName = "9.24.105.23";        // code or import your hostname here
      portNum  = "8083";         // code or import your port num here
      url      = "jdbc:" + db + "://" + hostName + ":" + portNum + "/" +  dbName;
      user     = "db2inst1";        // code or import your userid here
      password = "linux0";        // code or import your password here
      owner    = "db2inst1";        // code or import your owner here

      try                         // Here we establish the Connection pool for this database
      {
         // Create JDBC connection specification.
         spec = new IBMJdbcConnSpec
                (poolName,    // pool name from Webmaster
                 true,        // waitRetry
                 jdbcDriver,  // Remaining four
                 url,         // parameters are
                 user,        // specific for a
                 password);   // JDBC connection.

         // Here we obtain a reference to the connection manager just created.

         connMgr = IBMConnMgrUtil.getIBMConnMgr();
      }
      catch(Exception ex)
      {
         System.out.println("set connection spec, get connection manager: " +
                            ex.getMessage());
      }
  }
```

Figure 147. DB2netServlet2.java, (part 2 of 7)

```
// ********************************************************************************************
// * Service Method
// ********************************************************************************************
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        IBMJdbcConn cmConn   = null;
        Connection dataConn  = null;
        Statement stmt       = null;
        ResultSet rs         = null;
        String query         = null;
        ServletOutputStream out = res.getOutputStream();

        // fetch the database query string enterd by the user
        query = req.getParameter("query");

        res.setContentType("text/html");
        printPageHeader(out);

        // if no parameters, just print the form
        if (query == null) {
            printPageFooter(out);
            return;
        }

        out.println("<hr><h3>Previous Query</h3>");
        out.println("<pre>");
        out.println("Database Name : "+dbName);
        out.println("User Id       : "+user);
        out.println("Password      : "+password);
        out.println("SQL Query     : "+query);
        out.println("Web Url       : "+url);
        out.println("Driver        : "+jdbcDriver);
        out.println("</pre>");

        try {
           // Get an IBMJdbcConn object (cmConn) meeting JDBC
           // connection specs, from the connection manager pool.

           cmConn = (IBMJdbcConn)connMgr.getIBMConnection(spec);

           // Get a Connection object (dataConn).  This is
           // an object from the java.sql package and it is used
           // for JDBC access.

           dataConn = cmConn.getJdbcConnection();

           // We now have an open connection
        }
        catch(Exception ex) {
           System.out.println("get connection, process statement: " +
                              ex.getMessage());
        }
        out.println("<hr>");
        out.println("<h3>JDBC Driver and Database Messages</h3>");
```

Figure 148. DB2netServlet2.java, (part 3 of 7)

```
        try {
            DatabaseMetaData dma = dataConn.getMetaData();
            out.println("Connected to " + dma.getURL() + "<br>");
            out.println("Driver       " + dma.getDriverName() + "<br>");
            out.println("Version      " + dma.getDriverVersion() + "<br>");
        }
        catch (SQLException ex) {
            out.println("<hr>*** SQLException caught ***<p>");
            while (ex != null) {
               out.println("SQLState: " + ex.getSQLState() + "<br>");
               out.println("Message:  " + ex.getMessage() + "<br>");
               out.println("Vendor:   " + ex.getErrorCode() + "<br>");
               ex = ex.getNextException();
            }
        }
        // create and execute the query
        try {
            stmt = dataConn.createStatement();
            rs = stmt.executeQuery(query);
            dispResultSet(rs, out);      // print out the result
            rs.close();
            stmt.close();
            out.println("<hr>");
        }
        catch (SQLException ex) {
            out.println("<hr>*** SQLException caught ***<p>");
            while (ex != null) {
               out.println("SQLState: " + ex.getSQLState() + "<br>");
               out.println("Message:  " + ex.getMessage() + "<br>");
               out.println("Vendor:   " + ex.getErrorCode() + "<br>");
               ex = ex.getNextException();
            }
        }
        catch (java.lang.Exception ex) {
            ex.printStackTrace();
        }
        finally {
            try {
                if(stmt != null) {
                   stmt.close();
                }
            }
            catch(SQLException ex){};
            if(cmConn != null) {
                try {
                   cmConn.releaseIBMConnection();
                }
                catch(IBMConnMgrException ex) {
                   System.out.println("release connection: " + ex.getMessage());
                }
            }
            printPageFooter(out);
        }
    }
```

Figure 149. DB2netServlet2.java, (part 4 of 7)

```
/*********************************************************************************************
 * Return servlet info
 */
public String getServletInfo() {
   return "A simple WebSphere servlet to connect to DB2 SAMPLE or other, Database";
}


/*********************************************************************************************
 * Display results in html table format
 */
private void dispResultSet(ResultSet rs, ServletOutputStream out)
   throws SQLException, IOException
{
   int i;

   // metadata can supply information about the schema
   ResultSetMetaData rsmd = rs.getMetaData();
   int numCols = rsmd.getColumnCount();
   out.println("<hr>");
   out.println("<h3>Database Columns and Data</h3>");
   out.println("<table border=3>");
   out.println("<tr>");
   for (i=1; i<=numCols; i++) {
      out.println("<th>" + rsmd.getColumnLabel(i) + "</th>");
   }
   out.println("</tr>");

   // for entire data
   while (rs.next()) {
      out.println("<tr>");

      // for one row
      for (i=1; i<=numCols; i++) {
         dispElement(rs, rsmd.getColumnType(i), out, i);
      }
      out.println("</tr>");
   }
   out.println("</table>");
}
```

Figure 150. DB2netServlet2.java, (part 5 of 7)

```
This is screen.
/*********************************************************************************************
     *  Print one element
     */
    private void dispElement(ResultSet rs, int dataType,
                             ServletOutputStream out, int col)
          throws SQLException, IOException
    {

        // ask for data depending on the datatype
        switch(dataType) {
        case Types.DATE:
          java.sql.Date date = rs.getDate(col);
          out.println("<th>" + date.toString() + "</th>");
          break;
        case Types.TIME:
          java.sql.Time time = rs.getTime(col);
          out.println("<th>" + time.toString() + "</th>");
          break;
        case Types.TIMESTAMP:
          java.sql.Timestamp timestamp = rs.getTimestamp(col);
          out.println("<th>" + timestamp.toString() + "</th>");
          break;
        case Types.CHAR:
        case Types.VARCHAR:
        case Types.LONGVARCHAR:
          String str = rs.getString(col);
          out.println("<th>" + str + "</th>");
          break;
        case Types.NUMERIC:
        case Types.DECIMAL:
          java.math.BigDecimal numeric = rs.getBigDecimal(col, 10);
          out.println("<th>" + numeric.toString() + "</th>");
          break;
        case Types.BIT:
          boolean bit = rs.getBoolean(col);
          out.println("<th>" + new Boolean(bit) + "</th>");
          break;
        case Types.TINYINT:
          byte tinyint = rs.getByte(col);
          out.println("<th>" + new Integer(tinyint) + "</th>");
          break;
        case Types.SMALLINT:
          short smallint = rs.getShort(col);
          out.println("<th>" + new Integer(smallint) + "</th>");
          break;
        case Types.INTEGER:
          int integer = rs.getInt(col);
          out.println("<th>" + new Integer(integer) + "</th>");
          break;
        case Types.BIGINT:
          long bigint = rs.getLong(col);
          out.println("<th>" + new Long(bigint) + "</th>");
          break;
```

Figure 151. DB2netServlet2.java, (part 6 of 7)

```
        case Types.REAL:
          float real = rs.getFloat(col);
          out.println("<th>" + new Float(real) + "</th>");
          break;
        case Types.FLOAT:
        case Types.DOUBLE:
          double longreal = rs.getDouble(col);
          out.println("<th>" + new Double(longreal) + "</th>");
          break;
        case Types.BINARY:
        case Types.VARBINARY:
        case Types.LONGVARBINARY:
          byte[] binary = rs.getBytes(col);
          out.println("<th>" + new String(binary, 0) + "</th>");
          break;
        }
    }
    /*********************************************************************************
     *  Print header
     */
    private void printPageHeader(ServletOutputStream out)
                throws IOException {
        out.println("<html>");
        out.println("<head>");
        out.println("<tltle>WebSphere DB2 Servlet Test (net driver)</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<center><font size=5>" +
                    "<b>WebSphere to DB2 Servlet Using 'net' Driver</b><br>" +
                    "</font>" +
                    "(COM.ibm.db2.jdbc.net.DB2Driver and using ConnManager)" +
                    "</center>");
        out.println("<hr>");
        out.println("<form action=\"/servlet/DB2netServlet2\" method=\"get\">");
        out.println("<pre>");
        out.println("SQL Query     : <input type=textarea name=query>  (Example: select * from
employee)");
        out.println("</pre>");
        out.println("<input type=submit>");
        out.println("</form>");
    }
    /*********************************************************************************
     *  Print footer
     */
    private void printPageFooter(ServletOutputStream out)
                throws IOException {
        out.println("<br>End of Servlet Page<br>");
        out.println("</body>");
        out.println("</html>");
        out.flush();
    }
```

Figure 152. DB2netServlet2.java, (part 7 of 7)

Appendix A. Installing IBM HTTP Server Beta 3 with SSL

This section will discuss the procedures and components for installing the Secure Sockets Layer (SSL) module for IBM HTTP Server Beta 3. IBM HTTP Server Beta 3 is downloadable from:

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

A.1 IBM HTTP Server

IBM HTTP Server features include:

- Easy installation
- Support for SSL secure connections
- Customer-designed services
- IBM support
- Help information that uses the easy-to-navigate design that is common to all WebSphere products

A.1.1 SSL Protocol

This protocol, implemented using IBM security libraries, ensures that data transferred between a client and a server remains private. Once your server has a digital certificate, SSL-enabled browsers like Netscape Navigator and Microsoft Internet Explorer can communicate securely with your server using the SSL protocol.

The IBM HTTP Server powered by Apache supports client authentication, configurable cipher specifications, and session ID caching for improving SSL performance on the UNIX platforms.

A.1.2 Install IBM HTTP Server

In 5.7.1, “IBM HTTP Server install steps” on page 33, we showed the steps to install IBM HTTP Server. Before you proceed with the remaining section, please be sure you have successfully installed this Web server.

Another prerequisite is that you should install Java Development Kit (JDK) 1.1.7 V3 or higher and use THREADS_FLAG system variable set to "green thread". There are other system dependency files that you *may* need to install depending on your Linux distribution. Please refer to the readme.linux file in the tar file.

A.1.3 Installing global security kit (GSK)

GSK is packaged in tar files: gsk4bas-XXX.rpm and gsk4ikm-XXX.rpm, where XXX is the 4.0-1.1.i386 version number at the time of this writing. These packages include the library modules and documentations associated with the Key Management (IKEYMAN) utility for creating and maintaining certificates. The IKEYMAN utility creates server certificates for SSL.

To set up SSL secure connections, your public key must be associated either with a digitally signed certificate from a certificate authority (CA) or with a self-signed certificate created using IBM IKEYMAN utility. CA is an external signed certificate provider that is designated as a trusted CA on your server. IBM HTTP Server supports the following external CAs:

1. VeriSign
2. Thawte

Before you can install the SSL modules, you need to first install gsk4bas-4.0-1.1.i386.rpm and gsk4ikm-4.0-1.1.i386.rpm files. You can use your package manager on your distribution (gnorpm on Red Hat and kpackage on OpenLinux), or run the following commands:

```
rpm -ivh gsk4bas-4.0-1.1.i386.rpm
rpm -ivh gsk4ikm-4.0-1.1.i386.rpm
```

A.1.4 Install IBM SSL modules

Your installation source will include these two files: IBM_SSL_56-1.3.6-1.i386.rpm and IBM_SSL_Base-1.3.6-1.i386.rpm

The IBM Secure Sockets Layer (SSL) modules contain the encryption library files as well as the IKEYMAN utility program for creating and storing server certificates. The IBM SSL library modules provide encrypted HTTPS connection for the IBM HTTP Server. It uses the public key technology to negotiate a session key and crypto algorithms between client and server.

Use the package program on your distribution to install these rpm files, starting first with IBM_SSL_Base-1.3.6-1.i386.rpm file and then IBM_SSL_56-1.3.6-1.i386.rpm.

Alternatively, you can run these commands:

```
rpm -ivh IBM_SSL_Base-1.3.6-1.i386.rpm
rpm -ivh IBM_SSL_56-1.3.6-1.i386.rpm.
```

A.2 Prepare Server Certificate

In this section we will show the steps for creating a self-signed certificate for using SSL with IBM HTTP Server.

A.2.1 Creating a key database

Use IKEYMAN to create the key database file, public-private key pair, and certificate request. A key database is a file that the server uses to store one or more key pairs and certificates. You can use one key database for all your key pairs and certificates or create multiple databases.

IKEYMAN lets you create a self-signed certificate or store an authorized certificate. If you are using a Certificate Authority (CA) for your server certificate, after you receive the CA-signed certificate, use IKEYMAN to receive the certificate into the key database where you created the original certificate request.

Type `ikeyman` in an xterm to run the Java-based IBM Key Management program.

From IKEYMAN's menu choose **Key Database File -> New**; IKEYMAN presents a dialog box titled New. Fill out the following fields:

1. In the Key database type field, use the default value of CMS key database File.
2. In the File Name field, provide a key name (for example, mycert.kdb).
3. In the Location field, type the directory to store the key database file (for example, /opt/IBMHTTPServer/keys/).

Note

The directory /opt/IBMHTTPServer/keys/ is not created by default. You must create this subdirectory beforehand.

The next dialog box requires you to fill in the password and expiration time, and decide if you want to stash the password to a file:

1. Provide a password (do not lose the password; write it down).
2. Leave the expiration time field unchecked.
3. Be sure to check the **Stash password to a file** box. For a secure network connection, you must store the encrypted database password

in a stash file. Click **Help** for more details,or click **OK** to continue. See Figure 153.

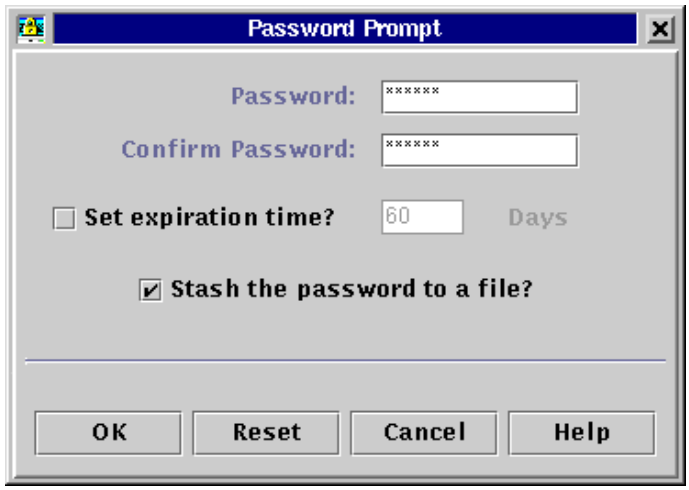


Figure 153. IKEYMAN dialog box; use stash to store the password

A.2.2 Create a self-signed certificate

From IKEYMAN's menu select **Create -> New Self-Signed Certificate**; IKEYMAN presents a dialog box titled Create New Self-Signed Certificate (see Figure 154). Fill out the following fields:

1. In the Key label field, type in any label. This label identifies the key and certificate in the key database.
2. In the Version field , click the drop-down list, and select the value X509 V2.

   Note: The production version supports both X509 V2 and X509 V3. In this beta version of IBM HTTP Server, we will use X509 V2.
3. In the Common Name field, IKEYMAN picks up the default domain name of the server. Leave this field as is.
4. In the Organization field, type in the organization name.
5. In the Validity Period field, use the default value of 365 days.

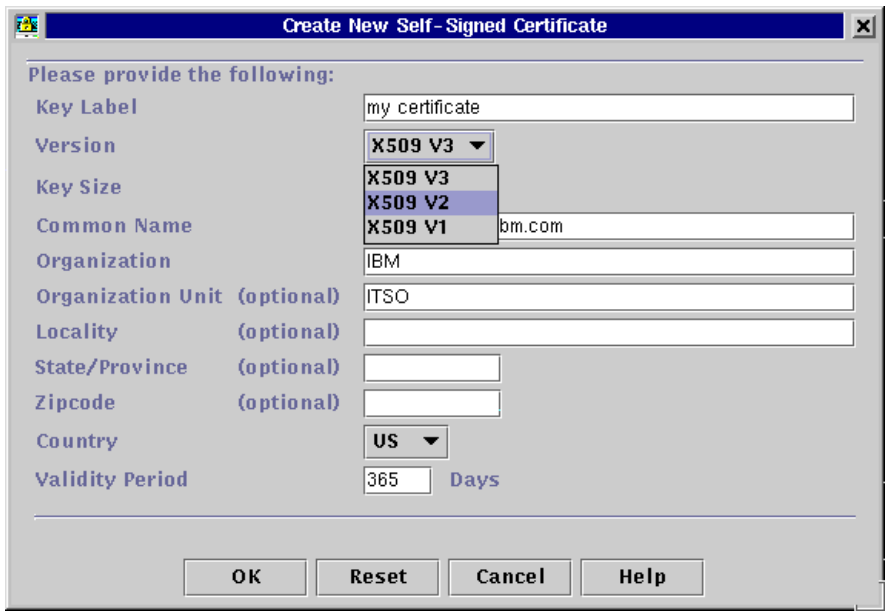


Figure 154. Create a self-signed certificate

A.3 Register key database with IBM HTTP Server

After creating a server key database, a self-signed certificate is created. For IBM HTTP Server to use this server certificate to encrypt SSL connection with a client browser, the following steps must be completed:

1. Register the server key database with the server.

   If you just installed IBM HTTP Server and have not modified your httpd.conf file, you can replace your existing httpd.conf with httpd.conf.sample, which is a template file for setting the SSL configuration for your HTTP Server. Otherwise, reference httpd.conf.sample to edit your existing httpd.conf file.

   To use the template file, copy it as httpd.conf, and edit httpd.conf as follows:

   1. Uncomment the LoadModule directive for `/libexec/mod_ibm_ssl_56.so`. See Figure 155.

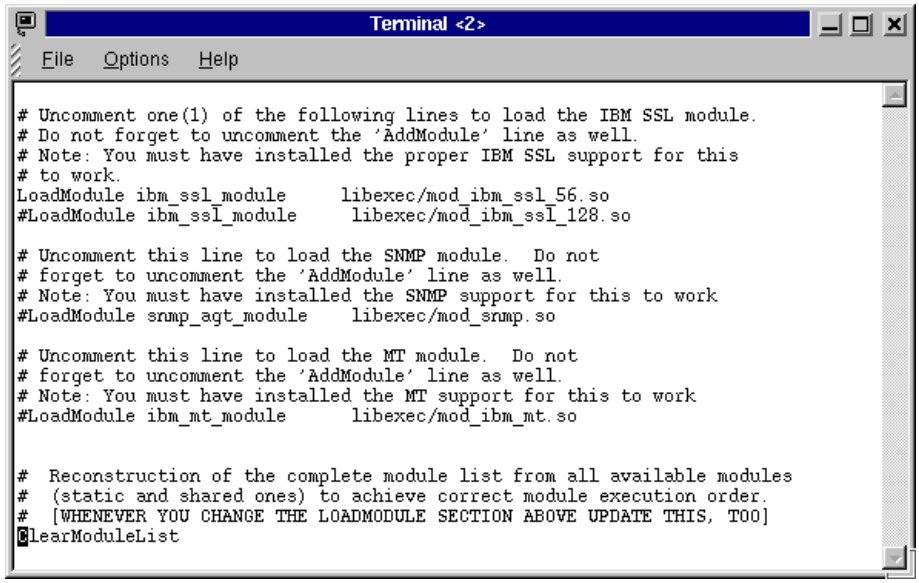


Figure 155. LoadModule directive enables 56-bit encryption module

2. Uncomment `AddModule for mod_ibm_ssl.c`. See Figure 156.

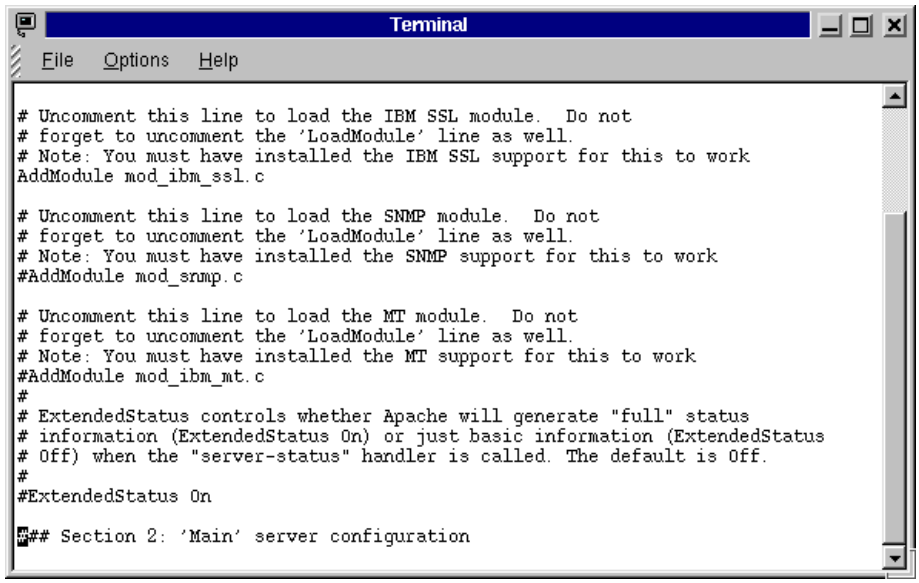


Figure 156. AddModule directive is used for loading SSL library

3. Ensure that the line `Listen 443` is uncommented.

4. Place the host name of the server in the virtual host stanza for port 443. See Figure 157.

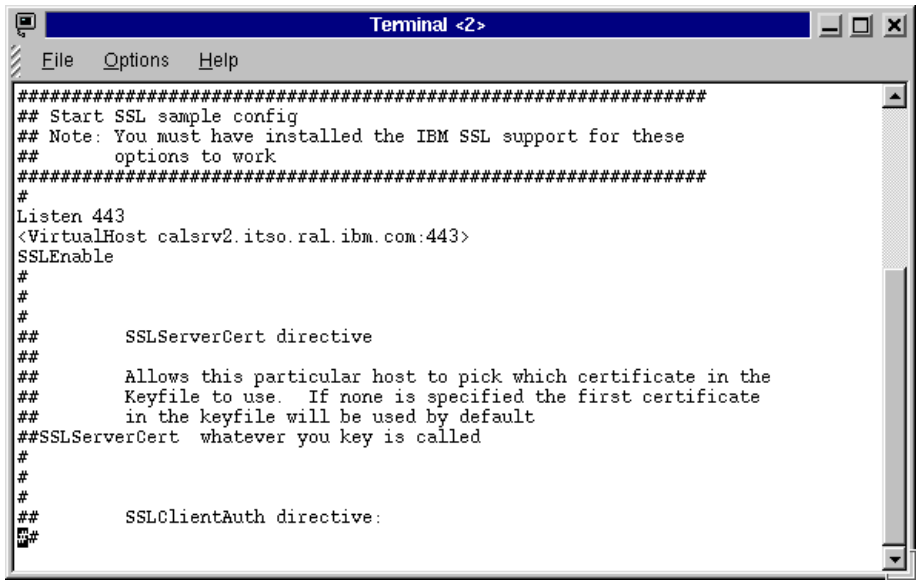


Figure 157. Use VirtualHost tag to listen to port 443

5. Ensure that the `SSLEnable` line is uncommented in the virtual host stanza.
6. Uncomment the </VirtualHost> tag.
7. Set the Keyfile directive; it belongs outside of the virtual host stanza. See Figure 158 on page 272.

Note

The directory /opt/IBMHTTPServer/keys/ was created manually using the command line: mkdir /opt/IBMHTTPServer/keys during creation of the key database. Enter your directory path where you store your key file for this directive.

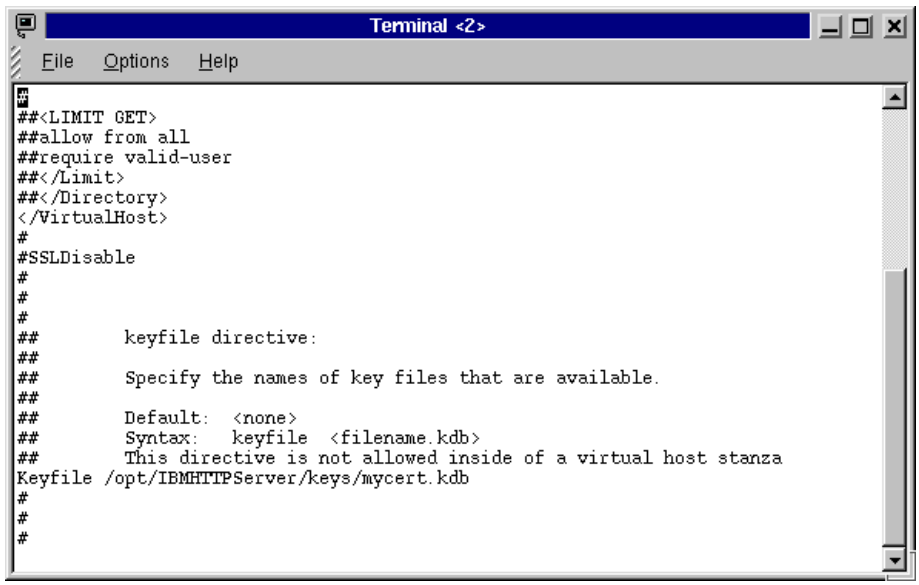


Figure 158. Register the server certificate for IBM HTTP

2. Restart the server. Type `reboot` at a command line. As an alternative you can restart the server by typing the following commands at the console:

```
/etc/rc.d/init.d/ibmhttpd stop
/etc/rc.d/init.d/ibmhttpd start
```

To access the server using SSL, use the HTTPS protocol, Type the following into your browser's URL location bar:

```
https://<hostname>
```

where `<hostname>` is the domain name or IP address set in the httpd.conf file. For example, `https://calsrv2.itso.ral.ibm.com` in Figure 159 on page 273.

Figure 159. SSL-enabled IBM HTTP Server using 56-bit encryption

Appendix B. Special notices

This publication is intended to help customers, business partners and IBM employees implement WebSphere Application Server and DB2 Universal Database on the Linux operating system. The information in this publication is not intended as the specification of any programming interfaces that are provided by WebSphere Application Server and DB2 Universal Database. See the PUBLICATIONS section of the IBM Programming Announcement for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee

that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX	AS/400
CICS	DataJoiner
DB2	Distributed Relational Database Architecture
IBM	IMS
MQSeries	Net.Data
Netfinity	OS/2
OS/390	RS/6000
S/390	System/390
TechConnect	VisualAge
WebSphere	

The following terms are trademarks of other companies:

TurboLinux, Inc., TurboLinux, and TurboLinux logo are trademarks of TurboLinux Incorporated.

Red Hat is a registered trademark, the Red Hat Shadow Man logo and RPM are trademarks of Red Hat Software, Inc.

SuSE is a trademark of SuSE GmbH.

Caldera Systems, Inc. is a registered trandmark of Caldera, Inc. OpenLinux is a trademark of Caldera, Inc.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United

States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

SET and the SET logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

Appendix C. Related resources

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook:

- *The Cathedral and the Bazaar* by Eric S Raymond, found at `http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar.html`
- *The CGI Specification*, found at `http://hoohoo.ncsa.uiuc.edu/cgi/interface.html`

C.1 Redbooks on CD-ROMs

Redbooks are available on the following CD-ROMs. Click the CD-ROMs button at `www.redbooks.ibm.com/` for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

C.2 Referenced Web sites

The following World Wide Web sites may provide more information about the topics discussed in this redbook:

- `www.linux.org`
- `www.redhat.com`
- `www.caldera.com`
- `www.turbolinux.com`
- `www.suse.com`

- www.blackdown.org
- www.software.ibm.com/webservers/
- www.software.ibm.com/data/db2/linux/
- www.software.ibm.com/ad/vajava/
- www.alphaworks.ibm.com
- www.fsf.org
- learn.ibm.be/linux/
- www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar.html
- www.gnu.org/copyleft/gpl.html
- www.metalab.unc.edu/Linux/
- www.cert.org
- www.ctv.es/USERS/xose/linux/linux_ports.html
- www.tuxedo.org/~esr/halloween.html
- www.netcraft.com/survey
- www.sitemetrics.com/serversurvey/index.htm
- www.zdnet.com/sr/stories/news/0,4538,2309670,00.html
- www.ibm.com/linux/
- www.software.ibm.com
- www.javasoft.com
- www.javasoft.com/products/servlet/index.html
- www.software.ibm.com/webservers/
- www.javasoft.com/products/servlet/index.html
- www.javasoft.com/security/usingJavakey.html
- www.w3.org/Protocols/
- www.ibm.com/developer/xml/
- www.alphaworks.ibm.com
- java.sun.com/products/
- www.software.ibm.com/webservers/httpservers/
- www.landrover.com
- ftp.calderasystems.com/pub/

- www.metalab.unc.edu/pub/Linux/distributions/redhat/redhat-6.0/i386/RedHat/RPMS/
- ftp://ftp.calderasystems.com/pub/openlinux/2.2/col/contrib/RPMS/
- ftp.linuxland.de/pub/OpenLinux/crypto/2.2/RPMS/
- ftp.software.ibm.com/ps/products/db2/tools/

How to get ITSO redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** http://www.redbooks.ibm.com/

  Search for, view, download, or order hardcopy/CD-ROM redbooks from the redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this redbooks site.

  Redpieces are redbooks in progress; not all redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders by e-mail including information from the redbooks fax order form to:

	e-mail address
In United States	usib6fpl@ibmmail.com
Outside North America	Contact information is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl/

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl/

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl/

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the redbooks Web site.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and redbooks by accessing the IBM Intranet Web site at http://w3.itso.ibm.com/ and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at http://w3.ibm.com/ for redbook, residency, and workshop announcements.

IBM redbook fax order form

Please send me the following:

Title	Order Number	Quantity

First name	Last name	
Company		
Address		
City	Postal code	Country
Telephone number	Telefax number	VAT number

☐ Invoice to customer number _____

☐ Credit card number _____

Credit card expiration date	Card issued to	Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

List of abbreviations

API	application program interface
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
DB2	DATABASE 2
DCS	Dynamically Constructed Servlet
DRDA	Distributed Relational Database Architecture
EJB	Enterprise JavaBeans
GB	gigabyte
GNOME	GNU Network Object Model Environment
GPL	General Public License
GSK	global security kit
GUI	graphical user interface
HP	Hewlett Packard
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Secure HTTP
IBM	International Business Machines Corporation
IIOP	open database connectivity
IP	Internet Protocol
IT	information technology
ITSO	International Technical Support Organization
JDBC	Java database connectivity
JDK	Java Development Kit
DOM	Document Object Model
JSP	JavaServer Pages
KB	kilobyte
KDE	K Desktop Environment
LAN	local area network
MB	megabyte
MIME	Multipurpose Internet Mail Extensions
ODBC	open database connectivity
ORB	object request broker
RAM	random access memory
RDBMS	relational database management system
RMI	remote method invocation
RPM	Red Hat package manager
SAM	Site Activity Monitor
SAX	simple API for XML
SQL	structured query language
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
URI	Universal Resource Identifier
URL	Universal Resource Locator
WWW	World Wide Web
XML	eXtensible Markup Language

Index

R

S

T

V

W

X

ITSO redbook evaluation

Linux for WebSphere and DB2 Servers
SG24-5850-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at `http://www.redbooks.ibm.com/`
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to `redbook@us.ibm.com`

Which of the following best describes you?
_ Customer _ Business Partner _ Solution Developer _ IBM employee
_ None of the above

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

What other redbooks would you like to see published?

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

SG24-5850-00
Printed in the U.S.A.

Linux for WebSphere and DB2 Servers

SG24-5850-00