IBM

Linux Web Hosting with WebSphere, DB2, and Domino

- Fast track for Web server development in Linux
- Domino and DB2 database side-by-side in action
- JSP and Servlet interaction explained

Budi Darmawan
Alistair le Roux
Yu-Phing Ong
Iwan Setiawan

Redbooks

ibm.com/redbooks

SG24-6007-00

International Technical Support Organization

Linux Web Hosting with WebSphere, DB2, and Domino

June 2000

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix G, “Special notices” on page 213.

First Edition (June 2000)

This edition applies to IBM WebSphere Application Server Standard Edition Version 2.03, IBM Database 2 Universal Database Version 6.1 and Lotus Domino Server Version 5.03 running on RedHat Version 6.2 and Caldera eDesktop Version 2.4.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. 0SJB Building 003 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493

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 is intended to assist Web server administrators utilize IBM software offerings to build and manage a Web server using Linux as the base operating system. It is assumed that the reader has basic Linux administration skills, an understanding of general HTML syntax, and is knowledgeable in Java programming.Therefore, this redbook is not aimed at providing basic installation and configuration information on Linux or IBM software on Linux.

The approach taken in this redbook is to create a Web server on Linux using IBM software as a case study. The Web server that we build is a customized IBM Redbook homepage, and we use the following:

- Linux, using the RedHat version 6.2 and Caldera eDesktop 2.4
- IBM HTTP server version 1.3.6
- IBM Database 2 version 6.1 with FixPack 3
- IBM WebSphere Application Server version 2.03
- Lotus Domino Server version 5.03

The topics that we discuss are:

- Creating dynamic Web pages using Java Server pages (JSP), servlets and Java bean
- Providing dynamic user and user profile management using an interface to DB2
- Integrating access to Lotus Domino database using the DIIOP/CORBA interface
- Operating the Web server

The team that wrote this redbook

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

Budi Darmawan is a system specialist at the International Technical Support Organization, Austin Center. He writes extensively and teaches IBM and Tivoli classes worldwide on all areas of database, programming, and system management. Before joining the ITSO, he worked in IBM Indonesia as solution architect for the services team. He has worked many years in database and general system management in OS/390 and distributed

systems. His current interest is in Tivoli system management software and Web and Java programming.

Alistair le Roux is a Senior CE in South Africa. He has two years of experience with IBM in RS/6000 hardware and AIX. His areas of interest include Linux, AIX, DB2, and e-commerce products. This is his first Redbook.

Yu-Phing Ong is a Senior Software Specialist in IBM Australia. He has six years of experience with IBM in the fields of database, business intelligence, and e-commerce. His areas of expertise include WINTEL, Unix, and programming. He has written extensively on DB2 and Net.Commerce in redbooks and certification guides.

Iwan Setiawan is a System Engineer in PT Sistelindo Mitralintas, Indonesia. He has three years of experience in the networking field. He holds a degree in Physics from University of Indonesia. His areas of expertise include Linux and Windows NT system administration.

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

John Owczarzak
International Technical Support Organization, Austin Center

Catherine A. Webb
IBM Integrated Solution PDT leader

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 “IBM Redbooks review” on page 229 to the fax number shown on the form.
- Use the online evaluation form found at `http://www.redbooks.ibm.com/`
- Send your comments in an Internet note to `redbook@us.ibm.com`

Chapter 1. Project introduction

In this chapter, we discuss background information on the overall project. The discussion is put forward in the following sections:

- Section 1.1, “Linux and IBM” on page 1 shows IBM’s commitment in Linux as one of the supported operating systems.
- Section 1.2, “Pattern for e-business” on page 2 shows the structured approach on building e-business within the enterprise using patterns as the basic building blocks.
- Section 1.3, “Penguins on the Web” on page 4 provides the background information on this redbook and the project.
- Section 1.4, “Document organization” on page 4 gives the overall document structure of the redbook and an expanded Table of Contents.

1.1 Linux and IBM

As stated in the Linux at IBM Web site, `http://www.ibm.com/linux`, IBM is committed to supporting Linux as an option in the customer’s operating system:

> *IBM is committed to supporting your choice of platform and operating systems -- a commitment we're extending to the support of Linux, the open-source operating system. To meet increased demand for this dynamic operating system, we provide the most comprehensive hardware, software and support solutions. Linux users can draw on IBM's extensive experience, skills and resources to address their technical needs. Customers needing Linux support today can call IBM Global Services for all their Linux support needs, from consulting and enablement assistance to Operational Support Services. Customers can call a 24-hour telephone hot line or access a web-based help center to receive Linux support.*

There are several IBM products that are currently available in Linux. There are also a lot of other IBM products that are currently in beta test for Linux. The key IBM alliance for Linux distributions are:

- Caldera Systems
- Red Hat
- Turbo Linux
- SuSE

However, since availability of some of the products is limited, we only tested scenarios in RedHat Version 6.2 and Caldera eDesktop Version 2.4.

1.2 Pattern for e-business

The patterns for e-business aim to provide a readily available design pattern on building an e-business system, including the system architecture and guidelines. These patterns are cataloged in the following schemes:

- User-to-business
- User-to-online buying
- Business-to-business
- User-to-data
- User-to-user
- Application integration

For more information on these patterns for e-business, see the IBM developerWorks Web site at:

`http://www.ibm.com/software/developer/web/patterns`

We have chosen to implement the most common subset of the patterns, the user-to-online buying, which is a subset of the user-to-business pattern. The application topology that we choose is a Web-up approach, meaning that we create the application from scratch in the Web server with no backend integration. Figure 1 on page 3 shows the ideal run-time topology for this pattern.

In Figure 1 on page 3, the e-business system for an enterprise spans from the demilitarized zone (DMZ) to the enterprise's internal network. The following components comprises the e-business solution for the user-to-online buying pattern:

- The protocol firewall filters unwanted and unnecessary incoming protocol packages. It only allows incoming requests to certain machines and certain ports.
- The request dispatch acts as the load balancing for multiple Web server systems. It routes the requests to a certain Web server to facilitate a balanced overall load, thus enhancing the overall throughput.
- The Commerce server node is the Web server with its application server, which processes HTTP requests from the Internet.
- The domain firewall protects the internal network by only allowing certain machines in the DMZ to send request packages to the internal network.

- The DB server node provides the database support for the Web application. The data in the database is considered secure, as it is protected by both the protocol firewall and the domain firewall.

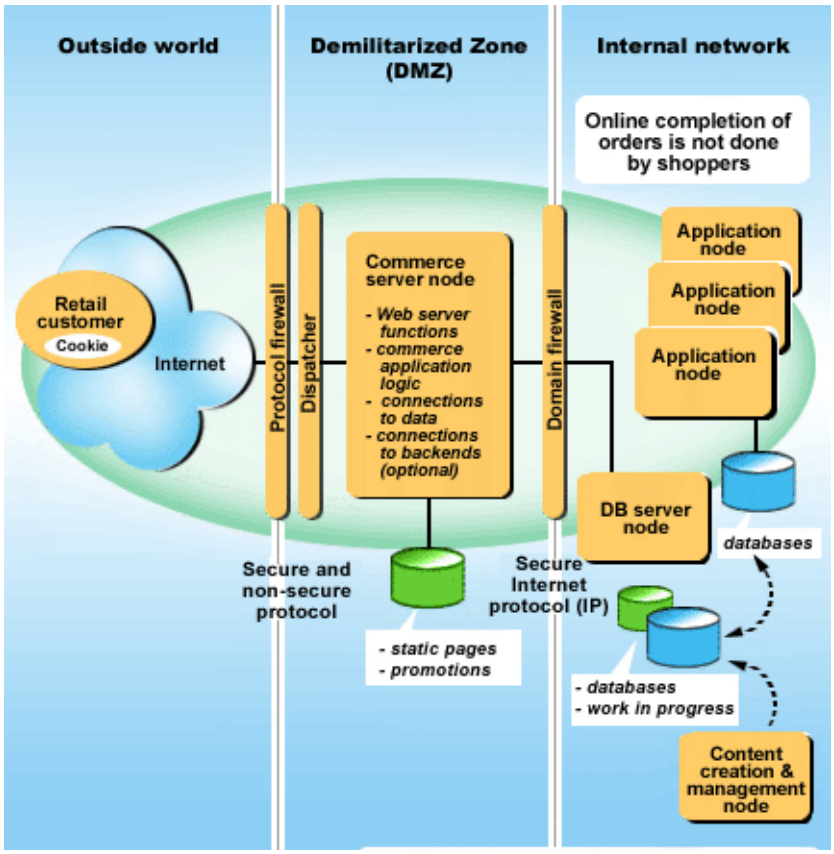

Figure 1. Run-time topology for user-to-online buying Pattern for e-business

The pattern for e-business also includes product selection. However, there is currently no Linux option in the product selection; so, we have to define the product selection ourselves. We implement the IBM HTTP server and IBM WebSphere Application Server in the commerce server node; whereas on the DB server node, we put IBM DB2 and the Lotus Domino Enterprise Server. For simplicity, we do not implement the firewall or the dispatcher.

For in-depth discussion on user-to-business and user-to-online buying patterns and their guidelines, refer to the *Patterns for e-business: User-to-Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition*, SG24-5864.

1.3 Penguins on the Web

This redbook is produced from a residency project at the ITSO in Austin, Texas. The redbook is aimed at showing IBM's ability to provide an advanced platform for generic Web hosting and e-business using Linux as the operating system.

The approach that we have taken is to actually build an imaginary Web server that provides some sample e-business functionality on Linux with IBM software. The Linux distributions that we use are the RedHat Linux Version 6.2 and Caldera eDesktop Version 2.4. The IBM software we utilize are:

- IBM WebSphere Application Server Version 2.03
- IBM HTTP server Version 1.3.6
- IBM Database 2 Version 6.1 FixPak 3
- Lotus Domino Version 5.03

The project is expanded further to provide some consideration on operating the Linux Web server. The consideration spans from basic daily Linux operation to product specific tasks that may need to be performed in administering a Linux Web server.

The sample application that we choose to provide is a customized Web site for the IBM Redbooks page. The Web application that we built consists of the following:

- Dynamic user and user profile management using IBM DB2
- Dynamic Web page using Servlet, Java Server Page (JSP), and Java beans
- Servlet and beans management using WebSphere Application server
- Access to the IBM Redbooks database in the Domino server

Putting all the above functions together shows a lot of the inter-operability issues that we experienced. There are several other redbooks that also discuss software and integration with Linux, but, as it turns out, integrating all software in building a working solution is another task in itself.

1.4 Document organization

This redbook is aimed at providing usage and administration information for Web administrators using the Linux Web server with IBM products. We will not cover the basic installation of products. It is assumed that the reader of

the redbook has basic HTML skill, Java programming, and Linux administration knowledge. This document has the following chapters:

- Chapter 1, “Project introduction” on page 1, provides some background information on the overall project.
- Chapter 2, “Setup and configuration” on page 7 provides an overview of the installation process and procedure that we performed on our system. The installation process is mostly described briefly except for some specific steps that are required to make the product work.
- Chapter 3, “Building the Web server” on page 53 discusses the steps that we took on building our Web server. This chapter discusses the programming structure that we used and the code that we built. The coding techniques that we cover include:
  - Access to DB2 using JDBC
  - Access to Domino using IIOP
  - Generic HTTP session access
  - Java Server Page implementation
  - Java beans programming
- Chapter 4, “Operational considerations” on page 107 shows some basic operational tasks that may need to be performed to ensure the operation of the Web server. The operation that is discussed here is the basic starting and stopping of the facilities, routine maintenance that must be performed and some monitoring and troubleshooting tips.
- Chapter 5, “Extending the case study” on page 139 discusses some things that can be achieved but is not currently implemented in this project because of time limitations or product limitations.

The appendices that we provide contains various source code and HTML files that we used throughout the projects.

We also provide a CD-ROM for all the source codes of the Java files, scripts, HTML files, and JSP files that we used. Refer to Appendix F, “Using the additional material” on page 211 for directions on using the CD-ROM.

Chapter 2. Setup and configuration

This chapter contains the basic setup and configuration tasks that we performed in preparing for the implementation of the scenario. The items we discuss here consists of:

- Section 2.1, “System configuration” on page 7 shows the system configuration that we used in our environment.
- Section 2.2, “Installation overview” on page 8 provides an overview of the overall setup and configuration tasks.
- Section 2.3, “Linux setup” on page 11 discusses the issues involved in setting up the Linux systems, including the hardware requirement, disk partitioning scheme, and actual consideration on RedHat and Caldera distributions.
- Section 2.4, “Application installation” on page 14 shows the installation and configuration of the IBM software that is used in this project.

2.1 System configuration

Based on the pattern of e-business user-to-online buying that we choose in Section 1.2, “Pattern for e-business” on page 2, we have a front-end server and a backend server. The system configuration that we used is shown in Figure 2.

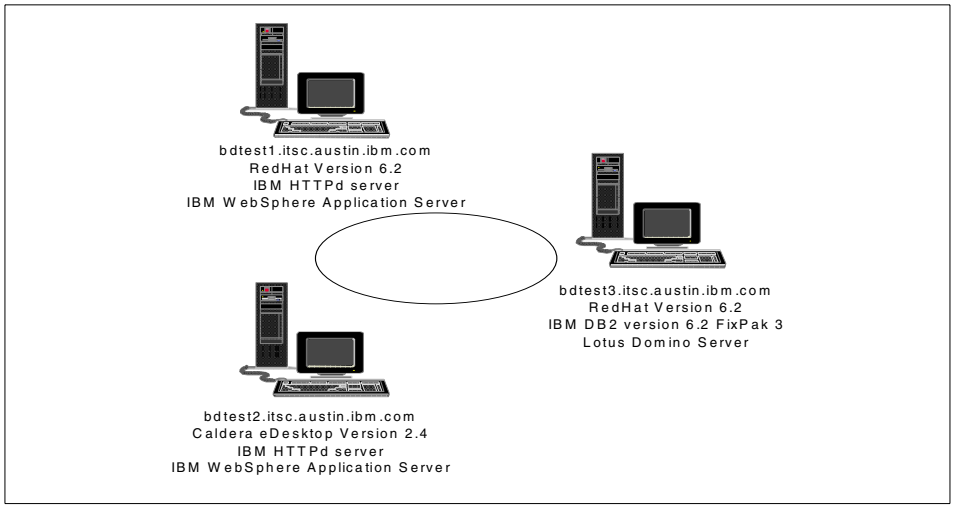

Figure 2. System configuration for the project

We have bdtest1 and bdtest2 as the front-end Web server, the interface to the world. All the application data resides in bdtest3 where the Domino and DB2 databases reside. We use both Caldera eDesktop Version 2.4 and RedHat Version 6.2 for the front-end server, and for the back-end server, we only use RedHat Version 6.2.

The machines that we used are IBM Netfinity 3000 servers. These Netfinity 3000 systems have the following specifications:

- Model: 8476-20U
- Flash EEPROM level: NTKT29AUS
- BIOS Date: 02/10/00
- Processor: Pentium II 350/100 Mhz
- Memory: 192MB
- SCSI: Adaptec AHA 2940 uWB Bios 1.32S8
- Disk: IBM 8 GB
- Video Adapter: S3 Trio3D 4 MB RAM
- Network: IBM PCI Token Ring Card

2.2 Installation overview

This section provides a series of global steps that were used to install the systems. The steps that are presented in this section can serve as either the overview or as a checklist to perform the installation. Detailed discussion of each step is provided in the subsequent sections.

We present the steps for the backend server first, ensuring the databases are available for access, then the steps for the front-end servers, and then verify that the front-end servers can access the backend database.

2.2.1 Installation of backend server

The backend server installation steps are:

1. Prepare the hardware and ensure that it satisfies all the required configuration for all the software that will be used.
2. Install our chosen Linux distribution.
   - Add the userID admin as a normal user.
   - Become familiarized with the preferred X-Window environment: KDE or GNOME.

3. Install DB2.
   - To install DB2, log in as the root user.
   - For RedHat installation, install rpm package for pdksh so that the DB2 installation program can run.
   - During installation, create the default DB2 instance and DB2 administration server instance.
   - Verify the DB2 installation by logging in as user db2inst1, then creating and accessing the sample database.
4. Install Domino Enterprise server from the console.
   - To install Domino, log in as the root user.
   - Stop the http, mail, and other services to prevent conflict with Domino.
   - After installation, log in as the notes userid, run the `httpsetup` command from Domino data directory, and use a Web browser to run the administration configuration of Domino server.
   - Run the `server` command from the Domino data directory and use a Web browser to test the Domino server installation.
   - Verify that HTTP, DIIOP, and Java components are running in Domino server.
   - Use the Domino administrator client or Web administration to configure Domino, such as CORBA access, HTTP port, and so on. Ensure that the Domino server is configured with the correct *fully qualified Internet host name*.
   - Test the new configuration using a Web browser.

2.2.2 Installation of front-end server

The front-end server installation steps are:

1. Prepare the hardware and ensure that it satisfies all the required configuration for all the software that will be used.
2. Install the chosen Linux distribution.
   - Add the admin name as a normal user.
   - Become familiarized with the preferred X-Window environment: KDE or GNOME.
3. To install Java, we used IBM JDK118.
   - To install Java, log in as the root user.
   - For Caldera installation, remove old Java versions.

- After installation, export the JAVA_HOME environment variable.

4. Install IBM HTTP Server 1.3.6
   - To install the IBM HTTP Server, log in as the root user.
   - Remove any other Web server installations, for example Apache, and its associated dependencies.
   - After installation, modify the run levels so that the IBM HTTP Server will start on reboot. Reboot the server. The IBM HTTP server's httpd daemon should now start normally.
   - Use a Web browser to test the IBM HTTP server.
5. Install WebSphere 2.0.3
   - To install WebSphere, log in as the root user.
   - Ensure the JAVA_HOME environment variable is set.
   - Install the core and Web server specific WebSphere rpm packages.
   - For JSP to work, change ownership of the pagecompile directory to user nobody.
   - Modify and verify various IBM HTTP Server and WebSphere Application server configuration files.
   - Restart IBM HTTP Server. This should load and run WebSphere.
   - Use a Web browser to test that WebSphere runs the snoop servlet, and test that WebSphere administration in localhost for port 9527.
6. Configure access to the backend databases
   - Install the DB2 Client Application Enabler (CAE) and create the default DB2 instance with Java support.
   - Verify connectivity to the sample database on the backend server.
   - Patch the bootstrap.properties file and the ibmhttpd startup script to access DB2.
   - Restart WebSphere and the IBM HTTP server, and then test DB2 connectivity to the sample database.
   - Copy the NCSO.jar file from the backend server.
   - Patch the WebSphere bootstrap.properties file to allow access to the Domino server.
   - Restart WebSphere and the IBM HTTP Server, and then test Domino connectivity.

2.3 Linux setup

This section discusses the issues involved in setting up the Linux operating systems on our Netfinity servers. This section is not intended to cover all aspects of Linux installation, but merely highlight some actions that we have performed to make the Linux system run. Refer to the bibliography for reference to several books that have more information on setting up Linux on Netfinity servers.

This section is divided into:

- Section 2.3.1, “Disk partitioning” on page 11, which discusses our disk partitioning strategy
- Section 2.3.2, “RedHat installation” on page 12, which discusses the RedHat Version 6.2 installation
- Section 2.3.3, “Caldera eDesktop installation” on page 13 which discusses the Caldera eDesktop Version 2.4 installation

2.3.1 Disk partitioning

Our disk partitioning was based on the defaults for each Linux distribution. For example, output from the `df` command for the RedHat installation is shown in Figure 3.

```
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/sda8               256667     31415    212000  13% /
/dev/sda1                23302      2875     19224  13% /boot
/dev/sda6              3842376      2096   3645092   0% /home
/dev/sda5              3842376    440776   3206412  12% /usr
/dev/sda7               256667      6117    237298   3% /var
```

Figure 3. Output of the df command

Specific directories will be used as follows:

- /home will contain the DB2 instance directory.
- /usr will contain the DB2 rpm installation
- Java, IBM HTTP Server, and WebSphere are installed under /opt. For simplicity, we make a symbolic link from /opt to /usr/opt rather than re-creating all the partitions by hand. Since the Domino installation uses a shell script that allows us to specify a directory for installation, we also choose to install Domino in /opt.

2.3.2 RedHat installation

Installation of RedHat is easy, simply boot off the RedHat 6.2 CD-ROM and choose the defaults for most options, with the following exceptions:

- Install type

  We chose Server for our installation type.

- Network Configuration

  Our network was token ring based; hence, we chose to set up the tr0 interface for the token ring. This option is available only from a GUI setup; it is not available when installing in text mode.

- Account Configuration

  As previously mentioned in 2.2, “Installation overview” on page 8, we created a userID called admin.

- X Configuration

  Since we would also need an X11 server to later configure WebSphere using a Web browser, we upgraded the Linux installation for the front-end server to include X11. Remember to install at least the following for X11:

  - XFree86
  - XFree SVGA server
  - A XFree86 server specific for the video card (in our case, we used the XFree S3 server)
  - X11R6 Contrib
  - Xconfigurator
  - various XFree86 fonts
  - XFree86 libraries
  - XFree86 xfs

  We found that choosing the combination of IBM G74 and S3 Trio3D with 4 MB RAM would not pass the X Configuration page’s test-configuration button test. The fix was to specify a custom monitor as SVGA non-interlaced, 50-90 Hz.

Once installation was complete, the CD was removed from the CD-ROM drive, and the server was rebooted.

We can now proceed to the application installation phase.

2.3.3 Caldera eDesktop installation

The Caldera eDesktop version 2.4 installation is easy, simply boot the Caldera eDesktop Version 2.4 CD-ROM and choose the defaults for most options, with the following exceptions:

- Network Configuration

  Our network was token ring based; hence, we chose to set up the tr0 interface for token ring. Caldera has scripts for ethernet network cards, but not for token rings. We modified the ethernet scripts to produce the required token ring interface. We used the following steps:

  a. Ensure the correct driver for the token ring has been loaded at boot time. This was the olympic driver for us. It is precompiled into kernel 2.2.14.

  Token ring olympic driver

  We had a problem with the olympic driver of the 2.2.14 in Caldera eDesktop 2.4; so, we changed the kernel rpm to the 2.2.14 kernel that is distributed for Caldera eServer 2.3 to activate the olympic

  b. Edit /etc/sysconfig/network file and add a token ring interface to IF_LIST as shown in Figure 4.

```
NETWORKING=yes
HOSTNAME=bdtest1.itsc.austin.ibm.com
IF_LIST='lo eth tr0 sl ppp
```

Figure 4. /etc/sysconfig/network file

  c. Copy /etc/sysconfig/network-scripts/ifcfg-eth0 to /etc/sysconfig/network-scripts/ifcfg-tr0 as shown in Figure 5.

```
[root@bdtest1 network-scripts]# cp ifcfg-eth0 ifcfg-tr0
```

Figure 5. Copy the eth0 script to tr0 script

  d. Edit the ifcfg-tr0 script to reflect our token ring network parameters. The content of ifcfg-tr0 is shown in Figure 6 on page 14

```
#!/bin/sh
#>>>Device type: token ring
#>>>Variable declarations:
DEVICE=tr0
IPADDR=192.168.1.2
NETMASK=255.255.255.0
NETWORK=192.168.1.0
BROADCAST=192.168.1.255
GATEWAY=none
ONBOOT=no
DYNAMIC=
#>>>End variable declarations
```

Figure 6. Content of the ifcfg-tr0 file

e. Issue the `ifup tr0` command to bring this network interface up.

- Account Configuration

  As previously mentioned in 2.2, “Installation overview” on page 8, we created a userID called admin.

- X Configuration

  X server is installed, by default, on Caldera e-Desktop Version 2.4. The process was painless, and it worked the first time using a P50 monitor attached to an S3 TRIO 3D, 4 Mb card with a resolution of 1024x768 at 70Hz.

Once installation was complete, the CD was removed from the CD-ROM drive, and the server was rebooted.

We can now proceed to the application installation phase

2.4 Application installation

The application installations are documented in the following order:

- Section 2.4.1, “DB2” on page 15 shows how to install DB2 Version 6.1 at fixpack level 3 in our backend server
- Section 2.4.2, “Domino Enterprise server” on page 19 shows how to install the Domino Version 5.03 in our backend server.
- Section 2.4.3, “IBM JDK 118” on page 32 shows how to install IBM Java Development Kit Version 1.1.8.

- Section 2.4.4, “IBM HTTP Server” on page 33 shows how to install IBM HTTP Server Version 1.3.6 in our front-end server.
- Section 2.4.5, “WebSphere” on page 39 shows how to install IBM WebSphere Application Server Version 2.03 in our front-end server.
- Section 2.4.6, “Configuring front-end access to backend databases” on page 44 concludes the installation by providing the connectivity from the front-end system to the backend databases

2.4.1 DB2

Before installing DB2 on a Linux-based workstation, we must ensure that our distribution of Linux has a kernel of 2.035 or higher. We also need to have the following packages installed:

- RPM (Red Hat package manager)
- pdksh package (public domain Korn shell)
- glibc Version 2.07 or higher
- libstdc++ Version 2.8.0

For RedHat Version 6.2 and Caldera eDesktop Version 2.4, all these requirements were met except for the existence of the pdksh package.

2.4.1.1 Installing pdksh

The pdksh shell is used by the DB2 installer and setup scripts. It is not normally installed as part of a server installation of Linux. Hence, we had to install it using the `rpm` command. Figure 7 shows the command we used to install the pdksh package from the CD-ROM.

```
[root@bdtest3 /root]# rpm -i /mnt/cdrom/RedHat/RPMS/pdksh-5.2.14-2.i3
86.rpm
```

Figure 7. Installing the pdksh package

2.4.1.2 Installing DB2

There are known installation problems with DB2 6.1 at fixpack level 1. Therefore, we chose to install DB2 v6.1 with fixpack 3 downloaded from IBM. The steps for installation are as follows:

1. Log in as root.
2. Change the working directory to the directory where the DB2 Version 6.1 fixpack 3 code is stored.

3. Execute the `db2setup` command. The Install DB2 V6 screen opens.
4. Using the Tab key, change the highlighted option to **DB2 UDB Enterprise Edition**. Press the **Enter** key to select this option. Tab the highlight to **OK**, and press **Enter** to continue.
5. In the next screen, Create DB2 Services, choose **Create a DB2 instance** by tabbing the highlight to that option, and pressing the **Enter** key.
6. Accept all defaults in the DB2 instance screen that appears next, by using the Tab key to move to **OK** and pressing the **Enter** key. Note that the system generated password that will be used is ibmdb2.
7. Similarly for the Fenced User screen, choose **OK** and note that the system generated password that will be used is also ibmdb2. This will now return to the DB2 Instance screen.
8. Since you may want to use the DB2 Administration Server, choose to install it now. Highlight the option to **Create the Administration Server**, and press the Enter key.
9. Once again, accept all defaults, press **OK**, and note that the system generated password is ibmdb2. A further notice states that the DB2 variable DB2SYSTEM will be set to the hostname of our server, in this case bdtest3. Press **OK** to continue.
10. Back to the DB2 Instance screen, you can proceed further by choosing **OK**.
11. This brings you to the Summary Report of the DB2 Installer screen. Choose **Continue**.
12. Choose **OK** to start the installation. The installation should proceed as shown in Figure 8 on page 17.

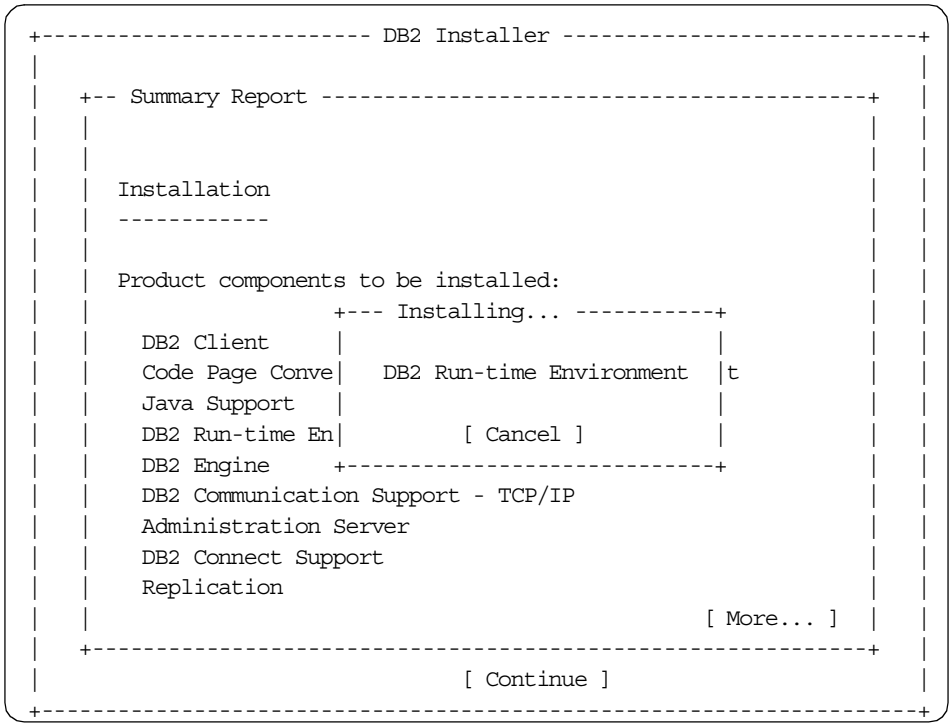

Figure 8. DB2 Installer Summary Report and installation window

13. When the installation is complete, choose **OK** to proceed to the Status Report screen. Verify that all components have installed successfully, and choose **OK** to return to the initial DB2 Installer screen
14. Choose **Close** to exit the DB2 installer, and **OK** to dismiss the final warning screen about exiting the DB2 installer.

2.4.1.3 Verifying the DB2 installation

Now that we have successfully installed DB2, we verify the installation by creating and accessing the sample database. To do this, perform the following steps:

1. Log on to the server as the db2 instance owner, db2inst1. Since you chose the defaults, you should be able to log on as db2inst1 with a password of ibmdb2.
2. Check that DB2 is running by issuing the `db2start` command.
3. Create the sample database by running the `db2sampl` command.

4. Once this command has completed, connect to sample database. This is shown in Figure 9.

```
[db2inst1@bdtest3 db2inst1]$ db2start
SQL1026N  The database manager is already active.
[db2inst1@bdtest3 db2inst1]$ db2sampl
[db2inst1@bdtest3 db2inst1]$ db2 connect to sample

   Database Connection Information

 Database server        = DB2/LINUX 6.1.0
 SQL authorization ID   = DB2INST1
 Local database alias   = SAMPLE
```

Figure 9. Creating and connecting to the DB2 SAMPLE database

5. Verify that the DB2 SQL engine is working by issuing a select from a table in the SAMPLE database. For our example, we used:

   ```
   db2 "select * from department"
   ```

   The results are shown in Figure 10.

```
[db2inst1@bdtest3 db2inst1]$ db2 "select * from department"

DEPTNO DEPTNAME                      MGRNO  ADMRDEPT LOCATION
------ ------------------------------ ------ -------- ----------------
A00    SPIFFY COMPUTER SERVICE DIV.  000010 A00      -
B01    PLANNING                      000020 A00      -
C01    INFORMATION CENTER            000030 A00      -
D01    DEVELOPMENT CENTER            -      A00      -
D11    MANUFACTURING SYSTEMS         000060 D01      -
D21    ADMINISTRATION SYSTEMS        000070 D01      -
E01    SUPPORT SERVICES              000050 A00      -
E11    OPERATIONS                    000090 E01      -
E21    SOFTWARE SUPPORT              000100 E01      -
```

Figure 10. An SQL query from the DEPARTMENT table of the SAMPLE database

6. Reset the database connection with:

   ```
   db2 connect reset
   ```

This has verified that the DB2 installation is working.

2.4.2 Domino Enterprise server

In our project, we installed the Domino server on the backend server, bdtest3. Before beginning the installation of the Domino server, you ensure the following:

- Verify library structure and availability

  Ensure that the kernel version is 2.2.5 or higher, and that glibc 2.1.1 or greater and libstdc++ 2.9.1 or greater was installed on the machine. More importantly, the distribution must follow the libstdc++-libc6 naming convention.

- Stop any unnecessary services on the Linux box.

  We have to make sure that services, such as http server or sendmail that may have started automatically, are stopped.

- Check terminal size

  Ensure that you have a terminal size of 80 columns by 24 rows, as this is required by the Domino installation program. You can check your terminal size by issuing the `echo $COLUMNS` and `echo $LINES` commands from the Linux prompt where you will perform the installation.

2.4.2.1 Installing Domino

Use the following steps to start installing Domino on your machine:

1. Log in as user root.
2. Create the userID for running Domino. We used userID called notes with group notes.
3. Change to the directory of the Domino installation package, have Domino in the /home/src/domino503src directory.
4. Type `./install` from the command prompt. The Domino installation screen appears.

> **Domino installation key usage**
>
> In the installation process we have to use the Tab key to continue, the Esc key to return to the previous screen, and the Spacebar key to make a choice of options.

5. Read pass the license agreement and select **Yes** to accept the license agreement.
6. Select the **Domino Enterprise Server** as the installation package that you want to install using Spacebar key.

7. Specify the installation target directory. We put the Domino server in the /opt/lotus directory.
8. If you want to install more than one Domino server in your machine, you can choose the option **Yes** in this screen to let Domino create this additional servers. In our project, we choose **No** because we just want to install one Domino server in our machine.
9. Specify the Domino data files directory. We installed the Domino data files in /local/notesdata directory.
10. Specify the userid and group that will own Domino data files and run the Domino server as notes as described in step 2 on page 19.
11. After that process, Domino displays the screen shown in Figure 11.
12. Press the **Tab** key and the Installation script will perform the installation tasks.

```
================================================================================
                        Domino Server Installation
================================================================================
Installation settings:

    Installation type    : Domino Enterprise Server

    Program directory    : /opt/lotus
    Data directory       : /local/notesdata
      UNIX user          :   notes
      UNIX group         :   notes


Press the Escape key to re-configure the settings
  or
Press the Tab key to perform the installation...
```

Figure 11. Domino installation screen

13. When the installation finishes successfully, the install program reminds you to log in as the userID notes to run the program rather than running Domino as root.

2.4.2.2 Running the httpsetup program

The initial configuration of Domino server is performed using a Web browser interface. The following steps discuss the process:

1. Log in with the notes userID.
2. Change the directory to the Domino data directory.

3. To start the HTTP task, issue the command: `/opt/lotus/bin/http httpsetup`

   If everything worked, we will see a screen similar to Figure 12.

```
[notes@bdtest3 notesdata]$ /opt/lotus/bin/http httpsetup
04/28/2000 06:00:10 PM  Created new log file as /local/notesdata/l
og.nsf
04/28/2000 06:00:10 PM
****************************************
*      Lotus Domino Server Setup       *
* To setup this server, please connect  *
*     your web browser to port 8081     *
* Example: http://this.server.com:8081  *
****************************************
04/28/2000 06:00:12 PM  JVM: Java Virtual Machine initialized.
04/28/2000 06:00:13 PM  HTTP Web Server started
```

Figure 12. Screen of Domino http setup

4. Start up a Web browser and connect to the local machine on port 8081 . The Web configuration window is shown in Figure 13.

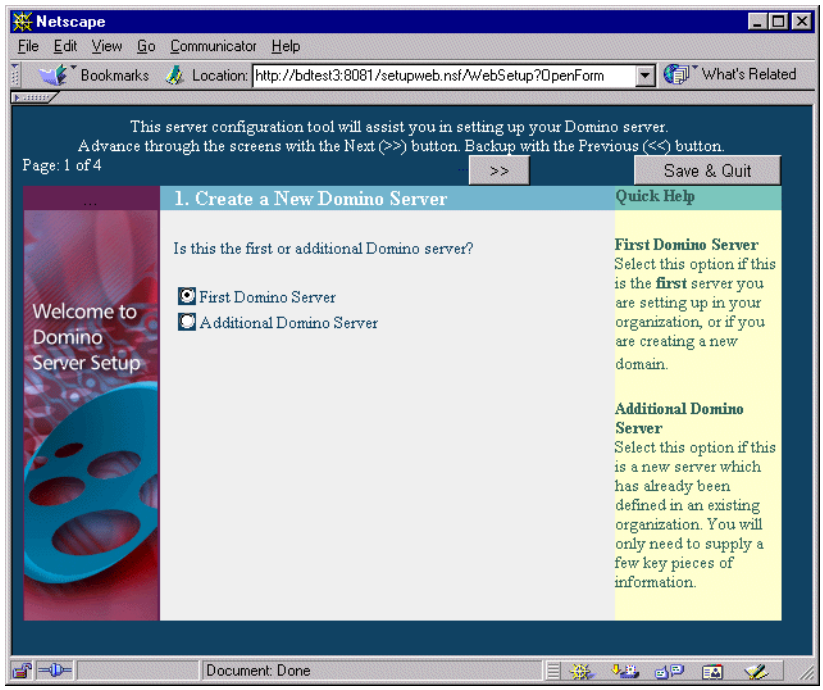

Figure 13. Domino Web installation

5. Click the >> button to continue to the Server Audience window. The services that we selected are shown in Figure 14.

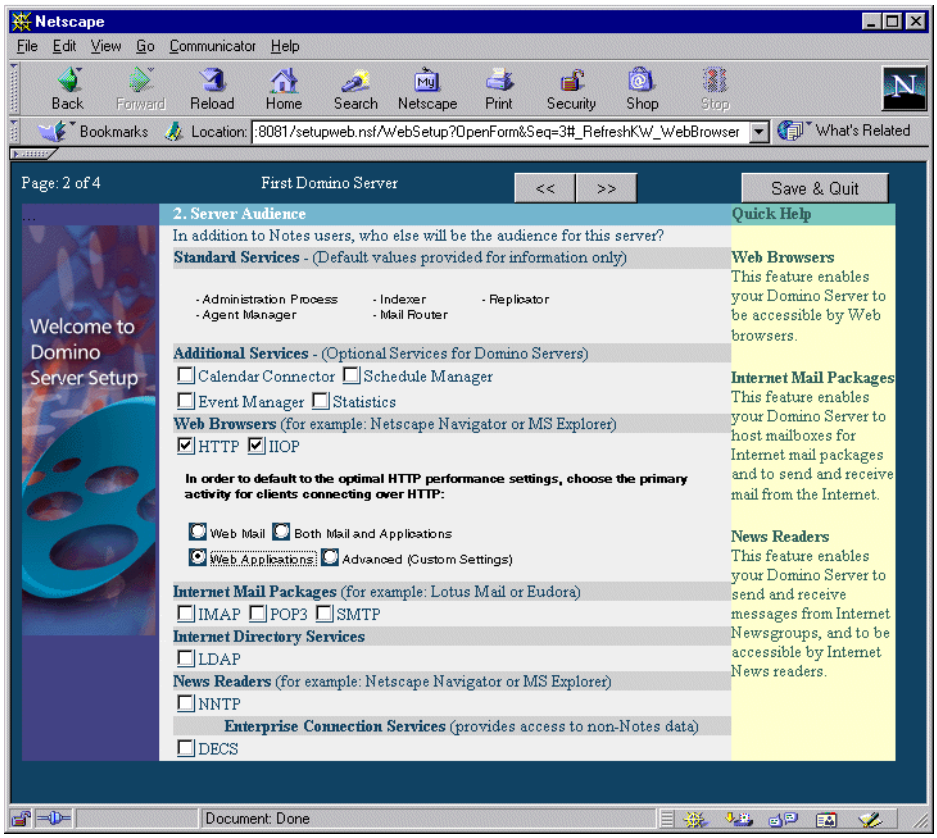

Figure 14. Server audience setting in our Domino server

6. The next screen that appears on the Web browser is Administration settings. We should fill in the appropriate fields to suit our settings. In our project, we filled the fields with the configuration as shown in Figure 15 on page 23.

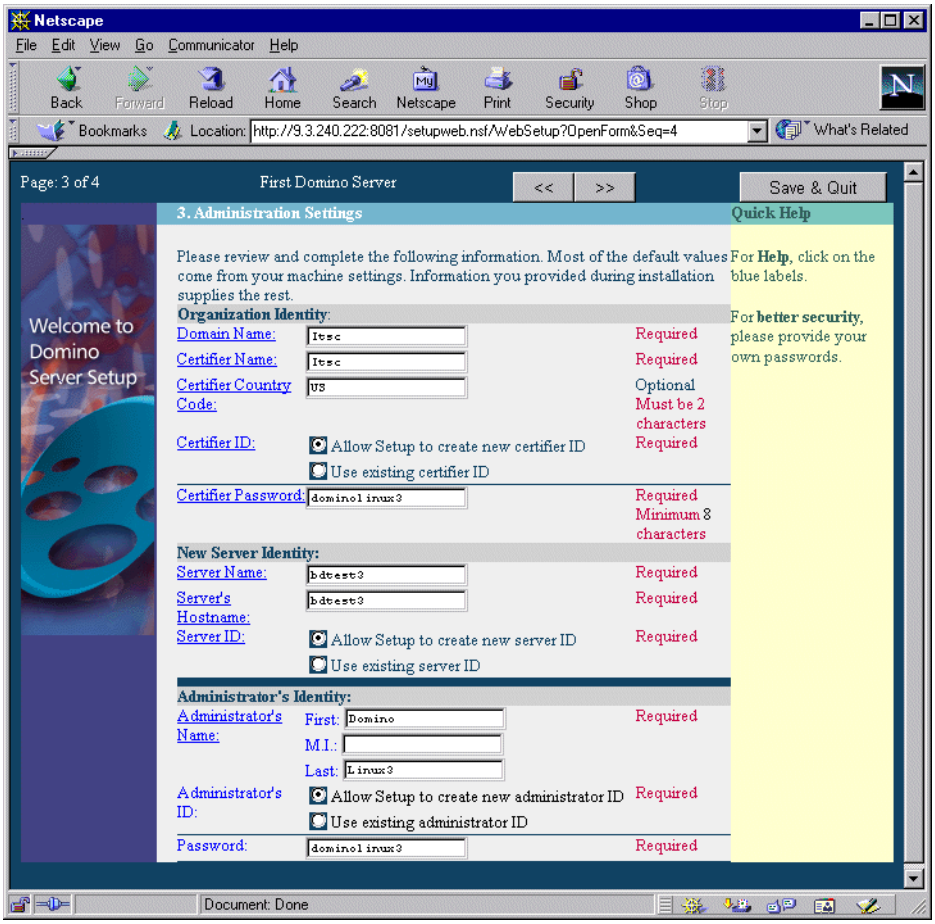

Figure 15. Administration setting in our Domino server

7. Specify the network and communication settings. We choose the installation default. After finishing filling in the field, click the **Finish** button.
8. A progress bar will appear in the Web browser, and when it is completed, click the **Exit** button from the Domino greeting window, and Domino will shutdown the Web server.
9. To start up the Domino server, we changed the current directory to the Domino data directory and typed in the following command:

   ```
   /opt/lotus/bin/server
   ```

   In Figure 16 on page 24 shows the screen with the Domino server startup command.

```
[notes@bdtest3 notesdata]$ /opt/lotus/bin/server

Lotus Domino (r) Server, Release 5.0.3 , March 21, 2000
Copyright (c) 1985-2000, Lotus Development Corporation, All Rights Reserved

05/09/2000 03:59:45 PM  Router: Unable to obtain Internet host and domain names
05/09/2000 03:59:45 PM  Mail Router started for domain ITSC
05/09/2000 03:59:45 PM  Router: Internet SMTP host bdtest3 in domain
05/09/2000 03:59:50 PM  Database Replicator started
05/09/2000 03:59:55 PM  Index update process started
05/09/2000 04:00:00 PM  Agent Manager started
05/09/2000 04:00:01 PM  JVM: Java Virtual Machine initialized.
05/09/2000 04:00:01 PM  AMgr: Executive '1' started
05/09/2000 04:00:05 PM  bdtest3/Itsc/US is the Administration Server of the
Domino Directory.
05/09/2000 04:00:05 PM  Administration Process started
05/09/2000 04:00:10 PM  JVM: Java Virtual Machine initialized.
05/09/2000 04:00:10 PM  HTTP Web Server started
05/09/2000 04:00:15 PM  DIIOP Server started on bdtest3
05/09/2000 04:00:15 PM  DIIOP port 63148 may not be available on this system,
will use port 60148 instead
05/09/2000 04:00:20 PM  Maps Extractor started
05/09/2000 04:00:25 PM  Database Server started
> 05/09/2000 04:00:25 PM  Maps Extractor: Building Maps profile
05/09/2000 04:00:25 PM  Maps Extractor: Maps profile built OK
```

Figure 16. Domino startup command

10. We can test our configuration by pointing the Web browser to our Domino server:

    `http://bdtest3/`

    If the Domino server is working properly, the window similar to Figure 17 on page 25 will appear in our Web browser.

Figure 17. The Domino Web server screen

2.4.2.3 Connecting a Domino client

We can perform many administration functions using a Web browser, but we still need the Domino Administrator and Designer client to perform any other tasks, such as creating IDs or designing new databases.

In our project, we install the Domino Administrator and Designer client in a separate machine since the Linux version of Domino Administrator and Designer is not available.

We need to extract the administrator ID file before performing any more administration on the Domino server with Domino client and move it to the machine we plan to use for administration.

1. Open the address book with a Web browser using the URL:
   `http://bdtest3.itsc.austin.ibm.com/names.nsf`

2. Enter the administrator user ID of Domino Linux3 and its password that is provided in the Administration setting window shown in Figure 15 on page 23.
3. Click to the **People view** and open the **Person** document for the administrator created earlier and download the USER.ID file to the administration machine.

Netscape users

If you are using a Netscape browser, you may have to rename the ID file to USER.ID.

To access the Domino server with our Domino Administrator or Designer client, follow these steps:

1. Open Notes Client and switch to the USER.ID file we downloaded earlier.
2. Click on the **Administrator** or **Designer** icon
3. For Designer, choose **File** -> **Database** -> **Open**, enter the server name or IP address of the new server, and click **Open**. After a few seconds, the Notes client displays the databases available for the new server.

   For Administrator, choose **File** -> **Open Server** and specify the server name or IP address of the new server and click **Open**.

2.4.2.4 Domino server Java and CORBA requirements

Before we can use a Java program to access Domino data with CORBA, we must set up and configure our Domino server to support the Java and CORBA environments.

HTTP and DIIOP

In Domino R5, the installation scripts should configure the server to run HTTP and DIIOP tasks by default, but we must ensure that we have the following line in notes.ini file:

```
ServerTask=<any other task>, HTTP, DIIOP
```

Also, make sure that the lines shown in Figure 18 on page 26 appear on the server console when starting up the Domino server.

```
05/03/2000 01:50:40 PM  JVM: Java Virtual Machine initialized.
05/03/2000 01:50:41 PM  HTTP Web Server started
05/03/2000 01:50:45 PM  DIIOP Server started on bdtest1.itsc.austin.ibm.com
```

Figure 18. Verifying Java, HTTP, and DIIOP tasks in the Domino server console

These tasks can be started with the following console commands:

- To load the HTTP task:

  ```
  load http
  ```

- To load the DIIOP task:

  ```
  load diiop
  ```

2.4.2.5 Configure Domino server using a Web browser.

Now, we will explain how we configured our Domino server in our project. In this description, we used a Web browser to configure the Domino server. We can also configure the Domino server using the Domino Administration client.

We configure Domino to use port 8000, for anonymous access using the following steps:

1. Access the Domino Web administrator by pointing the Web browser to the appropriate URL on our backend server, in this case:

   ```
   http://bdtest3/webadmin.nsf
   ```

2. The Domino server will ask for the Administrator userid and password.
3. A successful login to the server will present a screen similar to Figure 19 on page 28 in the Web browser.

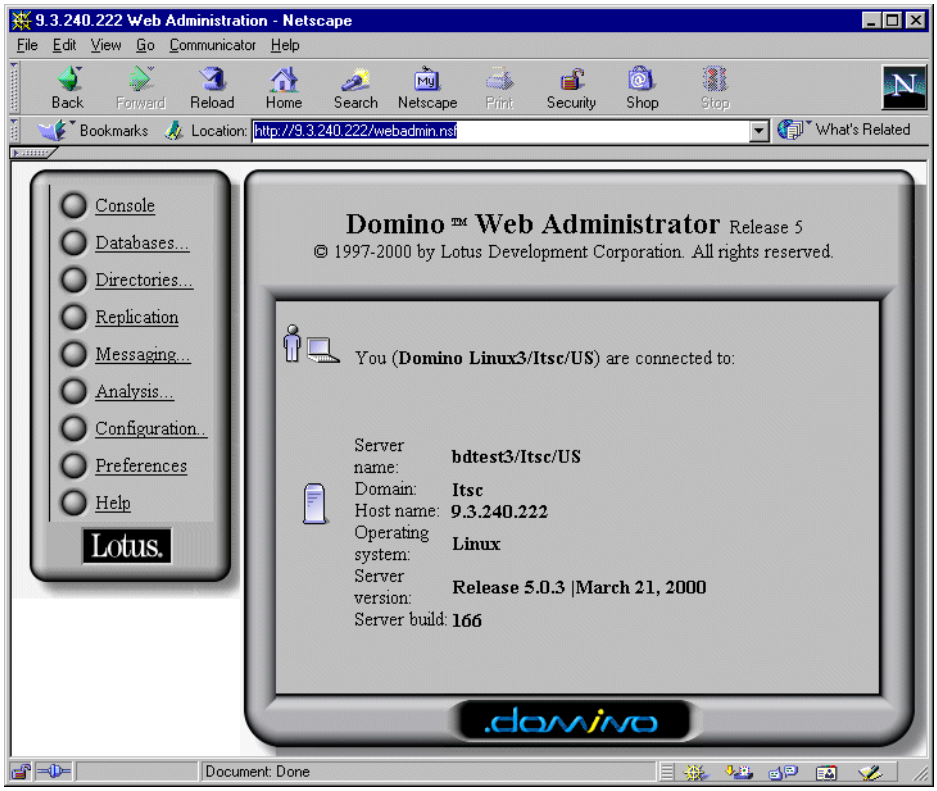

Figure 19. Domino Web Administrator

4. From the administrator window, as shown in Figure 19, click on **Configuration**.
5. Click on **Servers**, and you will get a window similar to Figure 20 on page 29.

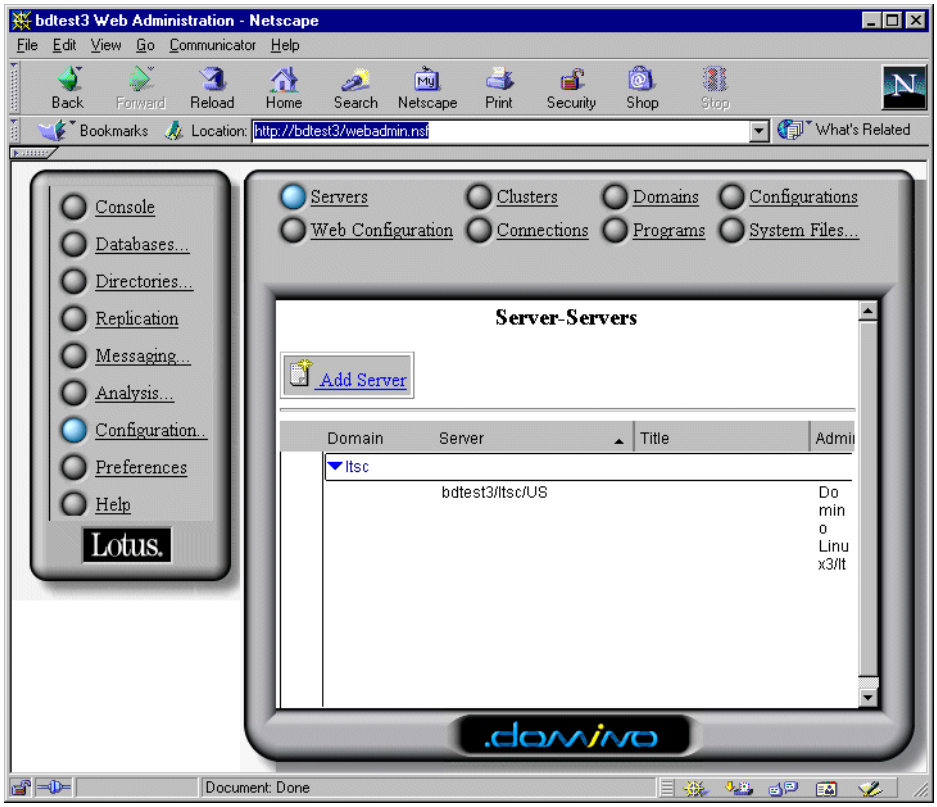

Figure 20. Configuration screen

6. Double click on the server name and after that, click on **Edit Server**, and a screen similar to Figure 21 on page 30 will appear on the Web browser.

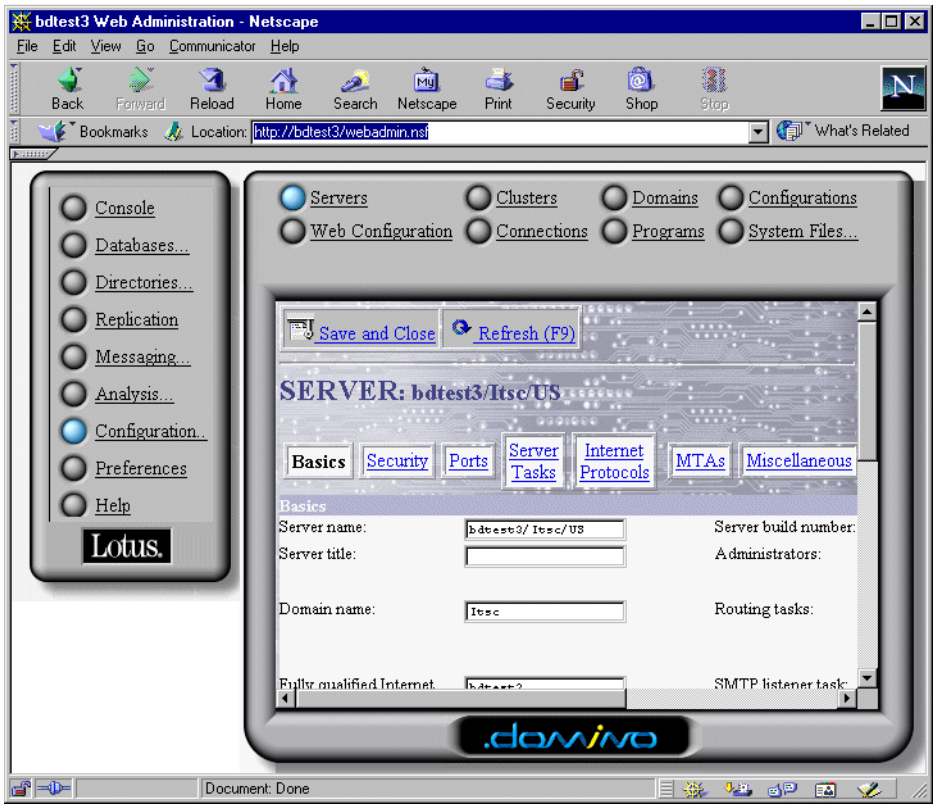

Figure 21. Edit server screen

7. In the Basic configuration, make sure that you have filled the **Fully qualified Internet host name** with the hostname, bdtest3.itsc.austin.ibm.com.

> **Fully qualified Internet host name**
>
> Ensure this field is filled in correctly. If not, Domino will not allow Java applications to connect using CORBA.

8. Click on **Ports** and then click on **Internet Port**.
9. Change the **TCP/IP port number** field from 80 to 8000.
10. Click the **IIOP Server** tab and complete these fields:
    - TCP/IP Port Number

This is the name of the port the DIIOP task listens on. Do not change this port unless you have the assigned port number 63148 (the default) to another task.

- TCP/IP port status: Enable

11. Click the **Internet Protocols** -> **IIOP** tab and complete this field:

    Number of Threads: This is the number of threads that you want to allow the DIIOP server task to process at the same time. The default is 10

12. Click the **Security** tab and complete these fields in the IIOP Restrictions section:

    - Run restricted Java/Javascript

      The name that the applet or application uses to access the server. Applet or application names entered in this field are allowed to run programs created using a restricted set of Java and Javascript features. If the applet or application logs on anonymously, enter the word **Anonymous** in this field.

    - Run unrestricted Java/Javascript

      The name that the applet or application uses to access the server. Applet or application names entered in this field are allowed to run programs created using all Java and Javascript features. If the applet or application logs on anonymously, enter the word **Anonymous** in this field.

    The security setting is shown in Figure 22 on page 32.

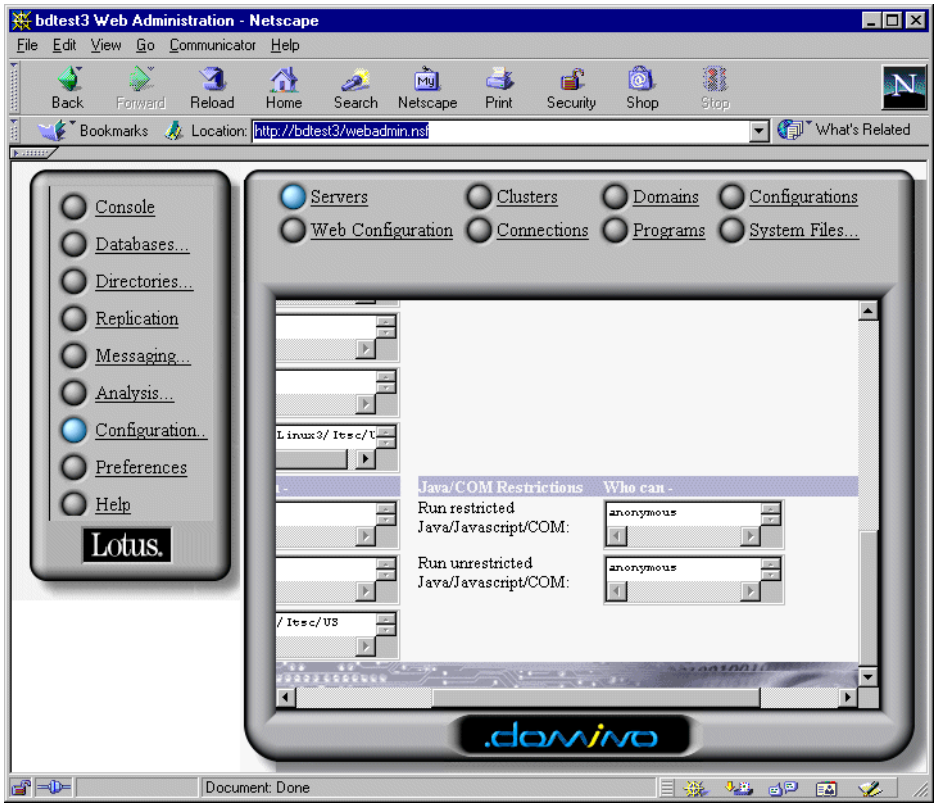

Figure 22. Security configuration in our project

13.Click on **Save and Close**.

After you have configured and saved the configuration, we must restart the Domino server. From the browser window in the webadmin.nsf page, click on **Console**. In the Domino Console field, type in the `restart server` command and click the **Send** button.

You can now test the new configuration by using a Web browser to point to the Domino server at port 8000.

2.4.3 IBM JDK 118

This section discusses the installation procedure for the Java Development Kit (JDK) in our project.

2.4.3.1 Pre-installation of IBM JDK 118

To start the pre-installation of IBM JDK 118, log in as the root user.

Before installing IBM JDK 118, you should ensure a clean installation of the Java environment. This involves removing any existing Java virtual machine and runtime environments, unless they are necessary for any other applications.

For example, we queried the Linux distributions with the command:

```
rpm -qa | grep jdk
```

We found that Caldera eDesktop 2.4 installs JDK 1.2.2-1 by default. Our scenario requires IBM JDK 118. Hence, the default JDK was removed with the command:

```
rpm -e jdk-1.2.2-1
```

2.4.3.2 Installing IBM JDK 118

Our JDK target installation is in the /opt directory.

The JDK118 distribution is in tar format and gzipped. Logged in as the user root, we change the working directory to the target installation directory, /opt, then used the `tar zxf` command to uncompress and untar this file, ibm-jdk-l118-linux-x86.tgz.

Since we will be installing IBM HTTP Server and WebSphere after installing the JDK, we also define and export the environment variable JAVA_HOME to indicate where the JDK has been installed.

Figure 23 shows the results of issuing the `tar` and `export` commands on our front-end server.

```
[root@bdtest2 /opt]# tar zxf /mnt/cdrom/Java/ibm-jdk-l118-linux-x86.tgz
[root@bdtest2 /opt]# export JAVA_HOME=/opt/jdk118/
```

Figure 23. Installing IBM JDK118 on the front-end server

2.4.4 IBM HTTP Server

IBM HTTP Server installation is performed while logged in as the root user.

We encountered some problems using the Apache Web server that comes bundled with RedHat 6.2 and Caldera eDesktop 2.4. If WebSphere is installed with the Apache Web server module, and Apache restarted, Apache complains about the Apache 1.3 API used by the WebSphere module. Figure 24 illustrates this error on RedHat 6.2.

```
[root@bdtest2 conf]# /etc/rc.d/init.d/httpd start
Starting httpd: [Fri May  5 11:43:05 2000] [warn] Loaded DSO
/opt/IBMWebAS/plugins/linux/mod_app_server.so uses plain Apache 1.3
API, this module might crash under EAPI! (please recompile it with
-DEAPI)
                                                          [FAILED]
```

Figure 24. Apache failure loading WebSphere module

Because of this error, Apache will not start. Hence, we had to remove the Apache Web server and install IBM HTTP Server 1.3.6.

2.4.4.1 Removing Apache Web server

The Apache module should be removed, using the `rpm -e` command, in the following order:

1. Remove the mod_perl package with the command:

   ```
   rpm -e mod_perl
   ```

2. Remove the php package with the command:

   ```
   rpm -e php
   ```

   Removing php package in Caldera

   For Caldera, the command is:

   ```
   rpm -e mod_php3-3.0.12-5
   ```

3. Remove the Apache package with the command:

   ```
   rpm -e apache
   ```

   The following directories relating to the Apache Web server should be removed as follows:

   - For directory /var/log/httpd, use the command:

     ```
     rm -f /var/log/httpd
     ```

   - For directory /home/httpd, use the command:

     ```
     rm -f /home/httpd
     ```

   - For directory /etc/httpd, use the command:

     ```
     rm -f /etc/httpd
     ```

Removing /etc/httpd on Caldera

On Caldera installations, the /etc/httpd directory is not used. Hence there is no requirement to remove this directory (as it does not exist).

Figure 25 shows the results of issuing these commands on our front-end server with a RedHat installation.

```
[root@bdtest2 /opt]# rpm -e mod_perl
[root@bdtest2 /opt]# rpm -e php
[root@bdtest2 /opt]# rpm -e apache
cannot remove /var/log/httpd - directory not empty
cannot remove /home/httpd/html - directory not empty
cannot remove /home/httpd - directory not empty
cannot remove /etc/httpd/conf - directory not empty
[root@bdtest2 /opt]# rm -rf /var/log/httpd
[root@bdtest2 /opt]# rm -rf /home/httpd
[root@bdtest2 /opt]# rm -rf /etc/httpd
```

Figure 25. Uninstalling Apache Web server

We can now proceed with installing the IBM HTTP Server.

2.4.4.2 IBM HTTP Server installation

The installation steps are as follows:

1. Install the IBM HTTP Server rpm package. In our instance the file is called IBM_HTTP_SERVER-1.3.6.1.i386.rpm. See Figure 26 for the results of the `rpm -i` command.

```
[root@bdtest2 /opt]# rpm -i /mnt/cdrom/ibmhttp/2.1/IBM_HTTP_Server-1.3
.6-1.i386.rpm
httpd: cannot determine local host name.
Use the ServerName directive to set it manually.
/opt/IBMHTTPServer/bin/apachectl start: httpd could not be started
execution of script failed
```

Figure 26. rpm installation of IBM HTTP Server

2. Set ServerName directive

   In the /opt/IBMHTTPServer/conf directory, we had to edit the httpd.conf, adding the ServerName directive with our hostname. Figure 27 on page 36

shows a partial content of the httpd.conf file where we have added our ServerName entry.

```
# ServerName: allows you to set a host name which is sent back to
# your server if it's different than the one the program would get
# "www" instead of the host's real name).
# Note: You cannot just invent host names and hope they work. The name
# define here must be a valid DNS name for your host. If you don't know
# this, ask your network administrator.
# If your host doesn't have a registered DNS name, enter its IP address
# You will have to access it by its address (e.g., http://123.45.67.89/)
# anyway, and this will make redirections work in a sensible way.
#
#ServerName localhost
ServerName bdtest2.itsc.austin.ibm.com
```

Figure 27. Adding the ServerName directive to httpd.conf

3. Getting IBM HTTP Server to start at boot time

   To set the IBM HTTP Server to start at boot time using a X-window workstation, use the following steps:

   a. Run the `tksysv` command. The Runlevel manager window is shown in Figure 28 on page 37

   b. Verify that ibmhttpd is listed in the Available listbox and is listed in both the start and stop for runlevel 2.

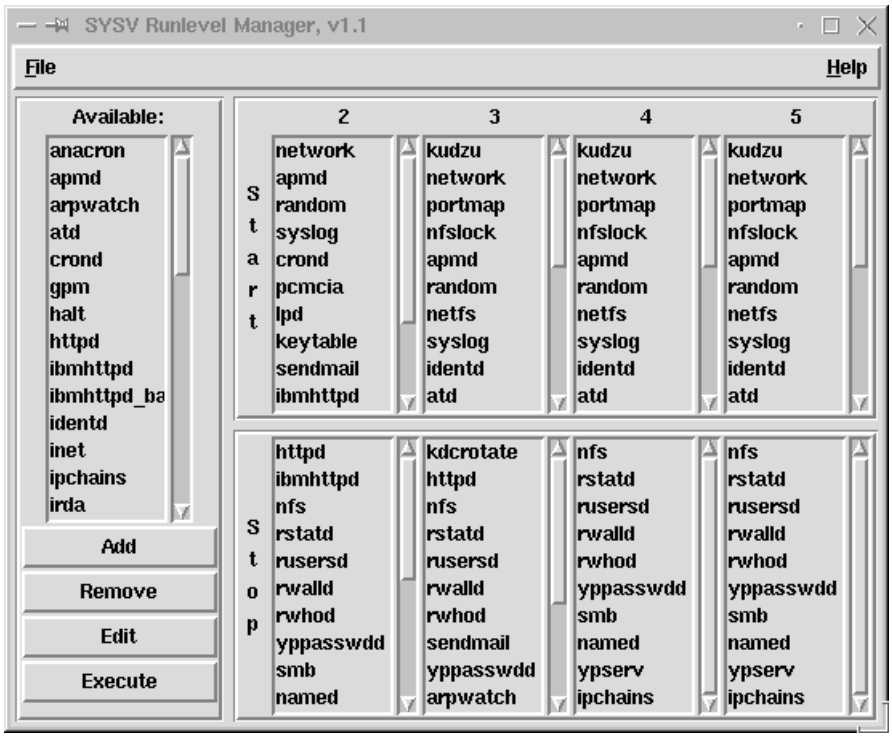

Figure 28. The output of the tksysv command

c. To add ibmhttpd to a required run level, highlight it in the Available: listbox and click on **Add**. A screen similar to Figure 29 on page 38 will pop up and prompt you to start or stop the service.

d. Select **run level 2** and click on **Done**. This procedure will need to be completed for each run level that you wish to have httpd configured for.

Figure 29. Adding the httpd to a run level

If there is no X-window capable workstation, you can add ibmhttpd in the startup by using the following steps:

a. There should be an executable script file called ibmhttpd located in /etc/rc.d/init.d. If not, copy this script file over from /opt/IBMHTTPServer/bin.

b. Issue the following commands to create the symbolic links in the various rc.d directories for the appropriate runlevels:

- `ln -s /etc/rc.d/init.d/ibmhttpd /etc/rc.d/rc0.d/K15ibmhttpd`
- `ln -s /etc/rc.d/init.d/ibmhttpd /etc/rc.d/rc1.d/K15ibmhttpd`
- `ln -s /etc/rc.d/init.d/ibmhttpd /etc/rc.d/rc2.d/K15ibmhttpd`
- `ln -s /etc/rc.d/init.d/ibmhttpd /etc/rc.d/rc3.d/S85ibmhttpd`
- `ln -s /etc/rc.d/init.d/ibmhttpd /etc/rc.d/rc4.d/S85ibmhttpd`
- `ln -s /etc/rc.d/init.d/ibmhttpd /etc/rc.d/rc5.d/S85ibmhttpd`

Now, you can start the IBM HTTP Server. The IBM HTTP Server should also start at boot time now. Check this by restarting the system and attempting to connect to port 80 with a Web browser. We verified that our installation worked when we could obtain the IBM HTTP Server default Web page in our Web browser as shown in Figure 30 on page 39.

Figure 30. IBM HTTP Server default Web page

2.4.5 WebSphere

We document here the steps to install WebSphere on the front-end server, configure various configuration files for WebSphere and IBM HTTP Server, and verify that IBM HTTP Server will load and run Java servlets.

2.4.5.1 Installing WebSphere

WebSphere must be installed as the root user. It requires a JDK be installed and a JAVA_HOME environment be set. It then requires 2 components to be installed. These are the installation steps:

1. Export JAVA_HOME=/opt/jdk118, since we had previously installed IBM JDK 118 (see Section 2.4.3, “IBM JDK 118” on page 32).
2. Install the WebSphere core, which has a file name of IBMWebAS-core-2.03-1.i386.rpm
3. We can now install the Web server specific component. Since we are using IBM HTTP Server, we will use the file IBMWebAS-ibm-http-2.03-1.i386.rpm.

WebSphere dependencies

The rpm package of WebSphere for IBM HTTP Server has a dependency on IBM HTTP Server 1.3.6; so, ensure that IBM HTTP Server 1.3.6 has already been installed.

Figure 31 shows our installation of the WebSphere application server following the installations steps documented before.

```
[root@bdtest2 /opt]# export JAVA_HOME=/opt/jdk118
[root@bdtest2 /opt]# rpm -i /mnt/cdrom/was/IBMWebAS-core-2.03-1.i386.rp
m
A copy of the configuration file was made and is located at /opt/IBMWebA
S/properties/bootstrap.properties.bak
[root@bdtest2 /opt]# rpm -i /mnt/cdrom/was/IBMWebAS-ibm-http-2.03-1.i38
6.rpm
A backup of your httpd.conf file has been made and is being saved at
/opt/IBMHTTPServer/conf/httpd.conf
```

Figure 31. Installing WebSphere application server

4. Finally, to ensure that JSP pages can be compiled, we change the ownership of the pagecompile directory, located in subdirectory /opt/IBMWebAS/servlets, to have an owner and group owner of nobody. We use the `chown` command in the following form:

   ```
   chown nobody.nobody /opt/IBMWebAS/servlets/pagecompile
   ```

2.4.5.2 Verifying WebSphere installation

We verify that WebSphere has been installed successfully by:

- Visually verifying the IBM HTTP Server configuration files.
- Visually verifying the WebSphere properties file.
- Using a Web browser to verify WebSphere is operational.

Verifying the IBM HTTP Server configuration files

The IBM HTTP Web server has a configuration called httpd.conf, located in /opt/IBMHTTPServer/conf. Visually inspect this file and check for the following:

- LoadModule_app_server_module

  There should be a LoadModule definition for the app_server_module to load WebSphere in the list of dynamic shared object support modules

after the comment `#Dynamic Shared Object (DSO) Support` under the extra modules section after the comment `#Extra Modules`. The load module statement is shown in Figure 32.

```
...
#
# Dynamic Shared Object (DSO) Support
#
...
# LoadModule usertrack_module   libexec/mod_usertrack.so
LoadModule unique_id_module   libexec/mod_unique_id.so
LoadModule setenvif_module    libexec/mod_setenvif.so
LoadModule ibm_app_server_module /opt/IBMWebAS/plugins/linux/mod_ibm
_app_server.so
...
```

Figure 32. LoadModule entry for app_server_module in httpd.conf

- AddModule mod_app_server.c

  There should be an AddModule definition for mod_app_server.c in the modules load list. This list is prefixed by an appropriate comment and the ClearModuleList directive. Under the Extra Modules section, we will find this AddModule definition for mod_app_server.c as shown in Figure 33.

```
...
#  Reconstruction of the complete module list from all available
#  (static and shared ones) to achieve correct module execution
#  [WHENEVER YOU CHANGE THE LOADMODULE SECTION ABOVE UPDATE THIS,
ClearModuleList
...
# AddModule mod_usertrack.c
AddModule mod_unique_id.c
AddModule mod_so.c
AddModule mod_setenvif.c
AddModule mod_app_server.c
...
```

Figure 33. AddModule entry for mod_app_server.c in httpd.conf

- At the end of the httpd.conf file, we verify that it contains the IBM WebSphere virtual directory and configuration entries as shown in Figure 34 on page 42.

```
Alias /IBMWebAS/samples/  /opt/IBMWebAS/samples/
Alias /IBMWebAS/  /opt/IBMWebAS/web/
NcfAppServerConfig BootFile  /opt/IBMWebAS/properties/bootstrap.propert
ies
NcfAppServerConfig LogFile  /opt/IBMWebAS/logs/apache.log
NcfAppServerConfig LogLevel  TRACE|INFORM|ERROR
```

Figure 34. WebSphere entries in httpd.conf

Verifying the WebSphere configuration file

WebSphere uses the bootstrap.properties file, located in /opt/IBMWebAS/properties, for configuration. Visually verify that it has sent up the Java configuration correctly.

In our installation, we noticed that there is an invalid entry for the Java.libpath. The Java.libpath is referring to /opt/jdk118/lib/i586/green_threads, while it should be /opt/jdk118/lib/linux/native_thread. The correct statement is shown in bold in Figure 35.

```
##
# The classpath for the JVM. This classpath is appended to the front
# of the WebSphere Runtime classpath.  This classpath should contain
# the JVM Classes and any user classes. The WebSphere classpath is
# built dynamically from the contents of the <Install Root>\lib
# directory
Java.classpath=/opt/jdk118/lib/classes.zip:/opt/IBMWebAS/classes:/opt/I
BMWebAS/web/classes

##
# The location of the JVM libraries.
Java.libpath=/opt/jdk118/lib/linux/native_threads/
```

Figure 35. Corrected Java configuration in WebSphere bootstrap.properties

Using a Web browser to verify WebSphere is operational

To start WebSphere application server for the first time, we need to restart the IBM HTTP Server. Just in case, stop all Java processes with the `killall Java` command, and then restart the IBM HTTP server with the commands shown in Figure 36 on page 43.

```
[root@bdtest2 /opt]# killall Java
Java: no process killed
[root@bdtest2 /opt]# /etc/rc.d/init.d/ibmhttpd stop
/etc/rc.d/init.d/ibmhttpd stop: httpd stopped
[root@bdtest2 /opt]# /etc/rc.d/init.d/ibmhttpd start
/etc/rc.d/init.d/ibmhttpd start: httpd started
```

Figure 36. Starting up IBM WebSphere Application Server for the first time

Now we can access our server to run the snoop servlet that comes with WebSphere. Figure 37 shows the results of running the snoop servlet.

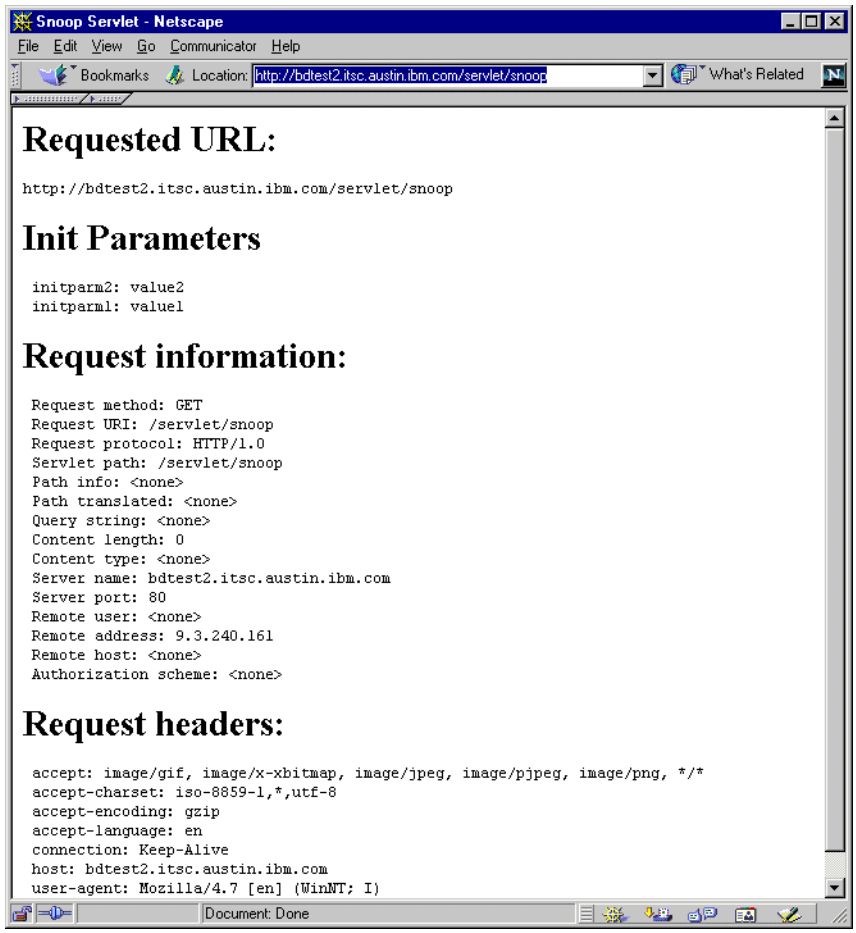

Figure 37. Testing WebSphere installation with the snoop servlet

We also verify that the WebSphere Administration server is running by connecting to our server on port 9527. Figure 38 shows the results of this.

Figure 38. Verifying the WebSphere admin server is running on port 9527

2.4.6 Configuring front-end access to backend databases

At this point, our front-end server is not yet ready to access the backend DB2 and Domino databases. We must perform further configuration on the front-end server. See the following sections:

- Section 2.4.6.1, “Configuring front-end access to DB2” on page 44
- Section 2.4.6.2, “Configuring front-end access to Domino” on page 49

2.4.6.1 Configuring front-end access to DB2

In order for IBM WebSphere and IBM HTTP Server, installed on our front-end server bdtest2, to have access to the DB2 Java class libraries and the DB2 server on our backend server, we must perform the following steps on the front-end server:

1. Install the DB2 Client Application Enabler (CAE).

2. Verify the DB2 CAE installation.
3. Modify the IBM WebSphere configuration file to include the DB2 Java libraries.
4. Modify the IBM HTTP Server to identify the DB2 instance and the relevant DB2 libraries.
5. Verify the access to the backend DB2 database.

We describe these steps in the following sections.

Install IBM DB2 Client Application Enabler

To perform this task:

1. Log in as root.
2. Change the working directory to the directory where the DB2 6.1 at fixpack 3 code for the DB2 CAE is stored.
3. Execute the `db2setup` command. The Install DB2 V6 screen opens.
4. After running the `db2setup` command, depending on the distribution of DB2, either the DB2 Administration Client or the DB2 Runtime Client are available for installation. Choose whichever is available to install the CAE. In our scenario, this is available as the DB2 Administration Client.
5. Press the **Enter** key to select the **DB2 Administration Client**. The screen should be as presented in Figure 39.

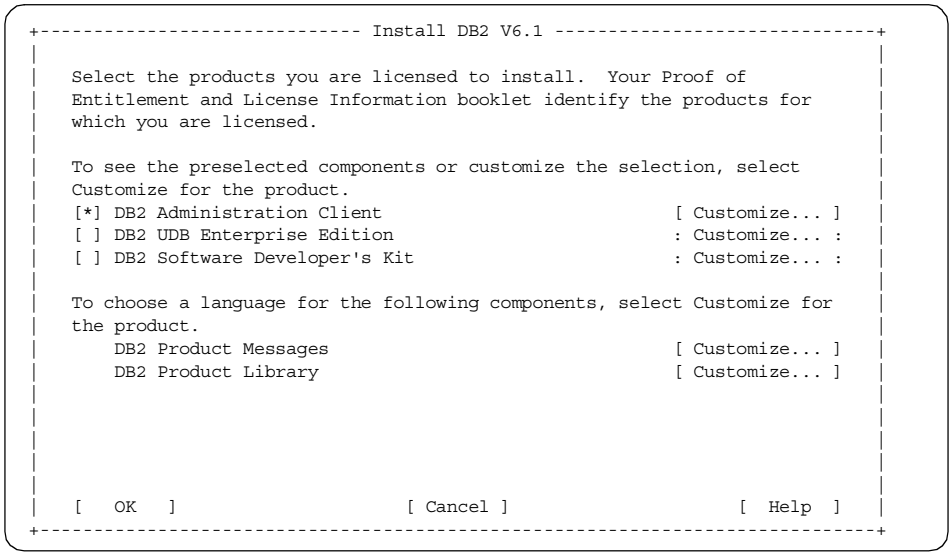

Figure 39. Installing the DB2 CAE with the db2setup command

6. To ensure the Java components are installed, we move the cursor over to the **Customize** option with Tab, then press **Enter**. We verify the Java support component will be installed as shown in Figure 40.

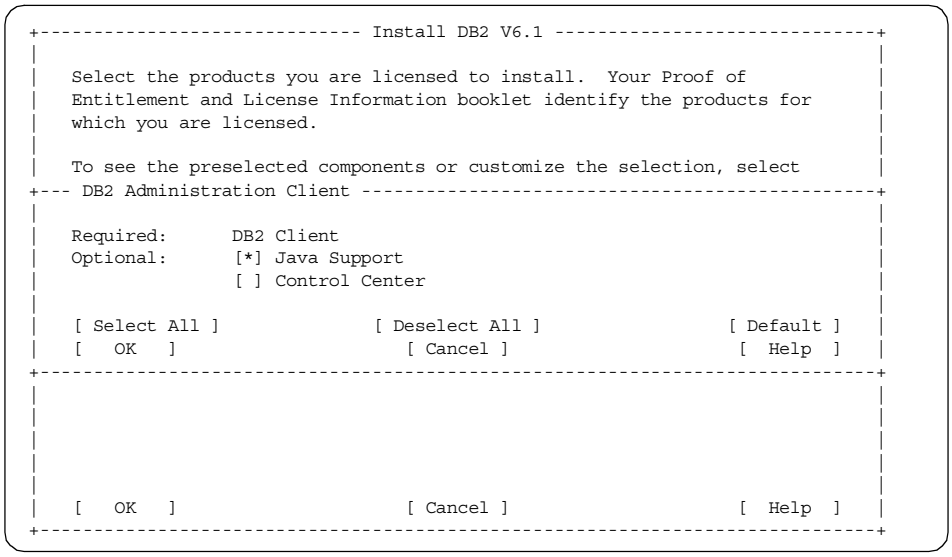

Figure 40. Customizing the DB2 CAE installation for Java support

7. Select **OK** to continue.
8. Back at the Install DB2 screen, choose **OK** to continue on to instance creation.
9. There are two methods of accessing DB2 with JDBC, the app driver and the net driver. Refer to Section 3.5.3, “Accessing DB2 via JDBC” on page 83 for more information. We use the Java app driver; therefore, we choose to create a DB2 instance.
10. Accept all defaults in the DB2 instance screen. Choose **OK** and note the system generated password, ibmdb2. Press **Enter** to acknowledge this, and return to the Create DB2 Services screen.
11. Select **OK** to lead to the Summary Report screen. Choose **Continue**.
12. To start the installation, choose **OK**. The installation will proceed.
13. When the installation is complete, choose **OK** to proceed to the Status Report screen. Verify that all components have installed successfully and choose **OK** to return to the initial DB2 Installer screen.
14. Choose **Close** to exit the DB2 installer and **OK** to dismiss the final warning screen about exiting the DB2 installer.

Verifying the DB2 CAE installation

Once the DB2 CAE has been installed, verify it is working as follows:

1. Log in as the DB2 instance owner, db2inst1.
2. Access the SAMPLE database in the backend server with the following commands in the order presented:

   ```
   db2 catalog tcpip node bdtest3 remote 9.3.240.222 server 50000
   db2 catalog database sample at node bdtest3
   ```

   Figure 41 shows the result of running these commands.

```
[db2inst1@bdtest2 db2inst1]$ db2 catalog tcpip node bdtest3 remote
9.3.240.222 server 50000
DB20000I  The CATALOG TCPIP NODE command completed successfully.
DB21056W  Directory changes may not be effective until the directory
cache is refreshed.
[db2inst1@bdtest2 db2inst1]$ db2 catalog database sample at node
bdtest3
DB20000I  The CATALOG DATABASE command completed successfully.
DB21056W  Directory changes may not be effective until the directory
cache is refreshed.
```

Figure 41. Cataloging the SAMPLE database on the front-end server

3. Verify the connection with the `db2 connect` command as follows:

   ```
   db2 connect to sample user db2inst1 using ibmdb2
   ```

4. Finally, verify that an SQL select statement works by issuing the following command:

   ```
   db2 "select * from department"
   ```

   See Figure 10 on page 18 for a example of the result set for this query.

Modifying IBM WebSphere configuration files

We must add the DB2 classpath for db2java.zip to the WebSphere Java.classpath parameter. Do this as follows:

1. The boostrap.properties file is located on the front-end server in directory /opt/IBMWebAS/properties.
2. Log on as root, edit this file, and find the following entry:

   ```
   Java.classpath
   ```

3. Add the following path to this entry:

   ```
   /usr/IBMdb2/V6.1/Java/db2Java.zip
   ```

4. The entry in the bootstrap.properties file should now look like Figure 42.

```
############################################################################
# Java Virtual Machine configuration                                       #
############################################################################

##
# The classpath for the JVM. This classpath is appended to the front
# of the WebSphere Runtime classpath.  This classpath should contain
# the JVM Classes and any user classes. The WebSphere classpath is
# built dynamically from the contents of the <Install Root>\lib
# directory
Java.classpath=/opt/jdk118/lib/classes.zip:/opt/IBMWebAS/classes:/opt/I
BMWebAS/web/classes:/usr/IBMdb2/V6.1/Java/db2Java.zip
```

Figure 42. Adding db2Java.zip to the WebSphere Java.classpath parameter

Modifying IBM HTTP server configuration files

In order for the JDBC connection to work, the DB2INSTANCE environment parameter must be set, and the library paths must be modified to include the DB2 library. We modify the IBM HTTP Server startup file, /etc/rd.d/initd/ibmhttpd, and added the environment variables shown in Figure 43.

```
...
# |||||||||||||||||| START CONFIGURATION SECTION  ||||||||||||||||||
# ------------------                               -------------------
#
# The path to DB2 stuff
export DB2INSTANCE=db2inst1
export LIBPATH=/usr/IBMdb2/V6.1/lib:$LIBPATH
export LD_LIBRARY_PATH=$LIBPATH:$LD_LIBRARY_PATH
...
```

Figure 43. DB2INSTANCE and LIBPATH in the ibmhttpd file

Restart IBM HTTP Server to activate these settings.

Verifying access to the backend DB2 database

We can now verify the access to our DB2 SAMPLE database with the TestDB2 servlet code found in Appendix B.1, “Java servlet TestDB2” on page 147. Register the servlet in WebSphere, then access the servlet. The output from the servlet is shown in Figure 44 on page 49.

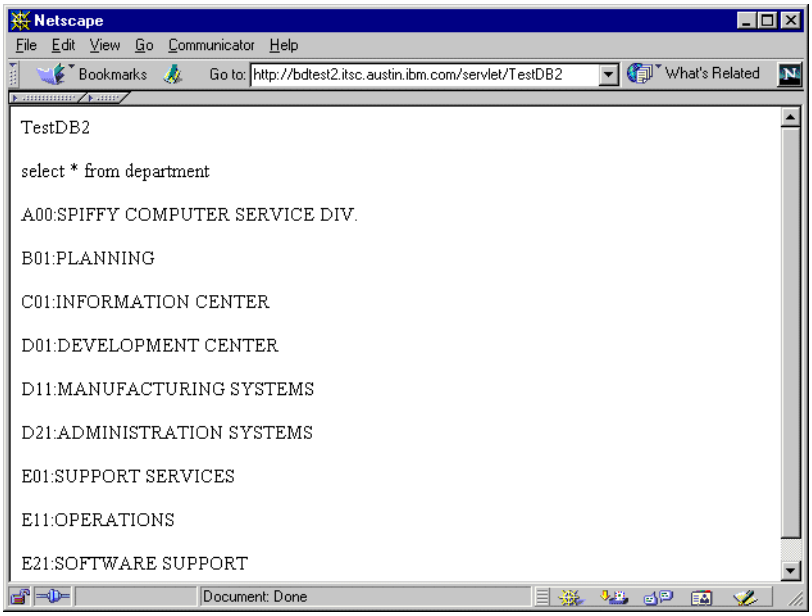

Figure 44. Testing the DB2 connection with servlet TestDb2

2.4.6.2 Configuring front-end access to Domino

In order to access the Domino database from our front-end server, bdtest2, we must to configure WebSphere Application server with the following tasks:

1. Copy the NCSO.jar file from /local/notesdata/domino/Java/ directory on Domino server to the front-end server. We put the file on the /opt/domino directory.
2. Edit the bootstrap.properties file. We have a bootstrap.properties located on /opt/IBMWebAS/properties.
3. Put in the full path of NCSO.jar in the classpath:

   ```
   Java.classpath=...:/opt/domino/NCSO.jar
   ```

   Figure 45 on page 50 will show our modified bootstrap.properties.

```
####################################################################
# Java Virtual Machine configuration                               #
####################################################################

##
# The classpath for the JVM. This classpath is appended to the front
# of the WebSphere Runtime classpath.  This classpath should contain
# the JVM Classes and any user classes. The WebSphere classpath is
# built dynamically from the contents of the <Install Root>\lib
# directory
Java.classpath=/opt/jdk118/lib/classes.zip:/opt/IBMWebAS/classes:/opt/IBMWebAS/w
eb/classes:/usr/IBMdb2/V6.1/Java/db2Java.zip:/opt/domino/NCSO.jar
```

Figure 45. Adding the NCSO.jar classpath to bootstrap.properties

4. Restart the IBM HTTP server with the following commands:

   ```
   killall Java
   /etc/rc.d/init.d/ibmhttpd stop
   /etc/rc.d/init.d/ibmhttpd start
   ```

2.4.6.3 Verifying WebSphere to access Domino server.

To verifying the WebSphere to access the Domino database, we need a simple Java servlet program. We create a simple servlet on WebSphere that trying to open session to the Domino server and shows the title of sample database.

1. We write a servlet TestDomino that is provided in Appendix C.1, “Java servlet TestDomino” on page 159.
2. Compile the servlet with Java compiler.
3. Put the servlet class file to WebSphere servlets directory.
4. Configure the WebSphere to load the servlet.
5. To verify the connection between WebSphere and Domino, run the servlet from the Web browser. We use the URL:

   ```
   http://bdtest2/servlet/TestDomino?db=homepage&host=bdtest3&port=8000
   ```

   where homepage is the homepage.nsf database file that is available by default in a Domino server.

   The result is similar to Figure 46 on page 51.

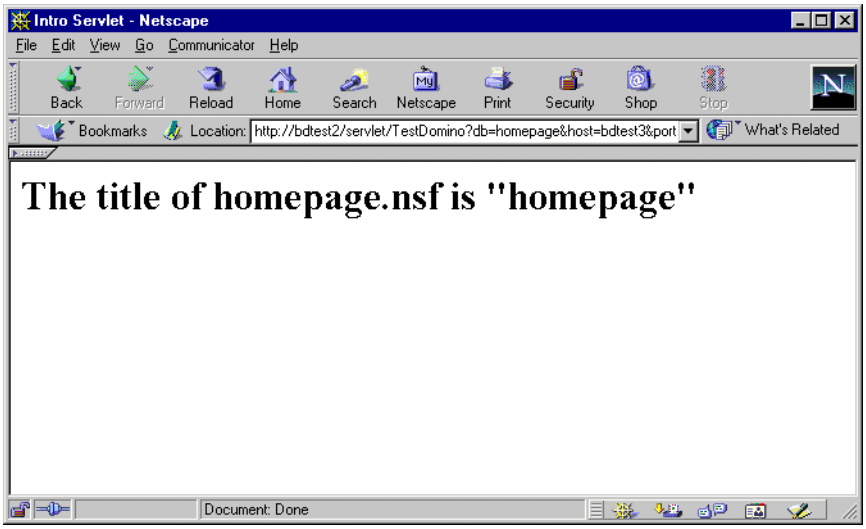

Figure 46. The TestDomino servlet result

Chapter 3. Building the Web server

In this chapter, we explore, ,in detail, the steps that we took in building our Web server. The discussion is divided into:

- Section 3.1, “External functionality” on page 53 describes the external structure of the created Web site.
- Section 3.2, “Internal design” on page 60 shows the components and building blocks that we used to build the Web site.
- Section 3.3, “DB2 database” on page 61 discusses the design and creation of our DB2 database structure.
- Section 3.4, “Domino database” on page 69 provides an overview of Domino database and our Redbooks database setup.
- Section 3.5, “Servlet programs” on page 78 gives a detailed discussion on the making of the servlets that we use.
- Section 3.6, “JSP files” on page 98 discusses the implementation of dynamic Web pages using JSP.
- Section 3.7, “Accessing data through a Java bean” on page 103 provides an overview of Web programming using a simple Java Bean.

3.1 External functionality

In this section we discuss the external functionality of our Web server. We walk you through our Web server as a typical user. There are two types of users for the Web site, the non-registered user and the registered user.

The Web site has the facility to:

- Browse and search the Redbooks database
- Profile management, register, and update the user
- Book ordering function

We provide a series of ScreenCam videos in Windows media format that illustrate the Web site’s flow. These ScreenCam videos are available in the accompanied CD-ROM. Refer to Appendix F, “Using the additional material” on page 211 for more information on the CD-ROM.

Figure 47 on page 54 illustrates the flow for a non-registered user.

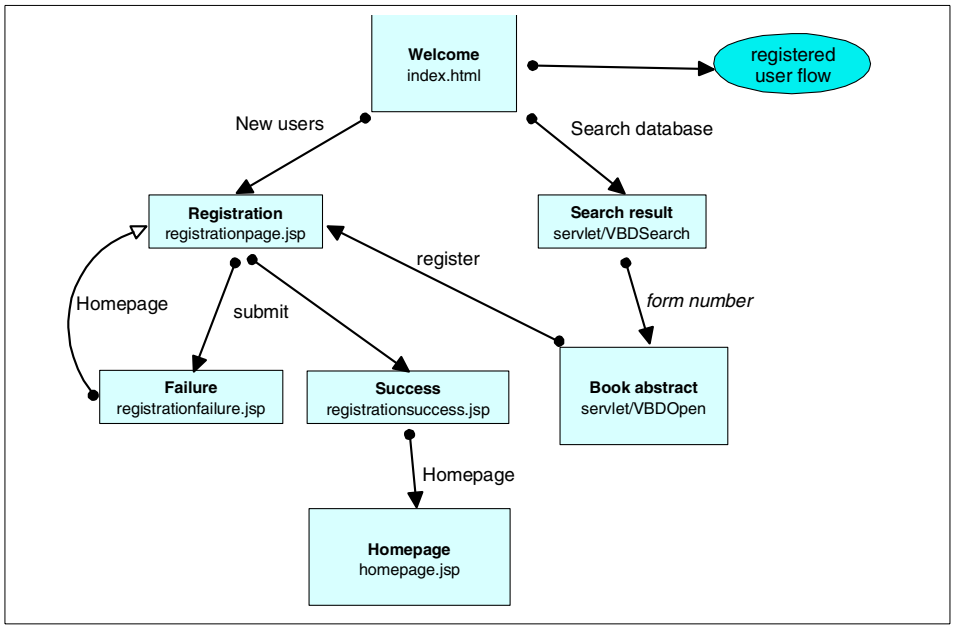

Figure 47. HTML flow for a non registered visitor

As indicated by Figure 47, the initial page that is shown in this Web site is the welcome page, which is shown in Figure 48 on page 55.

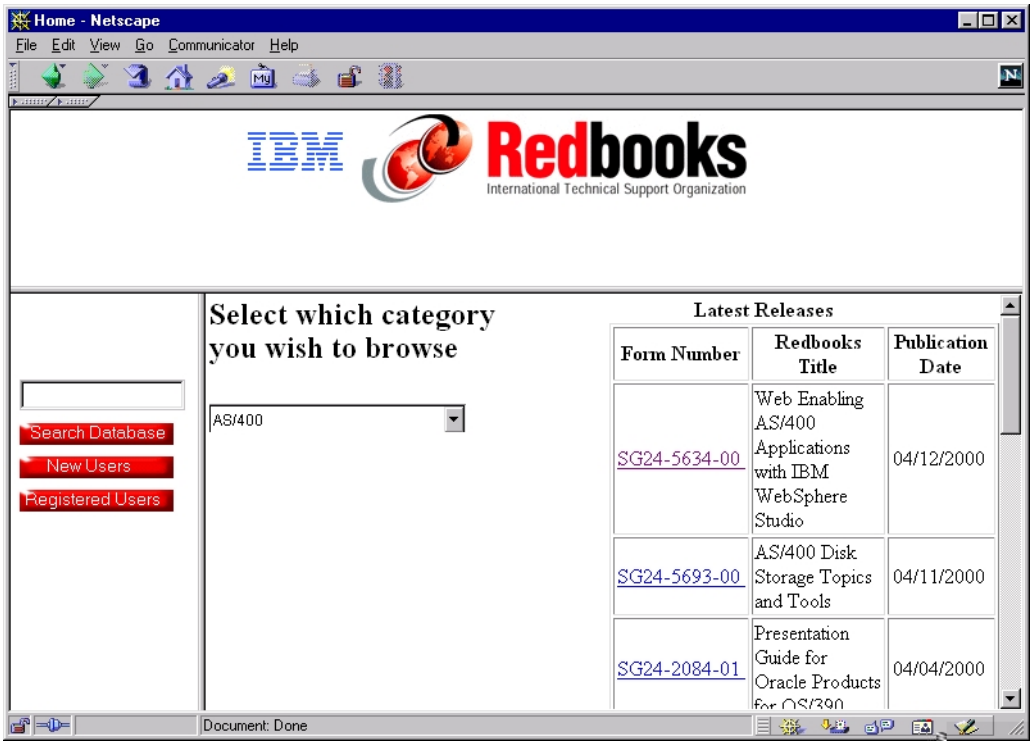

Figure 48. Welcome page

From the welcome page, shown in Figure 48, a non-registered user only has limited menus and functions that can be accessed. These functions are:

- Click on **Search Database**, which would allow searching the database for a specific topic in the redbooks database. This does not require a username or password.
- Click on **New Users**, which would present the registration page to start the registration process.

  The registration window is shown in Figure 49 on page 56. The registration page requires you to fill in your personal information and interests on what category of books you are interested in. These book categories are displayed as menu options on your home page. You also need to choose a username and a password. If there is a failure, an error and instructions will be displayed; otherwise, you are presented with a page showing your username and password. You will then have the option to continue or log out. If you choose to continue, you will be presented with your customized home page.

Figure 49. Registration page

- Click on **Registered Users**, which presents the login page and requires a valid user name and password. This selection is used by the registered users to log in.
- Click on the Form Number (SGxx-xxxx) of any of the Latest Releases table entries in the body of the Welcome page to go to the Abstract page of that particular book.

  The abstract of the book is shown in Figure 50 on page 57. The non-registered user will not be able to order a book without first registering and logging on. A link is provided from the abstract page of the book to the registration page, from where you can register, log in, and order.

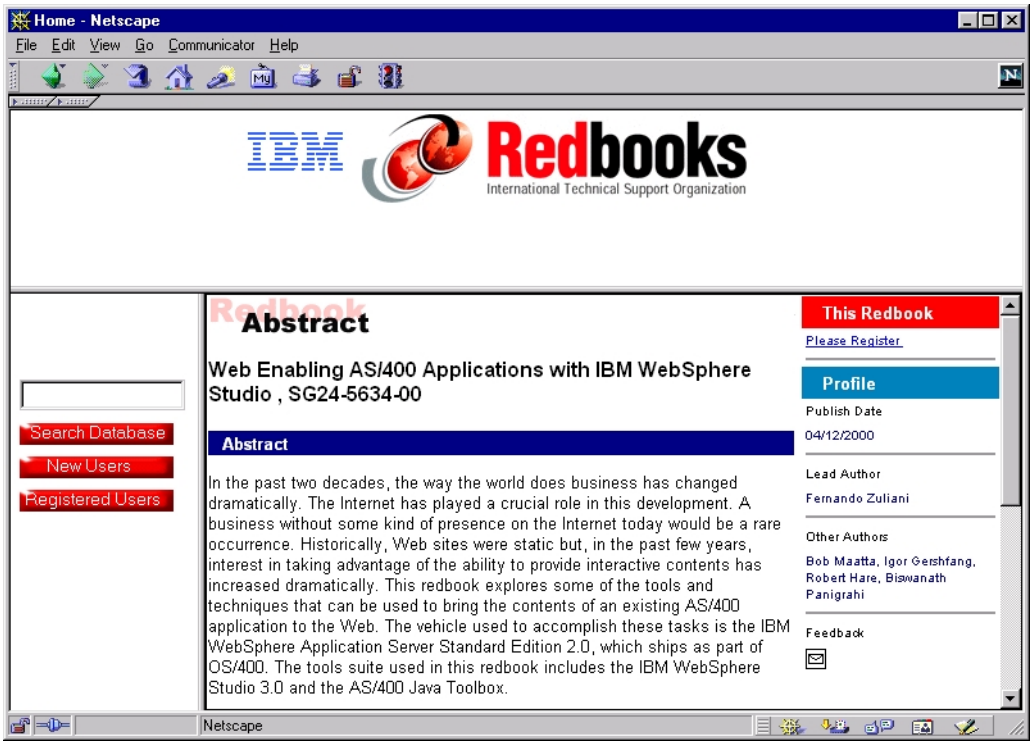

Figure 50. Abstract displayed after search for a book

- Select from a drop down list to browse a specific category of redbook.

Figure 51 on page 58 describe the flow of the Web site for a registered user.

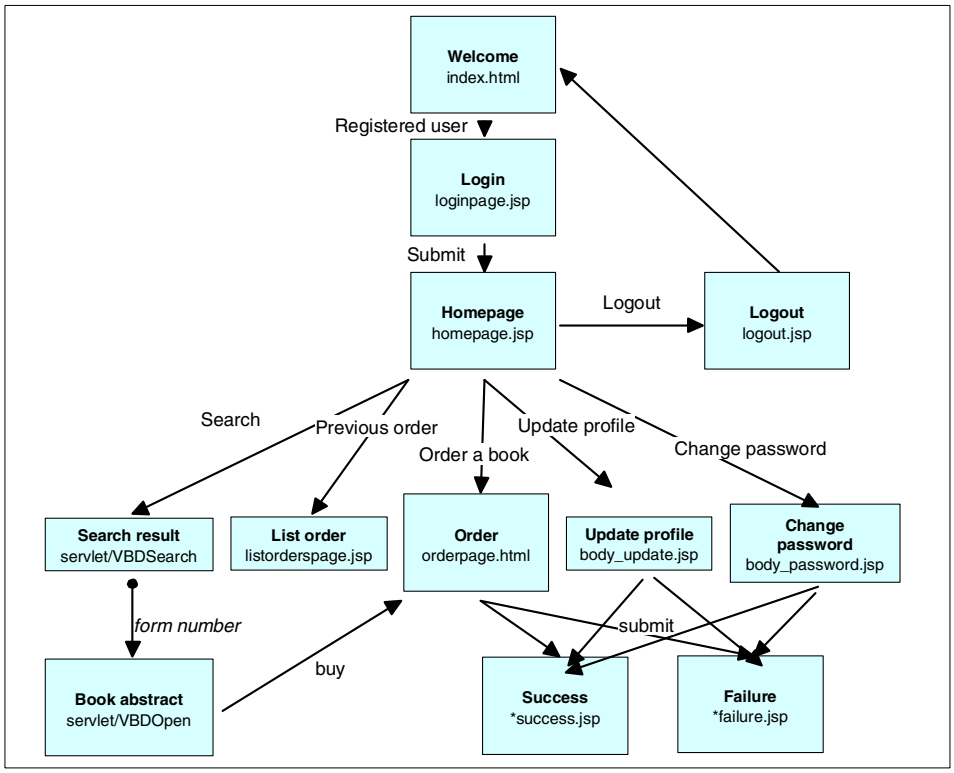

Figure 51. HTML flow for a registered user

A registered user would choose the **Registered users** button from the Welcome page shown in Figure 48 on page 55 to go to the login page. You need to present a valid user name and password to be entered. If the login is unsuccessful, you are returned to the login page and allowed to try again. If the login is successful, you will be presented with your customized home page, similar to the window shown in Figure 52 on page 59, which contains your interest profile.

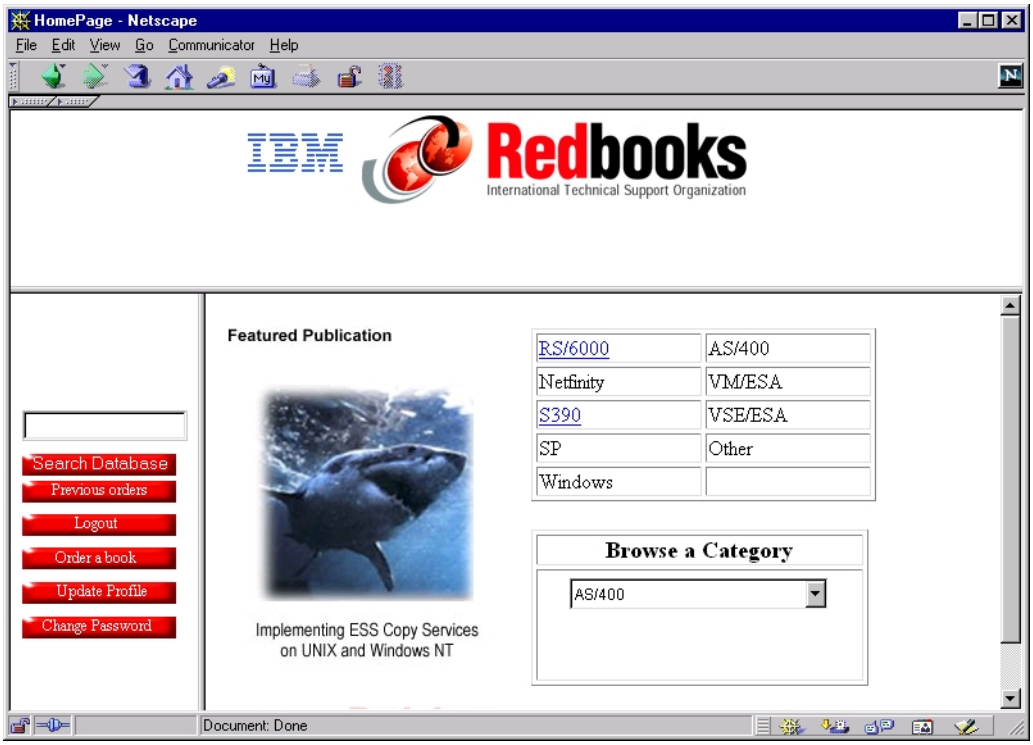

Figure 52. Home Page for a registered user

As a registered user, you have several options to choose from your home page as shown in Figure 52.

- The Search Database button is similar to the search for the non-registered user.
- Click on **Previous Orders**, which brings you to the List order page where the orders are that you have submitted using a Java Bean.
- Click on **Order a book**, which brings you to the Order page to initiate the ordering process.
- Click on **Update Profile**. This enables the update of personal information and interest categories.
- Click on **Change Password**. This provides the password change screen.

From the order page, update profile page, and change password page, you may get either a success page or failure page. On failure, clicking **continue** brings you back to the previous screen. On success, clicking **continue** brings you back to the homepage.

As shown in Figure 51 on page 58, the order page can be reached from the book abstract page or directly to the order page if you know the form-number. A successful order will display an order number and the shipping details. Failures will again result in an error message and further instructions being displayed.

3.2 Internal design

Our Web application structure is then developed to be able to provide the presentation flow discussed in Section 3.1, “External functionality” on page 53. As we have also decided to use the pattern for e-business user-to-online buying topology, the application is then developed to include a Commerce server node in the front-end and a Database server node in the backend. The front-end runs the Web server and the business logic using WebSphere, while the backend contains the data. Figure 53 shows our internal configuration.

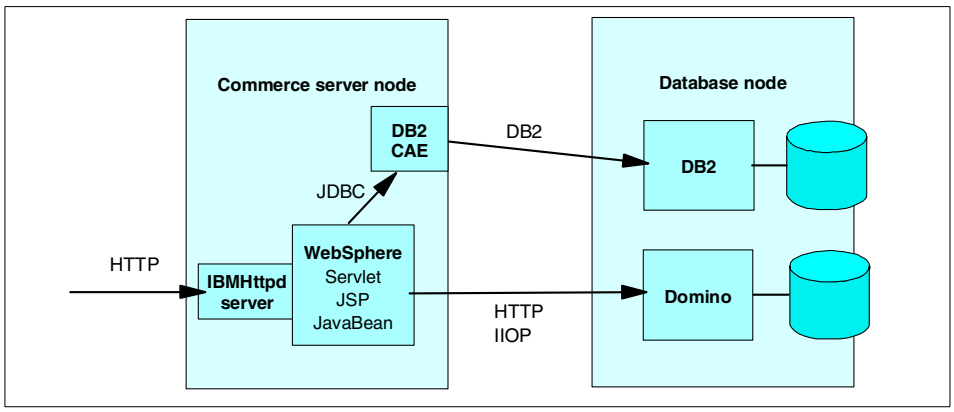

Figure 53. Internal configuration

The design is started from the Database server structure. We use two data sources, the DB2 database to store the order and user profiles and the Lotus Domino database to store the book information. The structure of the DB2 database is given in Section 3.3, “DB2 database” on page 61. The discussion on the Lotus Domino’s redbook database is given in Section 3.4, “Domino database” on page 69.

The Commerce server node is implemented using RedHat Version 6.2 and Caldera eDesktop Version 2.4. The HTML codes that we used are available in Appendix D, “HTML and JSP files listing” on page 173. We describe, in this

redbook, the development of some Java related ingredients of the Web server. These discussions are put in:

- Section 3.5, “Servlet programs” on page 78, which discusses the Servlet programs that we develop. The servlets are used primarily to maintain a session, to access the DB2 database, and to access the Domino database.
- Section 3.6, “JSP files” on page 98, which discusses the Java Server Pages (JSPs) that we use. The JSP provides the ability to create a dynamic Web page without having to code everything in a servlet.
- Section 3.7, “Accessing data through a Java bean” on page 103, which provides an overview of the Java Beans that we use in representing an arbitrary amount of dynamic data in a Web page for listing the orders.

3.3 DB2 database

This section discusses, in detail, the steps required to build the DB2 database that we use. The steps that we perform are discussed in the following:

- Section 3.3.1, “Modelling the database” on page 61 where we provide the design considerations that we face in developing the database.
- Section 3.3.2, “Creating the database” on page 66 shows the actual database creation.

3.3.1 Modelling the database

To achieve the objective of the Web site, our database must, at the very least, model the following entities:

- Users

  Users are the customers who come to the Web site to browse, log in with a username and password, register a profile of redbook category interests, and order redbooks.

- Orders

  These entities reflect the order of a product by a user. For simplicity’s sake, a user may currently only order one product at a time (that is, there is no shopping basket concept where one or more products of differing quantities may be placed before an order is finalized).

We examine each of these entities in more specific detail.

3.3.1.1 Users

The entity users must have the following attributes:

- Username and password

  Each username is considered to be unique. Hence, we validate that a user is registered by verifying the combination of the username and password. Each and every time an action is to be performed on the database, whether in changing or accessing user passwords, information, and profiles, the username and password must be verified.

  The username and password information needs to be built separately for performance and implementation reasons. For performance reasons, we need to separate the access to this information from all others to be able to provide fast and efficient access to the data. From an implementation point of view, we would also find it more convenient to later change password encryption algorithms and access methods.

- User information

  The user information contains information of the user's first and last names, address, email, and so on. Once a user is registered, this information is usually static. These data are not usually changed.

  We added the date joined attribure to track when the user is registered.

- Profile information

  The profile information reflects the user's interests in terms of categories of redbooks. This list of categories of interest to the user may change often, depending on the user. The list of categories may also change in time.

Based on the differences of the attributes, we decide to implement these attributes in different tables.

3.3.1.2 Orders

Orders have the following attributes:

- A username associated with the order
- The timestamp of the order
- The product that has been ordered
- A unique order identification number

  This unique order ID is required, as we can no longer use username as a unique identifier (as a user can, or should be able to, order more than one product).

3.3.1.3 Complete design

Based on the discussions for the user and order items, we will create four database tables. Each table has a set of fields, or column names, where each field has a data type, a length, and a nullable condition.

- The data type

  There are various data types that are available in DB2. We use the following data types in our database tables:

  - varchar: Provides variable length string type
  - char: Provides fixed length string type
  - date: Provides a date-only data type
  - timestamp: Provides a date and time up to a microsecond
  - int: Provides a whole number type

- The field length

  This represents the length of the field, if appropriate. For example, a data type of char or varchar requires a length, but a data type of date, timestamp, or int does not.

- The nullable condition

  This last attribute, the nullable condition, exists for us to track whether the field can be left empty or undefined or must be filled in. For example, a “” is a 0 length string; it is not a null string.

The structure of the database that we built is represented by the Entity Relationship diagram shown in Figure 54.

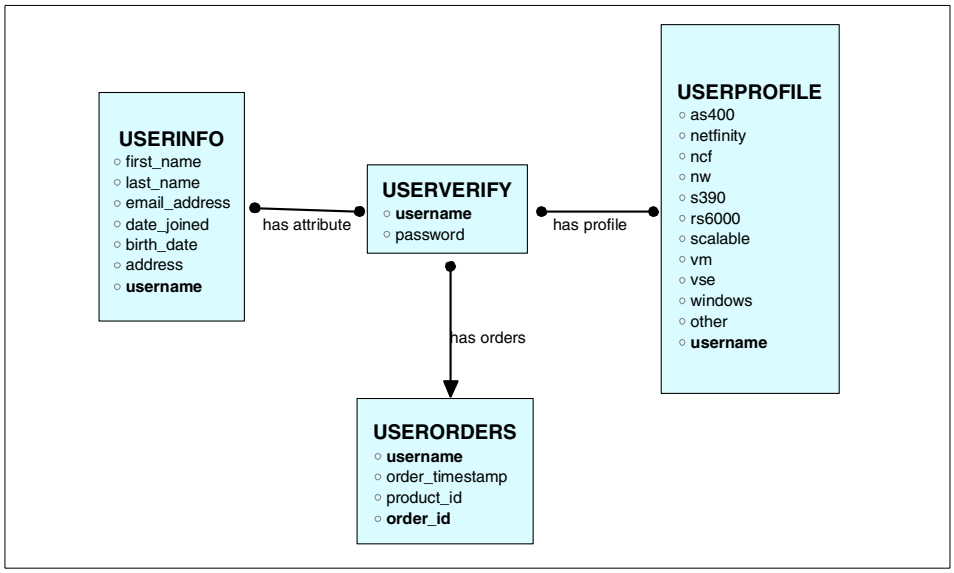

Figure 54. Entity Relationship diagram of our DB2 database structure

We can now list the field names and their attributes for each of the four database tables in the following sub-sections.

USERVERIFY

The USERVERIFY table contains the username and password. The fields of this table are shown in Table 1.

Table 1. The USERVERIFY database table

Field name	Data type	Length	Nullable
username	varchar	20	no
password	varchar	20	no

The username field will be the search key in this table; hence, we will also allocate it to be the primary key in the table.

Neither username nor password are allowed to be empty.

USERINFO

The USERINFO table contains the user information. The fields of this table are shown in Table 2.

Table 2. The USERINFO database table

Field name	Data type	Length	Nullable
first_name	varchar	50	no
last_name	varchar	50	no
email_address	varchar	50	no
date_joined	date	-	no
birth_date	date	-	no
address	varchar	200	no
username	varchar	20	no

The username field will be the search key in this table; hence, we will also allocate it to be the primary key in the table.

None of these fields are allowed to be empty.

USERPROFILE

This contains the list of platform categories that a user would be interested in. Each registered user will have an entry in this table. All fields are set to be nullable except for the username. If a registered user has indicated an interest in a platform, the field is set to a single character, typically a letter y. The fields for the USERPROFILE table are shown in Table 3.

Table 3. The USERPROFILE database table

Field name	Data type	Length	Nullable
as400	char	1	yes
netfinity	char	1	yes
ncf	char	1	yes
nw	char	1	yes
s390	char	1	yes
rs6000	char	1	yes
scalable	char	1	yes
vm	char	1	yes

Field name	Data type	Length	Nullable
vse	char	1	yes
windows	char	1	yes
other	char	1	yes
username	varchar	20	no

The username field will be the search key in this table; hence, we will also allocate it to be the primary key in the table.

USERORDERS

This contains the user orders. Each order, apart from having the username, product_id ordered, and a timestamp for the order, will also have a unique order_id. The fields for the USERORDERS table are shown in Table 4.

Table 4. The USERORDERS database table

Field name	Data type	Length	Nullable
username	varchar	20	no
order_timestamp	timestamp	-	no
product_id	varchar	20	no
order_id	int	-	no

The order_id field will be the search key in this table; hence, we will also allocate it to be the primary key in the table.

We require all fields in this table be filled.

3.3.2 Creating the database

Creating the database consists of the following steps:

- Creating the DB2 database
- Creating the DB2 tables in the database

We assume DB2 has been installed, and the installation has been verified successfully as discussed in Section 2.4.1, “DB2” on page 15.

3.3.2.1 Creating the DB2 database

We can create the DB2 database using either the DB2 Control Center GUI interface or via the command line.

We demonstrate how to create the database via the command line, as follows:

1. On the server, log in as user `db2inst1` (the default DB2 instance owner).
2. Execute the `db2 create database` command to create the USERSTUF database:

   ```
   db2 create database USERSTUF
   ```

DB2 database names

DB2 database names in DB2 v6.1 must be eight characters or less in length. Hence, we have named our database USERSTUF (instead of USERSTUFF)

Figure 55 on page 67 shows the results of the `create database` command, creating the USERSTUF database with default settings. We verify the database has been created by connecting to it with the DB2 `connect to database` command.

```
[db2inst1@bdtest1 db2inst1]$ db2 create database USERSTUF
DB20000I  The CREATE DATABASE command completed successfully.
[db2inst1@bdtest1 db2inst1]$ db2 connect to userstuf

   Database Connection Information

 Database server        = DB2/LINUX 6.1.0
 SQL authorization ID   = DB2INST1
 Local database alias   = USERSTUF
```

Figure 55. Creating the USERSTUF database on server bdtest1

This command creates a database called USERSTUF in the default location, /home/db2inst1.

DB2 database backups

Always back up a DB2 database using the `db2 backup database` command. Backing up the file system, or database files at a file system level, will NOT accomplish the same result. See Chapter 4, “Operational considerations” on page 107 for more detail on the DB2 operation.

3.3.2.2 Cataloging database USERSTUF on the front-end server

As will be discussed in 3.5.3, “Accessing DB2 via JDBC” on page 83, the front-end server will use the DB2 native API type 2 JDBC driver to access the DB2 USERSTUF database. For this to work, we must catalog the USERSTUF database on our front-end server.

We logged in as the DB2 instance owner db2inst1 on the front-end server by issuing the DB2 commands in the following order:

```
db2 catalog tcpip node bdtest3 remote 9.3.240.221 server 50000
db2 catalog database userstuf at node bdtest3
```

Verify the connection by connecting to the DB2 database with the command:

```
db2 connect to userstuf
```

3.3.2.3 Creating the DB2 tables in the database

We can create the database tables for the USERSTUF table either using the DB2 Control Center GUI or via the DB2 command line.

We demonstrate how to create the tables using the command line interface. We have already created four SQL script files to be used for this process (see Appendix A, “SQL scripts for creating the USERSTUF tables” on page 145). We demonstrate creating the USERINFO table here. The script file is named userinfo.ddl, and the contents are shown in Figure 56.

```
create table userinfo (
   first_name varchar(20) not null,
   last_name varchar(20) not null,
   email_address varchar(50) not null,
   date_joined date not null,
   birth_date date not null,
   address varchar(200) not null,
   username varchar(20) not null,
   primary key (username)
);
```

Figure 56. The userinfo.ddl file

The steps to create the USERINFO table are as follows:

1. Log in as the default DB2 instance owner, db2inst1.
2. Connect to the USERSTUF database with the command:

   ```
   db2 connect to userstuf
   ```

3. Run the `db2` command with the `-tvf` options, specifying the USERINFO script file called `userinfo.ddl` in the current path as follows:

```
db2 -tvf userinfo.ddl
```

Figure 57 on page 69 shows the results of the `connect` and `create table` commands on the bdtest1 server.

```
[db2inst1@bdtest1 db2inst1]$ db2 connect to userstuf

   Database Connection Information

 Database server        = DB2/LINUX 6.1.0
 SQL authorization ID   = DB2INST1
 Local database alias   = USERSTUF

[db2inst1@bdtest1 db2inst1]$ db2 -tvf userinfo.ddl
create table userinfo( first_name varchar(20) not null, last_name
varchar(20) not null, email_address varchar(50) not null, date_joined
date not null, birth_date date not null, address varchar(200) not null,
username varchar(20) not null, primary key(username) )
DB20000I  The SQL command completed successfully.
```

Figure 57. Creating the USERINFO table on server bdtest1

Repeat this step to create the other three database tables, USERORDERS, USERVERIFY, and USERPROFILE, replacing the userinfo.ddl script file name with the script names for the other database tables, as appropriate.

The default location for these tables is in the DB2 tablespace USERSPACE1; so, we will find the associated database table files in the directory db2inst1/NODE0000/SQL00002.

3.4 Domino database

In this section, we discuss the Domino database that we use to store the redbook information. We copy the Redbooks.nsf database as a subset of the actual redbook database that is used by ITSO. In this section, we discuss an overview of the Domino database concept in Section 3.4.1, “Domino database concept” on page 70 and the structure of our Redbooks database in Section 3.4.2, “The Redbooks database” on page 71.

3.4.1 Domino database concept

A Domino database is a collection of related information stored in a single file. A Domino application uses at least one database. A database holds information about its design as well as data. Domino data is organized as documents. A document is defined as an object containing text, graphics, video, or audio objects, or any other kind of rich text data.

Domino databases contain documents, as opposed to relational databases, which contain tables. Unstructured or structured data, as well as associated programming logic, are stored within the database.

Domino databases are essentially a complete application with its associated data. They contain all of the forms, views, agents, fields, and code required to store and manage data. The databases can reside on a Domino server or Lotus Notes workstation.

There are many design features that are available for Domino databases.We will evaluate the forms and views as the main design element that we are concerned with in this project.

3.4.1.1 Form

The form is the skeleton provided to users to enable them to enter data, either by typing or by using other control buttons. A form can also be thought of as an abstract structure of a document. It defines all the fields and its type for a document. There is at least one form in a database, although a typical business application will have many forms, each targeted to the type of information that the user wants to save in the database.

The form contains all the design elements: Fields to store the user's information, static text, buttons, sections, images, and subforms that help the user enter the data into the database.

3.4.1.2 View

A view shows a list of documents stored in a Domino database and can be thought of as a table of contents of a database. Views provides a means to organize documents in a database to be presented to the users or a program. Each row listed in a view represents data taken from a single document. Each column represents a field or a result of a formula taken from that document.

Domino databases have at least one view, but most of them have more. A view can display all the documents in the database, or it can display a subset of the documents. Views can present documents sorted on different fields, for example, sorted by topic, creation date, or author.

3.4.1.3 Documents

A Domino document is a container for data and is similar to a row in a database management system (DBMS). Each document is a collection of fields. A unique identifier is given to each document upon its creation and is used by the Domino database as a primary key.

A document's design is held within a Domino form, which specifies what a document can contain in each field, for example, a form number, a date, text, or rich text.

3.4.2 The Redbooks database

The Redbooks.nsf database is replicated from the current redbooks view. The important feature that is available from the redbook database is the Redbook form and the views available. This section is divided into:

- Section 3.4.2.1, "Redbook form" on page 71, which discusses the format of the redbook documents.
- Section 3.4.2.2, "Redbook views" on page 74, which discusses the views that we created to support our Web application.

3.4.2.1 Redbook form

The redbook is available in two forms, a standard notes form and the abstract Web page template. The IIOP access utilizes the standard notes form, while the direct Web access uses the Web page template.

Figure 58 shows the form lists of our Redbooks database.

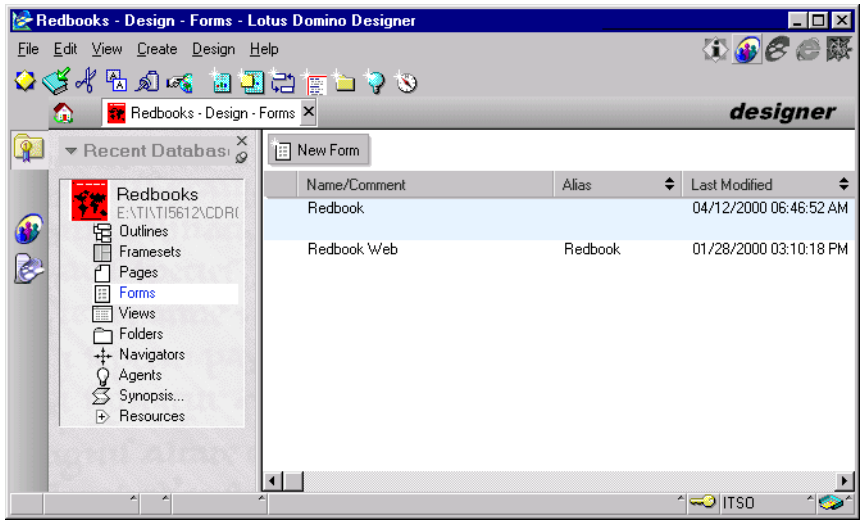

Figure 58. Redbook forms in the Redbooks.nsf

The redbook form itself is organized in a graphical way as shown in Figure 59 on page 73.

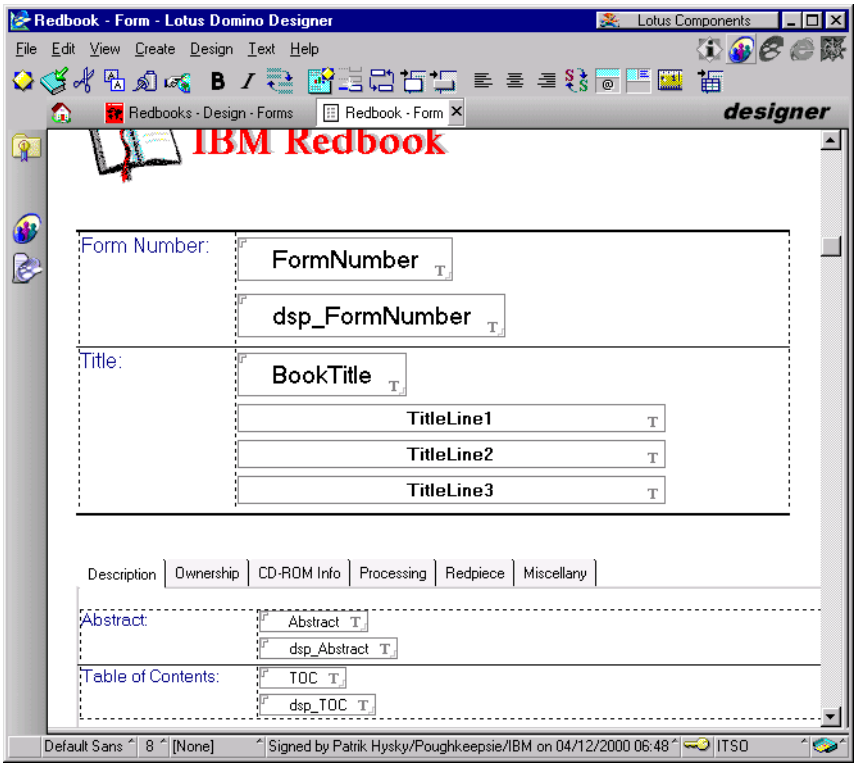

Figure 59. The IBM Redbook form

The redbook document has the following fields that we used:

- FormNumber: The IBM publication number
- BookTitle: The document title
- PublishDate: The publication date of the redbook, for sorting
- Platforms: The platform that is used for the redbook as the categories
- Abstract: The abstract text of the book

Figure 60 on page 74 shows the Web subform for a redbook document, which we emulate using our servlet, VBDOpen, in Appendix C.5, “Java servlet VBDOpen” on page 168.

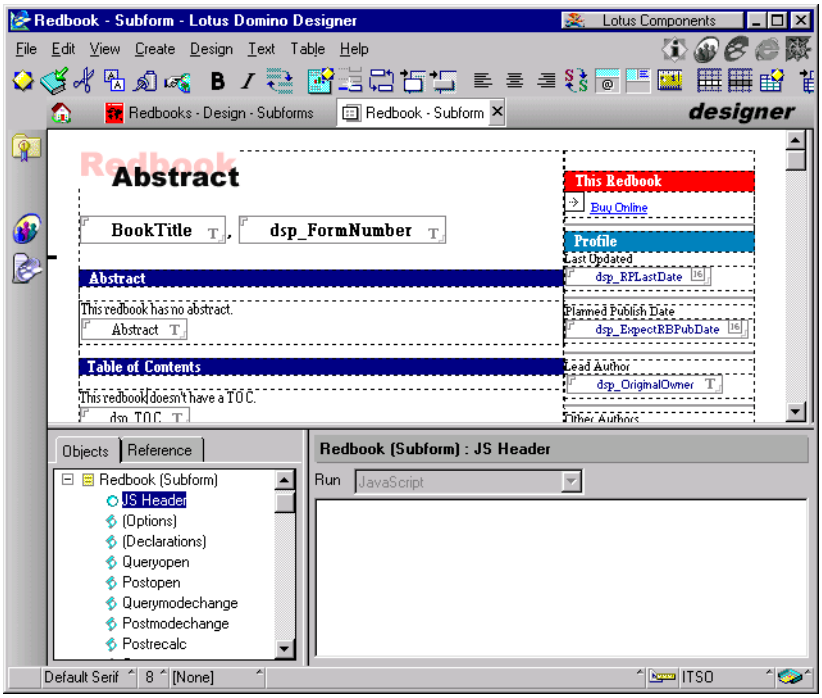

Figure 60. The Redbook subform in our workshop

3.4.2.2 Redbook views

Views, as the primary means of searching and navigating in the Domino database, must be created according to the programming needs. We created two views in our project, these are:

- The platform sorted view

  This view shows all the redbooks sorted by their platform and then by their publishing date in each platform's category. The platforms that we have are:

  - AS/400
  - Netfinity and Software
  - Network Computing Framework
  - Networking Hardware
  - OS/390 and S/390
  - RS/6000 and AIX
  - Scalable Parallel
  - VM/ESA
  - VSE/ESA

- Windows and WinNT
- Other

Figure 61 shows the platform sorted view in the Redbooks database through the Domino Designer client.

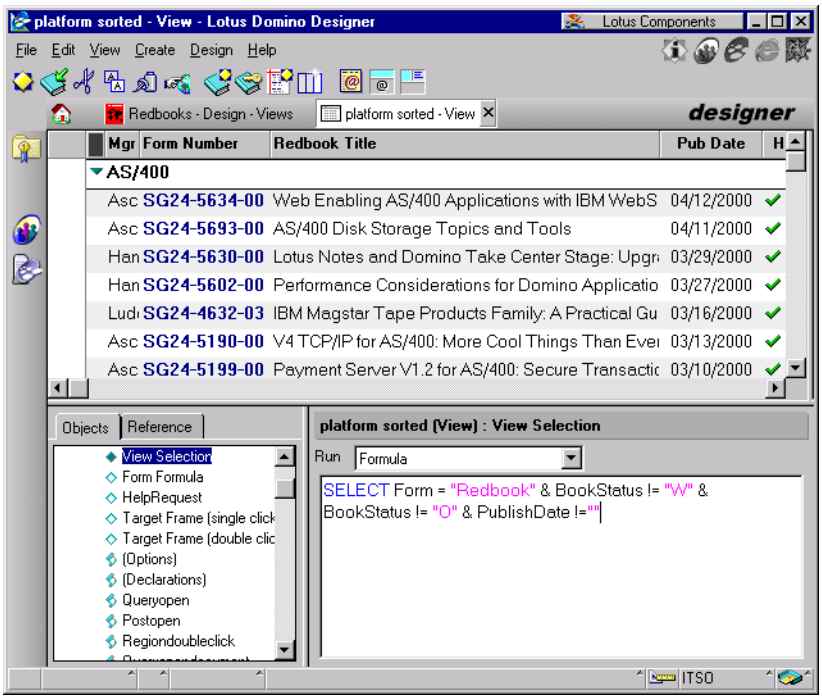

Figure 61. The design of the platform sorted view

- The All books view

  This view is not sorted by platforms; it just shows all documents sorted by their publishing date. Figure 62 on page 76 shows the All books view from the Redbooks database with the Domino Designer client.

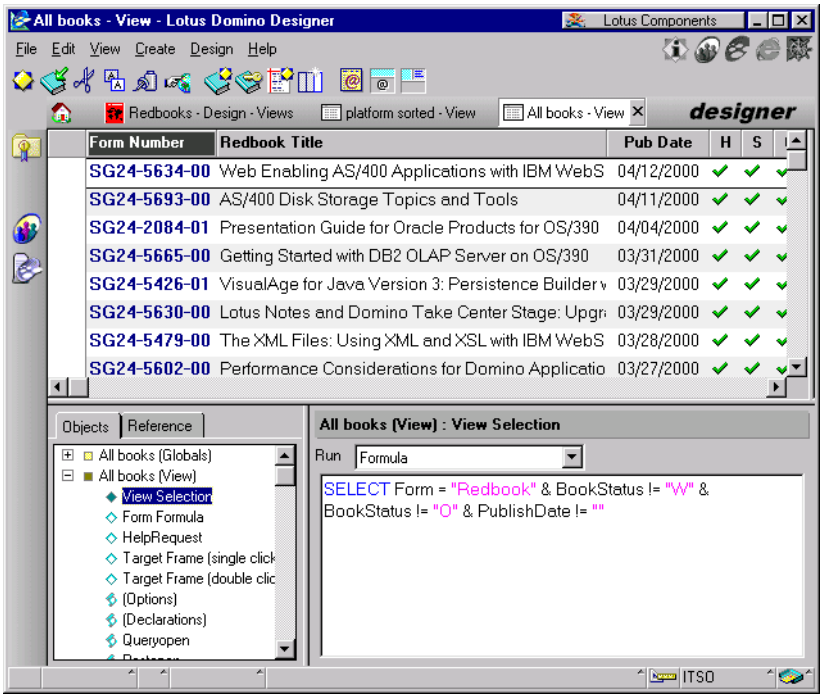

Figure 62. The design of the All books view

The following steps explain the creation process for our platform sorted view:

1. On the Domino Designer client, click on **Views** from the list of objects in the database, and click on the **New View** button.
2. Click on **Copy from** and select the view that you want to copy it from and click **OK**. The Create view window, such as that shown in Figure 63 on page 77, will appear.

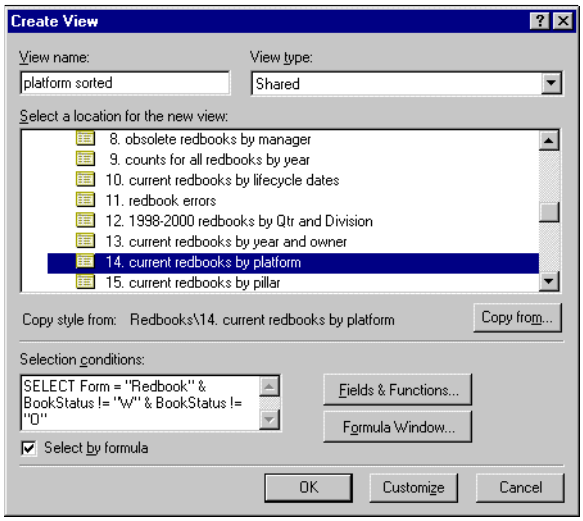

Figure 63. Create View platform sorted

3. Put in the view name, determine the view location, and enter the Selection condition and click **OK** in the Create View window shown in Figure 63.
4. Double click on the **newly created platform sorted** view, and you will get the screen similar to the one in Figure 61 on page 75.
5. Select the **PubDate** column, right click the column header and select **Column properties**. The a pop-up window for Column property is opened as shown in Figure 64 on page 78.

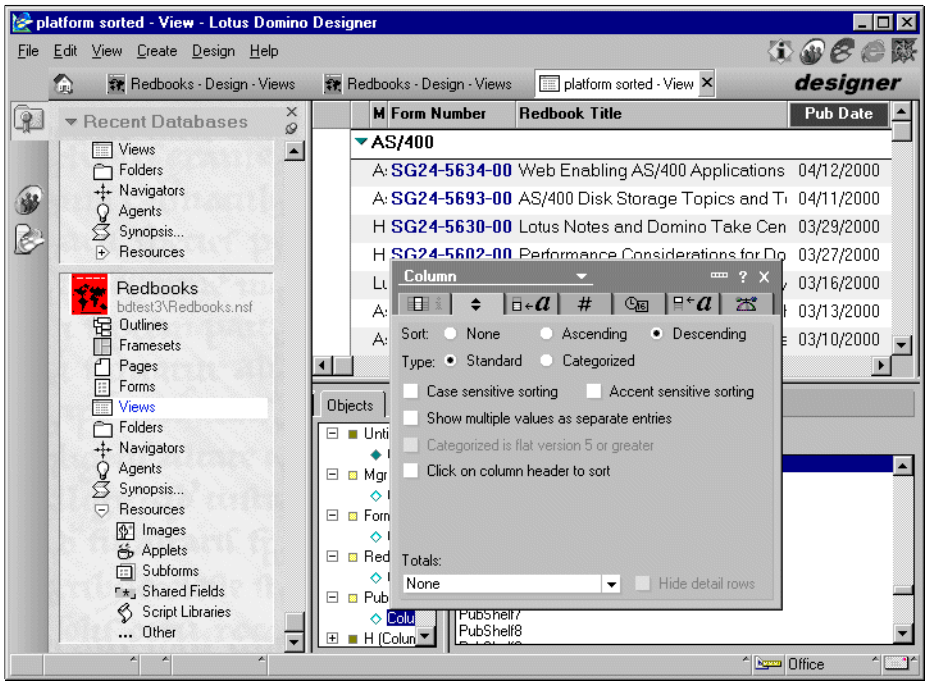

Figure 64. Column properties of PubDate

6. Define the sort option for the PubDate column as shown in Figure 64.
7. Check the other columns and make sure that the only sorted column is the Category and the PubDate column.
8. Click **File** and **Save**.

3.5 Servlet programs

In our project, we utilize a lot of servlets to provide business logic and programming capability to our Web server. These servlets access the DB2 database, Domino database, HTTP session information and Java bean object. In this section, we discuss mostly the behavior of our servlet and how we code the servlets to perform what we want. The discussion in this section is separated into:

- Section 3.5.1, “Servlets” on page 79, which discusses an overview of servlet and gives the list of servlets that we use in the project.
- Section 3.5.2, “Maintaining state in HTTP” on page 81, which shows the way the servlet handles session attributes.

- Section 3.5.3, “Accessing DB2 via JDBC” on page 83 provides a discussion on accessing DB2 database using the servlet.
- Section 3.5.4, “Accessing Domino data in Linux with Java” on page 93 discusses the way we access a Domino data using the servlet.

3.5.1 Servlets

Servlet is the Java replacement of the cgi programming for Java enabled Web servers. The servlet can be viewed as a server-side Java program that performs enhanced functions for the Web server. Servlet is implemented based on the Sun specification that can be found at:

```
http://java.sun.com/docs/servlet
```

The following codes are required when creating a servlet:

- Creating a Java servlet requires the following imports:

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

- The class must extend the HttpServlet class as follows:

```
public class <classname> extends HttpServlet
```

- We must at least implement either the doGet or doPost method.

Therefore, the minimal code required to implement a servlet and service an HTTP POST request is shown in Figure 65.

```
import javax.servlet.*;
import javax.servlet.http.*;
public class UselessServlet extends HttpServlet {
   public void doPost(HttpServletRequest req, HttpServletResponse res)
   throws ServletException {
   // service POST here
   }
}
```

Figure 65. Minimum code for servlet implementing HTTP GET request

As another example, we want to implement both GET and POST operation similarly. Since both methods will access the same code base, we accomplish this by implementing both doGet and doPost and passing the parameters to another method called doService. This is illustrated by the code in Figure 66 on page 80.

```
import javax.servlet.*;
import javax.servlet.http.*;
public class UselessServlet extends HttpServlet {
   public void doGet(HttpServletRequest req, HttpServletResponse res)
   throws ServletException {
      doService(req, res);
   }
   public void doPost(HttpServletRequest req, HttpServletResponse res)
   throws ServletException {
      doService(req, res);
   }
   public void doService(HttpServletRequest req, HttpServletResponse
res)
   throws ServletException {
   // Service GET or POST here
   }
}
```

Figure 66. Implementing doGet and doPost with doService

Parameters are passed to the servlet in the HttpServletRequest object. The getParameter method of the HttpServletRequest class is used to obtain these parameters. For example, if our servlet UselessServlet is accessed with a GET request of the following manner:

```
http://bdtest1/servlet/UselessServlet?password=elephant
```

we can obtain the parameter value for password using the following code:

```
String password=req.getParameter("password");
```

Although the parameter is not showed in the URL, the POST method also retrieve these parameters using the HttpServletRequest object. Therefore, the same code can be for both the GET and POST method.

Table 5 shows the list of servlets that we use and the environment that they run on.

Table 5. Servlet list

Servlet name	Function	Environment
TestDB2	Test DB2 connection	DB2
TestDomino	Test Domino connection	Domino

Servlet name	Function	Environment
AccessUserstuf	Primary interface to DB2. It handles both GET and POST method, and its action is determined by a parameter called action.	DB2 and Beans
VBDSearch	Performs search to the Redbooks database. The search argument is stored in a parameter called search.	Domino
VBDList10	List 10 latest book of a category. The category is indicated by its first argument.	Domino
VBDLatest	Provide 10 latest published book from all category in the Redbooks database.	Domino
VBDOpen	Access the abstract of a redbook and display it in HTML.	Domino

3.5.2 Maintaining state in HTTP

HTTP is a stateless protocol. To perform transactional activities, some kind of state must be maintained. There are many ways of maintaining state that has been developed: Using cookies, URL re-writing, hidden form fields in the HTML form, and so on. The Java servlet API implements access to state via the HttpSession object.

3.5.2.1 Session Tracking

HttpSession is a technological implementation that is used to manage sessions on the stateless HTTP protocol. It is part of the Java Servlet API, and it is accessible in the Java Servlet development kit.

The HttpSession object is the Java programing object that has a dictionary-like interface to store user-defined data. Data can be stored in the HttpSession object with any keyword, and the data is then retrieved with that keyword.

Each of the session objects has its own ID. HttpSessions objects are related to the client using its session ID. The session tracker uses the session ID to find the user's session object. The session object is not sent between the Web client and the server. Only the ID is passed back and forth between the Web client and the Web server. This means that the session object is suitable to store the user's secure data.

Since there are many different ways to represent a session data, the HttpSession provides a seamless, uniform way to access session data. It

uses a cookie if the client Web browser supports that, or it can also use URL rewriting to carry the session ID that is associated with the client.

The data that is stored in the HttpSession object is shared across requests from the same machine. Therefore, the HttpSession object is also suitable to store individual session profile data from the client.

The HttpSession object is not persistent data. It is deleted after a time out period or by the invalidate() method. In order to store data persistently, WebSphere provides a UserProfile (see Section 5.1.1.2, “User profile tracking” on page 140).

The HttpSession description can be found at

`http://java.sun.com/products/servlet/2.1/api/javax.servlet.http.HttpSession.html`

More information on session tracking can be found at:

`http://java.sun.com/docs/books/tutorial/servlets/client-state/session-tracking.html`

3.5.2.2 Using the HttpSession object

To use these session objects, we obtain a session object, get or put values to this object, and invalidate it if we need to remove the context.

Obtaining a session object

To obtain a session object associated with the client, we use the getSession method as follows (we assume `req` is of type HttpServletRequest):

```
boolean flag=true;
HttpSession session=req.getSession(flag);
```

This obtains the current valid session associated with this request from the client. The flag indicates whether a session object should be created if it is not found.

Getting values from a session object

For example, to get the username value from the session, we use the getValue method associated with the session object as follows:

```
String username=(String) session.getValue(“username“)
```

Putting values to a session object

We can also set values in the session using the putValue method. In the following example, we associated the password key with a value of `elephant`:

```
session.putValue(“password“,”elephant”)
```

Invalidating a session object
Finally, to invalidate the session instance, we use the invalidate method as follows:

```
session.invalidate();
```

This may be required during logout processing.

3.5.3 Accessing DB2 via JDBC

The initial setup for a Java class to access DB2 using JDBC requires importing the appropriate package, registering the DB2 driver, and creating the connection to the DB2 database. This only needs to be done once.

After that, each SQL statement to be execute requires a statement to be created, then executed, and finally closed.

Lastly, we give an example of what is required to write a Java servlet to access a DB2 database using JDBC.

3.5.3.1 DB2 JDBC Drivers

DB2 JDBC main usages are for executing dynamic SQL statements and for supporting upper data layers, such as SQLJ and IBM DataAccess beans.

DB2 provides two JDBC drivers: A native API Type 2 driver and a net protocol Type 3 driver. The Java class libraries for both drivers are found in db2java.zip.

- The native API driver is for server-side applications that run on the application server. The driver consists of Java parts and a C native API interface (DB2 CLI). The driver translates JDBC calls into DB2 CLI calls. It then gives the calls to the DB2 Client component, which passes them to the DB2 Server. After that, the driver returns the result back to the application. The class name for this JDBC driver is COM.ibm.db2.jdbc.app.DB2Driver.
- The net protocol driver is for creating an application using the Java applet model. The driver is a 100 percent Java program that is downloaded along with the applet to the client browser. The driver exchanges messages, by using the TCP socket protocol, to a JDBC Applet Server in the application server machine. The JDBC Applet Server translates applet messages into DB2 CLI calls. It sends the calls into the DB2 Client component, which passes them to the DB2 Server. Then, the JDBC Applet Server passes the results back to the applet. The class name for this JDBC driver is COM.ibm.db2.jdbc.net.DB2Driver.

The JDBC Applet Server is a stand-alone socket server application. To activate the JDBC Applet Server, enter the following command:

```
db2jstrt <port>
```

This will activate the JDBC Applet Server listening on port <port>. However, we do not activate this function since the DB2 database is only accessed through the WebSphere's servlet, not direct applet access.

3.5.3.2 Creating a JDBC program

To create a Java program that accesses DB2 using JDBC, we need to perform the following steps.

Importing the java.sql package

Creating a Java program to access DB2 via JDBC requires the following import:

```
import java.sql.*;
```

Registering the DB2 driver

We must then register the DB2 JDBC class. As mentioned in 3.5.3.1, "DB2 JDBC Drivers" on page 83, this can be one of two classes:

- COM.ibm.db2.jdbc.app.DB2Driver

  Use this driver if the DB2 CAE is installed on the client machine.

- COM.ibm.db2.jdbc.net.DB2Driver

  Use this driver if you do not wish to have the DB2 CAE installed on the client machine.

The code to register the driver, assuming you would be using this Java class with the native API driver, would be as follows:

```
try {
   Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
} catch (Exception e) {
   e.printStackTrace();
}
```

Creating the DB2 connection

A connection to the DB2 database is created with the getConnection method from the DriverManager class. For example, to connect to the userstuf database (we assume database userstuf has been cataloged correctly on this server), with the default userid and password, we would use the call method as follows:

```
String url="jdbc:db2:userstuf"
```

```
String uid="db2inst1";
String pwd="ibmdb2";
Connection con=null;
try {
   con=DriverManager.getConnection(url, uid, pwd)
} catch (SQLException e) {
   e.printStackTrace();
}
```

Creating the DB2 statement

Each time we need to execute an SQL statement, we must create a statement first. This is done with the following code (we assume the connection variable `con` has already been created and assigned to. See "Creating the DB2 connection" on page 84):

```
Statement stmt=con.createStatement();
```

Executing the DB2 statement

With this statement, we can now perform two types of SQL statements:

- Inserts/Update/Deletes

  SQL inserts, updates, and deletes return the number of rows affected by the SQL statement. We would use the executeUpdate method to execute these SQL statements. The method returns the number of rows affected by the SQL statement, of type integer. Hence, we could code the following to delete all rows from the USERINFO table where the username equals "alistair" as follows:

  ```
  String sqlStmt="delete from userinfo where username='alistair'";
  int rsi=stmt.executeUpdate(sqlStmt);
  ```

- Queries

  SQL queries return a set of rows. This is called a result set of rows. Use the method executeQuery from the statement, which returns a type of ResultSet. We could code a simple SQL query to select all rows from the USERINFO table as follows:

  ```
  String sqlStmt="select * from USERINFO"
  ResultSet rs=stmt.executeQuery(sqlStmt);
  ```

  To obtain each result set row, we use the next() method. This method returns true if there is a next result set row, and it is intially positioned before the first result set row. For example:

  ```
  if (rs.next()) {
     // process next result set row
  }
  ```

Each result set is described by its meta data. To get meta data for a result set, we use the getMetaData() method for the result set. For example:

```
ResultSetMetaData rsmd=rs.getMetaData();
```

We can illustrate this best with some sample code. For the following example code, we assume the connection `con` has been created (see , “Creating the DB2 connection” on page 84) and an output stream `out` defined.

a. First, we create the statement:

```
Statement=con.createStatement();
```

b. Next, we execute the statement of type query, with a return result of type ResultSet:

```
ResultSet rs=stmt.executeQuery(“select * from userinfo”);
```

c. To print out the first column of each result set row, we have the following code snippet:

```
while (rs.next()) {
   out.print(rs.getString(1));
}
```

d. To get even more fancy, we can print out all of columns of each result set row by obtaining the result set meta data, determining how many columns are in the result set, and then printing out each column with the getString method:

```
ResultSetMetaData=rs.getMetaData();
int numFields=rsmd.getColumnCount();
while (rs.next()) {
   for (int currentField=1; i<=numFields; currField++) {
      out.print(rs.getString(currField));
   }
   // next row
   out.println();
}
```

Fields/Columns start at 1

Note that the field/column count starts at 1. Hence, the for loop in the preceding sample code starts looping from 1.

The getString method can be used with a column number as we have done or with a column name, for example:

```
rs.getString(“first_name”)
```

You will see this method used in both forms in later examples.

Explicitly closing the DB2 statement
The DB2 statement is automatically closed when the last result row is reached or when the Java instance is terminated. In some cases, it is desirable to immediately release a statement's database and JDBC resources; the close method provides this immediate release as follows:

```
stmt.close();
```

The bare minimum required to execute a simple DB2 query statement on the USERINFO table, located in the USERSTUF database, is shown in Figure 67 on page 87.

Figure 67. Minimum code required to access a DB2 table from a servlet

3.5.3.3 DB2 servlet AccessUserstuf

The AccessUserstuf servlet is responsible for all actions involving the USERINFO, USERPROFILE, USERVERIFY, and USERORDERS database tables. As such, it must perform the following:

- At initialization (when it is first loaded by the WebSphere engine), it must register the DB2 class library and create a connection to the USERSTUF

"Accessing DB2 via JDBC" on page 83, and given example code in Figure 67 on page 87.

- At each invocation, via a POST method, it must then implement the action required. We will discuss the implementation of these actions in further detail in this section.

The DB2 servlet AccessUserstuf must act on POST methods. Hence, it is implemented as an HttpServlet. Each action can either succeed or fail. The page to redirect to, in either case, is passed as parameters in the Request object with the name of success and failure.

A brief discussion of the actions implemented in the AccessUserstuf servlet are as follows:

- login

  This action requires a valid username and password in Request object. It then put s the username, password, user information, and user profile in the HttpSession object

- changepassword

  This action requires a valid username and password in the HttpSession object and a new password (password1) with its validation (password2) in the Request object. It checks whether password1 is equals to password2 and updates the userverify table and the password value in the HttpSession object.

- register

  This action requires a non-existing valid username, a non-blank password, valid userinfo, and userprofile in the Request object. It then performs the following:

  - Inserts the new username and password into userverify table
  - Inserts the new userinfo into userinfo table
  - Inserts new userprofile into userprofile tables
  - Puts the username, password, userinfo, and userprofile in the HttpSession object

- update

  It requires a valid username, password in HttpSession object, and a valid userinfo and userprofile in Request object. It then updates userinfo table and userprofile table and the Session object.

- delete

This action requires a valid username and password in HttpSession object. It then delete the userverify entry and invalidates the HttpSession object.

- order

  This action requires a valid username, password in Session object, and product_id in Request object. It then inserts the new order into userorders table.

- listorders

  This action requires a valid username and password in HttpSession object. It then returns Orders object with a list of order_ids, associated timestamps and product_ids.

The complete listing of the AccessUserstuf servlet code is available in Appendix B.2, “Java servlet AccessUserstuf” on page 147. The complete discussion of all the method performed in the servlet is out of the scope of the book. We will demonstrate the detailed implementation of the login action in Section 3.5.3.4, “AccessUserstuf action=login” on page 89. The discussion of the listorders action is discussed with the Java bean discussion in Section 3.7, “Accessing data through a Java bean” on page 103.

3.5.3.4 AccessUserstuf action=login

The following sequence of tasks is performed when the servlet is invoked with the action parameter that contains the value of login:

1. The login action expects the username and password parameters to be set in the Request object and will access these parameters via the getParameter method. Assuming there is a HttpServletRequest object passed via variable `req`, the code to do this is as follows:

   ```
   String username=req.getParameter("username");
   String password=req.getParameter("password");
   ```

   For our purposes, this code is reused often; hence, we code a getUsername and getPassword private method to obtain these values. The method for getUsername is coded as follows:

   ```
   private String getUsername(HttpServletRequest req) {
      return req.getParameter("username");
   }
   ```

   Hence, we change our code as follows:

   ```
   String username=getUsername(req);
   String password=getPassword(req);
   ```

Overloading methods

There is another form of the getUsername and getPassword method that we use, as in the listorder action in Section 3.7, “Accessing data through a Java bean” on page 103, which retrieves the calue from the HttpSession object. So, we also have the following method for getUsername:

```
private String getUsername(HttpSession session) {
   return session.getValue("username");
}
```

We have now overloaded the getUsername and getPassword methods. These methods are distinguished by the parameter type being passed,

2. The next step is to verify if the username and password are valid. If not, we must then call some sort of error routine and pass control back to a failure page. Ideally, we could code it as follows:

```
if (!passwordsMatch(username, password)) {
   uidPwdError(username, password, failurepage, req, res);
}
```

We need the passwordsMatch method to do the following:

a. Access the USERSTUF database

   We assume the Connection object `con` has already been initialized.

b. Try to obtain a password for the provided username

   We code this by selecting a password from the USERVERIFY table based on the username. We then check that there is a result row returned with the rs.next() method call, and ensure a valid password has been returned by checking that the result set meta data has at least one column as a result.

c. Return a value of true if the passwords match, false otherwise.

   This is done by checking for string equality between the provided password and the password found in the USERVERIFY table for the username.

Java code for the passwordsMatch method is shown in Figure 68 on page 91.

```
   protected boolean passwordsMatch(String username, String password) {
      boolean match=false;
      try {
         String sqlStmt="select password from userverify where
   username='"+username+"'";
         Statement stmt=con.createStatement();
         ResultSet rs=stmt.executeQuery(stmt);
         if (rs.next()) {
            ResultSetMetaData rsmd=rs.getMetaData();
            if rsmd.getColumnCount()>0) {
               String thePassword=rs.getString(1);
               if (thePassword.equals(password)) match=true;
            }
         }
      } catch (SQLException e) e.printStacktrace();
      return match;
}
```

Figure 68. Java code for the passwordsMatch method

3. If the login is to be successful, we must populate the Session object, using the putValue method, with the following information:

   a. username, password

      This information is already available in the username and password variables. Hence, the code is simply:

      ```
      session.putValue("username", username);
      sessoin.putValue("password", password);
      ```

   b. user information

      The user information, for example, an address or e-mail, must be obtained from the USERINFO database table. We perform a select of the relevant information, then iterate through the list of USERINFO columns to be populated into the Session object.

      We predefine a string array called infoCols, which contains a list of values that we intend to obtain from the select statement, and then place it in the Session object (in effect, using this array of strings as a filter). The code looks something like Figure 69 on page 92.

```
private String infoCols[]={"first_name", "last_name",
"email_address", "date_joined", "birth_date", "address"};
...
String sqlStmt="select * from userinfo where
username='"+username+"'";
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery(sqlStmt);
if (rs.next) {
   for (int i=0; i<infoCols.length; i++) {
      String info=infoCols[i];
      session.putValue(info, rs.getString(info));
   }
}
```

Figure 69. Java code to populate user information in the Session object

c. user profile

In a similar fashion, we perform a select of the relevant columns from the USERPROFILE table. If the particular category (as400, netfinity and so on) does not have a null value, it means the user has expressed an interest in this category, and, thus, we should set the value in the Session object.

Once again, we use an array of strings called profileCols as a filter. The code looks something like Figure 70.

```
private String profileCols[]={"as400", "netfinity", "ncf", "nw",
"s390", "rs6000", "scalable", "vm", "vse", "windows", "other"};
...
String sqlStmt="select * from userprofile where
username='"+username+"'";
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery(sqlStmt);
if (rs.next) {
   for (int i=0; i<profileCols.length; i++) {
      String category=profileCols[i];
      if (category=! null) session.putValue(category, category);
   }
}
```

Figure 70. Java code to populate user profile information in the Session object

4. Finally, once login is successful, and the Session object populated with the appropriate information, we can redirect the flow of control to another page on our server, indicated by the successpage string variable:

   `res.callPage(successpage, req).`

3.5.4 Accessing Domino data in Linux with Java

There are several mechanisms for Java programs to access Domino data, Domino services, or data in other database management systems, such as:

- Domino Objects for Java through the lotus.domino package included with the NCSO.jar or notes.jar. The Domino objects represent and can be used to manipulate Domino data, such as item values, components, such as databases and views, and services, such as user registration functions.
- The Java Native Interface (JNI) used to call C or C++ API functions or programs. The Lotus C or C++ Domino and Notes APIs can be used to manipulate Domino data and services at a more granular level than what is provided by Domino objects.
- The Lotus Script Extender (LSX) toolkit and its Java Adapter. The LSX toolkit lets LotusScript developers create custom objects in C++ and call them from LotusScript. The Java Adapter provides the facilities to call those objects from Java. Since the language used to create the objects is C++, those C++ programs can make calls to the Domino and Notes APIs or other APIs to access data and services.

In this Redbook, we focus on using the Domino objects classes to access Domino data and services. For complete information about the Lotus C and C++ APIs, and the LSX toolkit, visit the Developer Central Web site at: `http://www.lotus-developer.com`.

The Domino Objects class architecture is based on a conceptual containment model, where the container defines an object's scope. A container object is always used to access the objects it contains. For example, we use a Session object to get Database objects, and a Database object to create Document objects.

Since one Domino object may be contained by several others, a full containment diagram is beyond the scope of this redbook. However, some of the key containment relationships are shown Figure 71 on page 94.

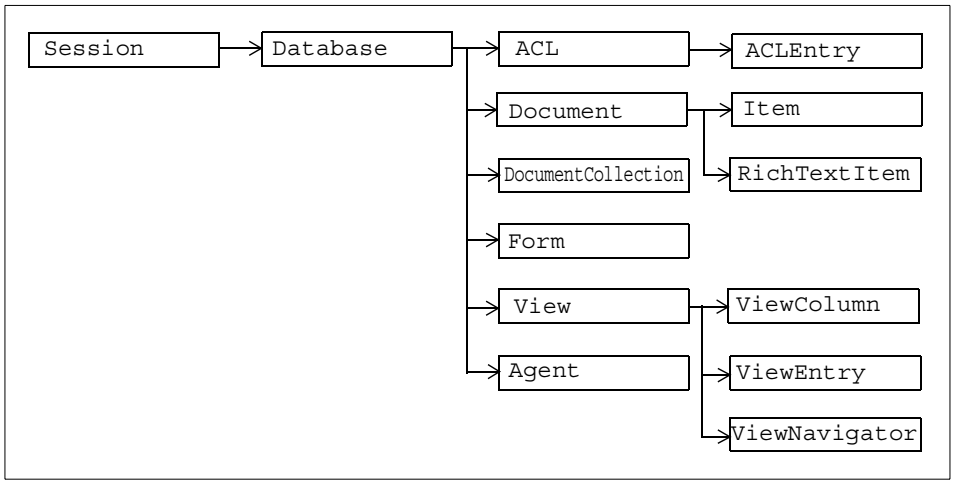

Figure 71. The Domino object containment architecture

The Domino Java classes are provided in the following packages:

- Domino notes.jar file contains lotus.domino classes for local Java access.
- Domino NCSO.jar file contains the CORBA implementation of the lotus.domino classes. It is used by remote Java programs only.

In Domino R5 for Linux, remote Java programs can access the Domino object classes on the Domino server using CORBA/IIOP network communication mechanisms.

More information about Domino CORBA programming is provided in the redbook, *Lotus Domino Release 5.0: A Developer's Handbook*, SG24-5331.

We use vi to write a servlet and compile the servlet with IBM JDK 1.1.6 for creating a servlet; however, you can develop the Java program using another Java development environment that supports JDK 1.1.6 or later, such as Sun JDK 1.1.6, IBM VisualAge for Java 3.0, or Symantec Visual Cafe 3.0.

The CLASSPATH for compiling and running a Domino servlet must include:

- Base Java classes in classes.zip
- Java servlet classes
- Domino classes in NCSO.jar for remote access or notes.jar for local access

In the following sections, we explain how we develop the Domino servlet. Only the important parts of the code are shown here. For a complete listing of

the servlets, refer to Appendix C, “Java code relating to Domino database” on page 159.

Import packages

The following packages needs to be imported to construct our servlet:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import lotus.domino.*;
import java.util.Vector;
```

Besides the standard Java packages and the servlet packages, the lotus.domino package is the Domino Object that we need to access the Domino database. We also use java.util.Vector to provide support for dynamic length array.

Access to Domino session and database

Session is the root in the containment hierarchy on the Domino object. We can open a session to Domino server using several method depending on how we will access the server.

We can have two option for accessing the server:

- Remote access

  For accessing the Domino database remotely, we need to put the hostname, userid, and password in the NotesFactory.createSession method. For example, we try to open database myDB on server host1 with userid myUser and password myPass:

  ```
  Session s = NotesFactory.createSession(“host1:8000”, “myUser”, “myPass”);
  Database db = s.getDatabase(“myDB”);
  ```

  Remember that the HTTP and DIIOP server must be running on the server. In our case, we use anonymous access to DIIOP so that we can just put the hostname in the NotesFactory.createSession method. For more information on how to set up DIIOP user access, please refer to Section 2.4.2.4, “Domino server Java and CORBA requirements” on page 26.

- Local access

  Local access is simpler, but the Domino server or client should be running on the machine the Java program is executing. For this local access, the Domino servlet manager must be enabled. The following example accesses the myDB database locally:

  ```
  Session s = NotesFactory.createSession();
  ```

```
Database db = s.getDatabase("myDB");
```

Access to View

After we success to open the database, we can access the views and folders in the database through getView and getViews. With the View class, we can locate documents within views and folders and perform operations there.

```
View view = db.getView(viewname);
```

The viewname must contain the name of a view or a folder in the Domino database. The view or folder name must be specified exactly.

Access to documents with ViewNavigator

ViewNavigator is an object that is created under the View object and selects the entries in a selected view and navigates the view to access a specific document in the view.

ViewNavigator can be created through several methods:

- createViewNav

  Creates a view navigator for all the entries in a view, for example:

  ```
  Session s = NotesFactory.createSession("bdtest3:8000");
  Database db = s.getDatabase("Redbooks");
  View view = db.getView("platform sorted");
  ViewNavigator viewnav = view.createViewNav();
  ```

  This program will get all entries from the Redbooks database on server bdtest1 from the view platform sorted. The content of viewnav may have a category or document depending on the structure of the database.

- createViewNavFromCategory

  This method will create a view navigator for all the entries in a view under a specified category. For example, we use this methods to get all entries in the selected category of the Redbooks database:

  ```
  Session s = NotesFactory.createSession("bdtest3:8000");
  Database db = s.getDatabase("Redbooks");
  View view = db.getView("platform sorted");
  ViewNavigator viewnav = view.createViewNavFromCategory("AS/400");
  ```

  This program will get all entries from the Redbooks database in server bdtest1 on the view platform sorted and at the category AS/400.

- createViewNavMaxLevel

  This method will create a view navigator for all entries in a view down to a specified level. For example, if we need to get all entries at the top level of view, we can use the following code:

```
Session s = NotesFactory.createSession("bdtest1:8000");
Database db = s.getDatabase("Redbooks");
View view = db.getView("platform sorted");
ViewNavigator viewnav = view.createViewNavMaxLevel(0);
```

Access the fields of a document

After we get all entries in Navigator, we can perform some methods to access the document from navigator:

```
ViewNavigator nav = view.createViewNavFromCategory("AS/400");
ViewEntry entry = nav.getFirst();
while (entry!=0) {
   Document doc = entry.getDocument();
}
```

This code will return documents associated with the view entry.

If the view entry is not a document, it returns null and also returns null if the document is deleted after the ViewEntry object is created.

The getDocument() method returns a document object. The document object has several methods that we can use to retrieve the fields value:

- getItemValueString(String)

  This method returns the value of an item with a single text value. If the item has no value, or the value is numeric or date-time, this method returns an empty string.

  ```
  out.println(entry.getDocument().getItemValueString("FormNumber");
  ```

  This code will return the text from column FormNumber.

- getItemValue(String)

  This method returns the value of the appropriate field from the document. The data type of the result is always java.util.Vector; however, the element of the Vector is dependent to the actual field type as shown in Table 6.

Table 6. Data type of Item from getItemValue()

Item type	Value return type
Rich text	java.util.Vector with a String element as plain text
Text (includes Names, Authors, and Readers item types) or text list	java.util.Vector with String elements
Number or number list	java.util.Vector with Double elements
Date-time or range of date-time values	java.util.Vector with DateTime elements

In our project, we found that the value of column PublishDate was a date-time value; so, we have to convert the data to a string with the following code:

```
Vector dM = entry.getDocument().getItemValue("PublishDate");
if ( dM.size() > 0 )
   DateTime dt = (DateTime)dM.elementAt(0);
String PublishDate = dt.getDateOnly();
```

3.6 JSP files

Java Server Pages (JSPs) are HTML files that provide dynamic content and improved functionality. They make use of Java or various other scripting languages.

The JSPs are compiled into servlets by the WebSphere Application server before they are used.

When you install the WebSphere application server on a Web server, the Web server's configuration is set by default to pass HTTP requests for JSP files (files with the extension .jsp) to the application server. The application server configuration is set by default to pass JSP files to its JSP processor, the pageCompile servlet. The JSP processor creates and compiles a servlet from each JSP file. The processor produces two files for each JSP file:

- .java file - Contains the Java language code for the servlet
- .class file - The compiled servlet

WebSphere stores the compiled JSP servlet in the /opt/IBMWebAS/servlets/pagecompile directory. The WebSphere application server can automatically detect changes in its JSPs and will recompile the JSP file as necessary. Remember that the owner of the pagecompile directory needs to be changed to nobody after the WebSphere installation.

3.6.1 The JSP components

A JSP file can have the extension .jsp, and it can contain a combination of:

- HTML tags
- JSP directives, such as `<%@LANGUAGE="Java" %>`
- Inline scripts: `<% ... %>`
- JSP tags such as `<SERVLET>`, `<BEAN>` and so on

3.6.1.1 JSP directives

JSP directive tags define the attributes that apply to the entire JSP page. We used `<%@ LANGUAGE="Java" %>` in our JSP pages, indicating that we will use Java as the default language. Figure 72 shows a portion of our logout.jsp file. Note that the JSP directive is located at the beginning of the file.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<%@ LANGUAGE="Java" %>
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;
charset=ISO-8859-1">
<TITLE>MainPage</TITLE>
<BASE TARGET="_parent">
```

Figure 72. Portion of logout.jsp

A complete listing of all relevant JSP directives can be obtained from Sun's JSP page at:

`http://java.sun.com/products/jsp`

3.6.1.2 Inline scripts

You can embed any valid Java code within a JSP file between the `<%` and `%>` tags. The Java codes can be intermingled with the HTML. Figure 73 illustrates how we embed HTML and Java into the JSP file.

```
<HEAD>
<TITLE>HomePage</TITLE>
<BASE TARGET="_parent">
</HEAD>
<% HttpSession session=request.getSession(true);
if (session.getValue("error") !=null){ %>
<B ALIGN=CENTER><FONT SIZE="+2">ERROR:</FONT></B>
<% out.println(session.getValue("error"));
}
session.removeValue("error");
%>
```

Figure 73. Embedded HTML and Java in a JSP file

In Figure 73, we embedded the HTML by ensuring that we enclosed the preceding Java statement with `<% %>`, provide some HTML codes, and then

put another set of Java code inside the <% %> tag. The HTML in Figure 73 on page 99 will not be displayed unless the condition in the Java code before it returns true.

3.6.1.3 JSP tags

The <SERVLET> tag can be used to call a servlet from within a JSP page.The syntax is:

```
<SERVLET NAME=servlet CODE=servlet_class_name CODEBASE=remote_url initparm=parameter>
<PARAM NAME=parameter_name VALUE=parameter_value>
</SERVLET>
```

We used this tag to include the latest releases from the Domino database, Figure 74 illustrates this simple use of the tag:

```
<table border=0 cellspacing=0 cellpadding=0 width=292>
   <tr align=center><td><B>Latest Releases</B></td></tr>
   <tr></tr>
   <tr valign=top align=left>
   <td width=291><servlet name="VBDGetLatest"></servlet></td>
   </tr>
</table>
```

Figure 74. Embedded servlet tag

The <BEAN> tag is used to reference a bean. We use a bean to return a list of previous orders that a user had made. The general form of the code to register the bean is:

```
<BEAN NAME="Bean_name" VARNAME="local_Bean_name"
TYPE ="class_or_interface_name" INTROSPECT="yes|no"
BEANNAME="ser_filename" CREATE="yes|no"
SCOPE="request|session|userprofile" >
<PARAM property_name="value">
</BEAN>
```

Figure 75 illustrates the usage of this bean tag.

```
<body>
<bean
  name="orders"
  type="penguins.access.db2.Orders"
>
</bean>
```

Figure 75. The bean tag

The bean is later accessed in the inline Java code from the JSP. For more discussion on beans, see Section 3.7, “Accessing data through a Java bean” on page 103.

3.6.2 Implementation of JSP in our Web application.

This section describes the way in which we implemented JSP in our Web application.

We opted to use a request to servlet response via the JSP approach. This is illustrated in Figure 76.

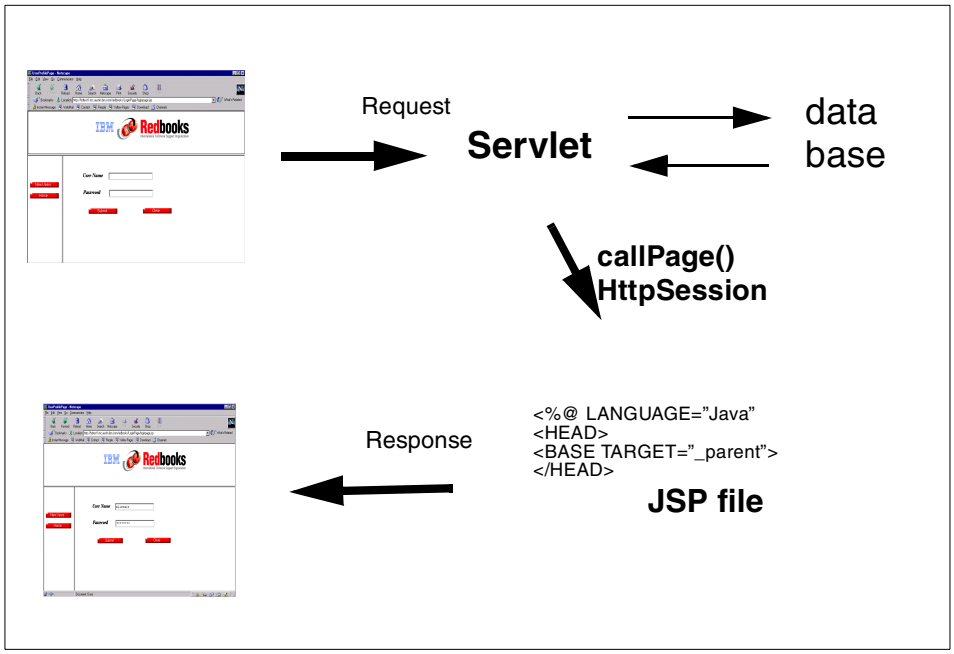

Figure 76. Request to servlet response via JSP approach

3.6.2.1 Invoking a servlet

Invoking a servlet from within a JSP is achieved by creating a URL that calls the servlet. It has the following general form:

```
http://hostname:port/servlet/servlet_name
```

We called our servlets from within an HTML form and called them as follows:

```
<FORM action=http://hostname/servlet/servlet_name METHOD=POST>
```

Figure 77 shows how we implemented this to process an HTML form.

```
 <FORM NAME="login" ACTION="http://bdtest1/servlet/AccessUserstuf"
METHOD=POST>
<INPUT TYPE=HIDDEN NAME="action" VALUE="login">
        <TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0 WIDTH=363>
. . .
```

Figure 77. Invoking a servlet that will process a form

In our servlet, called AccessUserstuf, we indicate what action is to be taken by adding a hidden parameter in the form. In Figure 77, this hidden parameter is called action and indicates that this form will be processed as a login. This action parameter indicates whether this is a login, update, registration, or other action. This code allows us to code only one servlet to perform various functions.

3.6.2.2 Generating dynamic HTML

The servlet, AccessUserstuf, with the hidden parameter set to login, verifies that the user is valid. If an error occurs during this validation, the servlet has to be able to communicate this back to the calling JSP. We chose to implement HttpSession objects to enable maintaining the connection information (see Section 3.5.2, “Maintaining state in HTTP” on page 81). The servlet puts the information into the HttpSession object and calls a JSP file. The JSP file then retrieves this information as follows:

- Create the session using HttpSession
- Use `getValue()` method to access information

Figure 78 illustrates this method

```
 </HEAD>
<% HttpSession session=request.getSession(true);
 if (session.getValue("error") !=null){
     %> <B ALIGN=CENTER><FONT SIZE="+2">ERROR:</FONT></B><%
out.println(session.
getValue("error"));
}
session.removeValue("error");
%>
```

Figure 78. Implementing the session object in a JSP

If the session.getValue() returns a null string, the error and an explanation is then printed to the client browser using out.println(). The error keyword is then removed from the session information.

In general, all the information that is passed to our JSP files from the servlet has been implemented using the HttpSession object. Figure 79 illustrates how this was achieved in our home page.

```
<% if (session.getValue("as400")!=null) {%>
<TD WIDTH=115 ><A HREF="/servlet/VBDList10?as400=item1">AS/400</A>
<% }%>
</TD>
```

Figure 79. Retrieving information from the session object

In Figure 79, we used the information contained in the HttpSession to dynamically generate a link to a new servlet. If the “value” returned by session.getValue() contains a null string, the link to /servlet/VBDList10?as400=item1 would not be displayed in the home page.

3.7 Accessing data through a Java bean

Java beans are Java components designed to be used on client systems. They are Java classes that conform to certain coding standards. They can be described in terms of their properties, methods, and events. Java beans may be packaged with a special descriptor class called a BeanInfo class and special property editor classes in a JAR file. Java beans may or may not be visual components. For more information, see the Java bean documentation at:

http://www.javasoft.com/beans/docs

A bean was chosen for one instance where the database was queried for information pertaining to previous orders. We create the following coding construct to access a bean for the list of orders:

- A bean is called from within a JSP by using the <BEAN. . . .></BEAN> tags. The JSP can then access the values contained within the bean using the request object. Figure 80 on page 104 illustrates how we accomplished this.

```
<bean
  name="orders"
  type="penguins.access.db2.Orders"
>
</bean>
```

Figure 80. Bean tag in the listordersPage.jsp

The JSP page, listordersPage.jsp, picks up the bean name of orders and finds the bean instance of penguins.access.db2. Orders from the Request object., the information contained in the bean, is accessible via the bean's public methods. See Section 3.7.0.1, "AccessUserstuf action=listorders" on page 104, where we discuss the Orders bean.

- We then use the beans to print the previous orders in a table. This is accomplished by iterating through all the orders contained in the bean with the following code:

```
for (Enumeration e=orders.orderIDs(); e.hasMoreElements();) {
  String key=(String) e.nextElement();
  out.println("<tr><td>"+key+"</td>");
  out.println("<td>"+orders.timestamp(key)+"</td>");
  out.println("<td>"+orders.productID(key)+"</td>");
}
```

The complete listing of listorderspage.jsp is provided in Appendix D.19, "Listorderspage.jsp" on page 196

3.7.0.1 AccessUserstuf action=listorders

The following sequence of tasks is performed when the servlet is invoked with the action parameter containing the value of listorders:

1. The listorders action retrieves the username and password from the Session object by using the getUsername and getPassword method:

   ```
   String username=getUsername(session);
   String password=getPassword(session);
   ```

2. We verify the username and password pair using the passwordsMatch method previously defined in Section 3.5.3.4, "AccessUserstuf action=login" on page 89. If successful, we proceed to the next step or redirect to an error page regarding the username and password authentication error.

3. The orders list for a user can be retrieved as an SQL query from the USERORDERS database table. It is a list of order IDs and associated timestamps and product IDs. Since we have no idea how long this list can grow for a user, we will use a Java bean called Orders to encapsulate this information and return the information via the bean through the Session object.

   Let us define what the Orders bean must be able to do now:

   - Add an order, which consists of the elements order_id, timestamp, and product_id
   - List all the orders
   - Retrieve the timestamp for a particular order_id
   - Retrieve the product_id for a particular order_id

   So the bean must have at least the following interface:

```
public class Orders {
   public void add(String order_id, String order_timestamp, String
product_id);
   public Enumeration orderIDs();
   public String productID(String orderID);
   public String timestamp(String orderID);
}
```

   We implement the list of order IDs as a vector and use hashtables to store the order timestamps and product IDs. The Java code for the Orders bean is presented in Figure 81 on page 106.

```
import java.util.*;
public class Orders {
    private Vector orderIDs=new Vector();
    private Hashtable timestamps=new Hashtable();
    private Hashtable productIDs=new Hashtable();
    public Orders() {
        super();
    }
    public void add(String order_id, String order_timestamp, String product_id) {
        orderIDs.addElement(order_id);
        timestamps.put(order_id, order_timestamp);
        productIDs.put(order_id, product_id);
    }
    public Enumeration orderIDs() {
        return orderIDs.elements();
    }
    public String productID(String orderID) {
        return (String) productIDs.get(orderID);
    }
    public String timestamp(String orderID) {
        return (String) timestamps.get(orderID);
    }
}
```

Figure 81. Java code for the Orders bean

Hence, for each resulting row of the query from the USERORDERS database table, we will add the appropriate order ID, timestamp, and product ID to an Orders bean. The code to perform this is as follows:

```
String sqlStmt="select order_id, order_timestamp, product_id from
userorders where username='"+username+"'";
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery(sqlStmt);
Orders orders=new Orders();
while (rs.next()) {
   orders.add(rs.getString("order_id"),
rs.getString("order_timestamp"), rs.getString("product_id));
}
```

4. Finally, we must set the Orders bean attribute in the Request object environment and pass control to a JSP page to indicate a successful retrieval of the Orders list. We do this with the setAttribute and callPage methods as follows:

```
req.setAttribute("orders", orders);
res.callPageA(successpage, req);
```

The JSP page redirected to the variable successpage will be able to pick up the bean name of “orders”, find the Java bean instance of type penguins.access.db2.Orders from the Request object environment, and will have access to the information via the bean’s public methods.

Chapter 4. Operational considerations

This chapter discusses the operational aspect of running a Linux Web server. The discussion here is aimed at keeping the product that constitutes the application that is running, which is categorized into:

- IBM HTTP server
- IBM WebSphere Application Server
- IBM Database 2
- Lotus Domino Enterprise Server

The discussion is divided into:

4.1 Facility administration

This section describe some administrative operations that we perform in maintaining the Web solution. Most of the startup and shutdown processes are performed automatically at boot or shutdown of the operating system, however, some knowledge of the manual operation is mandatory. These administration tasks are presented in the following sections:

4.1.1 IBM HTTP Server and WebSphere administration

Basically, the IBM HTTP Server is an Apache Web server. There are several excellent Web pages that discusses the administration of the HTTP server, such as:

- `http://www.ibm.com/software/Web servers`
- `http://www.apache.org`

This section provides some administration overview of the IBM HTTP Server and WebSphere in the following sections:

- Section 4.1.1.1, “Web server startup and shutdown” on page 108 shows the manual starting and stopping of the Web server.
- Section 4.1.1.2, “IBM HTTP Server administration server” on page 109 shows an overview of the IBM HTTP Server administration client.
- Section 4.1.1.3, “Servlet management with WebSphere” on page 113 provides an overview to the WebSphere servlet administration.

4.1.1.1 Web server startup and shutdown

WebSphere acts as a module plug-in for the IBM HTTP Server as discussed in Section 2.4.5.2, “Verifying WebSphere installation” on page 40. Therefore, the WebSphere functions are started and stopped together with the IBM HTTP Server.

The IBM HTTP Server starts a set of httpd processes, and WebSphere starts a set of Java processes. The manual commands for IBM HTTP Server are:

- Starting the HTTP server:

  ```
  /etc/rc.d/init.d/ibmhttpd start
  ```

- Restarting the HTTP server by sending the SIGHUP signal:

  ```
  /etc/rc.d/init.d/ibmhttpd restart
  ```

- Gracefully restarting the HTTP server:

  ```
  /etc/rc.d/init.d/ibmhttpd graceful
  ```

- Stopping the HTTP server:

  ```
  /etc/rc.d/init.d/ibmhttpd stop
  ```

The output from these commands is shown in Figure 82.

```
[root@bdtest1 init.d]# /etc/rc.d/init.d/ibmhttpd stop
/etc/rc.d/init.d/ibmhttpd stop: httpd stopped
[root@bdtest1 init.d]# /etc/rc.d/init.d/ibmhttpd start
/etc/rc.d/init.d/ibmhttpd start: httpd started
[root@bdtest1 init.d]# /etc/rc.d/init.d/ibmhttpd restart
/etc/rc.d/init.d/ibmhttpd restart: httpd restarted
[root@bdtest1 init.d]# /etc/rc.d/init.d/ibmhttpd graceful
/etc/rc.d/init.d/ibmhttpd graceful: httpd gracefully restarted
```

Figure 82. Output of the httpd control commands

If the server needs to be killed quickly, and the above commands fail for any reason, the server keeps the parent process ID in a file called httpd.pid. This file can be found in /opt/IBMHTTPServer/logs/. Sending the TERM signal to this process will stop the server ungracefully.

Once the required command has been issued, we can check to see whether it was successful by issuing the following command:

```
ps ax | grep httpd
```

Output from this command looks similar to Figure 83. Note that in our example, we have started seven httpd servers, and that it is sufficient to kill only the parent ID to stop all the processes.

```
[root@bdtest1 admin]# ps ax |grep httpd
12531 ?        S      0:00 /opt/IBMHTTPServer/bin/httpd
13165 ?        S      0:00 /opt/IBMHTTPServer/bin/httpd
13166 ?        S      0:00 /opt/IBMHTTPServer/bin/httpd
13167 ?        S      0:00 /opt/IBMHTTPServer/bin/httpd
13168 ?        S      0:00 /opt/IBMHTTPServer/bin/httpd
13169 ?        S      0:00 /opt/IBMHTTPServer/bin/httpd
13180 ?        S      0:00 /opt/IBMHTTPServer/bin/httpd
13256 pts/13   S      0:00 grep httpd
[root@bdtest1 admin]#
```

Figure 83. Output of the ps ax |grep httpd command

To ensure that the Java processes that is started by WebSphere is also terminated, issue the `killall java` command before stopping the HTTP server.

4.1.1.2 IBM HTTP Server administration server

This section provides the steps to install and log on to the IBM HTTP Server administration server. We do not go into any detail regarding the configuration of the Web server using this console. The steps below describe how we installed the server:

1. Install the `RPM` packages:

   ```
   rpm -i IBM_Admin_Server-1.3.6-1.i386.rpm
   rpm -i IBM_Admin_Server_Forms-1.3.6-1.i386.rpm
   ```

2. Change to the /opt/IBMHTTPServer/bin directory.
3. Create a group for the webadministrator

   ```
   groupadd -g 500 webadmin
   ```

4. Execute `the ./setupadm` command. This script is interactive, and the execution is given below showing our input:

```
************************************************************
Please supply a User ID to run the Administration Server
We will create the USERID using System Administration tools
************************************************************
[no default] -> httpuser
************************************************************
Please supply a GROUP NAME to run the Administration Server
We will create the Group using System Administration tools
************************************************************
[no default] -> webadmin
************************************************************
Please supply the Directory containing the files for
which a change in the permissions is necessary.
************************************************************
[default: /opt/IBMHTTPServer/conf] -> /opt/IBMHTTPServer/conf
************************************************************
We will flag 'SetUserID for Root' as well as
update the Group, Group access permissions and
Group execute permissions, for file:
        /opt/IBMHTTPServer/bin/admrestart
This interface is necessary for Administration Server requests
to restart manage Web servers
YES(default)-     ENTER 1
NO          -     ENTER 2
************************************************************
[default: YES  - 1] -> 1
************************************************************
You may use WildCards (i.e *.conf)
Please supply a File Name for permission changes
Default will change file permissions for ALL files in directory
************************************************************
[default: ALL FILES] ->
These are the file(s) and directory for which we will be changing
Group permissions:
-rw-rw-r--   1 root     webadmin     4137 Jul 29  1999 admin.conf
-rw-rw-r--   1 root     webadmin     4137 Jul 29  1999 admin.conf.default
-rw-rw-r--   1 root     webadmin     6246 Jul 29  1999 admin.msg
-rw-rw-r--   1 root     webadmin       24 May 18 15:19 admin.passwd
-rw-rw-r--   1 root     webadmin    31405 May 12 16:05 httpd.conf
-rw-rw-r--   1 root     webadmin    31028 May 12 16:05 httpd.conf.bak
-rw-rw-r--   1 root     webadmin    30989 Jul 29  1999 httpd.conf.default
-r--rw----   1 root     webadmin    46360 Jul 29  1999 httpd.conf.sample
-rw-rw-r--   1 root     webadmin    12441 Jul 29  1999 magic
-rw-rw-r--   1 root     webadmin    12441 Jul 29  1999 magic.default
-rw-rw-r--   1 root     webadmin     7350 Jul 29  1999 mime.types
-rw-rw-r--   1 root     webadmin     7350 Jul 29  1999 mime.types.default
drwxrwxr-x   2 root     webadmin     1024 May 18 09:55 /opt/IBMHTTPServer/conf
This is the file for which we will be adding 'set user ID' permission for Root:
-rwsr-x---   1 root     webadmin    46807 Jul 29  1999 /opt/IBMHTTPServer/bin/at
************************************************
CONTINUE - Perform Changes ENTER 1
QUIT -     No Changes      ENTER 2
************************************************
************************************************
[default: QUIT - 2] -> 1
Changes Completed
************************************************************
Configuration file: '/opt/IBMHTTPServer/conf/admin.conf'
will be saved as '/opt/IBMHTTPServer/conf/admin.conf.21:14:42_139'
Do you wish to update the Administration Server Configuration file
```

```
CONTINUE  enter 1
EXIT      enter 2
*****************************************************************
[default: QUIT - 2] -> 1
USER DONE
GRoup  DONE
Successfully updated configuration file
Old configuration file saved as '/opt/IBMHTTPServer/conf/admin.conf.21:14:42_139
```

5. Create a password for httpuser:

```
./htpasswd -b ../conf/admin.passwd httpuser webadmin
```

6. Start the administration server:

```
./adminctl start
```

7. Run a Web browser and point to port 8008 of our Web server. The initial administration screen is shown in Figure 84.

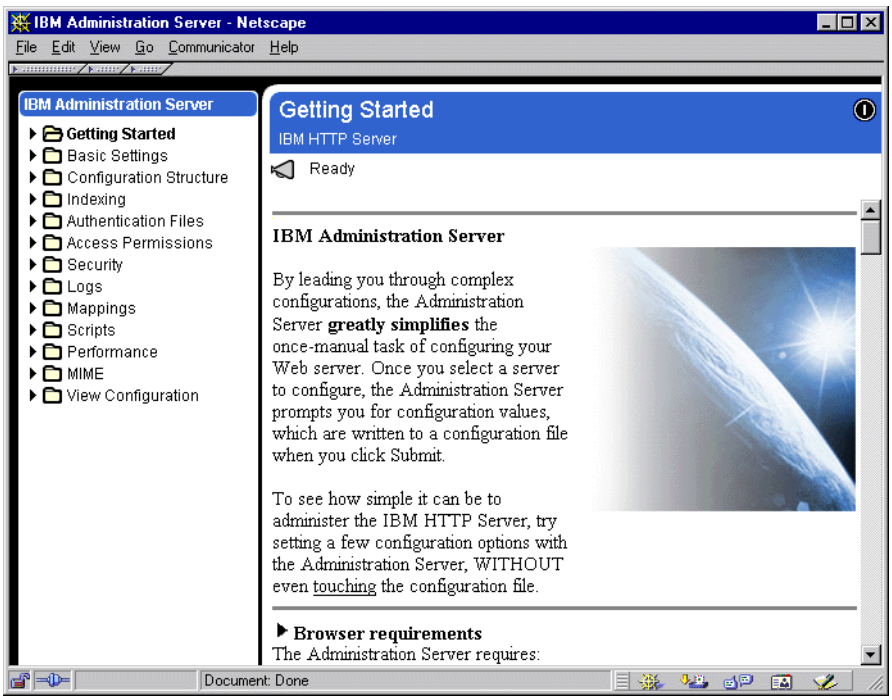

Figure 84. Administration Server Getting Started screen

To change the basic setting of the Web server, complete the following:

1. Click on **Basic Settings**
2. Click on **Core Setting**

This will display a screen similar to Figure 85.

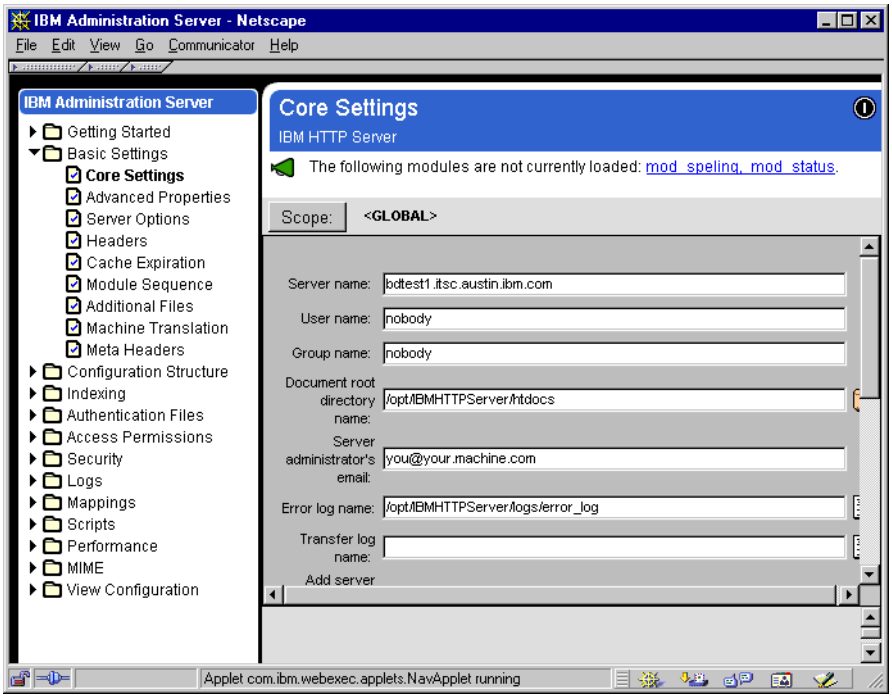

Figure 85. Basic settings for the Web server

The documentation for the Web server and the administration server is shown in Figure 86 on page 113 and can be accessed from the Web browser to the Web server in the manual subdirectory:

`http://bdtest2/manual/ibm/2tabContents.html`

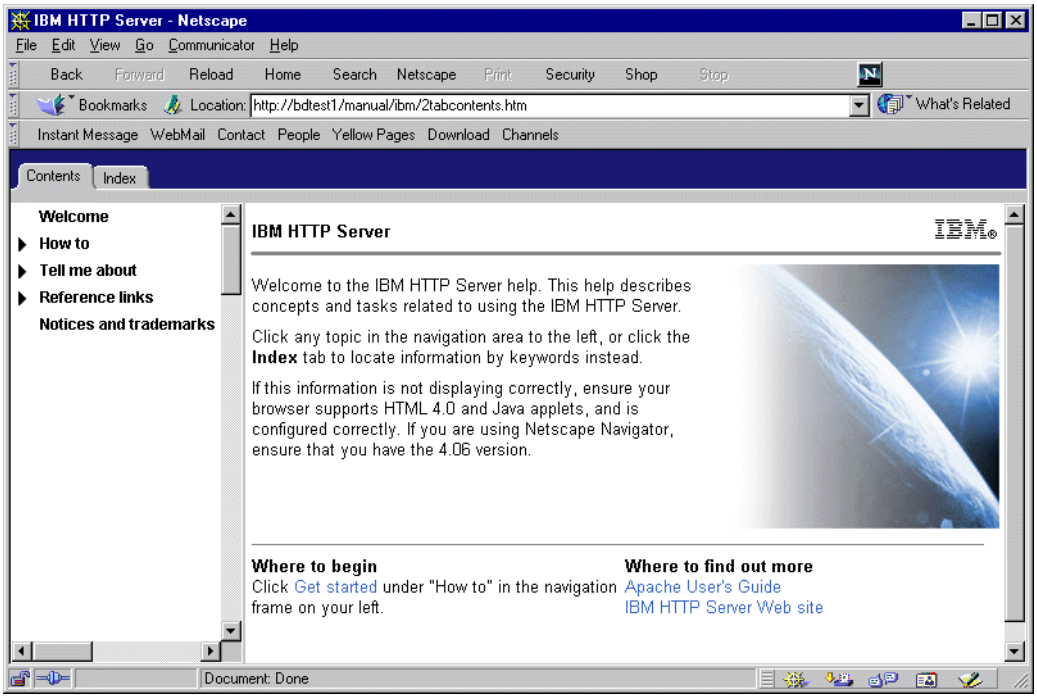

Figure 86. Documentation screen for IBM HTTP Server

4.1.1.3 Servlet management with WebSphere

Using the WebSphere administration in port 9527 of our Web server, we can manage the servlet that is in our environment. This section discusses adding, loading, unloading, and deleting servlets. A more complete coverage of WebSphere administration can be found at:

`http://www.software.ibm.com/Web servers/appserv/index.html`

Adding a servlet

Once the servlet has been written, it needs to be registered with WebSphere before use. This requires the following steps to be completed:

1. Put a copy of the compiled servlet class file in the servlet root directory. By default, the server looks for servlet class files in the servlet root directory, /opt/IBMWebAS/servlets.
2. If the servlet class belongs to a package, mirror the package structure as subdirectories under the servlets directory. For example, our servlet is called penguins.access.db2.AccessUserstuf; so, the class file AccessUserstuf.class is copied to the following directory structure: /opt/IBMWebAS/servlets/penguins/access/db2.

3. With WebSphere running, log into the administration tool in port 9527 using a Web browser. The administration screen looks like the window in Figure 87.

Figure 87. Administration login screen for WebSphere

4. Log in using admin as the username and password. We are now presented with the WebSphere introduction screen as shown in Figure 88 on page 115.

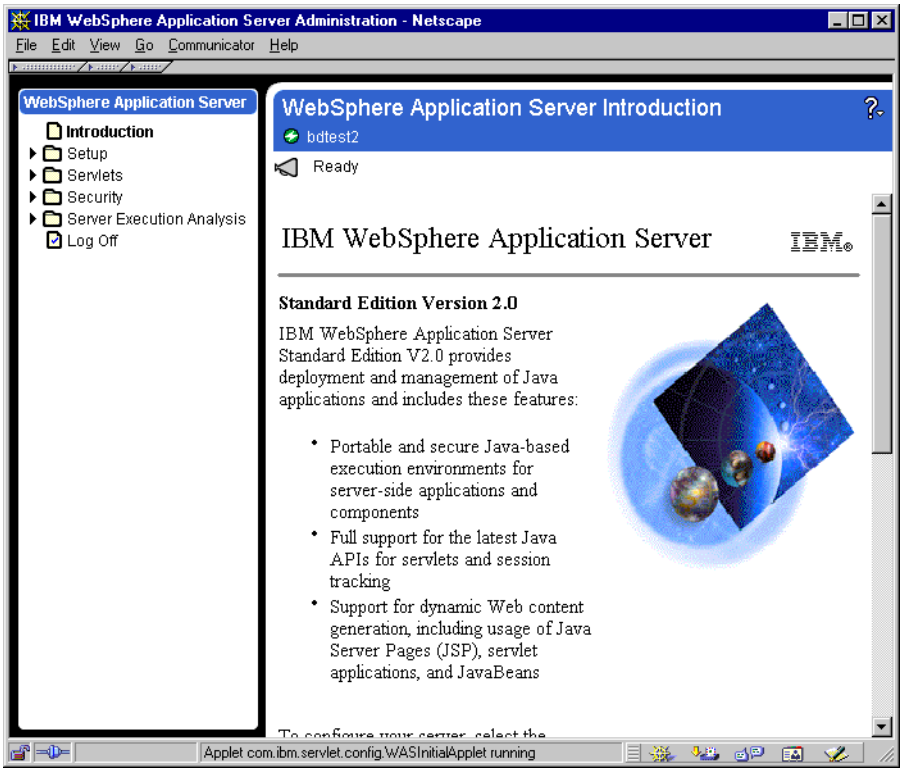

Figure 88. WebSphere introduction screen

5. Click on **Servlets** in the left frame.
6. Select **Configuration** and the Servlet Configuration window is shown as in Figure 89 on page 116.

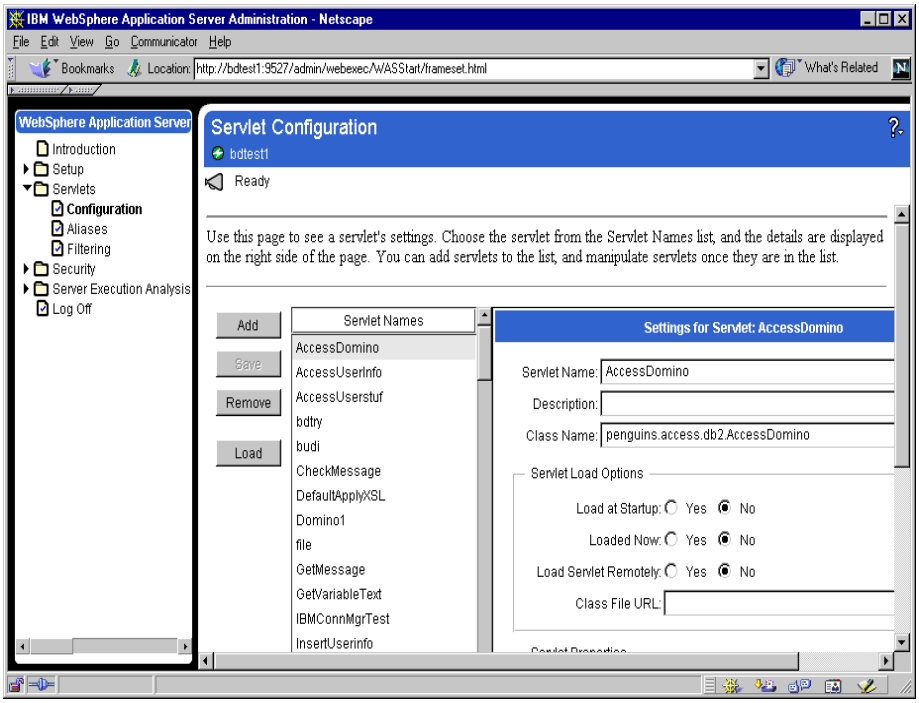

Figure 89. Servlet Configuration screen

7. Click on the **Add** button and type the servlet name in the popup box in Figure 90. It must be a unique name. It is the name that will be used to run the servlet from the Web browser.

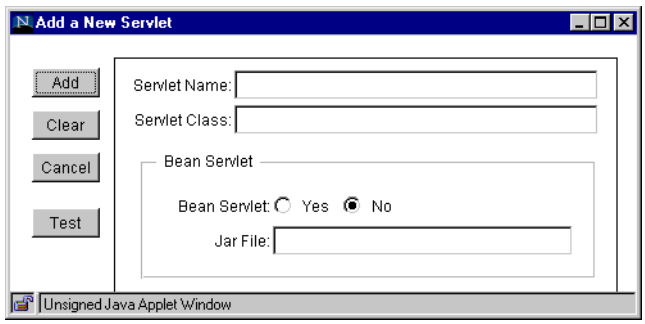

Figure 90. Adding A New Servlet

8. Fill the classname of the servlet. The classname is the full package name. For example, our full AccessUserstuf servlet name is called penguins.access.db2.AccessUserstuf.

9. If the servlet is a bean, fill in this information, select **yes**, and fill in the location of its jar file.
10. Click on **Test** to load the servlet and test it. If all goes well, it will be successful.
11. Click on **Add**. The servlet should be loaded automatically. The servlet is now registered and can be accessed.

Loading servlet

To load a servlet, complete the following steps:

1. Click on **Servlet** -> **Configuration** in the Servlet Configuration window as shown in Figure 89 on page 116
2. Select the servlet name from the Servlet Names column.
3. Click **Load**.

Unloading servlet

To unload a servlet, complete the following steps:

1. Click on **Servlet** -> **Configuration** in the Servlet Configuration window as shown in Figure 89 on page 116.
2. Select the servlet from the Servlet Names column.
3. Click **Unload**.

Deleting servlet

To delete a servlet, follow these steps:

1. Click on **Servlet** -> **Configuration** in the Servlet Configuration window as shown in Figure 89 on page 116.
2. Select the servlet from the Servlet Names column.
3. Click **Unload**.
4. Click **Yes** to confirm.

4.1.2 DB2 operation

This section discusses the manual startup and shutdown processes for a DB2 server.

4.1.2.1 Starting the DB2 instance

The DB2 instance is started with the command:

```
db2start
```

This command must be executed by the DB2 instance owner, in our case, db2inst1. See Figure 91 for an example of the execution of this command on our backend server, bdtest3.

```
[db2inst1@bdtest3 db2inst1]$ db2start
SQL1063N  DB2START processing was successful.
```

Figure 91. Starting the DB2 instance

4.1.2.2 Stopping the DB2 instance

The DB2 instance is stopped with the command:

```
db2stop
```

This command must be executed by the DB2 instance owner, in our case, db2inst1. See Figure 92 for an example of the execution of this command on our backend server, bdtest3.

```
[db2inst1@bdtest3 db2inst1]$ db2stop
SQL1064N  DB2STOP processing was successful.
```

Figure 92. Stopping the DB2 instance

In some cases, there may be active connections to a database in the DB2 instance. This DB2 database is considered to be active, and if the db2stop command is issued, the DB2 database manager for the DB2 instance will complain as shown in Figure 93.

```
[db2inst1@bdtest3 db2inst1]$ db2stop
SQL1025N  The database manager was not stopped because databases are
still active
```

Figure 93. Stopping the DB2 instance when a database is still active

We can ensure what connections to the database exist and decide if we will go ahead with halting the database instance. We use the `db2 list applications` command. A sample output of this command is shown in Figure 94 on page 119.

```
[db2inst1@bdtest3 db2inst1]$ db2 "list applications "

Auth Id  Application    Appl.      Application Id                DB       # of
         Name           Handle                                   Name     Agents
-------- -------------- ---------- ------------------------------ -------- -----
DB2INST1 java           1          0903F0DD.0706.000512165053     USERSTUF 1
DB2INST1 java           2          0903F0DD.0806.000512165055     USERSTUF 1
```

Figure 94. Listing connections to the DB2 instance

We can obtain even more information about these connections by using the `db2 list applications show detail` command. Figure 95 shows the result of running this command. The result is now showing more information for each database connection. In the case of a database instance shutdown, we would normally use this information to check on the status of the connections and determine if it is safe to forcefully disconnect these connections.

```
[db2inst1@bdtest3 db2inst1]$ db2 "list applications show detail"

Auth Id  Application Name     Appl.      Application Id                 Seq# Num
ber of  Coordinating Coordinator     Status                          Status Chang
e Time         DB Name  DB Path
                            Handle                                        Age
nts     Node Number  pid/thread
-------- -------------------- ---------- ------------------------------ ---- ---
------- ------------ --------------- ------------------------------ ------------
-------------- -------- --------------------
DB2INST1 java                 1          0903F0DD.0706.000512165053     0001 1
        0            16765           UOW Waiting                    Not Collecte
d              USERSTUF /home/db2inst1/db2inst1/NODE0000/SQL00002/
DB2INST1 java                 2          0903F0DD.0806.000512165055     0001 1
        0            16773           UOW Waiting                    Not Collecte
d              USERSTUF /home/db2inst1/db2inst1/NODE0000/SQL00002/
```

Figure 95. Listing connections, in detail, to the DB2 instance

Should we still choose to stop the database without waiting for all the connections to terminate, we have to use the command:

```
db2stop force
```

The output of this command, if successful, is exactly the same as that shown for the `db2stop` command without any parameters shown in Figure 92 on page 118.

4.1.3 Domino operation

This section discusses several features that are available to administer Domino server tasks. The topics that are covered here are:

- Section 4.1.3.1, “Domino console” on page 120 discusses the usage of the Domino console.
- Section 4.1.3.2, “Starting and stopping the Domino server” on page 122 shows the process of starting and stopping Domino, including cleaning up Domino resources after an abnormal termination.

4.1.3.1 Domino console

The Domino console is an important tool if we want to monitor our Domino server. We can access the console in several ways:

- Domino Administrator client

  As we described in Section 2.4.2.3, “Connecting a Domino client” on page 25, we can use the Domino Administrator client to monitor the Domino server. Currently, the Domino Administrator client is not available for Linux, forcing us to use the client on other platform, such as Windows NT.

  To get a console from the Domino Administrator client:

  a. Select the **Server** tab.

  b. Select the **Status** tab.

  c. Click on the **Live** icon.

  Figure 96 on page 121 shows a screenshot of the Domino Administrator console.

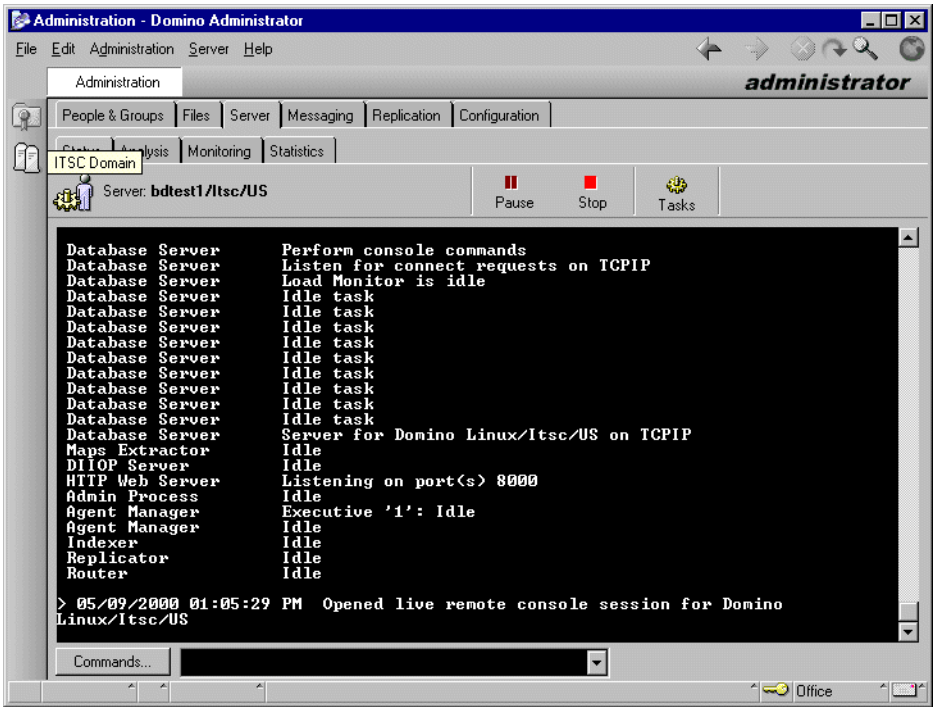

Figure 96. Domino Administrator console

- Domino character console

  Domino character console is a tool that provides a way to access the server console from the command line. We can telnet into our Domino server and invoke the character console using the `cconsole` command.

  To run the Domino character console for the first time, we complete the following tasks:

  a. Change the directory to our Domino data directory.

  b. Copy the server administrator ID file to the Domino data directory.

  c. Invoke the cconsole program:

     ```
     /opt/lotus/bin/cconsole -i -f /local/notesdata/user.id
     ```

     where `/opt/lotus/bin` is the Domino installation path, and `user.id` is the userid file for Domino server administrator.

  d. For a live console, issue the following command:

     ```
     live on
     ```

  e. To stop the console:

```
done
```

Figure 97 shows a screenshot of the Domino character console.

```
[notes@bdtest1 notesdata1]$ /opt/lotus/bin/cconsole -i -f /local/notesdata1/user
.id
Domino Character Console v0.0
Warning:  You are remotely connected to host bdtest1.itsc.austin.ibm.com.
If you have not taken precautions to secure this connection,
passwords will be exposed over the network.

Enter your password:
05/09/2000 01:20:56 PM  Initiating cconsole on bdtest1.itsc.austin.ibm.com
> show dbs


        Database Name            Refs Mod  FDs LockWaits/AvgWait #Waiters MaxWai
ters
/local/notesdata1/domlog.nsf        1   N   1        0       0        0       0
/local/notesdata1/mail.box          1   N   1        0       0        0       0
/local/notesdata1/names.nsf        46   N  12        0       0        0       0
/local/notesdata1/log.nsf           1   Y   1        0       0        0       0
>.
```

Figure 97. Domino character console

4.1.3.2 Starting and stopping the Domino server

The Domino startup process using a command line is as follows:

1. Log on with userID notes.
2. Change to the Domino data directory. We have the data directory in /local/notesdata.
3. Execute the command:

   `/opt/lotus/bin/server`

   where the `/opt/lotus` is the Domino server install path.

The server console is started, and it displays the startup status and other Domino messages.

To stop the Domino server from the server console, issue the `quit` command. The Domino server will shutdown after 10 seconds.

In order to be able to start Domino automatically, we develop a start_domino.sh script. The complete listing of the script is provided in Appendix E.1, “start_domino.sh” on page 209. The script performs the following sequence of tasks:

1. Ensure log on using the notes userid.

2. Ensure that the Domino server is not running.
3. Check and remove the IPC shared memory that is owned by userID notes.
4. Start the Domino server.

Having created the start_domino.sh script, we can start the Domino server automatically. We add an entry in the /etc/inittab file as follows:

```
notes:2:once:/bin/su - notes -c /home/notes/bin/start_domino.sh
```

If Domino exits abnormally, the Domino Inter-Process Communication (IPC) resources need to be cleaned up before restarting Domino. In order to clean up Domino after a crash, perform the following steps:

1. Kill all Domino processes of the server that failed from the userID notes.
2. Remove the associated Inter-Process Communication (IPC) resources.
3. List the corrupted shared memory and semaphores using the `ipcs` command. The `-m` option shows the shared memories, and the `-s` option shows the semaphores.
4. Remove any shared memory and semaphores. Use `ipcrm shm $id` to remove shared memory, and use `ipcrm sem $id` to remove semaphores.
5. Remove the ~notes.lck file from the Domino data directory.

The cleanup_domino script that performs the above steps is provided in Appendix E.2, “cleanup_domino.sh” on page 210.

4.2 Facility monitoring

This section provides an overview of monitoring the processes for the components of the Web solution the we built. Various other methods may be employed to ensure the availability and performance of the system.

4.2.1 Monitoring WebSphere

The loaded servlets can be viewed through the Server Execution Analysis screens as follows:

1. Log on to the WebSphere administration to get the WebSphere Introduction window as shown in Figure 88 on page 115.
2. Select **Server Execution Analysis**.
3. Select **Monitors**.
4. Select **Loaded Servlets** to display the monitor window as shown in Figure 98 on page 124.

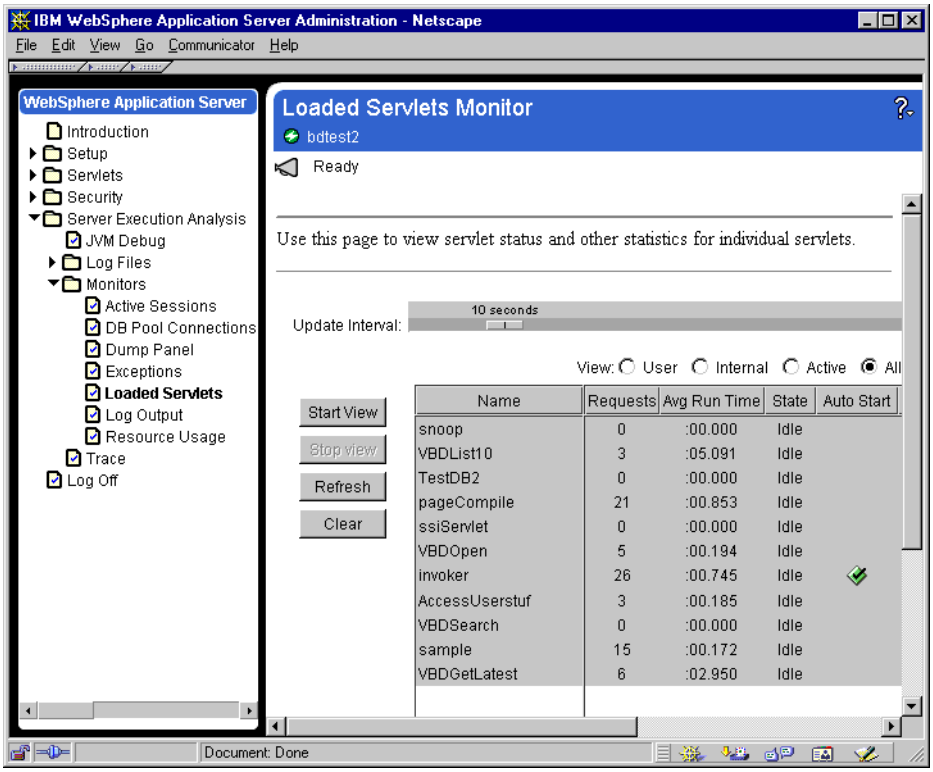

Figure 98. Loaded Servlets Monitor screen

All log files, debug information, resource usage, and database details can be viewed from this window by selecting the corresponding menu item.

4.2.2 DB2 server monitoring

There are two main monitoring tools included with DB2. These are:

- DB2 Performance Monitor

  The Performance Monitor provides information about the state of the DB2 Universal Database and the data that it controls. It is a graphical utility that can be customized for our database environment. We can define thresholds or zones that trigger warnings or alarms when the values being collected by the Performance Monitor are not within acceptable ranges.

  We use the tool when we need to monitor an existing problem or when we want to observe the performance of our system. It allows us to take a snapshot of database activity and performance data at a point in time.

These snapshots are used for comparison over time. Each point on the graph represents a data value.

This information can help us to identify and analyze potential problems or identify exception conditions that are based on thresholds that we set. We use the performance tool if we need to know the performance of the database manager and its database applications at a single point in time and to look at trends over time. We can also use it to get a visual overview of what elements are in a state of alarm. This helps us identify which parameters may need tuning. We can then look closely at the parameters that have been set for that element and change them to improve performance.

- DB2 event monitors

  In contrast to taking a point in time snapshot with the DB2 Performance Monitor, a DB2 event monitor collects information on database activities over a period of time. This collected information provides a good summary of the activity for a particular database event, for example, a database connection or an SQL statement.

  Event monitoring records the state of the database at the time that the specific events occur. It allows us to obtain a trace of the activity on the database. Event monitor records are stored and then analyzed after the data has been captured. We use the event monitor when we need to know how long a transaction took or, for example, how much CPU an SQL statement used. We then use the Event Analyzer to read the data recorded from the event monitor.

Apart from these two tools, we should also regularly back up and analyze the DB2 diagnostic log file, db2diag.log. See Section 4.4.2, “DB2 diagnostic log file” on page 136 for more information on this. Examine this log file for signs of potential problems, for example, running out of disk space.

4.2.3 Domino monitoring

Domino includes a number of powerful monitoring features that can be fairly comprehensive mechanisms for the automatic recording of statistics and automatic notification of events. This section explains the following monitoring features:

- Domino server monitor
- Domino statistic display
- Domino monitors
- Notes system diagnostic

4.2.3.1 Domino server monitor

Domino Administration server includes a server monitor that provides a visual representation of the status and availability of the Domino server, tasks, real-time system statistics, and status indicators.

Figure 99 shows a screenshot of the Domino server monitor. To invoke the Domino server monitor, perform the following tasks:

1. Run the Lotus Domino Administration client.
2. Click on **Server**.
3. Click on **Monitoring**.
4. Click on the **Start** icon.

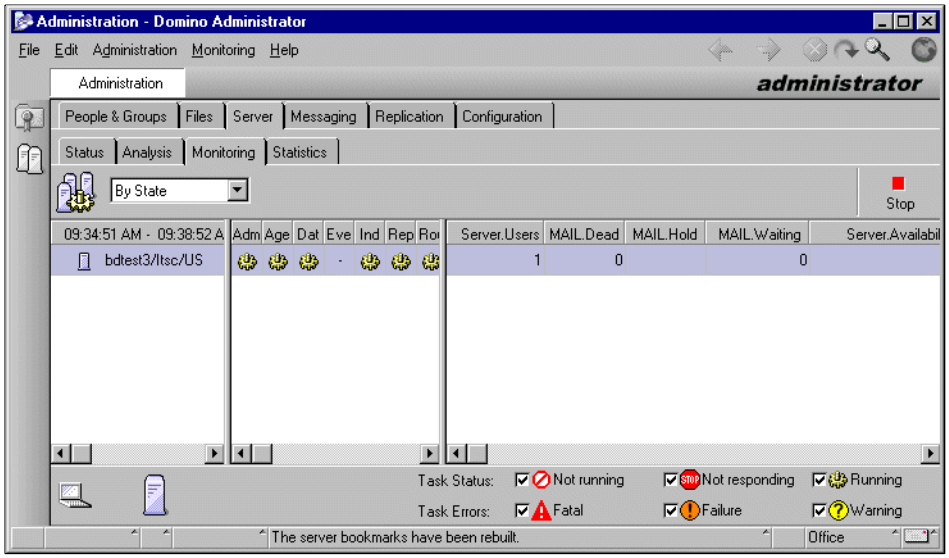

Figure 99. Domino server monitor

4.2.3.2 Domino statistic

We can get a snapshot of the Domino statistics from the Domino console. We need the Domino Administrator client or the character console as described in Section 4.1.3.1, “Domino console” on page 120. From the Domino console prompt, we can issue:

```
show statistic
```

It will display the statistics of the Domino server. Figure 100 on page 127 illustrates the output of this command.

```
 Server.Task.DB = Database Server: Idle task: [05/09/2000 13:26:20 CDT]
  Server.Task.DB = Database Server: Idle task: [05/09/2000 13:26:20 CDT]
  Server.Task.DB = Database Server: Idle task: [05/09/2000 13:26:20 CDT]
  Server.Task.DB = Database Server: Idle task: [05/09/2000 13:26:20 CDT]
  Server.Task.DB = Database Server: Idle task: [05/09/2000 13:26:20 CDT]
  Server.Task.DB = Database Server: Server for Domino Linux/Itsc/US on TCPIP: [05/09/20
CDT]
  Server.Time.Start = 05/09/2000 10:12:43 CDT
  Server.Title =
  Server.Trans.PerMinute = 0
  Server.Trans.PerMinute.Peak = 241
  Server.Trans.PerMinute.Peak.Time = 05/09/2000 10:27:32 CDT
  Server.Trans.Total = 613
  Server.Users = 1
  Server.Users.1MinPeak = 1
  Server.Users.1MinPeakTime = 05/09/2000 10:28:32 CDT
  Server.Users.5MinPeak = 3
  Server.Users.5MinPeakTime = 05/09/2000 10:28:32 CDT
  Server.Users.Peak = 4
  Server.Users.Peak.Time = 05/09/2000 10:27:10 CDT
  Server.Version.Notes = Release 5.0.3
  Server.Version.OS = Linux 2.2.14-5.0 #1 Tue Mar 7 2
  Stats.Time.Current = 05/09/2000 13:27:06 CDT
  Stats.Time.Start = 05/09/2000 10:12:41 CDT
>
```

Figure 100. Show statistic command output from Domino character console

4.2.3.3 Statistics and events

The statistic and events facility allows us to configure, create, collect, and report information about the Domino server. The statistic and event display is shown in the Domino Administrator client by:

1. Selecting the **Configuration** tab
2. Expanding the **Statistic & Event** category.

The statistic parameters for the Domino server are shown in Figure 101 on page 128.

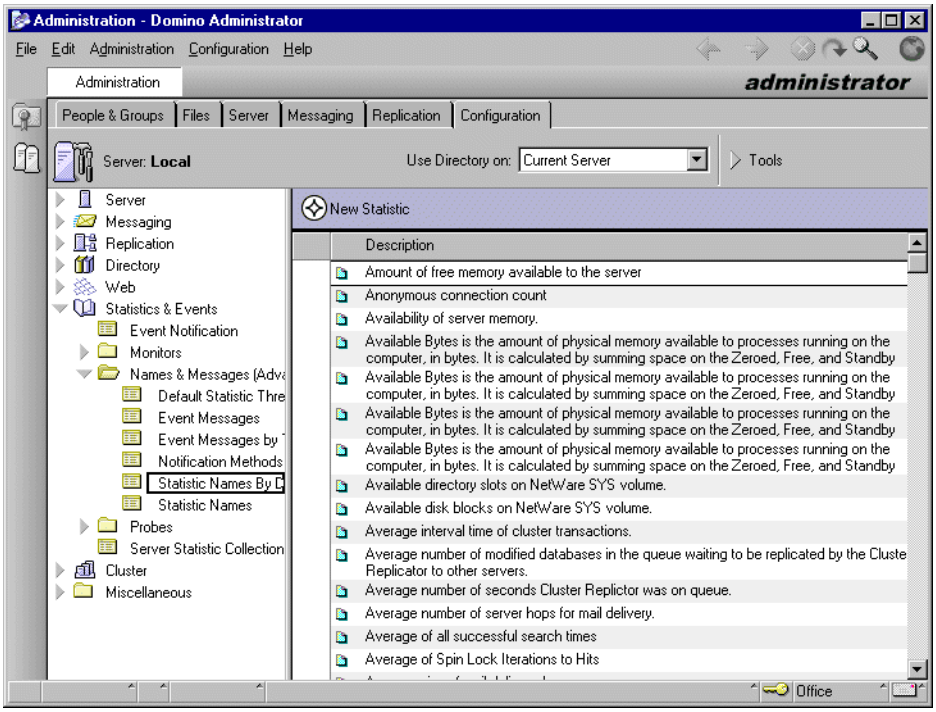

Figure 101. Statistics and events

Statistics and events are controlled by two server tasks, event and collect. By default, statistics are collected every hour. The information captured is stored in the following Domino databases:

- statrep.nsf, where statistical information is logged
- events4.nsf, where event handling and statistic monitoring is configured.

From the statistic and events, we can also create monitors to track server resources and network and system activity. Each monitor has an associated threshold. When the threshold is reached, the monitor then generates an event.

The monitor tools are available under the Statistics & Events category. Figure 102 on page 129 displays a screenshot of the monitor tool for file activity.

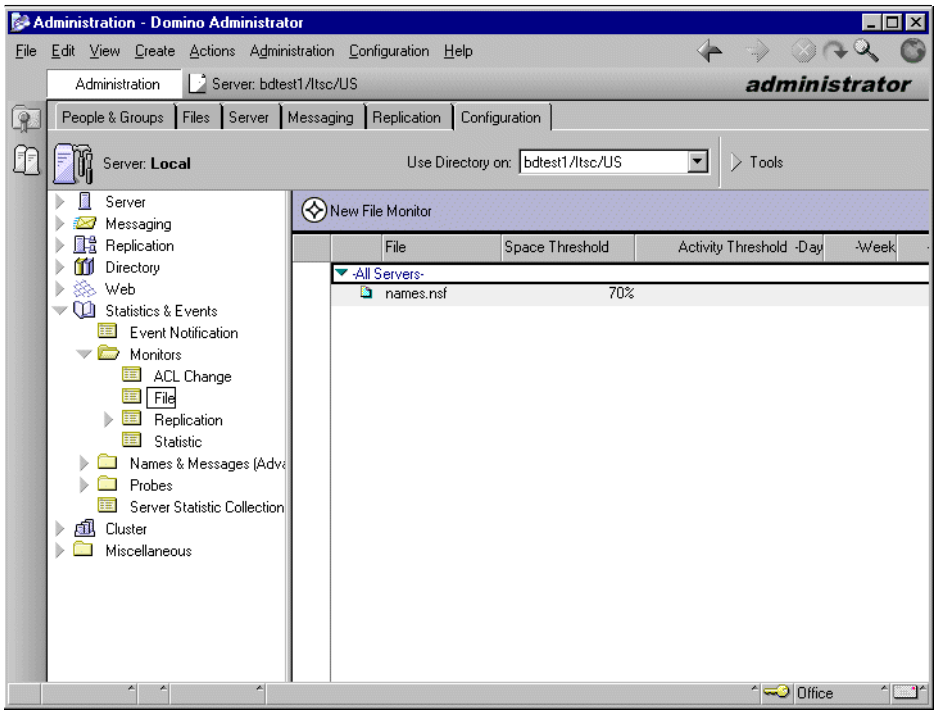

Figure 102. Monitor tool for Domino Administrator client

4.2.3.4 Notes System Diagnostics tool

The Notes System Diagnostic (NSD) tool is developed by Iris, and it is available only on the UNIX platform. NSD is included in the Domino server installation.

To run NSD, issue the `nsd` command and use the `-help` options for explanation of the available parameters. NSD will create the file nsd.log.<date and time>, where the date and time is the date and time the program was invoked. The NSD log file contains useful diagnostic information. Use any text editor to view this file.

4.3 Routine maintenance

This section discusses the routine maintenance tasks that need to be performed periodically to ensure the availability and performance of the system.

4.3.1 IBM HTTP Server maintenance

The following tasks are recommended to be performed for the IBM HTTP Server.

- Backup of the ServerRoot directory, in our case, /opt/IBMHTTPServer. This directory contains all the configuration files and any Web pages found in htdocs.

  You can use any backup method you choose, such as tar. The following `tar` command, when issued from the /opt/IBMHTTPServer directory, will copy all the IBM HTTP Server files to a backup file, HTTPServer.tar:

  ```
  tar cvf /HTTPServer.tar
  ```

- The log files contained in the /opt/IBMHTTPServer/logs directory are an important source of information on the server; however, they tend to grow quickly and need to be pruned.

  Issue the following commands from the /opt/IBMHTTPServer/logs directory to prune them:

  ```
  mv access_log access_log.old
  kill -1 `cat httpd.pid`
  ```

  The kill is necessary; otherwise, httpd will keep writing to the previous log file.

4.3.2 DB2 routine maintenance

This section illustrates some of the required tasks to ensure the availability and performance of the DB2 server. These tasks are discussed in:

- Section 4.3.2.1, “Backing up the DB2 database” on page 130, which explains an overview of DB2 backup process.
- Section 4.3.2.2, “DB2 database log files” on page 131, which discusses the DB2 log files.
- Section 4.3.2.3, “Table reorganization” on page 132, which illustrates the table reorganization procedure.
- Section 4.3.2.4, “DB2 statistic” on page 132, which discusses updating DB2 statistics.

4.3.2.1 Backing up the DB2 database

A DB2 database must be backed up using the `db2 backup database` command. Using a file system backup will not result in a database backup with database integrity intact.

The DB2 backup command is fairly complex. An online or off-line database backup can be taken with several types of backup media (disk, tape, ADSM), using different buffer sizes, etc. It is outside of the scope of this section to discuss all these various options. Refer to the *IBM DB2 Universal Database Command Reference*, SC09-2844, *IBM DB2 Universal Database Administration Guide Volume 1*, SC09-2839, and *IBM DB2 Universal Database Administration Guide Volume 2*, SC09-2840, for more comprehensive information.

We present the format of a simple db2 off-line backup database to disk, backing up our database USERSTUF to a directory on our server, /backup, as follows:

```
db2 backup database userstuf to /backup
```

4.3.2.2 DB2 database log files

All databases have logs associated with them. These logs keep records of database changes. In certain situations, depending on the configuration of the DB2 database manger on the backend server, we may need to back up certain logs files (inactive and archived log files) and delete them from their default locations.

Deleting active log files

Never delete active log files. These files are used by the DB2 database manager to keep track of current database transactions. Deleting these log files would result in a database left in an inconsistent state.

In our current DB2 instance configuration, the DB2 instance uses default circular logging. Hence, we have no requirement to back up and delete log files.

It is outside of the scope of this section to discuss, in detail, database logs. Refer to the *IBM DB2 Universal Database Administration Guide Volume 1*, SC09-2839, and *IBM DB2 Universal Database Administration Guide Volume 2*, SC09-2840, for more comprehensive information about the various types of log archiving (circular, log retain), different types of logs (active, online archived, off-line archived), log file locations, and so on.

It is also important to prune the content of the DB2 diagnostic log. Discussion on the DB2 diagnostic log is provided in Section 4.4.2, “DB2 diagnostic log file” on page 136.

4.3.2.3 Table reorganization

Every now and then, DB2 tables need to be reorganize to improve their structure, and, therefore, also their performance. The DB2 table is reorganized using the following procedure:

1. Invoke the DB2 command line using the `db2` command from the instance owner, db2inst1, in our case.
2. Connect to the database:

```
db2> connect to userstuf
```

3. Run reorganization for each table that you are interested in:

```
db2> reorg table userverify
db2> reorg table userinfo
db2> reorg table userprofile
db2> reorg table userorders
```

See *IBM DB2 Universal Database Command Reference*, SC09-2844, for more information on the commands used.

4.3.2.4 DB2 statistic

The DB2 statistic needs to be updated regularly to optimize the query running against the database. Run the DB2 statistic update against each table and database using the following procedure:

1. Invoke the DB2 command line using the `db2` command from the instance owner, db2inst1, in our case.
2. Connect to the database:

```
db2> connect to userstuf
```

3. Update the statistic of each table that you are interested in:

```
db2> runstats on table userverify
db2> runstats on table userinfo
db2> runstats on table userprofile
db2> runstats on table userorders
```

See *IBM DB2 Universal Database Command Reference*, SC09-2844, for more information on the commands used.

4.3.3 Routine Domino maintenance

This section provides an overview of the tasks that are needed to be performed for the Domino server.

4.3.3.1 Back up Domino file systems

The backup system is very important in daily operation because you never know when you might get into trouble.

In Domino, the minimum backup that we have to do is to back up all the Domino server data files, including databases, template files, and the notes.ini file and ID files, during a daily backup, and other files perhaps weekly.

Assuming that our Domino server was running perfectly, we now plan to make a backup of our Domino server. We need to back up the following directories and files:

/opt/lotus	This is the Domino software directory. We need to back it up only before we upgrade or install the new Domino server. If resources allow, it is recommended that we include this directory in our regular backups.
/local/notesdata	This is the Domino data directory. In this directory we will find the database, the templates, the mail files, and others. This directory may require one or more daily backups if our operations produce a lot of data. If we want to back up this directory, we must shutdown the Domino server first.
notes.ini, *.id	Keep a copy of these files on a removable media storage in a secure place. These files are present in the Domino data directory (/local/notesdata). The ID files include: cert.id, server.id, user.id.

We can use many tools to make a backup in Linux, such as tar, cpio, and so on, and we can choose the tool that will best suit our environment. But, remember that the goal of all backups is to minimize server downtime; so, in such cases, choose the solution that you are very familiar with.

4.3.3.2 Other maintenance tasks

Routine maintenance is needed in our daily operation to minimize the risk of potential problems and ensure that our Domino server continues to function reliably.

The redbook, *A Roadmap for Deploying Domino in The Organization*, SG24-5617, contains a useful checklist of routine maintenance that we can perform. Please refer to this book for more information.

4.4 Troubleshooting overview

This section provides an overview on troubleshooting some of the basic components of our Web server. The discussion is separated into:

- Section 4.4.1, “IBM HTTP Server and WebSphere log files” on page 134
- Section 4.4.2, “DB2 diagnostic log file” on page 136

4.4.1 IBM HTTP Server and WebSphere log files

The following log files are very useful in finding a specific problem for problem determination and resolution in the Web server. WebSphere logs a large amount of information to its log files. If something is not functioning correctly, these logs should be looked at. We briefly discuss some of these log files. All log files can be found in /opt/IBMWebAS/logs. In this directory, you will typically find one or more of these:

- jvm_stderr.log

  The JVM standard error log is probably the most useful log of all the WebSphere logs. In this log, the standard error output from the main WebSphere Java process appears.

- jvm_stdout.log

  The JVM standard output log is the second most useful one in WebSphere. The standard output log contains all of the output written by Java applications to standard output using the methods of the static java.lang.System.out object. The format of this log is application dependent. Use this log to inspect debugging and informational messages from your Java applications writing to System.out. This log is always enabled.

- apache.log.INFORM.*

  The IBM HTTP Server information log contains informational and error messages generated by the IBM HTTP Server included with WebSphere. Each time the server is re-initialized, a new log is generated with a different number used as the last component of the file name. The major part of each of these logs is taken up with messages showing the properties with which WebSphere is initialized. This is useful in confirming the parameter values that a particular invocation of WebSphere is using. Each line contains the following components:

  - An ID number specific to the particular invocation of WebSphere. This is the same number used as a suffix to the log file name.
  - A timestamp.

- The word Property.
- The property name and value separated by an = sign.

- apache.log.ERROR.* and apache.log.TRACE.*

  The IBM HTTP Server error log contains error messages generated by the IBM HTTP server included with WebSphere. The format and content is similar to that of the information log, but the messages are restricted to those describing error conditions. These messages are also included in the information log. The HTTP Server trace log contains additional trace information when tracing is activated.

- WebSphere_trace.log

  The WebSphere trace log contains the output of the different tracing functions. Each log entry is the result of an event picked up by one of the tracing monitors and can contain any of the following six components:

  - A timestamp
  - The name of the tracer that is the source of the event
  - The ID of the thread that caused the event
  - The severity level of the tracer message
  - Any exception generated by the event
  - The message text

- servlet/adminservice/access_log
- servlet/adminservice/error_log
- servlet/adminservice/event_log

  The servlet admin service error log contains messages that relate to events and errors that occur during the execution of the WebSphere administration service. This includes errors that may occur while running any tracing programs.

- servlet/servletservice/access_log
- servlet/servletservice/error_log
- servlet/servletservice/event_log

  The servlet service error log contains error information generated by the WebSphere servlet service in attempting to run servlets. It includes both errors pertaining to particular servlets and information pertaining to the servlet service itself. System.out and System.err output that is produced by servlets is placed in the JVM standard output and standard error logs, respectively.

This log can be used to help diagnose both the cause of servlet failures and failures of the servlet service. This log will record any exceptions thrown by WebSphere in trying to load or configure servlets; so, it is a good place to start diagnosing these sorts of problems. It will also contain information relevant to determining the causes of servlet service start failures.

The servlet service event log allows you to see when servlets are loaded, when they are called, and with what parameters they are called. This is useful in trying to diagnose why a particular servlet is failing under certain conditions but not others since it shows you how it is being called. You may want to add some debug writes to your servlet that include a timestamp so that you can match the debug output that your servlet is producing in standard output or standard error to the messages in the event log that already have a timestamp.

- The WebSphere engine tracing log shows detailed information about the activity on the server. This log is stored in the directory that you specified when activating the tracing function from the WebSphere administration. Each action that the server takes is recorded. The format of this log is:
  - A timestamp
  - A message

  This log is good for diagnosing CLASSPATH problems since it provides detailed information on which directories and JAR files have been found and verified in the classpath.

4.4.2 DB2 diagnostic log file

DB2 keeps a diagnostics log file, called db2diag.log. This file is located in the instance home directory under subdirectory sqllib/db2dump directory. In our scenario, on our backend server bdtest3, it would be located in /home/db2inst1/sqllib/db2dump. This file is appended to, or created and appended to if it does not already exist, with diagnostic information.

The amount of diagnostic information is determined by the DIAGLEVEL parameter found in the DB2 database manager configuration. Log in as the DB2 instance owner and access the database manager configuration using the command:

```
db2 get dbm cfg
```

Figure 103 on page 137 shows the result of running this command on our backend server, bdtest3, while logged in as the DB2 instance owner, db2inst1. The DIAGLEVEL information line has been highlighted in bold.

```
[db2inst1@bdtest3 db2inst1]$ db2 get dbm cfg | more

          Database Manager Configuration

     Node type = Database Server with local and remote clients

 Database manager configuration release level            = 0x0900

 CPU speed (millisec/instruction)             (CPUSPEED) = 1.637460e-06

 Max number of concurrently active databases     (NUMDB) = 8
 Data Links support                          (DATALINKS) = NO
 Federated Database System Support           (FEDERATED) = NO
 Transaction processor monitor name        (TP_MON_NAME) =

 Default charge-back account           (DFT_ACCOUNT_STR) =

 Java Development Kit 1.1 installation path (JDK11_PATH) =

 Diagnostic error capture level              (DIAGLEVEL) = 3
 Diagnostic data directory path               (DIAGPATH) = /home/db2inst1/sqllib
/db2dump

--More--
```

Figure 103. DIAGLEVEL parameter in the DB2 database manager configuration

The db2diag.log file can grow quickly if the DIAGLEVEL is set to 4 or if many errors are encountered. Hence, ensure the size of this file is manageable by backing up and deleting this file at regular intervals (the file will be created and appended to when necessary by the DB2 database manger process). Backups can be deleted once you are certain the diagnostic information provided in the files is no longer useful.

Chapter 5. Extending the case study

This chapter discuss some possible extension of our implementation. We explain these possibilities without being able to perform them ourselves. The discussion is divided into:

- Section 5.1, “Utilizing advanced features” on page 139, which discusses the available features that we do not use because of the complexity in implementing them.
- Section 5.2, “Product limitations in Linux” on page 142, which discusses the features that are not available in the Linux version of the software that we use.

5.1 Utilizing advanced features

This section discusses the advanced features in some of the software products that are not exploited because the lack of implementation time. However, these feature have already been covered in detail in other redbooks, such as *Linux for WebSphere and DB2 Servers*, SG24-5850, and *Patterns for e-business: User-to-Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition*, SG24-5864. Refer to these redbooks for more information.

5.1.1 WebSphere

The following advanced features of WebSphere are available but are not utilized in our project:

- Section 5.1.1.1, “DB connection pooling” on page 139
- Section 5.1.1.2, “User profile tracking” on page 140

5.1.1.1 DB connection pooling

Connecting and disconnecting from a data server causes the inefficient use of resources. WebSphere offers a database connection pooling facility that can be used by programers to manage connections to a database. This is an API that contains methods for:

- Establishing a pool of open connections to a given database
- Requesting a connection from the managed pool
- Returning the connection to the managed pool

This pool of connections for a given database is managed through the WebSphere administration facility.

The connection manager allows you to connect and disconnect to a data server more efficiently by pooling connections and reusing them. This reduces the overhead of connecting and disconnecting. The servlets use the connection pool as follows:

- When a user makes a request over the Web to a servlet, the servlet uses an existing connection from the pool, meaning the user request does not incur the overhead of a data server connection.
- When the request is satisfied, the servlet returns the connection to the connection manager pool for use by other servlets.

The user request, therefore, does not incur the overhead of a data server disconnection.

The connection manager also allows you to control the number of concurrent connections to a data server product. This is very useful if the data server license agreement limits you to a certain number of concurrent users.

There is a good introduction to connection manager in the WebSphere documentation under the IBMWebAS directory. In our bdtest1 system, it is at URL:

```
http://bdtest1/IBMWebAS/doc/whatis/iccmgr.html
```

5.1.1.2 User profile tracking

The WebSphere Application Server has an API, called UserProfile, that makes it easy to maintain persistent information about your Web site visitors. The UserProfile enables you to store the user's data in a database without coding any SQL.

The UserProfile contains the interface to store the user's personal data, such as name, postal and e-mail addresses, telephone numbers, and shopping cart information. For further information, see the UserProfile API Reference in the WebSphere documentation under the IBMWebAS directory. In our bdtest1 system, it is at URL:

```
http://bdtest1/IBMWebAS/doc/apidocs/Package-com.ibm.servlet.personalizatio
n.userprofile.html
```

5.1.2 IBM HTTP Server

The following advanced features of IBM HTTP Server are available but not utilized in our project:

- Section 5.1.2.1, “SSL” on page 141
- Section 5.1.2.2, “Caching and dynamic pages” on page 141

5.1.2.1 SSL

The IBM HTTP Server features include easy installation support for SSL secure connections.

Secure Sockets Layer (SSL) is a protocol that provides privacy and integrity between two applications using TCPIP. HTTP can use SSL for secure communications.

The data traveling between the client and server are encrypted using a secret key algorithm. A public-key algorithm is used for the exchange of the secret encryption key and for encrypting the digital signature in the servers certificate. With the server certificate, the client can verify the server's identity.

Once your server has a digital certificate, SSL-enabled browsers, such as 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. For further info on SSL, refer to:

- `http://home.netscape.com/eng/ssl3/`
- `http://www.ibm.com/software/webservers/httpservers/`

5.1.2.2 Caching and dynamic pages

By default, most Web browsers and proxy servers cache Web pages. In the case of JSP files, SHTML files, and servlets, these Web pages are dynamically generated and may change for each attempted access.

In order to prevent this caching, and thus access the newly generated dynamic data, we can add the following code in a Java Server Page or servlet to prevent Web browsers from caching JSP/SHTML files and servlets:

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

Use the following code to prevent a proxy server from sharing cached pages:

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

Both of these headers can be combined to prevent caching.

5.2 Product limitations in Linux

The level of WebSphere in Linux is lagging from the available version in other platforms (such as Windows NT or AIX). This poses a considerable restriction to our ability to utilize several of the new key features that really make the e-business development easier, such as the RequestDispatcher class. See Section 5.2.1, “The RequestDispatcher class” on page 142 for a more comprehensive discussion of this class.

In our project, we access the Lotus Domino database using CORBA. Lotus Domino server can also be accessed by JDBC. However, the Lotus Domino Driver for JDBC is only available for a limited number of platforms, and Linux is not among them. See Section 5.2.2, “JDBC connections for Domino” on page 143 for a more comprehensive discussion of this feature from Lotus.

5.2.1 The RequestDispatcher class

The basic use of a RequestDispatcher class is to pass on the current request to another program (servlet or JSP). A Request Dispatcher has two primary methods for including the response of another program or forwarding the request of the current program to another.

- Including the response from another program requires the use of the include method:

  ```
  aRequestDispatcher.include(request, response)
  ```

  We would use this if we want to call another servlet from our current servlet, then perform more processing in the current servlet after calling this other servlet

- Forwarding the request to another program requires the use of the forward method:

  ```
  aRequestDispatcher.forward(request, response)
  ```

  We would use this to forward the request on to another servlet, and then allow this new servlet to return the response back to the client directly.

Unfortunately for us, the RequestDispatcher class is only available in the Java JSDK 2.1 API for the javax.servlet class (supported and extended in WebSphere 3.02). In our servlets, to be run under WebSphere 2.03 (which only supports and extends JSDK 2.0), we were forced to use the deprecated callPage method found in package com.sun.server.http. This also explains the use of the HttpServiceResponse and HttpServiceRequest objects in our Java code in servlet AccessUserstuf.

Refer to the URL http://novocode.de/doc/servlet-essentials/chapter0.html for a good tutorial about the RequestDispatcher class. Refer to the URL http://java.sun.com for official specifications and documentation for the various Java API specifications.

5.2.2 JDBC connections for Domino

JDBC(TM) is an object interface that allows Java applications, applets, and agents to retrieve and manipulate data in database management systems using SQL. The interface allows a single application to connect to many different types of databases through a standard protocol.

The Lotus Domino Driver for JDBC implements the JDBC 1.0 specification. It can be for the Windows 95, Windows 98, and Windows NT 4.0 operating systems. This is a Type 2 driver, which means that it is comprised of a combination of Java classes and non-Java compiled code. It is currently not available for any other platform (including Linux).

The Lotus Domino Driver for JDBC allows Java programmers to access Domino/Notes databases using JDBC. Users can work with any JDBC enabled reporting tool to access Lotus Domino databases. Java programmers can use any Java Database Connectivity (JDBC) standard enabled application tool to access Lotus Domino databases as easily as any relational database. When running this JDBC driver with Domino R5, Java programmers can write Servlets that access Domino data. The Lotus Domino Driver for JDBC makes Domino databases look like another relational back-end source to the SQL tool or application interface by producing result sets that mirror the relational model. An application can also perform a SQL Join of data from Domino with data from a relational database, such as Oracle, Sybase, or DB/2.

Product features include:

- Java standard access to Domino 4.5x, 4.6x, and R5x databases.
- Compatible with Netscape Communicator 4.05 and higher.
- Compatible with Microsoft Internet Explorer 4.01 with service pack 1 and higher.
- Includes signed and unsigned version for maximum flexibility when creating Java applets that use JDBC
- Year 2000 ready; so, it correctly reads and interprets dates in two digit year (mm/dd/yy) and four digit year (mm/dd/yyyy) format.
- Tested with IBM's Visual Age for Java Version 2.0, IBM's WebSphere Application Server 2.0, Lotus eSuite DevPack 1.5, Borland's JBuilder 3

Professional, Jinfonet Software's JReport Professional 1.3, EnterpriseSoft's Enterprise Reports 3.0, Symantec's Visual Cafe Database Development Edition Version 3.0, and NetObjects BeanBuilder 1.0.

- Multi-threaded for high performance Web use.
- Works with clustered servers for high application availability via load balancing and fail over.
- Four sample programs (source .java files provided with documentation).
- Lotus Domino Driver for JDBC is another step in the Lotus commitment to open standards and accessibility to Domino databases and services so that customers can integrate Domino into any e-business solution regardless of their enterprise configuration or tools selection.

The JDBC API is included in the Sun JDK v1.1 and higher. The Lotus Domino Driver for JDBC (LDDJ) classes are provided when you download and install the Lotus Domino Driver for JDBC. The Lotus Domino Driver for JDBC requires Notes DLLs to connect to the Domino/Notes database. (This is the non-Java compiled code referred to above.)

- If your Java program is running on a client computer, then the Notes client must be installed.
- If your Java application is running on a server that is not running Domino, then the Notes client must be installed and configured to connect to the Domino server.
- If your Java application is running on the Domino server (for example, running as a Servlet on R5), then the Notes DLLs and API are already present and do not need to be installed.

Knowledge of Notes API

You do not need to know the Notes API. The Lotus Domino Driver for JDBC shields you from that and enables you to use the JDBC API to connect to Domino and SQL to query, update, and massage Domino data.

Refer to the URL `http://www.lotus.com` for more information about the Lotus Domino Driver for JDBC.

Appendix A. SQL scripts for creating the USERSTUF tables

This appendix provides the source file listings for creating the DB2 tables in our project.

A.1 The USERINFO table

```
create table userinfo (
   first_name varchar(20) not null,
   last_name varchar(20) not null,
   email_address varchar(50) not null,
   date_joined date not null,
   birth_date date not null,
   address varchar(200) not null,
   username varchar(20) not null,
   primary key (username)
);
```

A.2 The USERVERIFY table

```
create table userverify (
   username varchar(20) not null,
   password varchar(20) not null,
   primary key (username)
);
```

A.3 The USERPROFILE table

```
create table userprofile (
   username varchar(20) not null,
   as400 char(1),
   netfinity char(1),
   ncf char(1),
   nw char(1),
   s390 char(1),
   rs6000 char(1),
   scalable char(1),
   vm char(1),
   vse char(1),
   windows char(1),
   other char(1),
   primary key (username)
);
```

A.4 The USERORDERS table

```
create table userorders (
   username varchar(20) not null,
   order_timestamp timestamp not null,
   product_id varchar(20) not null,
   order_id int not null,
   primary key (order_id)
);
```

Appendix B. Java code relating to DB2

This appendix provides the code listings of the DB2 related Java codes.

B.1 Java servlet TestDB2

```
package penguins.access.db2;

import java.sql.*;
import javax.servlet.http.*;
import javax.servlet.*;
public class TestDB2 extends HttpServlet {
    static {
        try {
            Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public TestDB2() {
        super();
    }
    public void doGet(HttpServletRequest req, HttpServletResponse res) {
        try {
            String url = "jdbc:db2:sample";
            String uid = "db2inst1";
            String pwd = "ibmdb2";
            Connection con = DriverManager.getConnection(url, uid, pwd);
            ServletOutputStream out = res.getOutputStream();
            res.setContentType("text/html");
            out.print("<html><head>TestDB2</head><body><p>");
            out.print("select * from department<p>");
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("select * from department");
            while (rs.next()) {
                out.print(rs.getString(1) + ":" + rs.getString(2) + "<p>");
            }
            out.print("</body></html>");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

B.2 Java servlet AccessUserstuf

```
package penguins.access.db2;

import com.sun.server.http.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.sql.*;


/**
```

```
 * final version
 * call initDB from doService, this creates a database connection each call
 * instead of one per servlet creation (i.e. at load time)
 * This is in case connection is dropped due to database restart... so connection
 * is created, instead of having to have servlet reloaded.
 *
 * Creation date: (4/17/00 9:09:51 AM)
 * @author: Phing
 */
public class AccessUserstuf extends HttpServlet {
    private Connection con = null;
    // subset of column names for the userinfo table that we will use
    private String infoCols[] = {"first_name", "last_name", "email_address",
"date_joined", "birth_date", "address"};
    // subset of column names for the userprofile table that we will use
    private String profileCols[] = {"as400", "netfinity", "ncf", "nw", "s390", "rs6000",
"scalable", "vm", "vse", "windows", "other"};
    // register DB2 app class
    static {
        try {
            Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
/**
 * AccessUserstuf constructor comment.
 */
public AccessUserstuf() {
    super();
}
protected void debug(HttpServiceRequest req, HttpServiceResponse res) throws IOException
{
    try {
        ServletOutputStream out = res.getOutputStream();
        res.setContentType("text/html");
        out.println("<html><head><title>Form values : debug orders
</title></head><body>");
        // Request object parameters
        out.println("Parameters:");
        com.sun.server.ServletRequestImpl req2;
        out.println("<table border=1>");
        out.println("<th>Key</th><th>Value</th>");
        for (Enumeration e = req.getParameterNames(); e.hasMoreElements();) {
            String key = (String) e.nextElement();
            String parameter = req.getParameter(key);
            if (parameter == null) {
                parameter = "null parameter";
            }
            if (parameter.equals("")) {
                parameter = "empty string";
            }
            out.println("<tr><td>" + key + "</td><td>" + parameter + "</td></tr>");
        }
        out.println("</table><p>");
        // Session object values
        out.println("Session values:");
        HttpSession session = req.getSession(true);
        out.println("<table border=1>");
        out.println("<th>Key</th><th>Value</th>");
        String valueNames[] = session.getValueNames();
        for (int i = 0; i < valueNames.length; i++) {
            String key = valueNames[i];
```

```
            out.println("<tr><td>" + key + "</td><td>" + session.getValue(key) +
"</td></tr>");
        }
        out.println("</table><p>");
        // contents of database tables of most interest to us
        try {
            // contents of userinfo
            out.println("Contents of userinfo:<p>");
            String sqlStmt = "select * from userinfo";
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(sqlStmt);
            ResultSetMetaData rsmd = rs.getMetaData();
            int cols = rsmd.getColumnCount();
            while (rs.next()) {
                for (int col = 1; col <= cols; col++) {
                    out.print(rs.getString(col) + ":");
                }
                out.println("<p>");
            }
            stmt.close();
            out.println("<p>End contents of userinfo<p>");
            // contents of userverify
            out.println("Contents of userverify:<p>");
            sqlStmt = "select * from userverify";
            stmt = con.createStatement();
            rs = stmt.executeQuery(sqlStmt);
            rsmd = rs.getMetaData();
            cols = rsmd.getColumnCount();
            while (rs.next()) {
                for (int col = 1; col <= cols; col++) {
                    out.print(rs.getString(col) + ":");
                }
                out.println("<p>");
            }
            stmt.close();
            out.println("<p>End contents of userverify<p>");
        } catch (SQLException e) {
            e.printStackTrace();
            out.println(e.getMessage());
        }
        out.println("</body></html>");
        session.invalidate();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
public void doGet(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException {
    doService(req, res);
}
public void doPost(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException {
    doService(req, res);
}
protected synchronized void doService(HttpServletRequest req1, HttpServletResponse res1)
throws ServletException, IOException {
    // cast req and res to from ...Servlet... to ...Service...
    HttpServiceRequest req = (HttpServiceRequest) req1;
    HttpServiceResponse res = (HttpServiceResponse) res1;
    // obtain session object
    HttpSession session = req.getSession(true);
    // obtain action to perform
    String action = req.getParameter("action");
```

```
    // cleanup any previous errors
    session.removeValue("error");
    // obtain success and failure pages
    String successpage = req.getParameter("success");
    String failurepage = req.getParameter("failure");
    // obtain connection to database
    initDB();
    // debug action?
    if (action.equals("debug")) {
        debug(req, res);
        return;
    }
    try {
        // process an order, we expect a product_id cookie value
        if (action.equals("order")) {
            String username = getUsername(session);
            String password = getPassword(session);
            if (!passwordsMatch(username, password)) {
                uidPwdError(username, password, failurepage, req, res);
            } else {
                String product_id = req.getParameter("product_id");
                if (product_id == null || product_id.equals("")) {
                    raiseError("Product ID cannot be empty", failurepage, req, res);
                }
                if (product_id.length() > 20) {
                    raiseError("Product ID cannot be > 20 charactes", failurepage, req,
res);
                } else {
                    // generate new unique order_id
                    // if no orders prior, start at 15344!
                    int order_id = 15344;
                    String sqlStmt = "select max (order_id) from userorders";
                    Statement stmt = con.createStatement();
                    ResultSet rs = stmt.executeQuery(sqlStmt);
                    if (rs.next()) {
                        order_id = rs.getInt(1) + 1;
                    }
                    stmt.close();
                    // now insert a new record
                    sqlStmt = "insert into userorders (username, order_timestamp,
product_id, order_id) values ('" + username + "', current timestamp, '" + product_id +
"'," + order_id + ")";
                    stmt = con.createStatement();
                    int rsi = stmt.executeUpdate(sqlStmt);
                    if (rsi != 1) {
                        raiseError("USERORDERSINSERT rs!=1", failurepage, req, res);
                    }
                    stmt.close();
                    // put product_id into session object
                    session.putValue("product_id", product_id);
                    res.callPage(successpage, req);
                }
            }
            return;
        }
        // list orders for a given username
        if (action.equals("listorders")) {
            String username = getUsername(session);
            String password = getPassword(session);
            if (!passwordsMatch(username, password)) {
                uidPwdError(username, password, failurepage, req, res);
            } else {
```

```
            String sqlStmt = "select order_id, order_timestamp, product_id from
userorders where username='" + username + "'";
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(sqlStmt);
            Orders orders = new Orders();
            while (rs.next()) {
                orders.add(rs.getString("order_ID"), rs.getString("order_timestamp"),
rs.getString("product_id"));
            }
            stmt.close();
            // set bean in environment to pass back to jsp
            req.setAttribute("orders", orders);
            res.callPage(successpage, req);
        }
        return;
    }
    // login with username and password parameters in request stream
    if (action.equals("login")) {
        String username = getUsername(req);
        if (!usernameExists(username))
            usernameError(username, failurepage, req, res);
        String password = getPassword(req);
        if (!passwordsMatch(username, password))
            uidPwdError(username, password, failurepage, req, res);
        else {
            // set cookie values for userinfo
            String sqlStmt = "select first_name, last_name, email_address,
date_joined, birth_date, address from userinfo where username='" + username + "'";
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(sqlStmt);
            ResultSetMetaData rsmd = rs.getMetaData();
            if (rs.next()) {
                if (rsmd.getColumnCount() > 0) {
                    for (int i = 0; i < infoCols.length; i++) {
                        session.putValue(infoCols[i], rs.getString(infoCols[i]));
                    }
                } else {
                    raiseError("USERINFOSELECT: no records selected: " + sqlStmt,
failurepage, req, res);
                }
            }
            stmt.close();
            // set cookie values for userprofile
            sqlStmt = "select as400, netfinity, ncf, nw, s390, rs6000, scalable, vm,
vse, windows, other from userprofile where username='" + username + "'";
            stmt = con.createStatement();
            rs = stmt.executeQuery(sqlStmt);
            rsmd = rs.getMetaData();
            if (rs.next()) {
                for (int i = 0; i < profileCols.length; i++) {
                    if (rs.getString(profileCols[i]) != null) {
                        session.putValue(profileCols[i], profileCols[i]);
                    }
                }
            } else {
                raiseError("USERPROFILE: no records selected: " + sqlStmt, failurepage,
req, res);
            }
            stmt.close();
            // set cookie values for username and password
            session.putValue("username", username);
            session.putValue("password", password);
            // finally, go to the home page
```

```
                res.callPage(successpage, req);
            }
            return;
        } // end login
        // change user password based on cookie values for username and password
        // new password is found in the parameters for the request stream
        if (action.equals("changepassword")) {
            String newPassword = req.getParameter("password1");
            String newPasswordAgain = req.getParameter("password2");
            if (!newPassword.equals(newPasswordAgain)) {
                raiseError("Passwords \"" + newPassword + "\" and \"" + newPasswordAgain +
"\" do not match", failurepage, req, res);
            } else {
                Hashtable errors = new Hashtable();
                errors = validatePassword(newPassword, errors);
                if (!errors.isEmpty()) {
                    raiseError(toString(errors), failurepage, req, res);
                } else {
                    String username = getUsername(session);
                    String password = getPassword(session);
                    if (!passwordsMatch(username, password)) {
                        uidPwdError(username, password, failurepage, req, res);
                    } else {
                        String sqlStmt = "update userverify set password='" + newPassword +
"' where username='" + username + "'";
                        Statement stmt = con.createStatement();
                        int rs = stmt.executeUpdate(sqlStmt);
                        stmt.close();
                        if (rs != 1) {
                            raiseError("USERVERIFYUPDATE: result rows !=1 from " + sqlStmt,
failurepage, req, res);
                        }
                        // update password cookie
                        session.putValue("password", newPassword);
                        res.callPage(successpage, req);
                    }
                }
            }
            return;
        } // end changepassword
        // register with info provided as parameters in request stream
        if (action.equals("register") || action.equals("registration")) {
            String username = getUsername(req);
            if (usernameExists(username)) {
                raiseError("Username '" + username + "' already exists.  Please try again",
failurepage, req, res);
            }
            Hashtable errors = validateData(req);
            errors = validateUsername(username, errors);
            String password = getPassword(req);
            errors = validatePassword(password, errors);
            if (!errors.isEmpty()) {
                raiseError("The following error(s) have been detected.  Please correct and
try again:<p>" + toString(errors), failurepage, req, res);
            } else {
                Statement stmt = con.createStatement();
                // insert userinfo
                String sqlStmt = "insert into userinfo (first_name, last_name,
email_address, date_joined, birth_date, address, username) values ('" +
req.getParameter("first_name") + "','" + req.getParameter("last_name") + "','" +
req.getParameter("email_address") + "',current date,date('" +
req.getParameter("birth_date") + "'),'" + req.getParameter("address") + "','" + username
+ "')";
```

```
                int rs = stmt.executeUpdate(sqlStmt);
                stmt.close();
                if (rs != 1) {
                    raiseError("USERINFOINSERT: result rows !=1 from " + sqlStmt,
failurepage, req, res);
                    return;
                }
                // insert userverify
                sqlStmt = "insert into userverify (username,password) values ('" + username
+ "','" + password + "')";
                stmt = con.createStatement();
                rs = stmt.executeUpdate(sqlStmt);
                stmt.close();
                if (rs != 1) {
                    raiseError("USERVERIFYINSERT: result rows !=1 from " + sqlStmt,
failurepage, req, res);
                    return;
                }
                // insert userprofile
                sqlStmt = "insert into userprofile
(username,as400,netfinity,ncf,nw,s390,rs6000,scalable,vm,vse,windows,other) values ('" +
username + "',";
                for (int i = 0; i < profileCols.length - 1; i++) {
                    if (req.getParameter(profileCols[i]) != null) {
                        sqlStmt = sqlStmt + "'y',";
                    } else {
                        sqlStmt = sqlStmt + "null,";
                    }
                }
                if (req.getParameter(profileCols[profileCols.length - 1]) != null) {
                    sqlStmt = sqlStmt + "'y')";
                } else {
                    sqlStmt = sqlStmt + "null)";
                }
                session.putValue("SQLstmt", sqlStmt);
                stmt = con.createStatement();
                rs = stmt.executeUpdate(sqlStmt);
                stmt.close();
                if (rs != 1) {
                    raiseError("USERPROFILEINSERT: result rows !=1 from " + sqlStmt,
failurepage, req, res);
                    return;
                }
                // login automatically successful, so set cookie values for everything
                session.putValue("username", username);
                session.putValue("password", password);
                for (int i = 0; i < infoCols.length; i++) {
                    if (req.getParameter(infoCols[i]) != null) {
                        session.putValue(infoCols[i], req.getParameter(infoCols[i]));
                    }
                }
                for (int i = 0; i < profileCols.length; i++) {
                    if (req.getParameter(profileCols[i]) != null) {
                        session.putValue(profileCols[i], req.getParameter(profileCols[i]));
                    }
                }
                res.callPage(successpage, req);
            }
            return;
        } // end registration
        // delete takes username and password from cookie values
        if (action.equals("delete")) {
            String username = getUsername(session);
```

```
            String password = getPassword(session);
            if (passwordsMatch(username, password)) {
                Statement stmt = con.createStatement();
                /*
                we delete the userverify entry so they can no longer login
                but maintain the userprofile, userorders and userinfo so we can still
                track orders (from userorders) while maintaining the username integrity
                (i.e. after deleting a username, if somebody wants to reuse the username
                they can't
                ** this can be alleviated by using a unique userid rather than a username
                ** maybe implement this in a more complete design later
                */
                String sqlStmt = "delete from userverify where username='" + username +
"'";
                int rs = stmt.executeUpdate(sqlStmt);
                stmt.close();
                if (rs != 1) {
                    raiseError("USERVERIFYDELETE: result rows !=1 from " + sqlStmt,
failurepage, req, res);
                }
                // invalidate session, return to homepage
                session.invalidate();
                res.callPage(successpage, req);
            } else {
                uidPwdError(username, password, failurepage, req, res);
            }
            return;
        } // end delete
        // update user info based on parameters in request stream
        if (action.equals("update")) {
            String username = getUsername(session);
            String password = getPassword(session);
            if (passwordsMatch(username, password)) {
                Hashtable errors = validateData(req);
                if (errors.isEmpty()) {
                    // we assume that "anything" passed in the request environment barring
the username
                    // which can no longer be changed, is to be updated.
                    // remember that password is changed via action=changepassword
                    String sqlStmt = "update userinfo set (first_name, last_name,
email_address, birth_date, address)=('" + req.getParameter("first_name") + "','" +
req.getParameter("last_name") + "','" + req.getParameter("email_address") + "',date('" +
req.getParameter("birth_date") + "'),'" + req.getParameter("address") + "') where
username='" + username + "'";
                    Statement stmt = con.createStatement();
                    int rs = stmt.executeUpdate(sqlStmt);
                    stmt.close();
                    if (rs != 1) {
                        raiseError("USERINFOUPDATE: result rows !=1 from " + sqlStmt,
failurepage, req, res);
                    }

                    // update userprofile
                    sqlStmt = "update userprofile set
(as400,netfinity,ncf,nw,s390,rs6000,scalable,vm,vse,windows,other) = (";
                    for (int i = 0; i < profileCols.length - 1; i++) {
                        if (req.getParameter(profileCols[i]) != null) {
                            sqlStmt = sqlStmt + "'y',";
                        } else {
                            sqlStmt = sqlStmt + "null,";
                        }
                    }
                    if (req.getParameter(profileCols[profileCols.length - 1]) != null) {
```

```
                    sqlStmt = sqlStmt + "'y')";
                } else {
                    sqlStmt = sqlStmt + "null)";
                }
                sqlStmt = sqlStmt + "where username='" + username + "'";
                stmt = con.createStatement();
                rs = stmt.executeUpdate(sqlStmt);
                stmt.close();
                if (rs != 1) {
                    raiseError("USERPROFILEINSERT: result rows !=1 from " + sqlStmt,
failurepage, req, res);
                }
                // must set cookie here!
                // cookies for userinfo
                for (int i = 0; i < infoCols.length; i++) {
                    if (req.getParameter(infoCols[i]) != null) {
                        session.putValue(infoCols[i], req.getParameter(infoCols[i]));
                    }
                }
                // cookies for userprofile
                for (int i = 0; i < profileCols.length; i++) {
                    if (req.getParameter(profileCols[i]) != null) {
                        session.putValue(profileCols[i],
req.getParameter(profileCols[i]));
                    } else { // undef the interest category
                        session.removeValue(profileCols[i]);
                    }
                }
                res.callPage(successpage, req);
            } else {
                raiseError("The following error(s) have been detected.  Please correct
and try again:<p>" + toString(errors), failurepage, req, res);
                }
            } else {
                uidPwdError(username, password, failurepage, req, res);
            }
            return;
        }
        raiseError("ACTIONERROR: Please report the following: <p> ACTION \"" + action +
"\"not defined in URI " + req.getRequestURI() + "", failurepage, req, res);
        res.callPage(failurepage, req);
    } catch (SQLException e) {
        e.printStackTrace();
        session.putValue("error", "SQLERROR " + e.getMessage());
        res.callPage(failurepage, req);
    }
}
private String getPassword(HttpServiceRequest req) {
    return req.getParameter("password");
}
private String getPassword(HttpSession session) {
    return (String) session.getValue("password");
}
private String getUsername(HttpServiceRequest req) {
    return req.getParameter("username");
}
private String getUsername(HttpSession session) {
    return (String) session.getValue("username");
}
protected void initDB() {
    try {
        String url = "jdbc:db2:userstuf";
        String uid = "db2inst1";
```

```
        String pwd = "ibmdb2";
        con = DriverManager.getConnection(url, uid, pwd);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
protected boolean passwordsMatch(String username, String password) {
    boolean match = false;
    try {
        // verify username+password
        String sqlStmt = "select password from userverify where username=" + "'" +
username + "'";
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(sqlStmt);
        if (rs.next()) {
            ResultSetMetaData rsmd = rs.getMetaData();
            if (rsmd.getColumnCount() > 0) {
                // implement any password algorithm here for checking
                String thePassword = rs.getString(1);
                if (thePassword.equals(password)) {
                    match = true;
                }
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return match;
}
protected void raiseError(String errorText, String errorPage, HttpServiceRequest req,
HttpServiceResponse res) {
    try {
        HttpSession session = req.getSession(true);
        session.putValue("error", errorText);
        res.callPage(errorPage, req);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
private String toString(Hashtable errors) {
    Enumeration he = errors.keys();
    String errorText = "";
    String fieldName = "";
    while (he.hasMoreElements()) {
        fieldName = (String) he.nextElement();
        errorText = errorText + errors.get(fieldName) + "<p>";
    }
    return errorText;
}
private void uidPwdError(String username, String password, String redirectedPage,
HttpServiceRequest req, HttpServiceResponse res) {
    HttpSession session=req.getSession(true);
    raiseError("Username \"" + username + "\" with password \"" + password + "\" invalid",
redirectedPage, req, res);
}
private void usernameError(String username, String redirectedPage, HttpServiceRequest
req, HttpServiceResponse res) {
    raiseError("Username \"" + username + "\" does not exist", redirectedPage, req, res);
}
protected boolean usernameExists(String username) {
    boolean exists = false;
    try {
        if (username == null) {
```

```
            return false;
        }
        /*
        note here we get the username from userinfo not userverify
        if the username has been deleted before, the userverify entry will have been
deleted
        but the userinfo entry is still there, hence we avoid "duplicate" usernames by
selecting from userinfo
        */
        String sqlStmt = "select username from userinfo where username='" + username +
"'";
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(sqlStmt);
        if (rs.next()) {
            ResultSetMetaData rsmd = rs.getMetaData();
            if (rsmd.getColumnCount() > 0) {
                if (username.equals(rs.getString(1))) {
                    exists = true;
                }
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return exists;
}
protected Hashtable validateData(HttpServletRequest req1) {
    HttpServiceRequest req = (HttpServiceRequest) req1;
    Hashtable errors = new Hashtable();
    try {
        String aString = req.getParameter("first_name");
        if (aString == null || aString.equals("")) {
            errors.put("first_name", "First name cannot be empty");
        }
        aString = req.getParameter("last_name");
        if (aString == null || aString.equals("")) {
            errors.put("last_name", "Last name cannot be empty");
        }
        aString = req.getParameter("email_address");
        if (aString == null || aString.equals("")) {
            errors.put("email_address", "Email address cannot be empty");
        }
        aString = req.getParameter("birth_date");
        if (aString == null || aString.equals("")) {
            errors.put("birth_date", "Birth Date cannot be empty");
        }
        aString = req.getParameter("address");
        if (aString == null || aString.equals("")) {
            errors.put("address", "Address cannot be empty");
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return errors;
}
private Hashtable validatePassword(String password, Hashtable errors) {
    if (password == null || password.equals("")) {
        errors.put("password", "Password cannot be empty");
        return errors;
    }
    if (password.length()<4) {
        errors.put("password", "Password must be 4 characters or longer");
        return errors;
```

```
    }
    if (password.length() > 20) {
        errors.put("password", "Password cannot be longer than 20 characters");
        return errors;
    }
    return errors;
}
private Hashtable validateUsername(String username, Hashtable errors) {
    if (username == null || username.equals("")) {
        errors.put("username", "Username cannot be empty");
        return errors;
    }
    if (username.length() > 20) {
        errors.put("username", "Username cannot be longer than 20 characters");
        return errors;
    }
    return errors;
}
}
```

B.3 Java Bean orders

```
package penguins.access.db2;
import java.util.*;
public class Orders {
    private Vector orderIDs=new Vector();
    private Hashtable timestamps=new Hashtable();
    private Hashtable productIDs=new Hashtable();
    public Orders() {
        super();
    }
    public void add(String order_id, String order_timestamp, String product_id) {
        orderIDs.addElement(order_id);
        timestamps.put(order_id, order_timestamp);
        productIDs.put(order_id, product_id);
    }
    public Enumeration orderIDs() {
        return orderIDs.elements();
    }
    public String productID(String orderID) {
        return (String) productIDs.get(orderID);
    }
    public String timestamp(String orderID) {
        return (String) timestamps.get(orderID);
    }
}
```

Appendix C. Java code relating to Domino database

This appendix provides listings of Domino related Java codes.

C.1 Java servlet TestDomino

```
import Java.io.*;
import Java.util.*;
import Javax.servlet.*;
import Javax.servlet.http.*;
import lotus.domino.*;
public class TestDomino extends HttpServlet {
   public void doGet (HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException
   {
      String title=null;
      String dbName;
      String host;

      /* Get parameter name and value */
      dbName=req.getParameter("db");
      host=req.getParameter("host")+":"+req.getParameter("port");

      res.setContentType("text/html");
      ServletOutputStream out = res.getOutputStream();
      out.println("<html>");
      out.println("<head><title>Intro Servlet</title></head>");
      out.println("<body>");
      try{
         Session s=NotesFactory.createSession(host);
         Database db=s.getDatabase(s.getServerName(), dbNameValue);
         /* Retrieve database title */
         title=db.getTitle();
      }
      catch(NotesException ex) {
         out.println(ex.id + " " + ex.text);
      }
      out.println("<h1> The title of "+dbNameValue+".nsf is \""+title+"\"
</h1>");
      out.println("</body></html>");
   }
}
```

C.2 Java servlet VBDList10

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import lotus.domino.*;
import java.util.Vector;

public class VBDList10 extends HttpServlet {

   String viewname="platform sorted";
   String dbname="Redbooks";
   String host="bdtest3:8000";
   String category=null;
   String tm, dumb;
   DateTime dt;

   public void doGet (HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException {
      doService(req,res);
   }

   public void doPost (HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException {
      doService(req,res);
   }

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

      String catcat = null;
      int no;
      Vector listParam = new Vector();

      /* Get parameter name and value */

      for (Enumeration e=req.getParameterNames(); e.hasMoreElements(); ) {
         String param = (String)e.nextElement();
         String paramValue = (String)req.getParameter(param);
         listParam.addElement(paramValue);
      }
      for (int i=0; i<listParam.size(); i++)
      {
         catcat = (String) listParam.elementAt(i);
         if (catcat.equals("item1")) category = "AS/400";
```

```
        if (catcat.equals("item2")) category = "Netfinity and Software";
        if (catcat.equals("item3")) category = "Network Computing
Framework";
        if (catcat.equals("item4")) category = "Networking Hardware";
        if (catcat.equals("item5")) category = "OS/390 and S/390";
        if (catcat.equals("item6")) category = "RS/6000 and AIX";
        if (catcat.equals("item7")) category = "Scalable Parallel";
        if (catcat.equals("item8")) category = "VM/ESA";
        if (catcat.equals("item9")) category = "VSE/ESA";
        if (catcat.equals("item10")) category = "Windows and Win NT";
        if (catcat.equals("item11")) category = "";

        res.setContentType("text/html");
        ServletOutputStream out=res.getOutputStream();

        try {

           /* Create session and open the database  */
           Session s=NotesFactory.createSession(host);
           Database db=s.getDatabase(s.getServerName(), dbname);
           View view = db.getView(viewname);

           out.println("<html>");
           out.println("<head><title>Redbooks Database access
Test</title></head>");
           out.println("<body>");
           out.println("</p>");

           out.println("<blockquote>");
           out.println("<li><b><font size=+1></p>");
           if (category.equals(""))
              out.println("Not Categorized </font> </b> </li>
</blockquote>");
           else
              out.println(category+"</font></b></li></blockquote>");
           out.println("<blockquote>");
           out.println("<table BORDER COLS=3 WIDTH=95%>");
           out.println("<tr ALIGN=LEFT VALIGN=CENTER>");
           out.println("<td ALIGN=LEFT VALIGN=CENTER WIDTH=15%>");
           out.println("<center><b>Form Number</b></center></td>");
           out.println("<td>");
           out.println("<center><b>Redbooks Title</b></center></td>");
           out.println("<td WIDTH=20%>");
           out.println("<center><b>Publication Date </b> </center> </td>
</tr>");

           ViewNavigator navy = view.createViewNavFromCategory(category);
```

```
            ViewEntry entriy = navy.getFirst();
            int count=0;
            while ((entriy != null) && count < 10) {
               if (entriy.isCategory()) out.println("");
               if (entriy.isDocument()) {
                  Vector dM =
entriy.getDocument().getItemValue("PublishDate");
                  if (dM.size() > 0) {
                     dt = (DateTime)dM.elementAt(0);
                     tm = dt.getDateOnly();
                  }
                  if (tm == null) out.println("");
                  else {
                     out.println("<tr><td><center>");
                     out.println("<A HREF=/servlet/VBDOpen?docid="
+entriy.getNoteID() + " TARGET=body>");

out.println(entriy.getDocument().getItemValueString("FormNumber"));
                     out.println("</A>");
                     out.println("</center></td>");
                     out.println("<td>");

out.println(entriy.getDocument().getItemValueString("BookTitle"));
                     out.println("</td>");
                     out.println("<td><center>");
                     out.println(tm);
                     out.println("</center></td></tr>");
                     count++;
                  }
               }
               entriy = navy.getNext();
            }
            out.println("<tr><td VALIGN=CENTER WIDTH=20></td>");
            out.println("<td></td><td></td></tr></table></blockquote>");
            out.println("</body></html>");

         }
         catch(NotesException ex) {
            out.println(ex.id + " " + ex.text);
            out.println("<html><body>");
            out.println(ex.id+" "+ex.text);
            out.println("</body></html>");
         }
      }
   }
}
```

C.3 Java servlet VBDGetLatest

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import lotus.domino.*;
import java.util.Vector;

public class VBDGetLatest extends HttpServlet {

   String viewname="All books";
   String dbname="Redbooks";
   String host="bdtest3:8000";
   String category=null;
   String tm, dumb;
   DateTime dt;

   public void doGet (HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
   doService(req,res);
   }

   public void doPost (HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
   doService(req,res);
   }

   public void doService (HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException
   {
      int no;
      /* Get parameter name and value */

/*      category=req.getParameter("category");
      if (category.equals("All"))
         viewname="All books";
      else
         viewname="platform sorted";
*/
      try {

         /* Create session and open the database  */
         Session s=NotesFactory.createSession(host);
         Database db=s.getDatabase(s.getServerName(), dbname);
         View view = db.getView(viewname);
```

```
         res.setContentType("text/html");
         ServletOutputStream out=res.getOutputStream();
         out.println("<html>");
         out.println("<head><title>Redbooks Database access
Test</title></head>");
         out.println("<body>");
         out.println("</p>");

/*         out.println("<blockquote>");
         out.println("<li><b><font size=+1></p>");

         if (category.equals(""))
            out.println("Not Categorized</font></b></li></blockquote>");
         else
            out.println(category+"</font></b></li></blockquote>");
*/
         out.println("<blockquote>");
         out.println("<table BORDER COLS=3 WIDTH=95%>");
         out.println("<tr ALIGN=LEFT VALIGN=CENTER>");
         out.println("<td ALIGN=LEFT VALIGN=CENTER WIDTH=15%>");
         out.println("<center><b>Form Number</b></center></td>");
         out.println("<td>");
         out.println("<center><b>Redbooks Title</b></center></td>");
         out.println("<td WIDTH=20%>");
         out.println("<center><b>Publication Date</b></center></td></tr>");

//         ViewNavigator navy = view.createViewNavFromCategory(category);
    ViewNavigator navy = view.createViewNavMaxLevel(0);
         ViewEntry entriy = navy.getFirst();
         int ix = 0;
         while ((entriy != null) && (ix<10)) {
            if (entriy.isCategory()) out.println("");
            if (entriy.isDocument()) {
         Vector dM = entriy.getDocument().getItemValue("PublishDate");
         if (dM.size() > 0) {
               dt = (DateTime)dM.elementAt(0);
               tm = dt.getDateOnly();
               if (tm == null) out.println("");
               else {
                  ix++;
                  out.println("<tr><td><center>");
                  out.println("<A HREF=/servlet/VBDOpen?docid="
+entriy.getNoteID() + " TARGET=body>");

out.println(entriy.getDocument().getItemValueString("FormNumber"));
                  out.println("</A>");
```

```
                    out.println("</center></td>");
                    out.println("<td>");

out.println(entriy.getDocument().getItemValueString("BookTitle"));
                    out.println("</td>");
                    out.println("<td><center>");
                    out.println(tm);
                    out.println("</center></td></tr>");
                  }
               }
            }
           entriy = navy.getNext();
        }
        out.println("<tr><td VALIGN=CENTER WIDTH=20></td>");
        out.println("<td></td><td></td></tr></table></blockquote>");
        out.println("</body></html>");

     } catch(NotesException ex) {
        System.out.println(ex.id + " " + ex.text);
      res.setContentType("text/html");
      ServletOutputStream out=res.getOutputStream();
      out.println("<html><body>");
      out.println(ex.id+" "+ex.text);
      out.println("</body></html>");
     }
}

public String getServletInfo() {
   return "Create a page that displays the title of the database client
specified";
  }
}
```

C.4 Java servlet VBDSearch

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import lotus.domino.*;
import java.util.Vector;

public class VBDSearch extends HttpServlet {

   String viewname="All books";
   String dbname="Redbooks";
```

```
   String host="bdtest3:8000";
   String query=null;
   String tm, dumb;
   DateTime dt;

   public void doGet (HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
   doService(req,res);
   }

   public void doPost (HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
   doService(req,res);
   }

   public void doService (HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException
   {
      int no;
      /* Get parameter name and value */

      query = req.getParameter("search");

      try {

         /* Create session and open the database  */
         Session s=NotesFactory.createSession(host);
         Database db=s.getDatabase(s.getServerName(), dbname);
         View view = db.getView(viewname);

         res.setContentType("text/html");
         ServletOutputStream out=res.getOutputStream();
         out.println("<html>");
         out.println("<head><title>Redbooks Query Result</title></head>");
         out.println("<body>");
         out.println("</p>");

         out.println("Parameter = "+query+"</p>");

         ViewEntryCollection vec = view.getAllEntries();
         vec.FTSearch(query);
         ViewEntry entriy = vec.getFirstEntry();
         if (vec.getCount() != 0) {

           out.println("<blockquote>");
           out.println("<table BORDER COLS=3 WIDTH=95%>");
           out.println("<tr ALIGN=LEFT VALIGN=CENTER>");
```

```
          out.println("<td ALIGN=LEFT VALIGN=CENTER WIDTH=15%>");
          out.println("<center><b>Form Number</b></center></td>");
          out.println("<td>");
          out.println("<center><b>Redbooks Title</b></center></td>");
          out.println("<td WIDTH=20%>");
          out.println("<center><b>Publication
Date</b></center></td></tr>");

          while (entriy != null) {
           if (entriy.isCategory()) out.println("");
           if (entriy.isDocument()) {
        Vector dM = entriy.getDocument().getItemValue("PublishDate");
        if (dM.size() > 0) {
              dt = (DateTime)dM.elementAt(0);
              tm = dt.getDateOnly();
              if (tm == null) out.println("");
              else {
                 out.println("<tr><td><center>");
                 out.println("<A HREF=/servlet/VBDOpen?docid="
+entriy.getNoteID() +" TARGET=body>");

out.println(entriy.getDocument().getItemValueString("FormNumber"));
                 out.println("</A>");
                 out.println("</center></td>");
                 out.println("<td>");

out.println(entriy.getDocument().getItemValueString("BookTitle"));
                 out.println("</td>");
                 out.println("<td><center>");
                 out.println(tm);
                 out.println("</center></td></tr>");
              }
           }
        }
       entriy = vec.getNextEntry();
    }
    out.println("<tr><td VALIGN=CENTER WIDTH=20></td>");
    out.println("<td></td><td></td></tr></table></blockquote>");
    out.println("</body></html>");
   }
  else out.println("Not Found");
 } catch(NotesException ex) {
    System.out.println(ex.id + " " + ex.text);
  res.setContentType("text/html");
  ServletOutputStream out=res.getOutputStream();
  out.println("<html><body>");
  out.println(ex.id+" "+ex.text);
```

```
   out.println("</body></html>");
  }
}

public String getServletInfo() {
   return "Create a page that displays the title of the database client
specified";
  }
}
```

C.5 Java servlet VBDOpen

```
import java.io.*;
//import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import lotus.domino.*;
import java.util.Vector;

public class VBDOpen extends HttpServlet {

   String dbname="Redbooks";
   String host="bdtest3:8000";
   String tm;
   DateTime dt;

   public void doGet (HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException {
   doService(req,res);
   }

   public void doPost (HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException {
   doService(req,res);
   }

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

      /* Get parameter name and value */
     String docid=(String)req.getParameter("docid");
     HttpSession sess=req.getSession(true);
     String username = (String)sess.getValue("username");
     try {
```

```
        /* Create session and open the database  */
        Session s=NotesFactory.createSession(host);
        Database db=s.getDatabase(s.getServerName(), dbname);

        res.setContentType("text/html");
        ServletOutputStream out=res.getOutputStream();
        out.println("<html>");
        out.println("<head><title>Redbooks Abstract</title></head>");
        out.println("<body text=000000 BGCOLOR=FFFFFF>");
        out.println("<TABLE WIDTH=100% BORDER=0 CELLSPACING=0
CELLPADDING=0>");
        out.println("<TR VALIGN=top>");
   out.println("<TD WIDTH=76%>");
        out.println("<P><IMG SRC=/images/rbabs.gif WIDTH=126
HEIGHT=29><BR>");
   out.println("<BR>");
        Document doc;
        if (docid.length()>8) {
          doc=db.getDocumentByUNID(docid);
        } else {
          doc=db.getDocumentByID(docid);
        }
   out.println("<B><FONT
FACE=Arial>"+doc.getItemValueString("BookTitle"));
   out.println("</FONT></B><B><FONT FACE=Arial>, </FONT></B><B>");
   out.println("<FONT FACE=Arial>"+doc.getItemValueString("FormNumber"));
   out.println("</FONT></B><BR><BR>");

   out.println("<TABLE WIDTH=100% BORDER=0 CELLSPACING=0 CELLPADDING=0>");
   out.println("<TR VALIGN=top>");
   out.println("<TD WIDTH=100% BGCOLOR=000080><B>");
   out.println("<FONT SIZE=2 COLOR=000080 FACE=Arial>_</FONT></B>");
   out.println("<B><FONT SIZE=2 COLOR=FFFFFF FACE=Arial>Abstract");
   out.println("</FONT></B></TD></TR>");

   out.println("<TR VALIGN=top><TD WIDTH=100%>");
   out.println("<FONT SIZE=1 COLOR=FFFFFF FACE=Arial>_</FONT></TD></TR>");

   out.println("<TR VALIGN=top><TD WIDTH=100%><FONT SIZE=2 FACE=Arial>");
   out.println(toHtml(doc.getItemValueString("Abstract")));
   out.println("</FONT></TD></TR>");

   out.println("<TR VALIGN=top><TD WIDTH=100%>");
   out.println("<FONT SIZE=1 COLOR=FFFFFF FACE=Arial>_</FONT></TD></TR>");

   out.println("<TR VALIGN=top><TD WIDTH=100% BGCOLOR=000080>");
   out.println("<B><FONT SIZE=2 COLOR=000080 FACE=Arial>_</FONT></B>");
```

```
   out.println("<B><FONT SIZE=2 COLOR=FFFFFF FACE=Arial>");
   out.println("Table of Contents</FONT></B></TD></TR>");

   out.println("<TR VALIGN=top><TD WIDTH=100%>");
   out.println("<FONT SIZE=1 COLOR=FFFFFF FACE=Arial>_</FONT></TD></TR>");

   out.println("<TR VALIGN=top><TD WIDTH=100%><FONT SIZE=2 FACE=Arial>");
   out.println(toHtml(doc.getItemValueString("TOC"))+"</FONT></TD></TR>");

   out.println("<TR VALIGN=top><TD WIDTH=100%>");
   out.println("<FONT SIZE=2 COLOR=FFFFFF FACE=Arial>_</FONT></TD></TR>");

   out.println("<TR VALIGN=top><TD WIDTH=100%>");
   out.println("<FONT SIZE=1 COLOR=FFFFFF FACE=Arial>_</FONT></TD></TR>");
   out.println("</TABLE>");
   out.println("</TD><TD WIDTH=0%>");
   out.println("<IMG SRC=/icons/blk.gif BORDER=0 HEIGHT=1 WIDTH=1 ALT=>");
   out.println("</TD><TD WIDTH=23%>");
   out.println("<TABLE HEIGHT=16 CELLPADDING=3 WIDTH=100% BORDER=0
CELLSPACING=0 CELLPADDING=0>");

   out.println("<TR VALIGN=top><TD WIDTH=100% BGCOLOR=FF0000><B>");
   out.println("<FONT COLOR=FF0000 FACE=Arial>_</FONT></B>");
   out.println("<B><FONT SIZE=2 COLOR=FFFFFF FACE=Arial>This Redbook");
   out.println("</FONT></B></TD></TR>");

   if (username==null || username.equals("")) {
          out.println("<TR VALIGN=top><TD WIDTH=100%>");
          out.println("<FONT SIZE=1 COLOR=0000FF FACE=Arial>");
     out.println("<A
HREF=/redbooks/RegistrationPage/body_registrationpage.jsp
TARGET=body>Please Register</FONT>");
          out.println("</U></A></TD></TR>");
   } else {

     out.println("<TR VALIGN=top><TD WIDTH=100%>");
     out.println("<A HREF=/redbooks/OrderPage/body_orderpage.html
TARGET=body>");
     out.println("<IMG SRC=/images/ra_kb.gif WIDTH=53 HEIGHT=22 BORDER=0
NAME=BUYING>");
     sess.putValue("FormNumber", doc.getItemValueString("FormNumber"));
     out.println("<FONT SIZE=1 COLOR=0000FF FACE=Arial>  </FONT><U>");
     out.println("<FONT SIZE=1 COLOR=0000FF FACE=Arial>Buy Online</FONT>");
     out.println("</U></A></TD></TR>");
   }
   out.println("<TR VALIGN=top><TD WIDTH=100%>");
   out.println("<IMG SRC=/images/dot.gif WIDTH=148 HEIGHT=5></TD></TR>");
```

```
   out.println("<TR VALIGN=top><TD WIDTH=100% BGCOLOR=0082BF><B>");
   out.println("<FONT COLOR=0082BF FACE=Arial>_</FONT></B>");
   out.println("<B><FONT SIZE=2 COLOR=FFFFFF FACE=Arial>Profile</FONT>");
   out.println("</B></TD></TR>");

   out.println("<TR VALIGN=top><TD WIDTH=100%>");
   out.println("<FONT SIZE=1 FACE=Arial>Publish Date</FONT></TD></TR>");

   out.println("<TR VALIGN=top><TD WIDTH=100%>");
   out.println("<FONT SIZE=1 COLOR=000080 FACE=Arial>");

        Vector dM = doc.getItemValue("PublishDate");
        DateTime dt = (DateTime)dM.elementAt(0);
        String tm = dt.getDateOnly();
        out.println(tm+"</FONT></TD></TR>");

        out.println("<TR VALIGN=top><TD WIDTH=100%>");
        out.println("<IMG SRC=/images/dot.gif WIDTH=148
HEIGHT=5></TD></TR>");

        out.println("<TR VALIGN=top><TD WIDTH=100%>");
        out.println("<FONT SIZE=1 FACE=Arial>Lead
Author</FONT></TD></TR>");

        out.println("<TR VALIGN=top><TD WIDTH=100%>");
        out.println("<FONT SIZE=1 COLOR=000080 FACE=Arial>");

out.println(doc.getItemValueString("OriginalOwner")+"</FONT></TD></TR>");

        out.println("<TR VALIGN=top><TD WIDTH=100%>");
        out.println("<IMG SRC=/images/dot.gif WIDTH=148
HEIGHT=5></TD></TR>");

        out.println("<TR VALIGN=top><TD WIDTH=100%>");
        out.println("<FONT SIZE=1 FACE=Arial>Other
Authors</FONT></TD></TR>");

        out.println("<TR VALIGN=top><TD WIDTH=100%>");
        out.println("<FONT SIZE=1 COLOR=000080 FACE=Arial>");
   out.println(doc.getItemValueString("OtherAuthors"));
   out.println("</FONT></TD></TR>");

        out.println("<TR VALIGN=top><TD WIDTH=100%>");
        out.println("<IMG SRC=/images/dot.gif WIDTH=148
HEIGHT=5></TD></TR>");
```

```
            out.println("<TR VALIGN=top><TD WIDTH=100%>");
            out.println("<FONT SIZE=1 FACE=Arial>Feedback</FONT></TD></TR>");

            out.println("<TR VALIGN=top><TD WIDTH=100%>");
            out.println("<a href=mailto:>");
      out.println("<img SRC=/images/email_kb.gif BORDER=0 height=16
width=16></a></td>");

      out.println("</tr></table></td></tr></table>");

         } catch(NotesException ex) {
            System.out.println("Error");
            sess.putValue("error",ex.id+" "+ex.text);
         }
      }

      private String toHtml(String curText) {
         int nl=-1;

         StringBuffer tempText=new StringBuffer(0);
         while (curText.indexOf("\n") != -1) {
            nl = curText.indexOf("\n");
            tempText.append(curText.substring(0,nl)+"<br>");
            curText = curText.substring(nl+1);
         }
         tempText.append(curText);
         return tempText.toString();
      }

      public String getServletInfo() {
         return "Create a page that displays Redbook abstract";
      }
}
```

Appendix D. HTML and JSP files listing

This appendix provides a listing of all HTML and JSP files used in our project.

D.1 Body_homepage.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<%@ LANGUAGE="java" %>
<HTML>
<HEAD>
<BASE TARGET="_parent">
<% HttpSession session=request.getSession(true);%>
</HEAD>
<BODY BGCOLOR="#FFFFFF" LINK="#0000FF" VLINK="#800080" TEXT="#000000" TOPMARGIN=2
LEFTMARGIN=2 MARGINWIDTH=2 MARGINHEIGHT=2>
<TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0 WIDTH=520>
<TR VALIGN=TOP ALIGN=LEFT>
<TD>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=233>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=14 HEIGHT=23><IMG SRC="../clearpixel.gif" WIDTH=14 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=340></TD>
<TD WIDTH=219 ALIGN=LEFT VALIGN=TOP><IMG ID="Picture37" HEIGHT=340 WIDTH=219
SRC="./ws.jpg" BORDER=0></TD>
</TR>
</TABLE>
</TD>
<TD>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=18 HEIGHT=24><IMG SRC="../clearpixel.gif" WIDTH=18 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=269>
<TABLE ID="Table1" BORDER=1 CELLSPACING=3 CELLPADDING=1 WIDTH=269>
<TR>
<TD WIDTH=125>
<% if (session.getValue("rs6000")!=null) {%>
<A HREF="/servlet/VBDList10?rs6000=item6" TARGET=body>RS/6000</A>
<% }else { %>
 RS/6000
<% } %>
</TD>
<TD WIDTH=125>
<% if (session.getValue("as400")!=null) {%>
<A HREF="/servlet/VBDList10?as400=item1" TARGET=body>AS/400</A>
<% } else { %>
AS/400
<% } %>
</TD>
</TR>
<TR>
<TD>
<% if (session.getValue("nw")!=null) {%>
```

```
<A HREF="/servlet/VBDList10?netfinity=item2" TARGET=body>Netfinity</A>
<% } else {%>
Netfinity
<% } %>
</TD>
<TD>
<% if (session.getValue("vm")!=null) {%>
<A HREF="/servlet/VBDList10?vm/esa=item8" TARGET=body>VM/ESA</A>
<%} else { %>
VM/ESA
<% } %>
</TD>
</TR>
<TR>
<TD>
<% if (session.getValue("s390")!=null) {%>
<A HREF="/servlet/VBDList10?s390=item5" TARGET=body>S390</A>
<%} else {%>
S390
<% } %>
</TD>
<TD>
<% if (session.getValue("vse")!=null) {%>
<A HREF="/servlet/VBDList10?vse=item9" TARGET=body>VSE/ESA</A>
<%} else {%>
VSE/ESA
<% } %>
</TD>
</TR>
<TR>
<TD>
<% if (session.getValue("scalable")!=null) {%>
<A HREF="/servlet/VBDList10?scalable=item7" TARGET=body>SP</A>
<%}else {%>
SP
<%}%>
</TD>
<TD>
<% if (session.getValue("other")!=null) {%>
<A HREF="/servlet/VBDList10?other=item11" TARGET=body>Other</A>
<%}else {%>
Other
<%}%>
</TD>
</TR>
<TR>
<TD>
<% if (session.getValue("windows")!=null) {%>
<TD><A HREF="/servlet/VBDList10?windows=item10" TARGET=body>Windows</A>
<%}else {%>
Windows
<%}%>
</TD>
<TD> </TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=18 HEIGHT=22><IMG SRC="../clearpixel.gif" WIDTH=18 HEIGHT=1 BORDER=0></TD>
<TD></TD>
```

```
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=263>
<TABLE ID="Table2" BORDER=1 CELLSPACING=3 CELLPADDING=1 WIDTH=263>
<TR>
<TD WIDTH=251><P ALIGN=CENTER><B><FONT SIZE="+1">Browse a Category</FONT></B></TD>
</TR>
<TR VALIGN=MIDDLE ALIGN=CENTER><TD>
<applet width="251" height="60" code="goURL.class">
<PARAM name="page0" value="AS/400">
<PARAM name="url0" value="http://bdtest2.itsc.austin.ibm.com/servlet/VBDList10?1=item1">
<PARAM name="page1" value="Netfinity and Software">
<PARAM name="url1" value="http://bdtest2.itsc.austin.ibm.com/servlet/VBDList10?1=item2">
<PARAM name="page2" value="Network Computing Framework">
<PARAM name="url2" value="http://bdtest2.itsc.austin.ibm.com/servlet/VBDList10?1=item3">
<PARAM name="page3" value="Networking Hardware">
<PARAM name="url3" value="http://bdtest2.itsc.austin.ibm.com/servlet/VBDList10?1=item4">
<PARAM name="page4" value="OS/390 and S/390">
<PARAM name="url4" value="http://bdtest2.itsc.austin.ibm.com/servlet/VBDList10?1=item5">
<PARAM name="page5" value="RS/6000 and AIX">
<PARAM name="url5" value="http://bdtest2.itsc.austin.ibm.com/servlet/VBDList10?1=item6">
<PARAM name="page6" value="Scalable Paralel">
<PARAM name="url6" value="http://bdtest2.itsc.austin.ibm.com/servlet/VBDList10?1=item7">
<PARAM name="page7" value="VM/ESA">
<PARAM name="url7" value="http://bdtest2.itsc.austin.ibm.com/servlet/VBDList10?1=item8">
<PARAM name="page8" value="VSE/ESA">
<PARAM name="url8" value="http://bdtest2.itsc.austin.ibm.com/servlet/VBDList10?1=item9">
<PARAM name="page9" value="Windows and Windows NT">
<PARAM name="url9"
value="http://bdtest2.itsc.austin.ibm.com/servlet/VBDList10?1=item10">
<PARAM name="page10" value="Other (Not Categorized)">
<PARAM name="url10"
value="http://bdtest2.itsc.austin.ibm.com/servlet/VBDList10?1=item11">
</applet>
<TD> </TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

D.2 Body_index.jsp

```
<%@ LANGUAGE="Java" %>
<HTML>
<HEAD>
<BASE TARGET="_parent">
</HEAD>
<BODY BGCOLOR="#FFFFFF" LINK="#0000FF" VLINK="#800080" TEXT="#000000" TOPMARGIN=2
LEFTMARGIN=2 MARGINWIDTH=2 MARGINHEIGHT=2>
<TABLE cellpadding=0 cellspacing=0 border=0 width=100%>
<TR VALIGN=top align=left>
<td>
<table border=0 cellspacing=0 cellpadding=0 width=256>
<tr valign=top align=left>
```

```
<td width=256 height=414>

<H2>Select which category you wish to browse<H2>
<applet width="200" height="27" code="goURL.class" color=#FF0000 bgcolor=#FF0000>
<PARAM name="page0" value="AS/400">
<PARAM name="url0" value="http://bdtest1.itsc.austin.ibm.com/servlet/VBDList10?1=item1">
<PARAM name="page1" value="Netfinity and Software">
<PARAM name="url1" value="http://bdtest1.itsc.austin.ibm.com/servlet/VBDList10?1=item2">
<PARAM name="page2" value="Network Computing Framework">
<PARAM name="url2" value="http://bdtest1.itsc.austin.ibm.com/servlet/VBDList10?1=item3">
<PARAM name="page3" value="Networking Hardware">
<PARAM name="url3" value="http://bdtest1.itsc.austin.ibm.com/servlet/VBDList10?1=item4">
<PARAM name="page4" value="OS/390 and S/390">
<PARAM name="url4" value="http://bdtest1.itsc.austin.ibm.com/servlet/VBDList10?1=item5">
<PARAM name="page5" value="RS/6000 and AIX">
<PARAM name="url5" value="http://bdtest1.itsc.austin.ibm.com/servlet/VBDList10?1=item6">
<PARAM name="page6" value="Scalable Paralel">
<PARAM name="url6" value="http://bdtest1.itsc.austin.ibm.com/servlet/VBDList10?1=item7">
<PARAM name="page7" value="VM/ESA">
<PARAM name="url7" value="http://bdtest1.itsc.austin.ibm.com/servlet/VBDList10?1=item8">
<PARAM name="page8" value="VSE/ESA">
<PARAM name="url8" value="http://bdtest1.itsc.austin.ibm.com/servlet/VBDList10?1=item9">
<PARAM name="page9" value="Windows and Windows NT">
<PARAM name="url9"
value="http://bdtest1.itsc.austin.ibm.com/servlet/VBDList10?1=item10">
<PARAM name="page10" value="Other (Not Categorized)">
<PARAM name="url10"
value="http://bdtest1.itsc.austin.ibm.com/servlet/VBDList10?1=item11">
</applet>
</td>
</tr>
</table>
</td>
<td>
<table border=0 cellspacing=0 cellpadding=0 width=292>
<tr align=center><td><B>Latest Releases</B></td></tr>
<tr></tr>
<tr valign=top align=left>
<td width=291><servlet name="VBDGetLatest"></servlet></td>
</tr>
</table>
</td>
</tr>
</table>
</BODY>
</HTML>
```

D.3 Body_loginpage.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<%@ LANGUAGE="java" %>
<HTML>
<HEAD>
<BASE TARGET="_parent">
</HEAD>
<BODY BGCOLOR="#FFFFFF" LINK="#0000FF" VLINK="#800080" TEXT="#000000" TOPMARGIN=2
LEFTMARGIN=2 MARGINWIDTH=2 MARGINHEIGHT=2>
<FORM NAME="login" ACTION="/servlet/AccessUserstuf" METHOD=POST>
<INPUT TYPE=HIDDEN NAME="action" VALUE="login">
<INPUT TYPE=HIDDEN NAME="success" VALUE="/Redbooks/HomePage/homepage.jsp">
<INPUT TYPE=HIDDEN NAME="failure" VALUE="/Redbooks/LoginPage/loginfailure.jsp">
```

```
<TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0 WIDTH=363>
<TR VALIGN=TOP ALIGN=LEFT>
<TD>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=176>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=67 HEIGHT=49><IMG SRC="../clearpixel.gif" WIDTH=67 HEIGHT=1 BORDER=0></TD>
<TD WIDTH=109><IMG SRC="../clearpixel.gif" WIDTH=109 HEIGHT=1 BORDER=0></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=109><P ALIGN=CENTER><B><I><FONT SIZE="+1">User Name</FONT></I></B></TD>
</TR>
</TABLE>
</TD>
<TD>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=187>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=11 HEIGHT=49><IMG SRC="../clearpixel.gif" WIDTH=11 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=22></TD>
<TD WIDTH=176><INPUT ID="FormsEditField1" TYPE=TEXT NAME="username" VALUE="" SIZE=22
MAXLENGTH=40></TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
<TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0 WIDTH=365>
<TR VALIGN=TOP ALIGN=LEFT>
<TD>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=170>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=64 HEIGHT=28><IMG SRC="../clearpixel.gif" WIDTH=64 HEIGHT=1 BORDER=0></TD>
<TD WIDTH=106><IMG SRC="../clearpixel.gif" WIDTH=106 HEIGHT=1 BORDER=0></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=106><P ALIGN=CENTER><B><I><FONT SIZE="+1">Password</FONT></I></B></TD>
</TR>
</TABLE>
</TD>
<TD>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=195>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=19 HEIGHT=29><IMG SRC="../clearpixel.gif" WIDTH=19 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=22></TD>
<TD WIDTH=176><INPUT ID="FormsEditField2" TYPE=PASSWORD NAME="password" VALUE="" SIZE=22
MAXLENGTH=40></TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=451>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=104 HEIGHT=34><IMG SRC="../clearpixel.gif" WIDTH=104 HEIGHT=1 BORDER=0></TD>
<TD></TD>
<TD WIDTH=107><IMG SRC="../clearpixel.gif" WIDTH=107 HEIGHT=1 BORDER=0></TD>
```

```
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=3 HEIGHT=1></TD>
<TD WIDTH=120 ROWSPAN=2 ALIGN=LEFT VALIGN=TOP><A
HREF="javascript:document.forms[0].reset()" TARGET="body"><IMG ID="Picture31" HEIGHT=17
WIDTH=120 SRC="../clear.gif" BORDER=0 ALT="clear"></A></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=16></TD>
<TD WIDTH=120 ROWSPAN=2><INPUT ID="FormsButton1" NAME="submit" TYPE=IMAGE BORDER=0
SRC="../submit.gif"></TD>
<TD></TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

D.4 Body_orderpage.jsp

```
<%@ LANGUAGE="java" %>
<HTML>
<HEAD>
<BASE TARGET="_parent">
<% HttpSession session=request.getSession(true);%>
</HEAD>
<BODY BGCOLOR="#FFFFFF" LINK="#0000FF" VLINK="#800080" TEXT="#000000" TOPMARGIN=2
LEFTMARGIN=2 MARGINWIDTH=2 MARGINHEIGHT=2>
<FORM NAME="orderpage" ACTION="/servlet/AccessUserstuf" METHOD=POST>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=515>
<INPUT TYPE=HIDDEN NAME="action" VALUE="order">
<INPUT TYPE=HIDDEN NAME="success" VALUE="/Redbooks/OrderPage/ordersuccess.jsp">
<INPUT TYPE=HIDDEN NAME="failure" VALUE="/Redbooks/OrderPage/orderfailure.jsp">
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=96 HEIGHT=17><IMG SRC="../clearpixel.gif" WIDTH=96 HEIGHT=1 BORDER=0></TD>
<TD WIDTH=419><IMG SRC="../clearpixel.gif" WIDTH=419 HEIGHT=1 BORDER=0></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=419><P ALIGN=CENTER><B><FONT SIZE="+2">ORDER INFORMATION</FONT></B></TD>
</TR>
</TABLE>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=56 HEIGHT=33><IMG SRC="../clearpixel.gif" WIDTH=56 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=457>
<TABLE ID="Table1" BORDER=1 CELLSPACING=3 CELLPADDING=1 WIDTH=457>
<TR>
<TD WIDTH=219><P>RedBook Product Number</TD>
<TD WIDTH=219><P><INPUT ID="FormsEditField2" TYPE=TEXT NAME="product_id" VALUE=<%=
(String)session.getValue("FormNumber")%> SIZE=27 MAXLENGTH=27> </TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
```

```
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=362>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=64 HEIGHT=22><IMG SRC="../clearpixel.gif" WIDTH=64 HEIGHT=1 BORDER=0></TD>
<TD WIDTH=298><IMG SRC="../clearpixel.gif" WIDTH=298 HEIGHT=1 BORDER=0></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=298><P ALIGN=CENTER><FONT SIZE="+1">Shipping Details</FONT></TD>
</TR>
</TABLE>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=62 HEIGHT=16><IMG SRC="../clearpixel.gif" WIDTH=62 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=515>
<TABLE ID="Table3" BORDER=1 CELLSPACING=3 CELLPADDING=1 WIDTH=515>
<TR>
<TD WIDTH=216><P>Customer Name</TD>
<TD WIDTH=280><P><%= session.getValue("last_name")%> </TD>
</TR>
<TR>
<TD WIDTH=216><P>Address</TD>
<TD WIDTH=280><P><%= (String)session.getValue("address")%> </TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=422>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=167 HEIGHT=37><IMG SRC="../clearpixel.gif" WIDTH=167 HEIGHT=1 BORDER=0></TD>
<TD></TD>
<TD WIDTH=15><IMG SRC="../clearpixel.gif" WIDTH=15 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><INPUT NAME="submit" TYPE=IMAGE BORDER=0
SRC="../submit.gif" BORDER=0></A></TD>
<TD></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A HREF="javascript:document.forms[0].reset()"
TARGET="body"><IMG ID="Picture20" HEIGHT=17 WIDTH=120 SRC="../clear.gif"
BORDER=0></A></TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

D.5 Body_password.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<HTML>
<HEAD>
<BASE TARGET="_top">
</HEAD>
<BODY BGCOLOR="#FFFFFF" LINK="#0000FF" VLINK="#800080" TEXT="#000000" TOPMARGIN=2
LEFTMARGIN=2 MARGINWIDTH=2 MARGINHEIGHT=2>
```

```
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=493>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=66 HEIGHT=26><IMG SRC="../clearpixel.gif" WIDTH=66 HEIGHT=1 BORDER=0></TD>
<TD WIDTH=47><IMG SRC="../clearpixel.gif" WIDTH=47 HEIGHT=1 BORDER=0></TD>
<TD WIDTH=277><IMG SRC="../clearpixel.gif" WIDTH=277 HEIGHT=1 BORDER=0></TD>
<TD WIDTH=103><IMG SRC="../clearpixel.gif" WIDTH=103 HEIGHT=1 BORDER=0></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2></TD>
<TD WIDTH=277><P ALIGN=CENTER><B><FONT SIZE="+2">Password update form</FONT></B></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=4 HEIGHT=3></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=300></TD>
<TD WIDTH=427 COLSPAN=3>
<FORM NAME="changepassword" ACTION="/servlet/AccessUserstuf" METHOD=POST>
<INPUT TYPE=HIDDEN NAME="action" VALUE="changepassword">
<INPUT TYPE=HIDDEN NAME="success" VALUE="/Redbooks/password/passwdsuccess.jsp">
<INPUT TYPE=HIDDEN NAME="failure" VALUE="/Redbooks/password/passwdfailure.jsp">
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=52 HEIGHT=35><IMG SRC="../clearpixel.gif" WIDTH=52 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=333>
<TABLE ID="Table4" BORDER=1 CELLSPACING=3 CELLPADDING=1 WIDTH=333>
<TR>
<TD WIDTH=157><P>Enter new password</TD>
<TD WIDTH=157><P><INPUT ID="FormsEditField1" TYPE=PASSWORD NAME="password1" VALUE=""
SIZE=19 MAXLENGTH=19> </TD>
</TR>
<TR>
<TD WIDTH=157><P>Re-enter new password</TD>
<TD WIDTH=157><P><INPUT ID="FormsEditField2" TYPE=PASSWORD NAME="password2" VALUE=""
SIZE=19 MAXLENGTH=19> </TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=359>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=58 HEIGHT=26><IMG SRC="../clearpixel.gif" WIDTH=58 HEIGHT=1 BORDER=0></TD>
<TD></TD>
<TD WIDTH=61><IMG SRC="../clearpixel.gif" WIDTH=61 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=3 HEIGHT=1></TD>
<TD WIDTH=120 ROWSPAN=2 ALIGN=LEFT VALIGN=TOP><A
HREF="javascript:document.forms[0].reset()" TARGET="body"><IMG ID="Picture34" HEIGHT=17
WIDTH=120 SRC="../clear.gif" BORDER=0></A></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=16></TD>
<TD WIDTH=120 ROWSPAN=2><INPUT ID="FormsButton2" NAME="FormsButton2" TYPE=IMAGE BORDER=0
SRC="../submit.gif"></TD>
<TD></TD>
```

```
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=1></TD>
<TD COLSPAN=2></TD>
</TR>
</TABLE>
</FORM>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

D.6 Body_registrationpage.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<%@ LANGUAGE="Java" %>
<HTML>
<HEAD>
<BASE TARGET="_parent">
</HEAD>
<BODY BGCOLOR="#FFFFFF" LINK="#0000FF" VLINK="#800080" TEXT="#000000" TOPMARGIN=2
LEFTMARGIN=2 MARGINWIDTH=2 MARGINHEIGHT=2>
<FORM NAME="registration" ACTION="/servlet/AccessUserstuf" METHOD=POST>
<INPUT TYPE=HIDDEN NAME="action" VALUE="register">
<INPUT TYPE=HIDDEN NAME="success"
VALUE="/Redbooks/RegistrationPage/registersuccess.jsp">
<INPUT TYPE=HIDDEN NAME="failure"
VALUE="/Redbooks/RegistrationPage/registerfailure.jsp">
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=30 HEIGHT=10><IMG SRC="../clearpixel.gif" WIDTH=30 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=459>
<TABLE ID="Table5" BORDER=1 CELLSPACING=3 CELLPADDING=1 WIDTH=459>
<TR>
<TD WIDTH=198><P>
<TABLE WIDTH=96 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>First Name</TD>
</TR>
</TABLE>
 </TD>
<TD WIDTH=242><P><INPUT ID="FormsEditField8" TYPE=TEXT NAME="first_name" VALUE="" SIZE=30
MAXLENGTH=30> </TD>
</TR>
<TR>
<TD WIDTH=198><P>
<TABLE WIDTH=95 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>Last Name</TD>
</TR>
</TABLE>
 </TD>
<TD WIDTH=242><P><INPUT ID="FormsEditField5" TYPE=TEXT NAME="last_name" VALUE="" SIZE=30
MAXLENGTH=30> </TD>
</TR>
<TR>
```

```
<TD WIDTH=198><P>
<TABLE WIDTH=95 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>E-Mail </TD>
</TR>
</TABLE>
 </TD>
<TD WIDTH=242><P><INPUT ID="FormsEditField3" TYPE=TEXT NAME="email_address" VALUE=""
SIZE=30 MAXLENGTH=30> </TD>
</TR>
<TR>
<TD WIDTH=198><P>
<TABLE WIDTH=96 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>Snail Mail</TD>
</TR>
</TABLE>
 </TD>
<TD WIDTH=242><P><TEXTAREA WRAP=PHYSICAL ID="FormsMultiLine1" NAME="address" VALUE=""
ROWS=5 COLS=28></TEXTAREA> </TD>
</TR>
<TR>
<TD WIDTH=198><P>
<TABLE WIDTH=99 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>Birth Date </TD>
</TR>
</TABLE>
 </TD>
<TD WIDTH=242><P><INPUT ID="FormsEditField6" TYPE=TEXT NAME="birth_date"
VALUE="dd/mm/yyyy" SIZE=18 MAXLENGTH=18> </TD>
</TR>
<TR>
<TD WIDTH=198 HEIGHT=19><P>User Name</TD>
<TD WIDTH=242><P><INPUT ID="FormsEditField9" TYPE=TEXT NAME="username" VALUE="Enter a
username" SIZE=18 MAXLENGTH=25> </TD>
</TR>
<TR>
<TD WIDTH=198><P>Password </TD>
<TD WIDTH=242><P><INPUT ID="FormsEditField10" TYPE=PASSWORD NAME="password" VALUE="enter
a password" SIZE=17 MAXLENGTH=8> </TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=445>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=26 HEIGHT=19><IMG SRC="../clearpixel.gif" WIDTH=26 HEIGHT=1 BORDER=0></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=419><P>In the Table below,  select which platform interests you. </TD>
</TR>
</TABLE>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=24 HEIGHT=9><IMG SRC="../clearpixel.gif" WIDTH=24 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=375>
```

```
<TABLE ID="Table1" BORDER=1 CELLSPACING=3 CELLPADDING=1 WIDTH=375>
<TR>
<TD WIDTH=179><P>
<TABLE WIDTH=125 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>AS/400</TD>
</TR>
</TABLE>
<INPUT ID="FormsCheckbox2" TYPE=CHECKBOX NAME="as400" VALUE="as400"> </TD>
<TD WIDTH=177><P>
<TABLE WIDTH=127 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>Networking</TD>
</TR>
</TABLE>
<INPUT ID="FormsCheckbox3" TYPE=CHECKBOX NAME="nw" VALUE="nw"> </TD>
</TR>
<TR>
<TD WIDTH=179><P>
<TABLE WIDTH=125 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>RS/6000</TD>
</TR>
</TABLE>
<INPUT ID="FormsCheckbox6" TYPE=CHECKBOX NAME="rs6000" VALUE="rs6000"> </TD>
<TD WIDTH=177><P>
<TABLE WIDTH=125 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>VM/ESA</TD>
</TR>
</TABLE>
<INPUT ID="FormsCheckbox9" TYPE=CHECKBOX NAME="vm" VALUE="vm"> </TD>
</TR>
<TR>
<TD WIDTH=179><P>
<TABLE WIDTH=125 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>S/390</TD>
</TR>
</TABLE>
<INPUT ID="FormsCheckbox7" TYPE=CHECKBOX NAME="s390" VALUE="s390"> </TD>
<TD WIDTH=177><P>
<TABLE WIDTH=125 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>VSE/ESA</TD>
</TR>
</TABLE>
<INPUT ID="FormsCheckbox8" TYPE=CHECKBOX NAME="vse" VALUE="vse"> </TD>
</TR>
<TR>
<TD WIDTH=179><P>
<TABLE WIDTH=125 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>SP</TD>
</TR>
</TABLE>
<INPUT ID="FormsCheckbox5" TYPE=CHECKBOX NAME="scalable" VALUE="scalable"> </TD>
<TD WIDTH=177><P>
<TABLE WIDTH=125 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>Unclassified</TD>
</TR>
</TABLE>
```

```
<INPUT ID="FormsCheckbox10" TYPE=CHECKBOX NAME="other" VALUE="other"> </TD>
</TR>
<TR>
<TD WIDTH=179><P>
<TABLE WIDTH=125 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>Netfinity</TD>
</TR>
</TABLE>
<INPUT ID="FormsCheckbox4" TYPE=CHECKBOX NAME="netfinity" VALUE="netfinity"> </TD>
<TD WIDTH=177><P>
<TABLE WIDTH=125 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>Windows</TD>
</TR>
</TABLE>
<INPUT ID="FormsCheckbox11" TYPE=CHECKBOX NAME="windows" VALUE="windows"> </TD>
</TABLE>
</TD>
</TR>
</TABLE>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=469>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=114 HEIGHT=8><IMG SRC="../clearpixel.gif" WIDTH=114 HEIGHT=1 BORDER=0></TD>
<TD></TD>
<TD WIDTH=115><IMG SRC="../clearpixel.gif" WIDTH=115 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120><INPUT ID="FormsButton2" NAME="FormsButton2" TYPE=IMAGE BORDER=0
SRC="../submit.gif"></TD>
<TD></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A HREF="javascript:document.forms[0].reset()"
TARGET="body"><IMG ID="Picture13" HEIGHT=17 WIDTH=120 SRC="../clear.gif"
BORDER=0></A></TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

D.7 Body_update.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<%@ LANGUAGE="Java" %>
<HTML>
<HEAD>
<BASE TARGET="_parent">
</HEAD>
<% HttpSession session=request.getSession(true);%>

<BODY BGCOLOR="#FFFFFF" LINK="#0000FF" VLINK="#800080" TEXT="#000000" TOPMARGIN=2
LEFTMARGIN=2 MARGINWIDTH=2 MARGINHEIGHT=2>
<FORM NAME="updatepage"  ACTION="/servlet/AccessUserstuf" METHOD=POST>
<INPUT TYPE=HIDDEN NAME="action" VALUE="update">
<INPUT TYPE=HIDDEN NAME="success" VALUE="/Redbooks/UpdatePage/updatesuccess.jsp">
<INPUT TYPE=HIDDEN NAME="failure" VALUE="/Redbooks/UpdatePage/updatefailure.jsp">
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=433>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=160 HEIGHT=30><IMG SRC="../clearpixel.gif" WIDTH=160 HEIGHT=1 BORDER=0></TD>
```

```
<TD WIDTH=273><IMG SRC="../clearpixel.gif" WIDTH=273 HEIGHT=1 BORDER=0></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=273><P ALIGN=CENTER><B><FONT SIZE="+1">USERPROFILE FOR :<%=
session.getValue("username")%></FONT></B></TD>
</TR>
</TABLE>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=40 HEIGHT=62><IMG SRC="../clearpixel.gif" WIDTH=40 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=459>
<TABLE ID="Table5" BORDER=1 CELLSPACING=3 CELLPADDING=1 WIDTH=459>
<TR>
<TD WIDTH=199><P>
<TABLE WIDTH=96 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>First Name</TD>
</TR>
</TABLE>
 </TD>
<TD WIDTH=241><P><INPUT ID="FormsEditField8" TYPE=TEXT NAME="first_name" VALUE="<%=
session.getValue("first_name")%>" SIZE=30 MAXLENGTH=30> </TD>
</TR>
<TR>
<TD WIDTH=199><P>
<TABLE WIDTH=95 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>Last Name</TD>
</TR>
</TABLE>
 </TD>
<TD WIDTH=241><P><INPUT ID="FormsEditField5" TYPE=TEXT NAME="last_name" VALUE="<%=
session.getValue("last_name") %>" SIZE=30 MAXLENGTH=30> </TD>
</TR>
<TR>
<TD WIDTH=199><P>
<TABLE WIDTH=95 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>E-Mail </TD>
</TR>
</TABLE>
 </TD>
<TD WIDTH=241><P><INPUT ID="FormsEditField3" TYPE=TEXT NAME="email_address" VALUE="<%=
session.getValue("email_address") %> " SIZE=30 MAXLENGTH=30> </TD>
</TR>
<TR>
<TD WIDTH=199><P>
<TABLE WIDTH=96 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>Snail Mail</TD>
</TR>
</TABLE>
 </TD>
<TD WIDTH=241><P><TEXTAREA WRAP=PHYSICAL ID="FormsMultiLine1" NAME="address" ROWS=5
COLS=28><%= session.getValue("address") %></TEXTAREA> </TD>
</TR>
<TR>
<TD WIDTH=199><P>
```

```
<TABLE WIDTH=99 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>Birth Date </TD>
</TR>
</TABLE>
 </TD>
<TD WIDTH=241><P><INPUT ID="FormsEditField6" TYPE=TEXT NAME="birth_date" VALUE="<%=
session.getValue("birth_date") %>" SIZE=18 MAXLENGTH=18> </TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=460>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=41 HEIGHT=3><IMG SRC="../clearpixel.gif" WIDTH=41 HEIGHT=1 BORDER=0></TD>
<TD WIDTH=419><IMG SRC="../clearpixel.gif" WIDTH=419 HEIGHT=1 BORDER=0></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=419><P>In the Table below,  select which platform interests you. </TD>
</TR>
</TABLE>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=41 HEIGHT=3><IMG SRC="../clearpixel.gif" WIDTH=41 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=375>
<TABLE ID="Table1" BORDER=1 CELLSPACING=3 CELLPADDING=1 WIDTH=375>
<TR>
<TD WIDTH=179><P>
<TABLE WIDTH=125 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>AS/400</TD>
</TR>
</TABLE>
<INPUT ID="FormsCheckbox2" TYPE=CHECKBOX <% if (session.getValue("as400")!=null) { %>
CHECKED <% } %>   NAME="as400" VALUE="as400"> </TD>
<TD WIDTH=177><P>
<TABLE WIDTH=127 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>Networking</TD>
</TR>
</TABLE>
<INPUT ID="FormsCheckbox3" TYPE=CHECKBOX <% if (session.getValue("nw")!=null) { %>
CHECKED <% } %> NAME="nw" VALUE="nw"> </TD>
</TR>
<TR>
<TD WIDTH=179 HEIGHT=19><P>
<TABLE WIDTH=125 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>RS/6000</TD>
</TR>
</TABLE>
<INPUT ID="FormsCheckbox6" TYPE=CHECKBOX <% if (session.getValue("rs6000")!=null) { %>
CHECKED <% } %> NAME="rs6000" VALUE="rs6000"> </TD>
<TD WIDTH=177><P>
<TABLE WIDTH=125 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>VM/ESA</TD>
```

```
</TR>
</TABLE>
<INPUT ID="FormsCheckbox9" TYPE=CHECKBOX <% if (session.getValue("vm")!=null) { %>
CHECKED <% } %> NAME="vm" VALUE="vm"> </TD>
</TR>
<TR>
<TD WIDTH=179><P>
<TABLE WIDTH=125 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>S/390</TD>
</TR>
</TABLE>
<INPUT ID="FormsCheckbox7" TYPE=CHECKBOX <% if (session.getValue("s390")!=null) { %>
CHECKED <% } %> NAME="s390" VALUE="s390"> </TD>
<TD WIDTH=177><P>
<TABLE WIDTH=125 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>VSE/ESA</TD>
</TR>
</TABLE>
<INPUT ID="FormsCheckbox8" TYPE=CHECKBOX <% if (session.getValue("vse")!=null) { %>
CHECKED <% } %> NAME="vse" VALUE="vse"> </TD>
</TR>
<TR>
<TD WIDTH=179><P>
<TABLE WIDTH=125 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>SP</TD>
</TR>
</TABLE>
<INPUT ID="FormsCheckbox5" TYPE=CHECKBOX <% if (session.getValue("scalable")!=null) { %>
CHECKED <% } %> NAME="scalable" VALUE="scalable"> </TD>
<TD WIDTH=177><P>
<TABLE WIDTH=125 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>Unclassified</TD>
</TR>
</TABLE>
<INPUT ID="FormsCheckbox10" TYPE=CHECKBOX <% if(session.getValue("other")!=null) { %>
CHECKED <% } %> NAME="other" VALUE="other"> </TD>
</TR>
<TR>
<TD WIDTH=179><P>
<TABLE WIDTH=125 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>Netfinity</TD>
</TR>
</TABLE>
<INPUT ID="FormsCheckbox4" TYPE=CHECKBOX <% if (session.getValue("netfinity")!=null) { %>
CHECKED <% } %> NAME="netfinity" VALUE="netfinity"> </TD>
<TD WIDTH=177><P>
<TABLE WIDTH=125 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>Windows</TD>
</TR>
</TABLE>
<INPUT ID="FormsCheckbox11" TYPE=CHECKBOX <% if(session.getValue("windows")!=null) { %>
CHECKED <% } %> NAME="windows" VALUE="windows"> </TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
```

```
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=407>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=75 HEIGHT=17><IMG SRC="../clearpixel.gif" WIDTH=75 HEIGHT=1 BORDER=0></TD>
<TD></TD>
<TD WIDTH=92><IMG SRC="../clearpixel.gif" WIDTH=92 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120><INPUT ID="FormsButton3" NAME="FormsButton3" TYPE=IMAGE BORDER=0
SRC="../submit.gif"></TD>
<TD></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A HREF="javascript:document.forms[0].reset()"
TARGET="body"><IMG ID="Picture13" HEIGHT=17 WIDTH=120 SRC="../clear.gif"
BORDER=0></A></TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

D.8 Header_defaultmasterborder.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
<META NAME="Generator" CONTENT="NetObjects Fusion 4.0.1 for Windows">
<BASE TARGET="_parent">
</HEAD>
<BODY BGCOLOR="#FFFFFF" LINK="#0000FF" VLINK="#800080" TEXT="#000000" TOPMARGIN=2
LEFTMARGIN=2 MARGINWIDTH=2 MARGINHEIGHT=2>
    <TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=772>
        <TR VALIGN=TOP ALIGN=LEFT>
            <TD WIDTH=188 HEIGHT=18><IMG SRC="./clearpixel.gif" WIDTH=188 HEIGHT=1
BORDER=0></TD>
            <TD></TD>
        </TR>
        <TR VALIGN=TOP ALIGN=LEFT>
            <TD HEIGHT=72></TD>
            <TD WIDTH=584 ALIGN=LEFT VALIGN=TOP><IMG ID="Picture18" HEIGHT=72 WIDTH=584
SRC="./redmast.gif" BORDER=0></TD>
        </TR>
    </TABLE>
</BODY>
</HTML>
```

D.9 Homepage.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<%@ LANGUAGE="Java" %>
<HTML>
<HEAD>
<FRAMESET ROWS="142,*">
    <FRAME NAME="header" SRC="../header_defaultmasterborder.html" SCROLLING=AUTO
MARGINWIDTH="2" MARGINHEIGHT="1" BORDER=5 NORESIZE>
    <FRAMESET COLS="152,*">
        <FRAME NAME="left" SRC="../HomePage/left_homepage.html" SCROLLING=AUTO
MARGINWIDTH="2" MARGINHEIGHT="1" BORDER=5 NORESIZE>
```

```
        <FRAME NAME="body" SRC="../HomePage/body_homepage.jsp" SCROLLING=AUTO
MARGINWIDTH=2 MARGINHEIGHT=2>
    </FRAMESET>
</FRAMESET>
</HEAD>
</HTML>
```

D.10 Index.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<HTML>
<HEAD>
<FRAMESET ROWS="142,*" >
    <FRAME NAME="header" SRC="./header_defaultmasterborder.html" SCROLLING=AUTO
MARGINWIDTH="0" MARGINHEIGHT="0" BORDER=0 NORESIZE>
    <FRAMESET COLS="152,*">
       <FRAME NAME="left" SRC="./left_index.html" SCROLLING=AUTO MARGINWIDTH="0"
MARGINHEIGHT="0" BORDER=0  NORESIZE>
       <FRAME NAME="body" SRC="./body_index.jsp" SCROLLING=AUTO MARGINWIDTH=0
MARGINHEIGHT=0>
    </FRAMESET>
</FRAMESET>
</HEAD>
</HTML>
```

D.11 Left_homepage.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<HTML>
<HEAD>
<BASE TARGET="_parent"
</HEAD>
<BODY BGCOLOR="#FFFFFF" LINK="#0000FF" VLINK="#800080" TEXT="#000000">
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=126>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=6 HEIGHT=56><IMG SRC="../clearpixel.gif" WIDTH=6 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP HEIGHT=22><FORM NAME="searchform"
ACTION="/servlet/VBDSearch" METHOD=POST TARGET=body>
<TR VALIGN=TOP ALIGN=LEFT>
</TR>
<TR VALIGN=TOP ALIGN=LEFT><TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP HEIGHT=22><INPUT ID="FormsEd
itField12"TYPE=TEXT NAME="search" VALUE="" SIZE=15 MAXLENGTH=30></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 HEIGHT=17><INPUT TYPE=IMAGE BORDER=0  SRC="../Search_database.gif" BORDER=0
ALT="search">
</TD>
</TR>
</FORM>
```

```
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A
HREF="/servlet/AccessUserstuf?action=listorders&success=/Redbooks/ListOrdersPage/listord
erspage.jsp&failure=/Redbooks/ListOrdersPage/listorderfailure.jsp" TARGET="body"><IMG
HEIGHT=17 WIDTH=120 SRC="../previous_orders.gif" BORDER=0 ALT="previous orders"></A></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A HREF="../logout.jsp"><IMG ID="Picture16" HEIGHT=17
WIDTH=120 SRC="../logout.gif" BORDER=0 ALT="home"></A></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A HREF="../OrderPage/orderpage.html"><IMG
ID="Picture30" HEIGHT=17 WIDTH=120 SRC="../orderbook.gif" BORDER=0 ALT="previous
orders"></A></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=10></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A HREF="../UpdatePage/body_update.jsp"
TARGET="body"><IMG ID="Picture33" HEIGHT=17 WIDTH=120 SRC="../update_profile.gif"
BORDER=0 ALT="update profile"></A></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=10></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A HREF="../password/body_password.jsp"
TARGET="body"><IMG ID="Picture35" HEIGHT=17 WIDTH=120 SRC="../changepassword.gif"
BORDER=0 ALT="ChangePassword"></A></TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

D.12 Left_index.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<HTML>
<HEAD>
<BASE TARGET="_parent">
</HEAD>
<BODY BGCOLOR="#FFFFFF" LINK="#0000FF" VLINK="#800080" TEXT="#000000">

<FORM NAME="search" ACTION="/servlet/VBDSearch" METHOD=POST TARGET=body>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=126>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=6 HEIGHT=56><IMG SRC="./clearpixel.gif" WIDTH=6 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
```

```
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP HEIGHT=22><INPUT ID="FormsEditField12"TYPE=TEXT
NAME="search" VALUE="" SIZE=15 MAXLENGTH=30></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ><INPUT ID"FormsButton6" NAME="FormsButton6" TYPE=IMAGE BORDER=0
SRC="./Search_database.gif"></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A
HREF="./RegistrationPage/registrationpage.jsp"><IMG ID="Picture15" HEIGHT=17 WIDTH=120
SRC="./New_Users.gif" BORDER=0 ALT="new users"></A></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A HREF="./LoginPage/loginpage.jsp"><IMG
ID="Picture17" HEIGHT=17 WIDTH=120 SRC="./Registered_users.gif" BORDER=0 ALT="registered
users"></A></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

D.13 Left_loginpage.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<HTML>
<HEAD>
<BASE TARGET="_parent">
</HEAD>
<BODY BGCOLOR="#FFFFFF" LINK="#0000FF" VLINK="#800080" TEXT="#000000">
<FORM NAME="login" ACTION="/servlet/AccessUserInfo" METHOD=POST>
<INPUT TYPE=HIDDEN NAME="action" VALUE="login">
```

```
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=126>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=6 HEIGHT=56><IMG SRC="../clearpixel.gif" WIDTH=6 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD>
</TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A HREF="../RegistrationPage/registrationpage.jsp"
TARGET="body"><IMG ID="Picture15" HEIGHT=17 WIDTH=120 SRC="../New_Users.gif"
TARGET="body" BORDER=0 ALT="new users"></A></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A HREF="../index.html"><IMG ID="Picture16" HEIGHT=17
WIDTH=120 SRC="../home.gif" BORDER=0 ALT="home"></A></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD></TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

D.14 Left_orderpage.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<HTML>
<HEAD>
<BASE TARGET="_parent">
</HEAD>
<BODY BGCOLOR="#FFFFFF" LINK="#0000FF" VLINK="#800080" TEXT="#000000">
<FORM NAME="LAYOUTFORM" ACTION="" METHOD=POST>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=126>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=6 HEIGHT=56><IMG SRC="../clearpixel.gif" WIDTH=6 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD
</TD>
</TR>
```

```
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A
HREF="../RegistrationPage/registrationpage.jsp"><IMG ID="Picture15" HEIGHT=17 WIDTH=120
SRC="../New_Users.gif" BORDER=0 ALT="new users"></A></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A HREF="../LoginPage/loginpage.jsp"><IMG
ID="Picture17" HEIGHT=17 WIDTH=120 SRC="../Registered_users.gif" BORDER=0 ALT="registered
users"></A></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A HREF="../HomePage/homepage.jsp"><IMG
ID="Picture16" HEIGHT=17 WIDTH=120 SRC="../home.gif" BORDER=0 ALT="home"></A></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A
HREF="/servlet/AccessUserstuf?action=listorders&success=/Redbooks/ListOrdersPage/listord
erspage.jsp" TARGET="body"><IMG ID="Picture30" HEIGHT=17 WIDTH=120 SRC="../orders.gif"
BORDER=0 ALT="previous orders"></A></TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

D.15 Left_registrationpage.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<HTML>
<HEAD>
<BASE TARGET="_parent">
</HEAD>
<BODY BGCOLOR="#FFFFFF" LINK="#0000FF" VLINK="#800080" TEXT="#000000">
<FORM NAME="registration" ACTION="/servlet/InsertUserinfo" METHOD=POST>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=126>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=6 HEIGHT=56><IMG SRC="../clearpixel.gif" WIDTH=6 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
```

```
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A HREF="../RegistrationPage/registrationpage.jsp"
TARGET="body"><IMG ID="Picture15" HEIGHT=17 WIDTH=120 SRC="../New_Users.gif" BORDER=0
ALT="new users"></A></TD>
</TR
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A HREF="../LoginPage/loginpage.jsp"><IMG
ID="Picture17" HEIGHT=17 WIDTH=120 SRC="../Registered_users.gif" BORDER=0 ALT="registered
users"></A></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A HREF="../index.html"><IMG ID="Picture16" HEIGHT=17
WIDTH=120 SRC="../home.gif" BORDER=0 ALT="home"></A></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD></TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

D.16 Left_registrationpage.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<HTML>
<HEAD>
<BASE TARGET="_parent">
</HEAD>
<% HttpSession session=request.getSession(true);%>

<BODY BGCOLOR="#FFFFFF" LINK="#0000FF" VLINK="#800080" TEXT="#000000">
<FORM NAME="registration" ACTION="/servlet/AccessUserInfo" METHOD=POST>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=126>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=6 HEIGHT=56><IMG SRC="../clearpixel.gif" WIDTH=6 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
```

```
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A
HREF="../RegistrationPage/registrationpage.jsp"><IMG ID="Picture15" HEIGHT=17 WIDTH=120
SRC="../New_Users.gif" BORDER=0 ALT="new users"></A></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A HREF="../LoginPage/loginpage.jsp"><IMG
ID="Picture17" HEIGHT=17 WIDTH=120 SRC="../Registered_users.gif" BORDER=0 ALT="registered
users"></A></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A HREF="../index.html"><IMG ID="Picture16" HEIGHT=17
WIDTH=120 SRC="../home.gif" BORDER=0 ALT="home"></A></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD></TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

D.17 Left_updatepage.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<HTML>
<HEAD>
<BASE TARGET="_parent">
</HEAD>
<BODY BGCOLOR="#FFFFFF" LINK="#0000FF" VLINK="#800080" TEXT="#000000">
<FORM NAME="LAYOUTFORM" ACTION="" METHOD=POST>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=126>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=6 HEIGHT=56><IMG SRC="../clearpixel.gif" WIDTH=6 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD>
</TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A
HREF="../RegistrationPage/registrationpage.html"><IMG ID="Picture15" HEIGHT=17 WIDTH=120
SRC="../New_Users.gif" BORDER=0 ALT="new users"></A></TD>
</TR>
```

```
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A HREF="../LoginPage/loginpage.html"><IMG
ID="Picture17" HEIGHT=17 WIDTH=120 SRC="../Registered_users.gif" BORDER=0 ALT="registered
users"></A></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A HREF="../HomePage/homepae.jsp"><IMG ID="Picture16"
HEIGHT=17 WIDTH=120 SRC="../home.gif" BORDER=0 ALT="home"></A></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD COLSPAN=2 HEIGHT=9></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP><A HREF="../OrderPage/orderpage.html"><IMG
ID="Picture30" HEIGHT=17 WIDTH=120 SRC="../orders.gif" BORDER=0 ALT="previous
orders"></A></TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

D.18 Listorderfailure.jsp

```
<HTML><HEAD><% HttpSession session=request.getSession(true);%>
</HEAD>
<BODY>
<CENTER>
<P><H3 ALIGN=CENTER>An error has occurred</H3></P>
<P><H4 ALIGN=CENTER><%= (String)session.getValue("error")%>
<% session.removeValue("error"); %></P><P>
<B>Return</B></P>
<P>
<A HREF="../OrderPage/orderpage.html"><IMG  HEIGHT=17 WIDTH=120 SRC="../home.gif"
BORDER=0 ALT="Homepage"></A></P><CENTER></BODY></HTML>
```

D.19 Listorderspage.jsp

```
<%@ LANGUAGE="java" %>
<html>
<head>
<base target="_parent">
</head>
<body>
<bean
  name="orders"
  type="penguins.access.db2.Orders"
>
</bean>
ListOrdersPage <p>
```

```
<table border=1>
<th>Order id</th><th>Timestamp</th><th>Product id</th>
<%@ import="java.util.*" %>
<%
for (Enumeration e=orders.orderIDs(); e.hasMoreElements();)
{
  String key=(String) e.nextElement();

out.println("<tr><td>"+key+"</td><td>"+orders.timestamp(key)+"</td><td>"+orders.productI
D(key)+"</td>");
}
%>
</table>
</body>
</html>
```

D.20 Loginfailure.jsp

```
<HTML>
<HEAD>
<% HttpSession session=request.getSession(true);%>
</HEAD>
<BODY>
<CENTER>
<P>
<H3 ALIGN=CENTER>An error has occurred</H3>
</P>
<P>
<H4 ALIGN=CENTER><%= (String)session.getValue("error")%>
<% session.removeValue("error"); %>
</P>
<P>
<B>Return</B>
</P>
<P>
<A HREF="../LoginPage/loginpage.jsp"><IMG  HEIGHT=17 WIDTH=120 SRC="../home.gif"BORDER=0
ALT="Homepage"></A>
</P>
<CENTER>
</BODY>
</HTML>
```

D.21 Loginpage.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<%@ LANGUAGE="Java" %>
<HTML>
<HEAD>
<FRAMESET ROWS="142,*">
<FRAME NAME="header" SRC="../header_defaultmasterborder.html" SCROLLING=AUTO
MARGINWIDTH="2" MARGINHEIGHT="1" BORDER=5 NORESIZE>
<FRAMESET COLS="152,*">
<FRAME NAME="left" SRC="../LoginPage/left_loginpage.html" SCROLLING=AUTO MARGINWIDTH="2"
MARGINHEIGHT="1" BORDER=5 NORESIZE>
<FRAME NAME="body" SRC="../LoginPage/body_loginpage.jsp" SCROLLING=AUTO MARGINWIDTH=2
MARGINHEIGHT=2>
</FRAMESET>
</FRAMESET>
</HEAD>
```

```
</HTML>
```

D.22 Logout.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<%@ LANGUAGE="Java" %>
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
<BASE TARGET="_parent">
<%
  HttpSession session=request.getSession(true);
  String username=(String)session.getValue("username");
  if (username == null || username.equals(""))
  {
    session.putValue("error","You are not logged on");
  } else {
    //session.removeValue("username");
    session.invalidate();
  }
%>
</HEAD>
<BODY>
<%
  HttpSession s1=request.getSession(true);
  String loguser=(String)s1.getValue("username");
  if (loguser == null || loguser.equals(""))
  {
%>
  <CENTER>
  <P><B>Logoff Successful</B>
  <P><A HREF="./index.html"><IMG SRC="./logout.gif" ALT="back"></A>
  </CENTER>
<%
  } else {
%>
  <CENTER>
  <P><B>Logoff failed
  <P>Username = <%= loguser %>
  </CENTER>
<%
  }
%>
</BODY>
</HTML>
```

D.23 orderfailure.jsp

```
<HTML>
<HEAD>
<% HttpSession session=request.getSession(true);%>
</HEAD>
<BODY>
<CENTER>
<P>
<H3 ALIGN=CENTER>An error has occurred</H3>
</P>
<P>
<H4 ALIGN=CENTER><%= (String)session.getValue("error")%>
```

```
<% session.removeValue("error"); %>
</P>
<P>
<B>Return</B>
</P>
<P>
<A HREF="../OrderPage/orderpage.html"><IMG  HEIGHT=17 WIDTH=120 SRC="../home.gif"
BORDER=0 ALT="Homepage"></A>
</P>
<CENTER>
</BODY>
</HTML>
```

D.24 Orderpage.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<HTML>
<HEAD>
<FRAMESET ROWS="142,*">
    <FRAME NAME="header" SRC="../header_defaultmasterborder.html" SCROLLING=AUTO
MARGINWIDTH="2" MARGINHEIGHT="1" BORDER=5 NORESIZE>
    <FRAMESET COLS="152,*">
        <FRAME NAME="left" SRC="../OrderPage/left_orderpage.html" SCROLLING=AUTO
MARGINWIDTH="2" MARGINHEIGHT="1" BORDER=5 NORESIZE>
        <FRAME NAME="body" SRC="../OrderPage/body_orderpage.jsp" SCROLLING=AUTO
MARGINWIDTH=2 MARGINHEIGHT=2>
    </FRAMESET>
</FRAMESET>
</HEAD>
</HTML>
```

D.25 Ordersuccess.jsp

```
<%@ LANGUAGE="java" %>
<HTML>
<HEAD>
<BASE TARGET="_parent">
<% HttpSession session=request.getSession(true);%>
</HEAD>
<BODY BGCOLOR="#FFFFFF" LINK="#0000FF" VLINK="#800080" TEXT="#000000" TOPMARGIN=2
LEFTMARGIN=2 MARGINWIDTH=2 MARGINHEIGHT=2>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=515>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=96 HEIGHT=17><IMG SRC="../clearpixel.gif" WIDTH=96 HEIGHT=1 BORDER=0></TD>
<TD WIDTH=419><IMG SRC="../clearpixel.gif" WIDTH=419 HEIGHT=1 BORDER=0></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=419><P ALIGN=CENTER><B><FONT SIZE="+2">ORDER INFORMATION</FONT></B></TD>
</TR>
</TABLE>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=56 HEIGHT=33><IMG SRC="../clearpixel.gif" WIDTH=56 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=457>
```

```
<TABLE ID="Table1" BORDER=1 CELLSPACING=3 CELLPADDING=1 WIDTH=457>
<TR>
<TD WIDTH=219><P>RedBook Product Number</TD>
<TD WIDTH=219><P><%= session.getValue("product_id")%>> </TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=362>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=64 HEIGHT=22><IMG SRC="../clearpixel.gif" WIDTH=64 HEIGHT=1 BORDER=0></TD>
<TD WIDTH=298><IMG SRC="../clearpixel.gif" WIDTH=298 HEIGHT=1 BORDER=0></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=298><P ALIGN=CENTER><FONT SIZE="+1">Shipping Details</FONT></TD>
</TR>
</TABLE>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=62 HEIGHT=16><IMG SRC="../clearpixel.gif" WIDTH=62 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=515>
<TABLE ID="Table3" BORDER=1 CELLSPACING=3 CELLPADDING=1 WIDTH=515>
<TR>
<TD WIDTH=216><P>Customer Name</TD>
<TD WIDTH=280><P><%= session.getValue("last_name")%> </TD>
</TR>
<TR>
<TD WIDTH=216><P>Address</TD>
<TD WIDTH=280><P><%= session.getValue("address")%> </TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=422>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=167 HEIGHT=37><IMG SRC="../clearpixel.gif" WIDTH=167 HEIGHT=1 BORDER=0></TD>
<TD></TD>
<TD WIDTH=15><IMG SRC="../clearpixel.gif" WIDTH=15 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD HEIGHT=17></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP>
</TD>
<TD></TD>
<TD WIDTH=120 ALIGN=LEFT VALIGN=TOP>
</TR>
</TABLE>
<CENTER>
<B>Return</B>
<P>
<A HREF="../HomePage/homepage.jsp"><IMG  HEIGHT=17 WIDTH=120 SRC="../home.gif" BORDER=0
ALT="HomePage"></A>
</BODY>
</HTML>
```

D.26 Passwdsuccess.jsp

```
<%@  LANGUAGE=”Java” %>
<HTML>
<HEAD>
    <% HttpSession session=request.getSession(true);%>
</HEAD>
<BODY>
<P>
<H3 ALIGN=CENTER>Your password has been successfully changed</H3>
</P>
<P>
<H4 ALIGN=CENTER>Your new password is:
<B ALIGN=CENTER><%= session.getValue(“password”) %></B></H4>
</P>
<P></P>
<CENTER>
<B> Return </B>
<P>
<A HREF=”../HomePage/homepage.jsp”><IMG  HEIGHT=17 WIDTH=120 SRC=”../home.gif” BORDER=0
ALT=”Homepage”></A>
</P>
</CENTER>
</BODY>
</HTML>
```

D.27 Passwdfailure.jsp

```
<HTML>
<HEAD>
<% HttpSession session=request.getSession(true);%>
</HEAD>
<BODY>
<CENTER>
<P>
<H3 ALIGN=CENTER>An error has occurred</H3>
</P>
<P>
<H4 ALIGN=CENTER><%= (String)session.getValue("error")%>
<% session.removeValue("error"); %>
</P>
<P>
<B>Return</B>
</P>
<P>
<A HREF="../HomePage/homepage.jsp"><IMG  HEIGHT=17 WIDTH=120 SRC="../home.g
if
"
BORDER=0 ALT="Homepage"></A>
</P>
<CENTER>
</BODY>
</HTML>
```

D.28 Registerfailure.jsp

```
<HTML>
<HEAD>
```

```
<% HttpSession session=request.getSession(true); %>
</HEAD>
<BODY>
<CENTER>
<P>
<H3 ALIGN=CENTER>An error has occurred</H3>
</P>
<P>
<H4 ALIGN=CENTER><%= (String)session.getValue("error")%>
<% session.removeValue("error"); %>
</P>
<P>
<B>Return</B>
</P>
<P>
<A HREF="../RegistrationPage/registrationpage.jsp"><IMG  HEIGHT=17 WIDTH=120
SRC="../home.gif
"
BORDER=0 ALT="Homepage"></A>
</P>
</CENTER>
</BODY>
</HTML>
```

D.29 Registersuccess.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<%@ LANGUAGE="Java" %>
<HTML>
<HEAD>
<BASE TARGET="_parent">
</HEAD>
<% HttpSession session=request.getSession(true);
%>

<BODY BGCOLOR="#FFFFFF" LINK="#0000FF" VLINK="#800080" TEXT="#000000" TOPMARGIN=2
LEFTMARGIN=2 MARGINWIDTH=2 MARGINHEIGHT=2>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=433>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=160 HEIGHT=30><IMG SRC="../clearpixel.gif" WIDTH=160 HEIGHT=1 BORDER=0></TD>
<TD WIDTH=273><IMG SRC="../clearpixel.gif" WIDTH=273 HEIGHT=1 BORDER=0></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=273><P ALIGN=CENTER><B><FONT SIZE="+1">USERPROFILE FOR : <%=
session.getValue("username")%></FONT></B></TD>
</TR>
</TABLE>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=40 HEIGHT=62><IMG SRC="../clearpixel.gif" WIDTH=40 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=459>
<TABLE ID="Table5" BORDER=1 CELLSPACING=3 CELLPADDING=1 WIDTH=459>
<TR>
<TD WIDTH=199><P>
<TABLE WIDTH=96 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
```

```
<TD><P>First Name</TD>
</TR>
</TABLE>
 </TD>
<TD WIDTH=241><P><%= session.getValue("first_name")%> </TD>
</TR>
<TR>
<TD WIDTH=199><P>
<TABLE WIDTH=95 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>Last Name</TD>
</TR>
</TABLE>
 </TD>
<TD WIDTH=241><P><%= session.getValue("last_name") %> </TD>
</TR>
<TR>
<TD WIDTH=199><P>
<TABLE WIDTH=95 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>E-Mail </TD>
</TR>
</TABLE>
 </TD>
<TD WIDTH=241><P><%= session.getValue("email_address") %> </TD>
</TR>
<TR>
<TD WIDTH=199><P>
<TABLE WIDTH=96 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>Snail Mail</TD>
</TR>
</TABLE>
 </TD>
<TD WIDTH=241><P><TEXTAREA WRAP=PHYSICAL ID="FormsMultiLine1" NAME="address" ROWS=5
COLS=28><%= session.getValue("address") %></TEXTAREA> </TD>
</TR>
<TR>
<TD WIDTH=199><P>
<TABLE WIDTH=99 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>Birth Date </TD>
</TR>
</TABLE>
 </TD>
<TD WIDTH=241><P><%= session.getValue("birth_date") %> </TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=460>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=41 HEIGHT=3><IMG SRC="../clearpixel.gif" WIDTH=41 HEIGHT=1 BORDER=0></TD>
<TD WIDTH=419><IMG SRC="../clearpixel.gif" WIDTH=419 HEIGHT=1 BORDER=0></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=419><P>In the Table below,  you have selected the following platforms
which interest you. </TD>
</TR>
</TABLE>
<TABLE width=55%>
```

```
<TR>
<TD> </TD>
<TD> </TD>
<TD> </TD>
<TD>
<TABLE  BORDER=1 CELLSPACING=3 CELLPADDING=1 WIDTH=70% >
    <TR>
        <TD WIDTH=72>AS/400 </TD>
        <TD WIDTH=73><% if (session.getValue("as400")!=null) {%>selected<%} else
 { %>not selected <%}%> </TD>
        <TD WIDTH=73>RS/6000 </TD>
        <TD WIDTH=73> <% if (session.getValue("rs6000")!=null) {%>selected
<%} else { %>not selected <%}%> </TD>
    </TR>
    <TR>
        <TD>Networking </TD>
    <TD><% if (session.getValue("nw")!=null) {%>selected<%} else {%>not
selected<%}%> </TD>
        <TD>VM </TD>
    <TD><% if (session.getValue("vm")!=null) {%>selected<%}else{%>not selected<%}%>
 </TD>
 </TR>
    <TR>
        <TD>VM/ESA </TD>
    <TD><% if (session.getValue("vse")!=null) {%>selected<%}else{%>not
selected<%}%> </TD>
        <TD>S390 </TD>
    <TD><% if (session.getValue("s390")!=null) {%>selected<%}else{%>not
selected<%}%> </TD>
    </TR>
    <TR>
        <TD>Other </TD>
<TD><% if (session.getValue("other")!=null) {%>selected<%}else{%>not
selected<%}%> </TD>
        <TD>Windows </TD>
<TD><% if (session.getValue("windows")!=null) {%>selected<%}else{%>not selected
<%}%> </TD>
    </TR>
    <TR>
        <TD>SP </TD>
<TD><% if (session.getValue("scalable")!=null){%>selected<%}else{%>not selected
<%}%> </TD>
        <TD> </TD>
        <TD> </TD>
   </TR>
</TABLE>
</TD>
</TR>
</TABLE>
<P>
<P>
<CENTER>
<B>Return<B>
<P>
<A HREF="../HomePage/homepage.jsp"><IMG  HEIGHT=17 WIDTH=120 SRC="../home.gif" B
ORDER=0 ALT="Homepage"></A>
</P>
</CENTER>
</BODY>
</HTML>
```

D.30 Registrationpage.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<%@ LANGUAGE="Java"%>
<HTML>
<HEAD>
<FRAMESET ROWS="142,*">
<FRAME NAME="header" SRC="../header_defaultmasterborder.html" SCROLLING=AUTO
MARGINWIDTH="2" MARGINHEIGHT="1" BORDER=5 NORESIZE>
<FRAMESET COLS="152,*">
<FRAME NAME="left" SRC="../RegistrationPage/left_registrationpage.html" SCROLLING=AUTO
MARGINWIDTH="2" MARGINHEIGHT="1" BORDER=5 NORESIZE>
<FRAME NAME="body" SRC="../RegistrationPage/body_registrationpage.jsp" SCROLLING=AUTO
MARGINWIDTH=2 MARGINHEIGHT=2>
</FRAMESET>
</FRAMESET>
</HEAD>
</HTML>
```

D.31 Updatefailure.jsp

```
<HTML>
<HEAD>
<% HttpSession session=request.getSession(true);%>
</HEAD>
<BODY>
<CENTER>
<P>
<H3 ALIGN=CENTER>An error has occurred</H3>
</P>
<P>
<H4 ALIGN=CENTER><%= (String)session.getValue("error")%>
<% session.removeValue("error"); %>
</P>
<P>
<B>Return</B>
</P>
<P>
<A HREF="../UpdatePage/body_update.jsp"><IMG  HEIGHT=17 WIDTH=120 SRC="../home.gif"
BORDER=0 ALT="Homepage"></A>
</P>
<CENTER>
</BODY>
</HTML>
```

D.32 Updatepage.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<HTML>
<HEAD>
<FRAMESET ROWS="142,*">
    <FRAME NAME="header" SRC="../header_defaultmasterborder.html" SCROLLING=AUTO
   MARGINWIDTH="2" MARGINHEIGHT="1" BORDER=5 NORESIZE>
    <FRAMESET COLS="152,*">
      <FRAME NAME="left" SRC="../UpdatePage/left_updatepage.html" SCROLLING=AUTO
   MARGINWIDTH="2" MARGINHEIGHT="1" BORDER=5 NORESIZE>
      <FRAME NAME="body" SRC="../UpdatePage/body_updatepage.jsp" SCROLLING=AUTO
   MARGINWIDTH=2 MARGINHEIGHT=2>
```

```
    </FRAMESET>
</FRAMESET>
</HEAD>
</HTML>
```

D.33 Updatesuccess.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<%@ LANGUAGE="Java" %>
<HTML>
<HEAD>
<BASE TARGET="_parent">
</HEAD>
<% HttpSession session=request.getSession(true);
%>

<BODY BGCOLOR="#FFFFFF" LINK="#0000FF" VLINK="#800080" TEXT="#000000" TOPMARGIN=2
LEFTMARGIN=2 MARGINWIDTH=2 MARGINHEIGHT=2>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=433>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=160 HEIGHT=30><IMG SRC="../clearpixel.gif" WIDTH=160 HEIGHT=1 BORDER=0></TD>
<TD WIDTH=273><IMG SRC="../clearpixel.gif" WIDTH=273 HEIGHT=1 BORDER=0></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=273><P ALIGN=CENTER><B><FONT SIZE="+1">USERPROFILE FOR : <%=
session.getValue("username")%></FONT></B></TD>
</TR>
</TABLE>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=40 HEIGHT=62><IMG SRC="../clearpixel.gif" WIDTH=40 HEIGHT=1 BORDER=0></TD>
<TD></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=459>
<TABLE ID="Table5" BORDER=1 CELLSPACING=3 CELLPADDING=1 WIDTH=459>
<TR>
<TD WIDTH=199><P>
<TABLE WIDTH=96 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>First Name</TD>
</TR>
</TABLE>
 </TD>
<TD WIDTH=241><P><%= session.getValue("first_name")%> </TD>
</TR>
<TR>
<TD WIDTH=199><P>
<TABLE WIDTH=95 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>Last Name</TD>
</TR>
</TABLE>
 </TD>
<TD WIDTH=241><P><%= session.getValue("last_name") %> </TD>
</TR>
<TR>
<TD WIDTH=199><P>
<TABLE WIDTH=95 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
```

```
<TR>
<TD><P>E-Mail </TD>
</TR>
</TABLE>
 </TD>
<TD WIDTH=241><P><%= session.getValue("email_address") %> </TD>
</TR>
<TR>
<TD WIDTH=199><P>
<TABLE WIDTH=96 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>Snail Mail</TD>
</TR>
</TABLE>
 </TD>
<TD WIDTH=241><P><TEXTAREA WRAP=PHYSICAL ID="FormsMultiLine1" NAME="address" ROWS=5
COLS=28><%= session.getValue("address") %></TEXTAREA> </TD>
</TR>
<TR>
<TD WIDTH=199><P>
<TABLE WIDTH=99 BORDER=0 CELLSPACING=0 CELLPADDING=0 ALIGN=LEFT>
<TR>
<TD><P>Birth Date </TD>
</TR>
</TABLE>
 </TD>
<TD WIDTH=241><P><%= session.getValue("birth_date") %> </TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=460>
<TR VALIGN=TOP ALIGN=LEFT>
<TD WIDTH=41 HEIGHT=3><IMG SRC="../clearpixel.gif" WIDTH=41 HEIGHT=1 BORDER=0></TD>
<TD WIDTH=419><IMG SRC="../clearpixel.gif" WIDTH=419 HEIGHT=1 BORDER=0></TD>
</TR>
<TR VALIGN=TOP ALIGN=LEFT>
<TD></TD>
<TD WIDTH=419><P>In the Table below,  you have selected the following platforms
which interest you. </TD>
</TR>
</TABLE>
<TABLE width=55%>
<TR>
<TD> </TD>
<TD> </TD>
<TD> </TD>
<TD>
<TABLE  BORDER=1 CELLSPACING=3 CELLPADDING=1 WIDTH=70% >
    <TR>
        <TD WIDTH=72>AS/400 </TD>
        <TD WIDTH=73><% if (session.getValue("as400")!=null) {%>selected<%} else
 { %>not selected <%}%> </TD>
        <TD WIDTH=73>RS/6000 </TD>
        <TD WIDTH=73> <% if (session.getValue("rs6000")!=null) {%>selected
<%} else { %>not selected <%}%> </TD>
    </TR>
    <TR>
        <TD>Networking </TD>
    <TD><% if (session.getValue("nw")!=null) {%>selected<%} else {%>not
selected<%}%> </TD>
        <TD>VM </TD>
```

```
    <TD><% if (session.getValue("vm")!=null) {%>selected<%}else{%>not selected<%}%>
 </TD>
 </TR>
    <TR>
        <TD>VM/ESA </TD>
    <TD><% if (session.getValue("vse")!=null) {%>selected<%}else{%>not
selected<%}%> </TD>
        <TD>S390 </TD>
    <TD><% if (session.getValue("s390")!=null) {%>selected<%}else{%>not
selected<%}%> </TD>
    </TR>
    <TR>
        <TD>Other </TD>
<TD><% if (session.getValue("other")!=null) {%>selected<%}else{%>not
selected<%}%> </TD>
        <TD>Windows </TD>
<TD><% if (session.getValue("windows")!=null) {%>selected<%}else{%>not selected
<%}%> </TD>
    </TR>
    <TR>
        <TD>SP </TD>
<TD><% if (session.getValue("scalable")!=null){%>selected<%}else{%>not selected
<%}%> </TD>
        <TD> </TD>
        <TD> </TD>
   </TR>
</TABLE>
</TD>
</TR>
</TABLE>
<P>
<P>
<CENTER>
<B>Return<B>
<P>
<A HREF="../HomePage/homepage.jsp"><IMG  HEIGHT=17 WIDTH=120 SRC="../home.gif" B
ORDER=0 ALT="Homepage"></A>
</P>
</CENTER>
</BODY>
</HTML>
```

Appendix E. Sample Domino scripts

This appendix provides the script file listings for the Notes operation.

E.1 start_domino.sh

```
#!/bin/bash

NOTES_USER=notes  # Notes user id
NOTES_PATH=/home/notes  # Home directory of Notes userid
NOTES_DATA=/local/notesdata/  # Domino data directory
NOTES_SERVER=/opt/lotus/  # Domino server location
SERVER_NAME=BDTEST3  # Domino server name
OUTPUT_LOG=/var/log/$SERVER_NAME.log  # Where we will put the log
PATH=$NOTES_PATH:/opt/lotus/bin:$HOME/bin:$PATH
if [ `whoami` != notes ];
then
   echo "Don't use root for running Domino server "
   su - notes
fi
if [ ! -x $NOTES_SERVER/bin/server ] ; then
  echo "Cannot found the server command - exiting"
  exit 1
fi
if [ ! -d $NOTES_DATA ] ;
then
  echo "Cannot access notes data directory - exiting"
  exit 1
fi
NOTES_RUNNING=`ps -fu notes | awk '{print $8}' | grep /opt/lotus | \
grep -v grep | cut -d"/" -f 1,2,3,4 | head -n1`
if ! $NOTES_RUNNING  ;
then
  echo "Domino Server is already running - exiting"
  exit 1
fi
cd $NOTES_DATA
rm -f ~notes.lck
mems=`ipcs -m | grep $NOTES_USER | awk '{ print $2 }'`
sems=`ipcs -s | grep $NOTES_USER | awk '{ print $2 }'`
for j in $mems; do if [ -n "$j" ];then ipcrm shm $j > tmp.lck;fi; done
for j in $sems; do if [ -n "$j" ];then ipcrm sem $j > tmp.lck;fi; done
rm -f tmp.lck

echo "Running Domino for LINUX"
```

```
/opt/lotus/bin/server > $OUTPUT_LOG 2>&1 &
```

E.2 cleanup_domino.sh

```
#!/bin/ksh
NOTES_USER=notes
NOTES_PATH=/local/notesdata
PATH=$NOTES_PATH:/pot/lotus/bin:$HOME/bin:$PATH

count=0
NOTES_RUNNING=`ps -fu $NOTES_USER | grep /opt/lotus | grep -v grep | grep
-v bash`
while [[ -n $NOTES_RUNNING ]] ; do
  sleep 10
  count=`expr $count + 1`
  echo "... waiting "$count"0 seconds"
  if [ $count -eq 1 ] ; then
    echo "Domino server is still running after 2 minutes"
    echo "Now ... he will dead"
    for i in `ps -fu $NOTES_USER |grep lotus|grep -v grep |\
      awk '{ print $2 }'`; do
      kill -9 $i
    done
    mems=`ipcs -m | grep $NOTES_USER | awk '{ print $2 }'`
    sems=`ipcs -s | grep $NOTES_USER | awk '{ print $2 }'`
    for j in $mems; do if [ -n "$j" ];then ipcrm shm $j>tmp.lck;fi; done
    for j in $sems; do if [ -n "$j" ];then ipcrm sem $j>tmp.lck;fi; done
    rm -f tmp.lck
    exit
  fi
  NOTES_RUNNING=`ps -fu $NOTES_USER | grep /opt/lotus | grep -v grep `
done
rm -f $NOTES_PATH/~notes.lck
echo Domino shutdown now completed
```

Appendix F. Using the additional material

This redbook also contains additional material in CD-ROM. See the appropriate section below for instructions on using or downloading the CD-ROM material.

F.1 Using the CD-ROM

The CD-ROM that accompanies this redbook contains the following:

File name or directory	*Description*
SG246007.html	Main HTML file
AUTORUN.*	Automatic run for Window based systems
***.html**	Various HTML files
***.gif**	Various images used in the HTML files
avi/	Directory for the ScreenCam videos
Java/	Direcotry for the source code of Java programs
htdocs/	Directory for the HTML and JSP files
script/	Directory for the various script files
sample/Redbooks.nsf	Sample redbooks Lotus Domino database

F.1.1 System requirements for using the CD-ROM

The following software configuration is recommended for optimal use of the CD-ROM.

Web browser	Such as Internet Explorer or NetScape Navigator
Adobe Acrobat Reader	For reading the PDF format of the redbook
Windows video/AVI player	For running the Screen Cam videos

F.1.2 How to use the CD-ROM

The CD-ROM should automatically open a Web browser window. Should you need to manually open the CD-ROM, you can access the contents of the CD-ROM or diskette by pointing your Web browser at the file, SG246007.html, in the CD-ROM root directory and following the links found there.

F.2 Locating the additional material on the Internet

The CD-ROM, diskette, or Web material associated with this redbook is also available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG246007

Alternatively, you can go to the IBM Redbooks Web site at:

http://www.redbooks.ibm.com/

Select the **Additional materials** and open the directory that corresponds with the redbook form number.

Appendix G. Special notices

This publication is intended to help system administrator or Web server developer to implement Linux based Web servers with IBM products. The information in this publication is not intended as the specification of any programming interfaces that are provided by all the IBM product that is used in this book. See the PUBLICATIONS section of the IBM Programming Announcement for IBM WebSphere Application server, IBM HTTPd server, IBM Database 2 and Lotus Domino Enterprise Server 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:

IBM	DB2
DB2 Connect	Netfinity
VisualAge	WebSphere

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

Caldera Systems, the C-logo, and OpenLinux are either registered trademarks or trademarks of Caldera Systems, Inc.

Domino, eSuite, Lotus eSuite, and Notes are trademarks of Lotus Development Corporation.

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.

Linux is a registered trademark of Linus Torvalds. All trademarks and registered trademarks on Linux.com are owned by their respective companies.

Lotus and Lotus Notes are registered trademarks of Lotus Development Corporation.

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.

Red Hat, the Red Hat "Shadow Man" logo, RPM, Maximum RPM, the RPM logo, Linux Library, PowerTools, Linux Undercover, RHmember, RHmember

More, Rough Cuts, Rawhide and all Red Hat-based trademarks and logos are trademarks or registered trademarks of Red Hat, Inc. in the United States and other countries.

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 other countries licensed exclusively through The Open Group.

SET, SET Secure Electronic Transaction, 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 H. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

H.1 IBM Redbooks

For information on ordering these publications see “How to get IBM Redbooks” on page 219.

- *Patterns for e-business: User-to-Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition*, SG24-5864
- *A Roadmap for Deploying Domino in the Organization*, SG24-5617
- *Connecting Domino to the Enterprise Using Java*, SG24-5425
- *Lotus Domino Release 5.0: A Developer's Handbook*, SG24-5331
- *Netfinity and Caldera Openlinux Integration Guide*, SG24-5861
- *Netfinity and Red Hat Linux Server Integration Guide*, SG24-5853
- *Linux for WebSphere and DB2 Servers*, SG24-5850
- *Domino R5 for Linux on IBM Netfinity*, SG24-5968 (IBM Redpiece)

H.2 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at http://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

H.3 Other resources

These publications are also relevant as further information sources:

- *WebSphere Application Server Guide*, SC34-4767 (see absract for ordering details)
- *DB2 Universal Database for UNIX Quick Beginning*, GC09-2836
- *DB2 Universal Database Administration Guide, Volume 1*, SC09-2839
- *DB2 Universal Database Administration Guide, Volume 2*, SC09-2840
- *DB2 Universal Database Command Reference*, Volume 6, SC09-2844
- *DB2 Universal Database Application Building Guide*, Volume 6, SC09-2842

H.4 Referenced Web sites

These Web sites are also relevant as further information sources:

- IBM Linux page: `http://www.ibm.com/linux`
- IBM Software homepages:
  - `http://www.ibm.com/software/developer/web/patterns`
  - `http://www.ibm.com/software/Web`
  - `http://www.ibm.com/software/webservers/httpservers`
- Sun's Java documentation pages:
  - `http://java.sun.com/docs/servlet`
  - `http://java.sun.com/products/servlet/2.1/api/javax.servlet.http.HttpSession.html`
  - `http://java.sun.com/docs/books/tutorial/servlets/client-state/session-tracking.html`
  - `http://java.sun.com/products/jsp`
  - `http://www.javasoft.com/beans/docs`
- Apache Web site: `http://www.apache.org`
- SSL documentation from Netscape: `http://home.netscape.com/eng/ssl3`
- Other source of servlet information: `http://novocode.de/doc/servlet-essentials/chapter0.html`
- Lotus information:
  - `http://www.lotus.com`
  - `http://www.lotus-developer.com`

How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM 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 IBM Redbooks fax order form to:

	e-mail address
In United States or Canada	pubscan@us.ibm.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 Redbooks 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.

Abbreviations and acronyms

AIX	Advanced Interactive Executive
API	Application Programming Interface
CAE	Client Application Enabler
CDROM	Compact Disk Read Only Memory
CE	Customer Engineer
CLI	Command Line Interface
CORBA	Common Object Request Broker Architecture
DB2	Database 2
DIIOP	Distributed IIOP
DLL	Dynamic Link Library
DMZ	Demiliterized Zone
GNOME	Gnu Network Object Model Environment
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transport Protocol
HTTPD	HTTP daemon
IBM	International Business Machines Corporation
IIOP	Internet Inter-ORB Protocol
IPC	Inter-process Communication
ITSO	International Technical Support Organization
JDBC	Java Database Conenctivity
JDK	Java Development Kit
JNI	Java Native Interface
JSDK	Java Servlet Development Kit
JSP	Java Server Page
JVM	Java Virtual Machine
KDE	K Desktop Environment
LDDJ	Lotus Domino Driver for JDBC
LSX	Lotus Script eXtender
NSD	Notes System Diagnostics
ORB	Object Request Broker
RPM	RedHat Package Manager
SHTML	Secure HTML
SQL	Structured Query Language
SSL	Secure Socket Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
UDB	Universal Database
URL	Uniform Resource Locator

Index

E

F

G

H

O

P

R

S

T

U

V

W

X

IBM Redbooks review

Your feedback is valued by the Redbook authors. In particular we are interested in situations where a Redbook "made the difference" in a task or problem you encountered. Using one of the following methods, **please review the Redbook, addressing value, subject matter, structure, depth and quality as appropriate.**

- Use the online **Contact us** review redbook form found at **ibm.com**/redbooks
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Document Number **Redbook Title**	SG24-6007-00 Linux Web Hosting with WebSphere, DB2, and Domino
Review	
What other subjects would you like to see IBM Redbooks address?	
Please rate your overall satisfaction:	O Very Good O Good O Average O Poor
Please identify yourself as belonging to one of the following groups:	O Customer O Business Partner O Solution Developer O IBM, Lotus or Tivoli Employee O None of the above
Your email address: The data you provide here may be used to provide you with information from IBM or our business partners about our products, services or activities.	O Please do not use the information collected here for future marketing or promotional contacts or other communications beyond the scope of this transaction.
Questions about IBM's privacy policy?	The following link explains how we protect your personal information. **ibm.com**/privacy/yourprivacy/

Linux Web Hosting with WebSphere, DB2, and Domino

Fast track for Web server development in Linux

Domino and DB2 database side-by-side in action

JSP and Servlet interaction explained

Linux, the open source operating system, is regarded as one of the main key operating systems in the future of e-business. IBM has provided a set of e-business applications on Linux to make it an advanced and robust e-business platform. The key offerings from IBM are the WebSphere application server, IBM Database 2, and Lotus Domino.

This book discusses implementing Linux as an e-business platform. Detailed information is provided on designing the system based on the e-business Pattern, setting up and configuring the software, developing the e-business solution, and operating the Web server. This book enables Web administrators and programmers to get up-to-speed with the exploding growth of e-business by using Linux platforms and IBM offerings. Examples are explained in great detail to ensure complete coverage of the programming tricks. This book also includes a CD-ROM that contains the source codes of all the components that we developed and used, which serves as an example of the e-business development that we performed on Linux.

This book is a must for Web administrators and programmers that want to deploy Linux based e-business configurations.

CD-ROM INCLUDED

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by IBM's International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6007-00 ISBN 0738428544