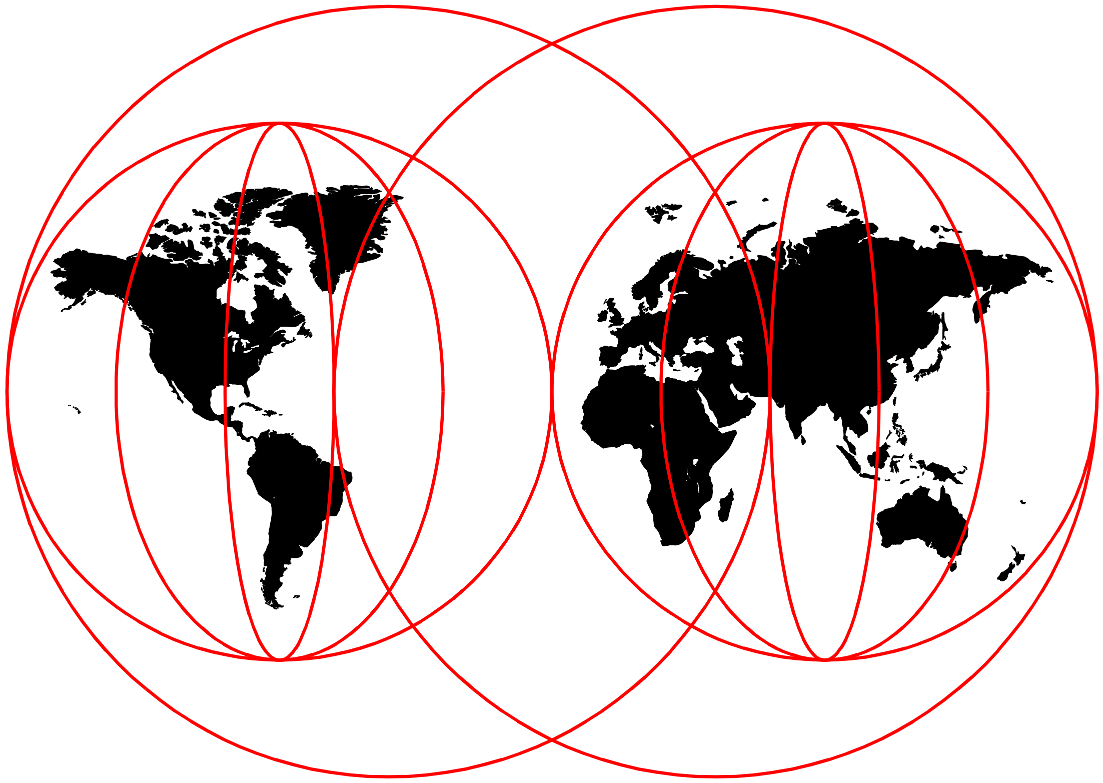
Building AS/400 Internet-Based Applications with Java

Bob Maatta, Markus Abegglen, Craig Pelkie, Brian Skaarup, Daniel Stucki



International Technical Support Organization

http://www.redbooks.ibm.com

SG24-5337-00

International Technical Support Organization

SG24-5337-00

Building AS/400 Internet-Based Applications with Java

January 1999

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix C, "Special Notices" on page 303.

First Edition (January 1999)

This edition applies to Version 3 Release 2 and later of OS/400

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JLU Building 107-2
3605 Highway 52N
Rochester, Minnesota 55901-7829

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

Figures

Tables

Preface

The AS/400 system, the Internet, and Java. What a powerful combination! If you are interested in enabling your company for the world of e-business, this redbook is for you. It is intended for anyone who wants to design and build AS/400 Internet- or intranet-based applications using Java.

This redbook focuses on building applets and servlets that access AS/400 resources. It provides many practical programming examples with detailed explanations of how they work. These examples are also available for you to download from our Internet site. This redbook gives you a fast start on your way to using Java, the Internet, and the AS/400 system.

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 Rochester Center.

Bob Maatta is a Senior Software Engineer from the United States at the International Technical Support Organization, Rochester Location. He writes extensively and teaches IBM classes worldwide on all areas of AS/400 client/server and application development. Before joining the ITSO in 1995, he worked in the U.S. AS/400 National Technical Support Center as a Consulting Market Support Specialist. He has over 20 years of experience in the computer field and has worked with all aspects of personal computers since 1983. He is a Sun Certified Java Programmer and a Sun Certified Java Developer.

Markus Abegglen is a Project Leader at DV Bern AG, an IBM Business Partner in Switzerland. He has eight years of experience in the AS/400 area, as well in object technology. He holds a higher national degree in Economics and Computer Science. He has worked on several projects that are based on VisualAge for Java.

Craig Pelkie is a consultant based in Southern California. He is the editor of the NEWS/400 newsletter *Client Access & Windows Solutions*, and the past editor of Midrange Computing’s *Client Access Expert*. He frequently leads seminars on Client Access, AS/400 web-enablement techniques and AS/400 client/server programming using Visual Basic. He is the author of the training manual *Using Microsoft Visual Basic with the AS/400* (http://www.vb400.com) which was used in labs at IBM Rochester Partners In Development.

Brian Skaarup is a software developer for EDB Gruppen Systems A/S, a Danish business partner. He has worked in the R&D department for the last eight years. His areas of expertise include client/server application design and development using C, C++, and Java.

Daniel Stucki is a Systems Engineer at DV Bern AG, an IBM Business Partner in Switzerland. He has nine years of experience in the AS/400 area as well in object technology. He also holds a degree in Computer Science from the Berne Institute of Technology. He has worked on several projects that are based on VisualAge for Smalltalk and VisualAge for Java.

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

Marcela Adan
ITSO Rochester

Chi Lam
IBM Rochester Laboratory

Gary Mullen-Schultz
IBM Rochester - Partners in Development

Schuman Shao
IBM Rochester Laboratory

Comments Welcome

Your comments are important to us!

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

- Fax the evaluation form found in “ITSO Redbook Evaluation” on page 317 to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Web sites:

For Internet users	`http://www.redbooks.ibm.com`
For IBM Intranet users	`http://w3.itso.ibm.com`

- Send us a note at the following address:

  `redbook@us.ibm.com`

Chapter 1. AS/400 Internet Application Development Overview

Let’s put our business on the Internet! Those words, spoken by the company president to the chief information officer (CIO), the CIO to the systems analysts, and the analysts to the programmers, are sure to cause a great deal of excitement, concern, confusion, and doubt. After all, it is one thing to surf the Web and find many examples of good and not-so-good sites to emulate. It is quite another issue to know where to start with your Internet development project and how to proceed.

There are many technical problems that must be resolved to reach the point where your customers can successfully use your corporate Web site to interact with your company. Some of the issues you need to address are:

- What hardware and software do you need on your AS/400 system so that it can be connected to the Internet and function as a Web server?
- What do you need to know about securing your AS/400 system, given that you need to allow access to data stored on the system?
- What skills are required to create dynamic Web pages that work with the AS/400 system? If you do not have the skills, what direction should you take to learn what you need to know?
- Where does Java fit into all of this? How does Java make it easier to work with Internet applications?

Although the answers to all of those questions are not in this redbook, this document shows you how Java can be used with the AS/400 system to create highly functional, dynamic Web pages. You learn how you can use Java in client-side applets and how Java servlets can be used on the AS/400 system to access the database and send responses to the client.

In this chapter, you explore four Internet application development approaches to use with the AS/400 system:

- Net.Data
- Common Gateway Interface (CGI) programming
- Java applets
- Java servlets

Upon finishing this chapter, you gain a better idea of which technique to use when you develop your Internet application.

1.1 From Server to Browser and Back—Fundamental Concepts

Before you can start coding an Internet application for the AS/400 system, you need to clearly understand how a request is sent from the browser to the IBM HTTP Server for AS/400, how the server invokes a program to process the request, and how the response from the program is returned to the browser. Although there are hundreds of configuration decisions that must be addressed, we assume that you already have the IBM HTTP Server for AS/400 connected to the Internet and configured to serve Web pages.

See how a Web serving request is processed. Figure 1 on page 2 shows how the request flows from the browser to the server, where the Web serving application accesses the database, prepares the response, and sends it back to the browser.

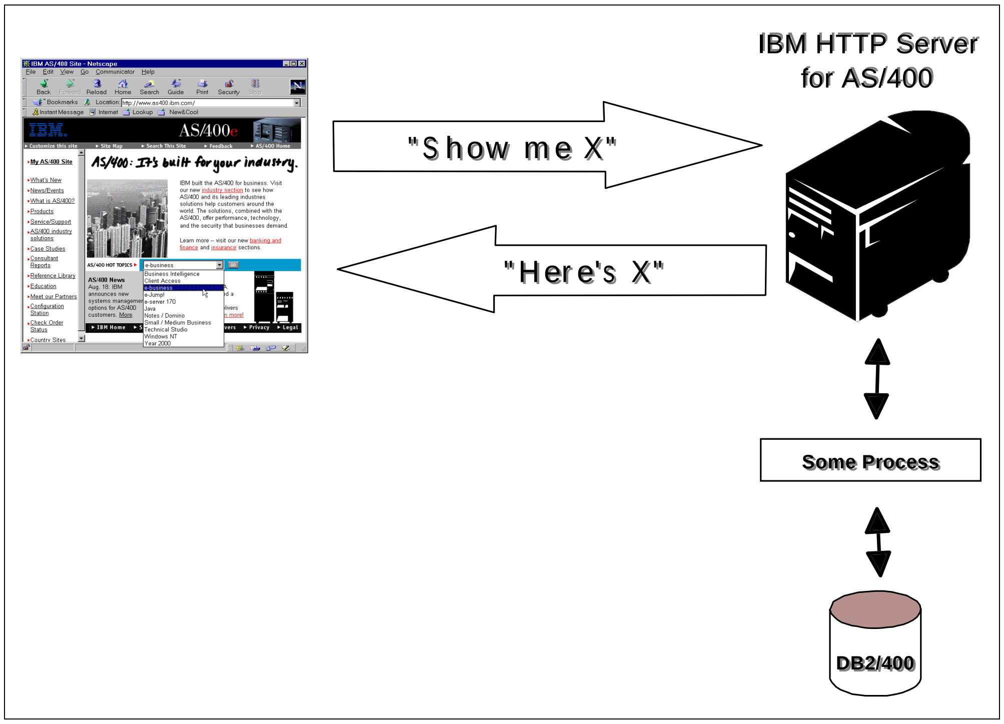


Figure 1. How an HTTP Server Processes a Request for a Dynamic Web Page

1.1.1 The Requesting Web Page

The user starts at the requesting Web page. The page appears when a user enters a URL in the browser's location or address field, or clicks a link on another Web page.

Web pages that allow user input are called *forms*. A form contains one or more elements that the user interacts with, for example, text fields, list boxes, check boxes, and buttons. When the user clicks a *submit button*, the browser formats a stream of data that contains the data the user entered or selected on the form. The stream of data is sent to the IBM HTTP Server for AS/400.

1.1.2 The Web Serving Application

Along with the data, the browser sends the name of the application that is to be invoked on the AS/400 system to process the data. The application name is included in the Hypertext Markup Language (HTML) code that was originally sent to the browser to display the Web page. The IBM HTTP Server for AS/400 invokes the application.

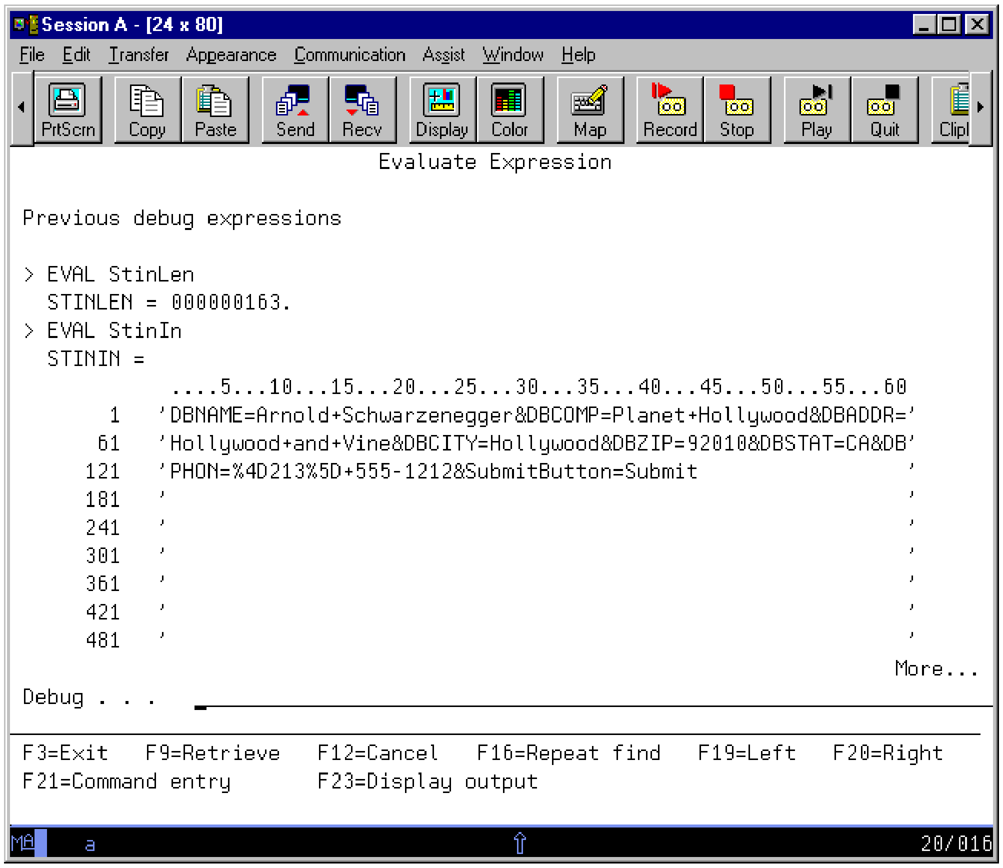
1.1.2.1 Input Data from the Form

There are two different methods used to make input data from the form available to the Web serving application program. The HTML form is coded to indicate which method to use.

The first method is the *Get method*. All of the data from the form is available to the Web serving application program in an environment variable called QUERY_STRING. The Web serving application program retrieves the contents of the environment variable so that it has access to the data. The Get method is limited to return 1024 bytes of data from the browser to the Web serving application program. Because the data also contains field names to identify each data element, the actual length of the data that can be entered by the user is less than 1024.

The second method is the *Post method*. With this method, data from the browser is available to the Web serving application program in a stream file called Standard Input (STDIN). Because the data is in a file, there is no limit to the length of the data that can sent from the browser. The Post method is the preferred method for returning form data from the browser.

Regardless of the method used, the data is HTTP encoded. Figure 2 shows a sample of data returned from a name and address Web form.


Session A - [24 x 80]
File Edit Transfer Appearance Communication Assist Window Help
PrtScrn Copy Paste Send Recv Display Color Map Record Stop Play Quit Clipb

```
                              Evaluate Expression

 Previous debug expressions

 > EVAL StinLen
   STINLEN = 000000163.
 > EVAL StinIn
   STININ =
             ....5...10...15...20...25...30...35...40...45...50...55...60
        1   'DBNAME=Arnold+Schwarzenegger&DBCOMP=Planet+Hollywood&DBADDR='
       61   'Hollywood+and+Vine&DBCITY=Hollywood&DBZIP=92010&DBSTAT=CA&DB'
      121   'PHON=%4D213%5D+555-1212&SubmitButton=Submit                  '
      181   '                                                             '
      241   '                                                             '
      301   '                                                             '
      361   '                                                             '
      421   '                                                             '
      481   '                                                             '
                                                                     More...
 Debug . . .  _

 F3=Exit   F9=Retrieve   F12=Cancel   F16=Repeat find   F19=Left   F20=Right
 F21=Command entry       F23=Display output

MA     a                                                              20/016
```

Figure 2. Form Data in HTTP Encoded Format

There are a few points to notice about the data in HTTP encoded format:

- The IBM HTTP Server for AS/400 takes care of the required ASCII to EBCDIC translation. Data that the user enters in the browser is sent to the server using the ASCII character set. Because the AS/400 works with the EBCDIC character set, there needs to be a translation between the two.

- The field names used on the form are included in the data and precede the value that the user entered. The field name and value are separated by an equals sign.
- Blank spaces that the user entered are replaced with plus signs (+).
- The end of each data field is delimited by the ampersand (&) character.
- Special characters are encoded with hexadecimal values. For example, the area code in the telephone number (line beginning 121 in Figure 2) has hexadecimal values for the parentheses:
  - `%4D` is used for the (character
  - `%5D` is used for the) character

One of the main jobs of the Web serving application program is to make sense of the HTTP encoded data. Because all AS/400 Web programmers have to deal with this requirement, Web programming tools that are available with the IBM HTTP Server for AS/400 include functions to interpret the HTTP encoded data and return it to the Web serving application program in data fields with which the program can work.

1.1.2.2 Processing the Request

Once the data is parsed into fields, the program can begin working on the request. At this point, conventional AS/400 programming techniques can be used. For example, the Web serving application program can:

- Use native database access techniques to retrieve records from the database.
- Use SQL statements to query the database or perform database updates.
- Call other programs on the AS/400 system to perform additional functions required to construct the response.
- Issue communications requests to other AS/400 systems or other database platforms to get additional data.

The Web serving application program has complete access to all of the AS/400 system facilities to which it is authorized.

1.1.2.3 Returning a Response to the Browser

Assume that the user’s request to the IBM HTTP Server for AS/400 was accepted by the server and that the server did not send an error response to the browser. It is now up to the Web serving application program to prepare a response that the browser understands and send it to the browser.

The response is HTML code. HTML is simply text that includes *tags* to identify elements on a Web page, such as the title, headers, tables, and graphics, and the *data* that is displayed on the page. Although most Web pages that you view when Web surfing are made up of *static HTML*, it is entirely possible and quite feasible to generate HTML on an "as-requested" basis, as in the example described in this chapter. It is a relatively easy programming job to construct HTML since it only requires simple string handling operations to put together the combination of HTML tags and data.

The response HTML and data can be as simple or as complex as required. For example, a simple listing of items can be done using the monospacing HTML tags. Or, you may choose to present the list in a nicely formatted table. The response may be another form to prompt the user for additional information. You can also include graphics, scripting, or applets in the response.

As each line of HTML code is constructed, it is written to the Standard Output (STDOUT) file. The STDOUT file is used by the Web serving application program as a conduit back to the browser, which is waiting for the response from the program. The IBM HTTP Server for AS/400 translates the response HTML in STDOUT from EBCDIC to ASCII and sends it back to the browser.

At this point, the cycle is complete. The user sent their request, which was successfully interpreted and responded to. The Web serving application program retrieved and formatted the requested data. The IBM HTTP Server for AS/400 sent the response back to the browser. The user is now free to review the response and make additional requests with their browser.

1.2 AS/400 Internet Application Development Techniques

Before you start working on your Internet application, you need to be clear about what your options are. If you are somewhat uncertain about what the best option is, you are not alone. Many developers are in the same position, trying to assess what is the best technique to use.

As with any other programming job, there is no definitive "best" technique, but rather a series of trade-offs. The technique you choose should be based on:

- Your understanding of the trade-offs
- The ability of the selected technique to let you accomplish what must be done
- The amount of effort it will take to develop the application
- The anticipated usage of the application
- The availability of other approaches that may be used to develop the application

We address the last point first, so that it is no longer a factor in your decision.

1.2.1 Other Approaches for Internet Applications

In addition to directly programming the AS/400 system for Web serving, which is the focus of this redbook, there are two other approaches to consider:

- Locating the application on a different server
- Using the Lotus Domino server available for the AS/400 system

1.2.1.1 Locating the Application on a Different Server

It is not unusual for a company to place their Web serving application on an entirely different host from the production database. For example, many companies use Windows NT Server or UNIX servers for Web applications. It is also possible to use another AS/400 system as the Web serving host. Since you are then developing the application on an AS/400 system, we consider that solution to be the same as developing the application on the AS/400 system that hosts the database.

When you introduce an entirely different type of server, you need to resolve the problem of making your production database available to that server. For real-time access, your application may use ODBC or communications programming to retrieve data from the AS/400 system database to the Web serving application. For applications that do not require real-time access to the database, you can replicate the database from the AS/400 system to the Web serving host as required.

If you adopt this approach, your programming tasks are simply relocated to the other server. Also, you probably need to perform more programming and certainly more management of the servers to coordinate access to the database. Nevertheless, you may choose to implement your Web serving application on another type of server because of capacity, availability of programming skills, or for security reasons.

1.2.1.2 Using the Lotus Domino Server for the AS/400 System

The AS/400 system now supports a native implementation of the Lotus Domino server. This provides a very attractive option. This is because performance and the ability to access the AS/400 system database is much enhanced compared to the implementation of Domino on the Integrated PC Server (IPCS).

Domino includes a full-featured application development environment, in addition to the rich functionality provided in the product itself. For many AS/400 system Web applications, Domino is the best choice.

An important point to note is that Domino and other AS/400 system Web development techniques are not mutually exclusive. If you choose to implement Domino for AS/400 system Web serving applications, you may still find the material in this redbook useful for applications that run outside of the Domino environment.

1.2.2 Four Tools for Internet Development on the AS/400 System

At this point, you can consider the tools that are available for developing an Internet application on the AS/400 system, using the IBM HTTP Server for AS/400. The tools that are examined include:

- Net.Data, included with the IBM HTTP Server for AS/400
- CGI programs, which can be written in traditional AS/400 system programming languages such as RPG and COBOL
- Java applets, which can use the AS/400 Toolbox for Java to communicate from an applet running in a browser to the AS/400 system
- Java servlets, which are available with OS/400 V4R3

Each of these tools require programming skills. In addition, you need to know enough HTML to achieve the effect you want, since you will use a tool to generate HTML to present the data retrieved from the AS/400 system.

1.3 Net.Data

Net.Data is a Common Gateway Interface (CGI) program provided with the AS/400 TCP/IP Connectivity Utilities (5769-TC1). Net.Data uses macros that you develop as input to the CGI program. The CGI program uses the macro to:

- Send HTML to your browser
- Run SQL commands
- Call system services such as programs compiled in other languages

Net.Data is the follow-on product to what was originally known as DB2WWW. The Net.Data CGI program itself is named DB2WWW. You occasionally see references to the DB2WWW product in the Net.Data documentation.

Note

The primary documentation for working with Net.Data on the AS/400 system is available at the AS/400 Net.Data Web site located at: http://www.as400.ibm.com/netdata

The following manuals are available at that site:

- *Net.Data Administration and Programming Guide*
- *Net.Data Language Environment Reference*
- *Net.Data Reference*

1.3.1 Why Use Net.Data

If you already know RPG, COBOL, C, or Java, you may wonder why you would use Net.Data, since you can develop AS/400 CGI programs using traditional AS/400 programming languages. However, when you use those languages, you are responsible for programming some of the low-level details required by the HTTP protocol. For example, you need to parse the input data that is sent to the server from an HTML form. You also need to take special care to send any required HTML headers back to the browser along with your application HTML and data, so that the browser knows how to work with the server response.

For RPG, COBOL, and C, you need to compile the programs before you can test them. If you need to make even minor changes, you need to repeat the edit, compile, and test cycle.

You may find that Net.Data provides a more simple alternative to working with AS/400 languages, especially if you simply want to retrieve and display data from the AS/400 database. Some of the characteristics of Net.Data that make it easier to use are:

- Net.Data is interpreted, not compiled. You can develop a Net.Data macro more rapidly than the equivalent compiled program. You can also make changes much more quickly. For example, adding or changing HTML statements in a Net.Data macro is trivial, and you can test the change immediately.
- Net.Data provides tremendous built-in support for working with the results of SQL queries. For example, Net.Data can automatically format the results of an SQL SELECT statement into an HTML table. You do not have to code the HTML for the table. You simply code the SELECT statement.
- Net.Data takes care of getting and parsing requests from the browser and preparing output to return to the browser. You simply code the HTML that you want sent to the browser and indicate what data is to be displayed. You do not have to code the lower-level HTML to deal with headers.

1.3.2 Net.Data Processing

To work with Net.Data effectively, you must understand how the CGI program DB2WWW interacts with:

- Your incoming request from the browser
- The IBM HTTP Server for AS/400
- Net.Data INI file
- The macro itself
- The DB2/400 database and other system services

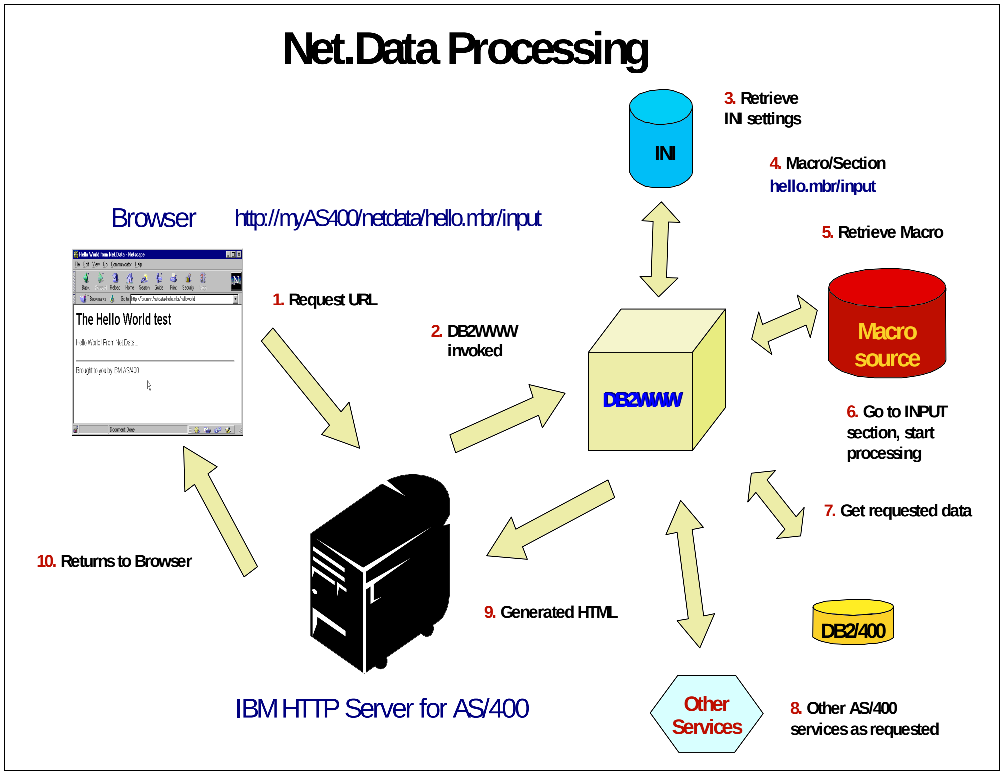


Figure 3. How Net.Data Processes a Macro and Generates a Response

Assuming that Net.Data is properly configured and you have a macro that you want to run, here is the process that occurs when you invoke a Net.Data macro. The following numbers correspond to the steps shown in Figure 3:

1. **A Net.Data macro is requested on an incoming URL.**

   You start a Net.Data macro by entering the URL containing the request in your browser's address entry area or you click on a link on a Web page that contains the request. The request is sent to the IBM HTTP Server for AS/400 the same as any other request.

2. **The DB2WWW CGI program is invoked.**

   The IBM HTTP Server for AS/400 determines from the incoming URL that the request is for Net.Data. At this point, the IBM HTTP Server for AS/400 invokes the DB2WWW CGI program.

   As with other requests to the IBM HTTP Server for AS/400, you need HTTP Server configuration directives so that the IBM HTTP Server for AS/400 knows how to handle the incoming request. To process Net.Data macros, you need at least one `EXEC` directive and you usually have at least one `MAP` directive.

3. **Net.Data configuration options are retrieved from the INI file.**

   Upon starting, the DB2WWW CGI program retrieves initialization options from the optional INI file, which is located in the same library as the DB2WWW program. If you do not create an INI file, the URLs that you use to invoke a Net.Data macro are considerably more complicated.

4. **The macro and start-at section within the macro are identified.**

   When the IBM HTTP Server for AS/400 invokes the DB2WWW CGI program, it passes the part of the incoming URL that identifies the macro to be invoked and the start-at section within the macro to the program. At this point, the DB2WWW program determines where on the system the macro source file is located. If you are using an INI file, the DB2WWW program uses the `MACRO_PATH` definition within the INI file to resolve the location of the macro.

5. **The macro is retrieved.**

   The DB2WWW CGI program now retrieves the macro. All Net.Data macros are stored in text format.

6. **DB2WWW starts the execution of the macro at the start-at section.**

   The DB2WWW CGI program parses the macro. Any global function calls and definitions in the macro are processed. Next, the DB2WWW CGI program goes to the start-at section that was specified on the incoming URL and starts processing the directives that are in that section.

   The start-at section is typically an HTML block that contains statements describing the initial page to be sent to the browser. For example, for a database query application, the start-at section may contain HTML that prompts the user for selection criteria.

7. **DB2/400 data is processed with SQL statements.**

   If there are any SQL statements or Net.Data function calls to other AS/400 system services, those statements or function calls are now processed.

8. **Other system services are invoked.**

   In a typical Net.Data macro, you embed SQL statements or function calls within HTML statements. Net.Data runs the SQL statement or function call at the point where it is encountered. The resulting HTML sent to the browser can include your headings and footings with the merged output of an SQL statement or function call.

9. **The resulting HTML is returned to the IBM HTTP Server for AS/400.**

   After processing the section and running SQL statements or other functions, the resulting HTML is returned from the DB2WWW CGI program to the IBM HTTP Server for AS/400.

10. **The resulting HTML is sent back to the browser.**

    The IBM HTTP Server for AS/400 sends the completed HTML page back to the browser. At this point, the process is complete. The user can request another Net.Data macro invocation, which starts the process over again.

1.3.3 How Net.Data Macros are Invoked

Net.Data macros are invoked from conventional URLs. The URL can be entered directly into the browser's address entry area or can be provided in the form of a link on an HTML page.

Here is a sample URL used to invoke the Net.Data sample application:

```
http://myAS400/netdata/nd_parts.mbr/input
```

There are three parts of the URL that are particularly important for Net.Data:

- **Location of DB2WWW CGI program**—In the sample URL, the `/netdata` part points to the location of the DB2WWW CGI program. The IBM HTTP Server for AS/400 expands this part of the URL using the `MAP` directive in the HTTP Server configuration file:

  ```
  Map /netdata/* /QSYS.LIB/NETDATA.LIB/DB2WWW.PGM/*
  ```

  The final asterisk character for both the source and replacement strings indicates that any input on the incoming URL following the `/netdata/` string is to be appended to the end of the expanded directory string. The expanded URL now looks like this:

  ```
  http://myAS400/QSYS.LIB/NETDATA.LIB/DB2WWW.PGM/nd_parts.mbr/input
  ```

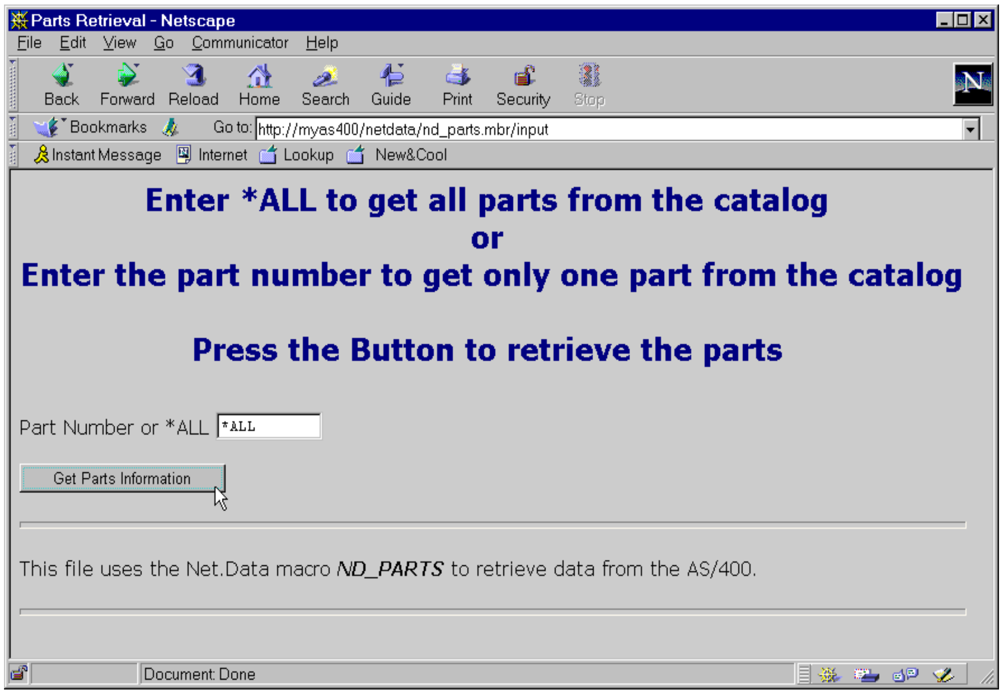
- **Name of macro to invoke**—The string `/nd_parts.mbr` identifies the name of the macro file that the DB2WWW CGI program is to load and process. Since the Net.Data macro is stored in an AS/400 source file in the AS/400 library file system, the `.MBR` suffix is required. If you request a macro that is stored in an AS/400 IFS directory, the suffix is the file extension. For example, if you store your Net.Data macros as `.TXT` files, include the `.TXT` suffix in the URL you use to invoke the macro.

- **Start-at section in macro**—Because a Net.Data macro can be quite long and include several different sections, you need to indicate to the DB2WWW CGI program where it is to start processing. In the sample URL, this is indicated with the `/input` string. It is conventional, although not required, that you start processing a Net.Data macro at a section labeled `INPUT`.

1.3.4 A Sample Net.Data Macro

The following sample Net.Data macro shows how you can use Net.Data to present a prompting page to the requester. Then, you can use input from the page to retrieve and display database records from the AS/400 system.

Figure 4 on page 11 shows the prompting page that is displayed when the user initially enters the URL.


Parts Retrieval - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Go to: http://myas400/netdata/nd_parts.mbr/input
Instant Message Internet Lookup New&Cool
Document: Done

Enter *ALL to get all parts from the catalog
or
Enter the part number to get only one part from the catalog

Press the Button to retrieve the parts

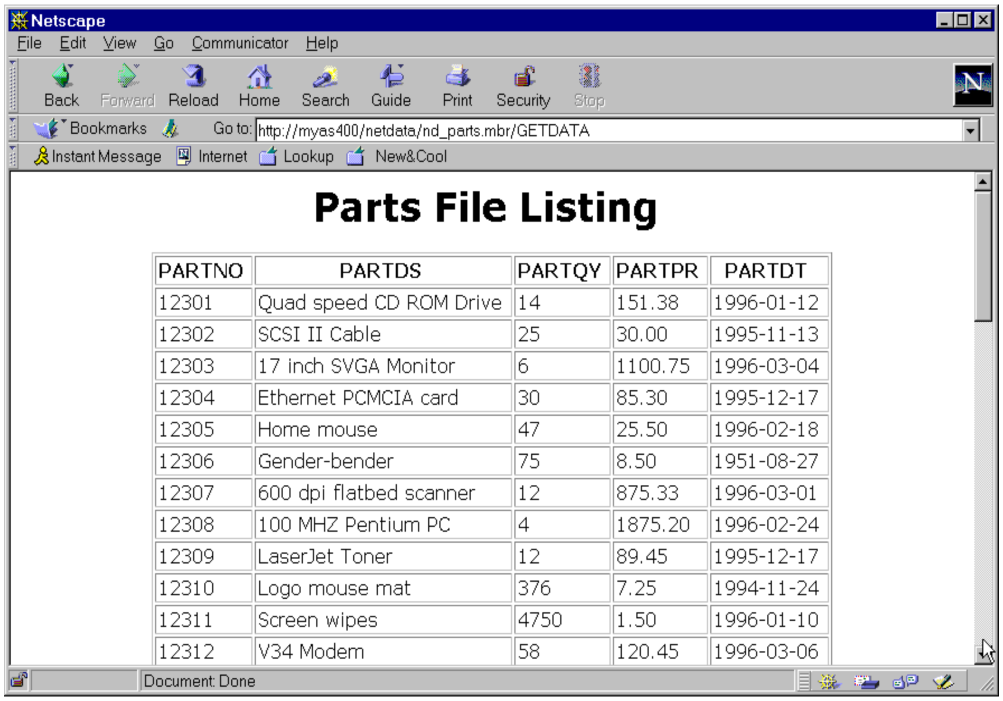
Part Number or *ALL *ALL

Get Parts Information

This file uses the Net.Data macro *ND_PARTS* to retrieve data from the AS/400.

Figure 4. Prompting Page Displayed by the Net.Data Macro

The user can accept the default of *ALL for the part number selection or enter a specific part number and then the Get Parts Information button. The Net.Data macro responds by querying the database and displaying the page shown in Figure 5.


Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Go to: http://myas400/netdata/nd_parts.mbr/GETDATA
Instant Message Internet Lookup New&Cool
Document: Done

Parts File Listing

PARTNO	PARTDS	PARTQY	PARTPR	PARTDT
12301	Quad speed CD ROM Drive	14	151.38	1996-01-12
12302	SCSI II Cable	25	30.00	1995-11-13
12303	17 inch SVGA Monitor	6	1100.75	1996-03-04
12304	Ethernet PCMCIA card	30	85.30	1995-12-17
12305	Home mouse	47	25.50	1996-02-18
12306	Gender-bender	75	8.50	1951-08-27
12307	600 dpi flatbed scanner	12	875.33	1996-03-01
12308	100 MHZ Pentium PC	4	1875.20	1996-02-24
12309	LaserJet Toner	12	89.45	1995-12-17
12310	Logo mouse mat	376	7.25	1994-11-24
12311	Screen wipes	4750	1.50	1996-01-10
12312	V34 Modem	58	120.45	1996-03-06

Figure 5. Parts File Listing Generated by the Net.Data Macro

1.3.5 Source Code for the Net.Data Sample Macro

The following source code is the complete Net.Data macro used to generate the prompting page and the parts list shown in Figure 4 and Figure 5 on page 11. If this is your first look at Net.Data, you probably think this is a strange language. However, if you glance through the code, you can start identifying different types of language elements and constructs:

- Comment lines start with the `%{` characters and end with the `%}` characters.
- Code blocks begin with `%define`, `%message`, `%function`, and `%html`.
- SQL statements are embedded in the macro and can include substitution variables (for example, `$(partno)`).
- HTML statements are embedded in the macro. Net.Data function calls can also be included with the HTML to generate additional HTML as the macro is processed.

Although we do not explain every detail of the macro in this redbook, we look at some of the sections of the macro so that you have an understanding of how it works.

```
%{------------------------------------------------------------%}
%{ Net.Data macro ND_PARTS -- display Parts in HTML table       %}
%{------------------------------------------------------------%}
%define{
    DATABASE       = "*LOCAL"
    DTW_HTML_TABLE = "YES"
%}

%message {
    -204      : "Error -204: Table not found"
     100      : "Warning 100: Record not found"       : Continue
     +default : "Warning $(RETURN_CODE)"              : Continue
     -default : "Unexpected SQL error $(RETURN_CODE)" : Exit
%}

%{------------------------------------------------------------%}
%{ Function RUNSQL - Called from GETDATA, get a part            %}
%{------------------------------------------------------------%}
%function(DTW_SQL) RUNSQL() {

  select * from apilib.parts where partno = $(partno)

%}

%{------------------------------------------------------------%}
%{ Function RUNSQLALL - Called from GETDATA, get all parts      %}
%{------------------------------------------------------------%}
%function(DTW_SQL) RUNSQLALL() {

  select * from apilib.parts order by partno

%}

%{------------------------------------------------------------%}
%{ HTML section INPUT - loaded by initial URL                   %}
%{------------------------------------------------------------%}
%html (INPUT) {

<HTML>
<HEAD>
    <TITLE>Parts Retrieval</TITLE>
</HEAD>

<BODY BGCOLOR="lightgrey">
<FORM action="GETDATA" method="POST">

<CENTER><B><FONT COLOR="darkblue" SIZE=+2>
    Enter *ALL to get all parts from the catalog
    <br>
    or
    <br>
    Enter the part number to get only one part from the catalog
    <br>
    Press the Button to retrieve the parts
</FONT></B></CENTER>

<br>
<br>
Part Number or *ALL
<INPUT TYPE="text" NAME="partno" VALUE="*ALL" SIZE=10>

<br>
<br>
<INPUT TYPE="submit" NAME="Submit" VALUE="Get Parts Information">
<p>
<HR SIZE="5">

<BR>This file uses the Net.Data macro <B><I>ND_PARTS</I></B>
to retrieve data from the AS/400.
<p>

<HR SIZE=5>
</FORM>
</BODY>
</HTML>
%}

%{--------------------------------------------------------------%}
%{ HTML section GETDATA - called when Submit button is clicked.  %}
%{--------------------------------------------------------------%}
%html (GETDATA) {

<html>
  <center>
    <h1>
      Parts File Listing
    </h1>

    %if ($(partno) == "*ALL")
      @RUNSQLALL()
    %else
      @RUNSQL()
    %endif
  </center>
</html>

%}
```

1.3.5.1 The HTML INPUT Section

Macro processing starts at the `INPUT` section. This is because the requesting URL identified the INPUT section as the start-at section (see the URL in Figure 4 on page 11).

Because this section is an HTML section (indicated by the `%html` block type identifier), the macro includes HTML statements that are sent to the browser. The HTML in the `INPUT` section is used to format the prompting Web page.

The `INPUT` section includes three statements that are used to process the prompting form:

- `<FORM action="GETDATA" method="POST">`
- `<INPUT TYPE="text" NAME="partno" VALUE="*ALL" SIZE=10>`
- `<INPUT TYPE="submit" NAME="Submit" VALUE="Get Parts Information">`

The `FORM` statement indicates two operations to the browser:

- Data entered on the prompting Web page is to be sent to the server using the `POST` method. The data is available to the server in the `STDIN` file.

- Processing the macro is to continue at the GETDATA section when the submit button is clicked.

The INPUT TYPE="text" statement is used to define the input field where the user requests all parts or a specific part. Note that the name of this field is partno.

The INPUT TYPE="submit" statement is used to define the button that the user clicks to submit the form for processing. When the submit button is clicked, the browser refers to the FORM statement to determine what it should do next.

1.3.5.2 The HTML GETDATA Section

The GETDATA section is used to retrieve the requested part data and format it for display. The section includes a mixture of HTML statements and Net.Data functions.

Because the user can request a listing of all parts or an individual part, the Net.Data %if construct is used to determine which to use. The value that was entered by the user is available in the $(partno) variable, which was defined in the INPUT section on the INPUT TYPE="text" statement.

Depending on the value of the $(partno) variable, the RUNSQLALL or RUNSQL function is invoked. The functions are defined near the beginning of the Net.Data macro. The important point to note here is that you can mix function calls with the HTML. The function call can generate additional HTML, which is substituted at the point of the function call.

1.3.5.3 The RUNSQL and RUNSQLALL Functions

The macro uses two SQL function blocks to process the user request. The function blocks start with the %function(DTW_SQL) identifier, which means that the function can contain only SQL statements and Net.Data functions.

The RUNSQL function is called if the user requests a particular part. The SQL SELECT statement uses the $(partno) variable to limit the query to the requested part. If the user requests a part that is not in the database, the resulting SQL error message is trapped by the %message block near the beginning of the macro. You can use the %message block to format your own messages to display to the user, rather than use the default error messages generated by Net.Data.

The RUNSQLALL function is called if the user requests a listing of all parts. In this function, there is no WHERE clause, so there is no limitation on the parts that are retrieved.

If you examine the code in these sections and the GETDATA section, look at the generated Web page shown in Figure 5 on page 11. You see that there are no HTML statements that define the table. This clearly illustrates one of Net.Data's strengths, which is its ability to automatically format queried data into a usable Web page.

1.3.6 When to Use Net.Data

Now that you have seen how Net.Data works on the AS/400 system and reviewed a sample Net.Data macro, you may wonder when you should consider using Net.Data. Here are some general guidelines about selecting Net.Data for your Web application.

1.3.6.1 You Need a Query Front-End to Run SQL Statements

Net.Data is a superb tool for creating simple HTML forms that act as front-ends to database queries. Using HTML in which you hand-code the Net.Data macro or HTML code that you include from an HTML editor of your choice, you can have a complete macro by simply adding an SQL section to run your query and automatically display tabular results.

In addition to the default behavior of Net.Data (display SQL results in an HTML table), you can customize the resulting table or use Net.Data functions to add other HTML options to the table, such as listbox fields and checkbox fields.

1.3.6.2 You Have Limited or No CGI Programming Support

If you do not have AS/400 programming skills (usually RPG, COBOL or C) or if you are uncomfortable with using the CGI APIs required to get, parse, and return data to the browser, you may find Net.Data easier to work with. Net.Data provides support for retrieving input from the browser and sending generated HTML back to the browser. You can concentrate on the application, rather than the mechanics of communicating with the browser.

1.3.6.3 One-Time or Short-Term Need

If the Web page that you need is not expected to have a long useful life, it does not make any sense to devote the effort required to create a CGI program or servlet. In this case, you may be willing to trade performance (generally better with CGI programs and servlets) for ease of creation and implementation (generally better with Net.Data).

1.4 Common Gateway Interface (CGI) Programming

On most non-AS/400 system Web serving platforms, CGI programming implies working with scripts written in the PERL language. Although there is an unsupported version of PERL available for the AS/400 system, most AS/400 CGI programs are created using ILE RPG, ILE COBOL, or ILE C. Because AS/400 CGI programs are compiled, they typically performs better than interpreted CGI programs such as Net.Data or PERL scripts.

All compiled AS/400 CGI programs must be invoked from the AS/400 library file system. CGI programs are created with traditional `CRTxxxPGM` commands using the AS/400 command line.

1.4.1 Why Use CGI Programming

The primary reason to use CGI programming on the AS/400 system is that you or your staff may already be familiar with one of the AS/400 system programming languages. For example, if you already know RPG, it is relatively simple to learn how to incorporate CGI processing techniques into an RPG program, compared with learning Net.Data or Java.

When you write a CGI program, you have access to all of the AS/400 system programming tools and constructs with which you are used to working. For example, you can use native database operations in your CGI programs. You can also use string handling operations in the language to create the exact HTML statements that you need. Finally, you can use the same debugging tools that you work with for other types of application programs to help you quickly put your CGI program into production.

1.4.2 CGI Processing

To work with CGI programs effectively, you must understand the following:

- How a CGI program is invoked in response to a request in the browser
- How form data is sent from the browser to the CGI program
- The IBM HTTP Server for AS/400
- How form data is made available to the CGI program while it is running
- How HTML generated in the CGI program is returned to the browser

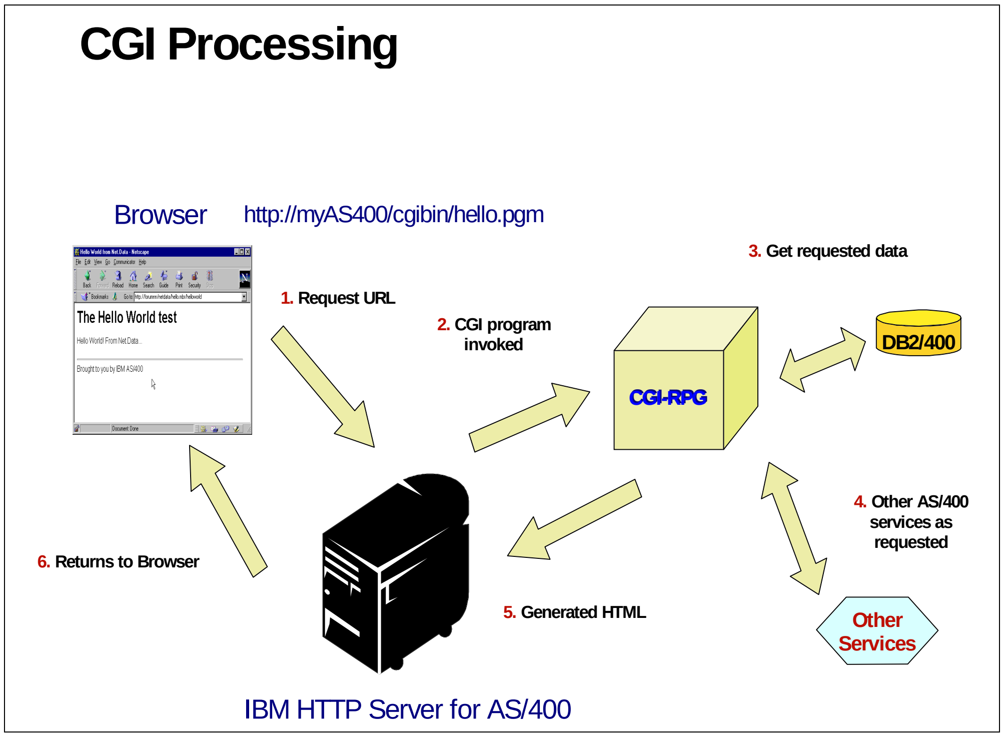


Figure 6. How CGI Processing Works

Assuming that your IBM HTTP Server for AS/400 is properly configured with the required MAP, PASS, and EXEC directives, here is the process that occurs when you invoke a CGI program. The step numbers correspond to the steps shown in Figure 6.

1. **A CGI program is requested on an incoming URL.**

   You start a CGI program by entering a URL containing the request in your browser's address entry area or you click on a link on a Web page that contains the request. The request is sent to the IBM HTTP Server for AS/400 along with any form data that was entered on the Web page.

2. **The CGI program is invoked.**

   The IBM HTTP Server for AS/400 uses the path on the incoming URL to locate the CGI program. The program is located based on a MAP, PASS, or EXEC directive that matches the `/cgibin` part of the URL.

   Note: You need at least one EXEC directive in your HTTP Server configuration file to enable CGI programs to be invoked. The EXEC directive may contain the replacement URL string that points to the library where the CGI program is located, or a MAP or PASS directive may contain the replacement URL.

For example, the following two directives in the HTTP Server configuration file may be used to allow the HELLO CGI program to be invoked:

```
Pass /cgibin/* /QSYS.LIB/CGIPGMS.LIB/*
Exec /QSYS.LIB/CGIPGMS.LIB/*
```

3. **The CGI program gets requested data.**

   Now that the CGI program is invoked, it runs like any other AS/400 program. It can open files, work with the DB2/400 database, run SQL statements, or call other AS/400 system services such as other programs, commands or APIs.

   At this point, the CGI program is conceptually similar to an AS/400 system workstation program in that it has received input from the browser form and is preparing a response to be sent back to the browser.

4. **The resulting HTML is returned to the IBM HTTP Server for AS/400.**

   After constructing the response HTML, the CGI program uses API calls to send the resulting HTML to the IBM HTTP Server for AS/400.

5. **The resulting HTML is sent back to the browser.**

   The IBM HTTP Server for AS/400 sends the completed HTML page back to the browser. At this point, the process is complete. The user can request another CGI program which starts the process over again.

1.4.3 APIs Used for CGI Programming

Section 1.1.2.1, “Input Data from the Form” on page 2 and Section 1.1.2.3, “Returning a Response to the Browser” on page 4 describe two files, STDIN and STDOUT, which are used with CGI programs. To review, remember these points:

- Form data sent from the browser is available to the CGI program in the STDIN file.
- Generated HTML to be sent from the CGI program back to the browser is written to the STDOUT file.

If you create an ILE C CGI program, you can work directly with the STDIN and STDOUT files. You do not need to work with the APIs described in this section.

However, if you use ILE RPG or ILE COBOL for your CGI programs, you cannot directly open files STDIN and STDOUT. You need to use APIs provided with the IBM HTTP Server for AS/400 to allow your CGI program to have access to the form data and to send its response.

1.4.3.1 The QHTTPSVR/QZHBCGI Service Program

Starting with OS/400 V4R3, the IBM HTTP Server for AS/400 product (5769-DG1) provides a Service Program (*SRVPGM), which includes several APIs that you use for CGI programming. The service program is QHTTPSVR/QZHBCGI. This is a complete replacement for the previously available service program QTCP/QTMHCGI.

The QHTTPSVR/QZHBCGI service program includes the following APIs:

- QtmhGetEnv—Get Environment Variable
- QtmhPutEnv—Put Environment Variable
- QtmhRdStin—Read from Stdin
- QtmhWrStout—Write to Stdout
- QtmhCvtDb—Convert using DB format

- QzhbCgiParse—Parse QUERY_STRING environment variable or POST data
- QzhbCgiUtils—Produce Full HTTP Response

If you create new CGI programs on a V4R3 AS/400 system, use service program QHTTPSVR/QZHBCGI. That service program includes all of the APIs that were previously available in QTCP/QTMHCGI, plus the new QzhbCgiParse API described in the following section.

1.4.3.2 The QzhbCgiParse API

Figure 2 on page 3 shows a sample of the form data that is available to your CGI program. The form data is in file STDIN and is in HTTP encoded format. To work with the form data, your CGI program needs to perform two tasks:

1. Read the data from the STDIN file.
2. Parse the data so that it can be used in the program. As you can see in the figure, the form field names are included with the data. Special characters are encoded using hexadecimal values.

Even though service program QHTTPSVR/QZHBCGI includes an API called QtmhRdStin to read from STDIN, you no longer need to use that API. Instead, you can use the new QzhbCgiParse API which both reads from STDIN and parses the data into field name and field value pairs.

The QzhbCgiParse API uses the six parameters, which are described in Table 1.

Table 1. Parameters Used with the QzhbCgiParse API

Parameter	Usage	Type	Description
Command string	Input	CHAR(20)	A list of flags and modifiers used to indicate the operation to be performed by the API. For example, to read form data from STDIN, the -POST flag is used.
Output format	Input	CHAR(8)	Specifies the format of the data to be returned to the target buffer. Must be one of these values: CGII0100 Free-form format (not parsed) CGII0200 CGI form variable output. Must be used with the -POST command string.
Target Buffer	Output	CHAR(*)	The output buffer that contains the POST data.
Length of Target Buffer	Input	BINARY(4)	Length of target buffer.
Length of Response	Output	BINARY(4)	The actual length of the POST data in the target buffer.

Parameter	Usage	Type	Description
Error Code	Input/Output	CHAR(*)	The standard AS/400 system API error structure. See "Error Code Parameter" in the *System API Reference* manual for additional information about this parameter.

1.4.3.3 How the QzhbCgiParse API Makes Form Data Available

The `QzhbCgiParse` API is similar to other OS/400 list APIs. With a list API, the data is written to a buffer along with information about where the data is located in the buffer. Using the location information, your program can iterate through the buffer and retrieve the data.

The advantage of the list processing technique is that variable length data can be easily accommodated. You do not have to know in advance how long a particular data element is, which is especially useful for character string data. The location information in the list not only indicates where each data element begins, but also how long the data element is. After extracting the data element from the list, it is available for use in your program.

The API retrieves both the field names and field values. The field names are based on the field names used on the HTML form. By default, the field names are returned to your program with the prefix `FORM_`. By appending the prefix, the field values do not replace current values in your program for fields that have the same name.

When you use the `QzhbCgiParse` API, the `CGII0200` format is "overlaid" on top of the target buffer. Your program works with the information available in the `CGII0200` format to determine where data elements can be located in the target buffer.

Table 2 shows the contents of the `CGII0200` format.

Table 2. The CGII0200 Format

Offset Decimal	Offset Hexadecimal	Type	Description
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(20)	Continuation handle
28	1C	BINARY(4)	Offset to first variable entry
32	20	BINARY(4)	Number of variable entries returned
36	24	CHAR(*)	Reserved
		BINARY(4)	Length of variable entry (see note)
		BINARY(4)	Length of variable name (see note)
		CHAR(*)	Variable name (see note)
		BINARY(4)	Length of variable value (see note)
		CHAR(*)	Variable value (see note)

Offset Decimal	Offset Hexadecimal	Type	Description
		CHAR(*)	Reserved (see note)
Note: These fields are repeated for each variable returned.			

1.4.3.4 The QtmhWrStout API

After retrieving form data with the `QzhbCgiParse` API, your CGI program prepares its response. The response typically includes many lines of HTML code that must be sent from the CGI program back to the browser. You need to send a response since the browser is locked, waiting for the response. If your CGI program does not respond, the browser eventually times-out so that the user can continue with other work.

Response HTML is sent from your CGI program to the browser in the `STDOUT` file. Because an ILE RPG or ILE COBOL program cannot directly open and write to `STDOUT`, CGI programs written in those languages must use the system API `QtmhWrStout` to send the response HTML back to the browser.

You can call the `QtmhWrStout` API as many times as needed to send a buffer of HTML to the browser.

Table 3 shows the parameter list used with the `QtmhWrStout` API.

Table 3. Parameters Used with the QtmhWrStout API

Parameter	Usage	Type	Description
Data variable	Input	CHAR(*)	Variable containing the data to write to STDOUT.
Length of data variable	Input	BINARY(4)	Length of the data to be written to STDOUT. Must be greater than 1.
Error Code	Input/Output	CHAR(*)	The standard AS/400 system API error structure. See "Error Code Parameter" in the *System API Reference* manual for additional information about this parameter.

1.4.4 A Sample RPG-CGI Program

To help you compare using CGI programming with the other techniques described in this manual (Net.Data, Java Servlets), the sample RPG-CGI program uses the same prompting form and queries the same database as the Net.Data macro. The user initially requests an HTML file that loads the prompting form, similar to Figure 4 on page 11. After selecting a part and clicking the submit button, the CGI program is invoked. The response sent back to the browser from the CGI program appears as shown in Figure 5 on page 11.

1.4.4.1 The Initial HTML File

Unlike the Net.Data macro sample, which includes the initial prompting page as part of the macro, the prompting page for the RPG-CGI program is a separate HTML file. The HTML code is shown in Figure 7.

```
<HTML>
<HEAD>
    <TITLE>Parts Retrieval</TITLE>
</HEAD>

<BODY BGCOLOR="lightgrey">
<FORM action="http://myas400/cgibin/cg_parts.pgm" method="POST">

<CENTER><B><FONT COLOR="darkblue" SIZE=+2>
    Enter *ALL to get all parts from the catalog
    <br>
    or
    <br>
    Enter the part number to get only one part from the catalog
    <br>
    Press the Button to retrieve the parts
</FONT></B></CENTER>

<br>
<br>
Part Number or *ALL
<INPUT TYPE="text" NAME="partno" VALUE="*ALL" SIZE=10>

<br>
<br>
<INPUT TYPE="submit" NAME="Submit" VALUE="Get Parts Information">
<p>
<HR SIZE="5">

<BR>This file uses the CGI-RPG program <B><I>CG_PARTS</I></B>
to retrieve data from the AS/400.
<p>

<HR SIZE=5>
</FORM>
</BODY>
</HTML>
```

Figure 7. HTML Code Used to Display the Prompting Page for the CGI Program

There are three HTML statements used to process the form:

```
<FORM action="http://myas400/cgibin/cg_parts.pgm" method="POST">
<INPUT TYPE="text" NAME="partno" VALUE="*ALL" SIZE=10>
<INPUT TYPE="submit" NAME="Submit" VALUE="Get Parts Information">
```

The form statement indicates the following information to the browser:

- Data entered on the prompting Web page is sent to the server using the `POST` method. The data is available to the server in the `STDIN` file.
- The program `CG_PARTS` is to be invoked. The IBM HTTP Server for AS/400 locates the program using the `/cgibin` part of the `ACTION` URL.

The `INPUT TYPE="text"` statement is used to define the input field where the user requests all parts or a specific part. Note that the name of this field is `partno`.

The `INPUT TYPE="submit"` statement is used to define the button that the user clicks to submit the form for processing. When the submit button is clicked, the browser refers to the `FORM` statement to determine what it should do next.

1.4.5 Source Code for the CG_PARTS CGI Program

The following source code is the complete ILE RPG program that is used to work with the input from the HTML prompting page and return the part data as shown in Figure 5 on page 11.

```
  ****************************************************************
 *  CGI-RPG program CG_PARTS -- display Parts in HTML table
 ***************************************************************

 ***************************************************************
 *  Use externally described data structure for field names
 ***************************************************************
D parts         e ds

 ****************************************************************
 * CGI APIs - from *SRVPGM QHTTPSVR/QZHBCGI
 ****************************************************************
D CGIParse        c                   'QzhbCgiParse'
D CGIWrite        c                   'QtmhWrStout'

 ****************************************************************
 * Parameter values for QzhbCgiParse API
 ****************************************************************
D zhbBuffer       s           1024
D zhbCommand      s             20    inz('-POST')
D zhbFormat       s              8    inz('CGII0200')
D zhbLen          s              9b 0 inz(%len(zhbBuffer))
D zhbRspLen       s              9b 0 inz(0)

 ****************************************************************
 * CGII0200 format
 *
 *    d2ybtrtn - bytes returned
 *    d2bytavl - bytes available
 *    d2conhnd - continuation handle
 *    d2offset - offset to first variable entry
 *    d2count  - number of variable entries returned
 ****************************************************************
D ds0200          ds                  based(ptr)
D    d2bytrtn                    9b 0
D    d2bytavl                    9b 0
D    d2conhnd                   20
D    d2offset                    9b 0
D    d2count                     9b 0

D ptr             s               *

 ****************************************************************
 * Parameter values for QtmhWrStout API
 ****************************************************************
D tmhOut          s            256
D tmhOutLen       s              9b 0 inz(%len(tmhOut))

 ****************************************************************
 * API error structure
 ****************************************************************
D QUSEC           ds
D    qusbprv                     9b 0
D    qusbavl                     9b 0
D    qusmsg                      7
D    qusrsvd                   200

 ****************************************************************
 * Other constants/variables
 ****************************************************************
D aFtrsize        c                   4
D aHdrsize        c                   16
D crlf            c                   x'15'

D aHdr            s             80    dim(aHdrsize) perrcd(1) ctdata
D aFtr            s             80    dim(aFtrsize) perrcd(1) ctdata
D getAllParts     s                   like(partno) inz(-1)
D PartRequest     s                   like(partno)

 ****************************************************************
 * HTML constants
 ****************************************************************
D htTdB           c                   '<td>'
D htTdE           c                   '</td>'
D htTrB           c                   '<tr>'
D htTrE           c                   '</tr>'

 ****************************************************************
 * Prototype for InitQUSEC procedure
 ****************************************************************
D InitQUSEC       pr

 ****************************************************************
 * Prototype for MakeFooter procedure
 ****************************************************************
D MakeFooter      pr

 ****************************************************************
 * Prototype for MakeHeader procedure
 ****************************************************************
D MakeHeader      pr

 ****************************************************************
 * Prototype for MakeHTML procedure
 ****************************************************************
D MakeHTML        pr
D   StringIn                   254    value

 ****************************************************************
 * Prototype for MakeTblDta procedure
 ****************************************************************
D MakeTblDta      pr
D   StringIn                    80    value

 ****************************************************************
 * Prototype for ParseForm procedure
 ****************************************************************
D ParseForm       pr                  like(partno)

 ****************************************************************
 * Prototype for RunSQL procedure
 ****************************************************************
D RunSQL          pr
D  sqPartno                           like(partno)  value

 ****************************************************************
 * Program mainline routine
 ****************************************************************

C                   callp     MakeHeader
C                   eval      PartRequest = ParseForm
C                   callp     RunSQL(PartRequest)
C                   callp     MakeFooter

C                   movel     *on           *inlr

 ****************************************************************
 * Procedure InitQUSEC
 *
 * Initialize bytes provided/available for QUSEC
 ****************************************************************
P InitQUSEC       b

C                   eval      qusbprv = %len(QUSEC)
C                   eval      qusbavl = 0

P InitQUSEC       e

 ****************************************************************
 * Procedure MakeFooter
 *
 * Write HTML for the page footer
 ****************************************************************
P MakeFooter      b
D    N            s              5  0

C                   do        aFtrsize      N
C                   callp     MakeHTML(%trim(aFtr(N)))
C                   enddo

P MakeFooter      e

 ****************************************************************
 * Procedure MakeHeader
 *
 * Write HTML for the page header. First line output is
 * the Content-Type header, which must be followed by the
 * newline character.
 ****************************************************************
P MakeHeader      b
D    N            s              5  0

C                   do        aHdrsize      N

C                   if        N = 1
C                   callp     MakeHtml(%trim(aHdr(N)) + crlf + crlf)
C                   else
C                   callp     MakeHTML(%trim(aHdr(N)))
C                   endif

C                   enddo

P MakeHeader      e

 ****************************************************************
 * Procedure MakeHTML
 *
 * Write the generated HTML string to STDOUT
 ****************************************************************
P MakeHTML        b

D   MakeHTML      pi
D     StringIn                 254    value

 *--------------------------------------------------------------
 *  Calculate length of HTML string
 *--------------------------------------------------------------

C                   eval      tmhOut    = %trim(StringIn)
C                   eval      tmhOutLen = %len(tmhOut)

 *--------------------------------------------------------------
 *  Use system API to write to STDOUT, send HTML to browser
 *--------------------------------------------------------------

C                   callp     InitQUSEC

C                   callb     CGIWrite                             99
C                   parm                    tmhOut
C                   parm                    tmhOutLen
C                   parm                    QUSEC

P MakeHTML        e

 ****************************************************************
 * Procedure MakeTblDta
 *
 * Make a table data element, write HTML string to STDOUT.
 ****************************************************************
P MakeTblDta      b

D   MakeTblDta    pi
D     StringIn                  80    value

C                   callp     MakeHTML(htTdB    +
C                                      StringIn +
C                                      htTdE)

P MakeTblDta      e

 ****************************************************************
 * Procedure ParseForm
 *
 * Get data from incoming request and parse
 ****************************************************************
P ParseForm       b

D    ParseForm    pi                  like(partno)

D  p1             s               *
D  p2             s               *
D  lenVE          s              9b 0
D  lenVN          s              9b 0
D  lenVV          s              9b 0
```

```
D  lenWork        s               9b 0 based(p2)
D  rtnVar         s                    like(partno)
D  varName        s              50
D  varValue       s              50
D  work           s            1024    based(p2)

C                   callb     CGIParse                             99
C                   parm                    zhbCommand
C                   parm                    zhbFormat
C                   parm                    zhbBuffer
C                   parm                    zhbLen
C                   parm                    zhbRspLen
C                   parm                    QUSEC

C                   if        (qusmsg = *blanks) and
C                             (*in99  = *off   )

 *  store offset to first variable (part request)

C                   eval      ptr = %addr(zhbBuffer)
C                   eval      p1  = ptr + d2offset

 *  get length of variable entry

C                   eval      p2 = p1
C                   eval      lenVE = lenWork

 *  get variable name

C                   eval      p2 = p2 + 4
C                   eval      lenVN = lenWork
C                   eval      p2 = p2 + 4
C                   eval      varName = %str(p2 : lenVN)

 *  get variable value

C                   eval      p2 = p2 + lenVN
C                   eval      lenVV = lenWork
C                   eval      p2 = p2 + 4
C                   eval      varValue = %str(p2 : lenVV)

 *  if variable entered on Web page is blank or *ALL,
 *  set value to -1 to request *ALL parts

C                   if        (varValue = *blanks) or
C                             (varValue = '*ALL')  or
C                             (varValue = '*all')
C                   eval      rtnVar = getAllParts

C                   else
C                   eval      varValue = %trim(varValue)
C                   movel     varValue      rtnVar
C                   endif

C                   return    rtnVar
C                   endif

P ParseForm       e

 ****************************************************************
 * Procedure RunSQL
 *
 * Run SQL for requested part(s)
 ****************************************************************
P RunSQL          b

D   RunSQL        pi
D    sqPartno                          like(partno)  value

C                   if        (sqPartno = getAllParts)
C/exec sql
C+    declare c1 cursor for
C+        select * from apilib/parts order by partno
C/end-exec
C                   else
C/exec sql
C+    declare c2 cursor for
C+        select * from apilib/parts where PARTNO = :sqpartno
```

```
     C/end-exec
     C                   endif

     C                   if        (sqPartno = getAllParts)
     C/exec sql open c1
     C/end-exec
     C                   else
     C/exec sql open c2
     C/end-exec
     C                   endif

     C/exec sql whenever not found goto EndOfFile
     C/end-exec

      *---------------------------------------------------------------
      * Create HTML table rows for part data
      *---------------------------------------------------------------

     C                   dow       sqlcod = 0

     C                   if        (sqPartno = getAllParts)
     C/exec sql
     C+  fetch c1 into  :partno, :partds, :partqy, :partpr, :partdt
     C/end-exec
     C                   else
     C/exec sql
     C+  fetch c2 into  :partno, :partds, :partqy, :partpr, :partdt
     C/end-exec
     C                   endif

     C                   callp     MakeHTML(htTrB)

     C                   callp     MakeTblDta(%trim(%editc(partno : 'X')))
     C                   callp     MakeTblDta(            partds        )
     C                   callp     MakeTblDta(%trim(%editc(partqy : 'Z')))
     C                   callp     MakeTblDta(%trim(%editc(partpr : 'J')))
     C                   callp     MakeTblDta(%char(       partdt       ))

     C                   callp     MakeHTML(htTrE)
     C                   enddo

     C     EndOfFile     tag

     C                   if        (sqPartno = getAllParts)
     C/exec sql close c1
     C/end-exec
     C                   else
     C/exec sql close c2
     C/end-exec
     C                   endif

     P RunSQL          e
** CTDATA aHdr
Content-Type: text/html
<html>
<head>
    <title>Query response from CG_PARTS</title>
</head>
<body>
    <center>
    <h1>Parts File Listing</h1>
    <table border cellpadding=2>
        <tr>
            <th>PARTNO</th>
            <th>PARTDS</th>
            <th>PARTQY</th>
            <th>PARTPR</th>
            <th>PARTDT</th>
        </tr>
** CTDATA aFtr
    </table>
    </center>
</body>
</html>
```

1.4.5.1 Understanding the Program Flow

When the program is invoked, it starts in the mainline routine:

```
      **************************************************************
     * Program mainline routine
     **************************************************************

    C                   callp     MakeHeader
    C                   eval      PartRequest = ParseForm
    C                   callp     RunSQL(PartRequest)
    C                   callp     MakeFooter

    C                   movel     *on           *inlr
```

The `MakeHeader` and `MakeFooter` routines write the contents of the `aHdr` and `aFtr` arrays to `STDOUT`. Because the response Web page header and footer contain HTML that does not change, that HTML can be coded as array elements and output without further processing.

The `ParseForm` procedure is invoked to parse data from file `STDIN`. The part number that the user entered on the prompting Web page is passed to the `RunSQL` procedure.

1.4.5.2 The ParseForm Procedure

The `ParseForm` procedure calls the `QzhbCgiParse` API to retrieve the input stream from file `STDIN`. After retrieving the input stream, the procedure works through the returned buffer to extract the value for the first field. That field contains the part number that the user wants to review or the special value `*ALL` to review all parts.

In this example, there is only one input field used on the prompting form. The field is named `partno` on the prompting form (see the HTML used for the prompting form in Figure 7 on page 21). The field name in the buffer is `FORM_partno`, since the default field name prefix is used when the `QzhbCgiParse` API is called.

Because the part number is defined as a numeric field in the program, the value `-1` is used to indicate that the user entered the special value `*ALL`.

Since there is only one field on the prompting form, the code does not iterate through the buffer. When you work with a form that has multiple input fields, you can use a program loop to parse all of the fields in the buffer. The `CGII0200` format includes a count field that indicates how many fields are available in the buffer.

1.4.5.3 The RunSQL Procedure

After parsing the input buffer, the program knows which part the user wants to review. When this procedure is called, the part number is either a specific part number or the special value `-1` to indicate that the user wants to review all parts.

Based on the part selection, one of the SQL `declare cursor` statements is used. After opening the selected cursor, the procedure loops and fetches one row at a time from the result set. As each row is retrieved, the procedures `MakeHTML` and `MakeTblDta` are called:

- Procedure `MakeHTML` is used to write the HTML tags `<TR>` and `</TR>` to delimit the beginning and end of the table row.
- Procedure `MakeTblData` is used to write the HTML tags `<TD>` and `</TD>` and the actual field values from the row.

The result of using those two procedures is that a complete HTML table row is written to `STDOUT` for each of the data rows fetched from the result set.

1.4.6 Summary of the CGI Sample Program

As you can see, there is considerably more code required for an RPG CGI program as compared to the Net.Data sample. As with any programming project, once you create a working RPG CGI program, you can easily copy code from one program to another.

The most difficult part of the RPG CGI program is the code that parses the buffer returned from the `QzhbCgiParse` API. You may find that after you develop several programs using this API, you can extract the routines and create your own service program to generalize and encapsulate the processing required for the API.

1.4.7 When to Use CGI Programs

CGI programming, using either ILE RPG or ILE COBOL, is a natural fit for many enterprises that have AS/400 systems. If you are familiar with ILE RPG or ILE COBOL, you can easily see how you can substitute HTML forms processing for traditional green-screen display file applications. The advantage of using HTML forms is that your application can be run in a browser on any number of client workstations without needing 5250 emulation support.

CGI programs typically have better performance than equivalent Net.Data macros. The overhead of invoking the CGI program is less than Net.Data. Also, because the CGI program is a compiled program, it runs faster than the interpreted Net.Data macro.

In short, CGI programming is a way to Web-enable your AS/400 system applications using existing programming skills. You need to learn only a few additional skills (HTML and the use of the APIs) to create CGI programs.

1.5 Java Applets

Java applets are programs that run inside a Java-capable browser. Browsers that provide support for applets include:

- Netscape Navigator version 4.04 or later
- Microsoft Internet Explorer version 4.01
- Sun Microsystems HotJava

Because an applet is an executable program, the interaction between the browser and the server is quite different from the Net.Data and CGI programming techniques. With Net.Data and CGI programming, the program running on the server is responsible for generating and sending HTML code to the browser. The browser simply displays the HTML.

Java applets are delivered to the browser in HTML files that contain a reference to the applet. Upon arriving in the browser, the applet begins execution. The applet usually displays what appears to the user as a Web page. However, all of the user interface elements are contained within the applet itself and are not rendered by HTML.

Most importantly, the applet can communicate directly back to the server. When the server responds, it is up to the applet to display the response. Because the applet is in control of its user interface elements, it can update them to display the server response, again without requiring HTML.

1.5.1 The Scripting Alternative

You may have heard of scripting languages that can be used for Web pages. The two most popular scripting languages are:

- JavaScript, which is syntactically similar to Java but does not provide as many features as Java
- VBScript, which is a subset of Microsoft Visual Basic

Scripting languages were introduced into browsers by Netscape and Microsoft as a method of providing more capabilities to Web pages displayed with HTML. The HTML on a Web page can include several user-interface components, such as push buttons, check boxes, lists, and clickable links. Scripting can be used to control animation effects on the page and also to provide edits of user-entered data.

1.5.1.1 JavaScript

JavaScript can have only limited interaction with the browser host or the server. For example, JavaScript is limited to storing only small amounts of information in "cookie" files on the browser host; JavaScript cannot store information in a text file for example. JavaScript also cannot access any files other than cookies on the browser host. These limitations are obviously required because of the risk inherent in running JavaScript code from unknown or untrusted Web servers.

Although JavaScript is generally regarded as "safe", it is possible to disable it in the browsers that support it. For example, both Netscape Navigator and Microsoft Internet Explorer let you disable JavaScript.

1.5.1.2 VBScript and ActiveX

Microsoft initially pushed a technology called ActiveX when they introduced VBScript with their Internet Explorer browser. The idea is that ActiveX components can be delivered to a Windows PC as part of the Web content associated with a page. ActiveX components are essentially the same as and extend other Windows components that are already present on a Windows PC. By having these additional components in place or delivered as part of the Web page when required, the VBScript on the Web page can present the viewer with a more Windows-like page, as opposed to the relatively less active HTML page.

However, the capabilities that give ActiveX its power also make it undesirable for most Web users. There is nothing to prevent an ActiveX component from accessing all areas of a user’s PC. The ActiveX component can possibly damage files or transmit data from a file to the server without the user’s consent. Microsoft tried to make ActiveX more acceptable by introducing a concept of digitally signed components. When you go to a Web site that tries to download an ActiveX component along with the HTML, the Internet Explorer browser prompts you to ask if you want to accept the component. Netscape browsers, which need a plug-in to provide ActiveX support, simply ignore ActiveX components. Part of the prompt includes information about the originator of the ActiveX component and whether the component possesses a signed digital certificate. At that point, you can accept or reject the component, based on the information that it presents to you.

Although it is possible to see how ActiveX components can provide you with greater Web interaction possibilities, ActiveX has not and most likely will not become a favored technique on the Internet for these reasons:

- ActiveX is currently only supported natively in the Microsoft Internet Explorer series of browsers. Users of other browsers, such as Netscape Navigator, cannot use ActiveX at all or need to install a plug-in to provide ActiveX support.
- ActiveX is a Microsoft Windows-based technology. Web users on other platforms cannot benefit from ActiveX components.
- Most people are leery of allowing unrequested components to install themselves on their PCs when they access the Internet.

1.5.2 How Applets are Different from Scripting

Java applets are delivered to your browser when you request an HTML file that includes the APPLET tag. The Web server sends the applet to your browser, which starts the applet. At that point, the browser acts as a container for the applet, which has control of the Web page.

There are two important concepts about browsers and applets that you should understand:

- Because an applet is written in Java, it requires the supporting Java runtime environment. That environment is provided by the browsers. Because the Java runtime environment is already on the browser host, the only code that needs to be transmitted from the server to the browser host is the code required for the application.
- Within the browser, the applet is constrained to run in the *sandbox*. The sandbox is an environment that limits what the applet can do. For example, the applet cannot access files on the browser host other than cookies. It cannot start network connections to hosts other than the server from which it was loaded. And, it cannot examine or change any configuration options on the browser host. The applet is limited to user-interaction within the browser and communication with the server.

Because applets are Java programs, they have the benefit of transportability. Since there is a base Java runtime environment that all browser vendors are required to implement to claim Java compliance, the applet can be served to Web users on completely different platforms. In all cases, the applet should appear the same to the user and should have the same interactions with the user and the server.

Both Netscape Navigator and Microsoft Internet Explorer provide browser configuration options to let the user loosen the restrictions imposed by the sandbox. However, it is up to the user to change the default configuration. Applets themselves cannot override the security that is imposed by the sandbox.

1.5.3 Applet Processing

One of the most important features of applets for AS/400 system developers is the capability of the applet to interact with the AS/400 system. When you create an applet, you can include Java classes from the AS/400 Toolbox for Java that let you access objects on the AS/400 system. For example, you can:

- Get records from the AS/400 system database using SQL statements or record-level access techniques.
- Call programs or invoke commands on the AS/400 system.
- Send and receive entries from data queues on the AS/400 system.

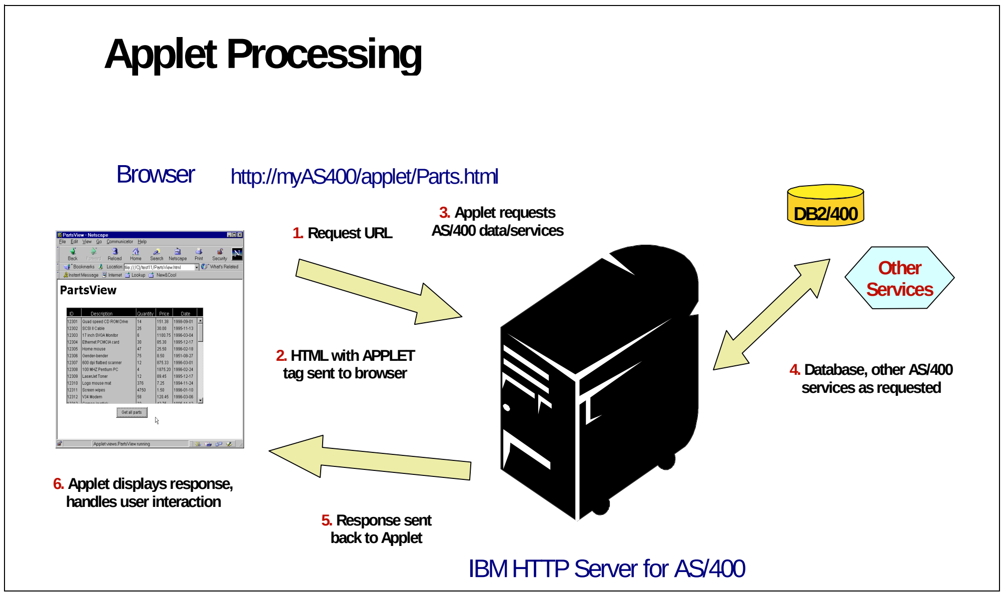

Applet Processing
Browser
http://myAS400/applet/Parts.html
1. Request URL
3. Applet requests AS/400 data/services
DB2/400
Other Services
2. HTML with APPLET tag sent to browser
4. Database, other AS/400 services as requested
PartsView
6. Applet displays response, handles user interaction
5. Response sent back to Applet
IBM HTTP Server for AS/400

Figure 8. Applet Processing

Assume that your IBM HTTP Server for AS/400 is properly configured with the required `MAP` and `PASS` directives and that your browser has Java support enabled. Here is the process that occurs when you invoke an applet. The step numbers correspond to the steps shown in Figure 8.

1. **An HTML page that contains an APPLET tag is requested on an incoming URL and returned to the browser.**

   You start an applet by requesting an HTML file that contains an `APPLET` tag. As the page is sent to your browser, the Java classes that are used in the applet are also sent to the browser. After receiving the applet, the browser starts it and passes control to the applet.

2. **A request is sent from the applet to the AS/400 system.**

   Once the applet has started, you interact with it as you would with other forms displayed in the browser. The applet may require that you enter data or make selections. You usually have one or more buttons in the applet that you can click. The buttons are not associated with a `FORM` statement in an HTML form, but rather are used to invoke methods in the applet's Java code. It is up to the Java code in the applet to use methods to send requests to the AS/400 system.

3. **The AS/400 system services the request from the applet.**

   The request from the applet is serviced the same as other types of program requests running on the AS/400 system. For example, if the request was to create an SQL result set, the SQL processor is invoked to query the AS/400 system database.

4. **The response is sent back to the applet.**

   After the AS/400 system processes the request, the results of servicing the request are sent back to the applet. The results are in the format that pertains to the request. For example, an SQL request generates a result set. A data queue read operation generates a packet of bytes that contains the data queue entry. The applet receives the results using the Java class and methods appropriate to the request.

5. **The applet displays the response data.**

   Once the results are available to the applet, it can display those results using any of the user interface components that were included in the applet when it was designed. The display is not limited to HTML only. After formatting and displaying the results, the applet is available for additional user interaction, which may include additional requests to the applet for more data from the AS/400 system.

1.5.4 How Applets are Different from Net.Data and CGI Programs

Earlier in this chapter, you learned how Net.Data and CGI programs can be used with the AS/400 system. The examples used SQL statements to query the AS/400 system database and return results to the browser in HTML forms. In each case, you worked with an initial HTML page that prompted you for the query to run, invoked the query on the AS/400 system, and received an HTML page in response to the query.

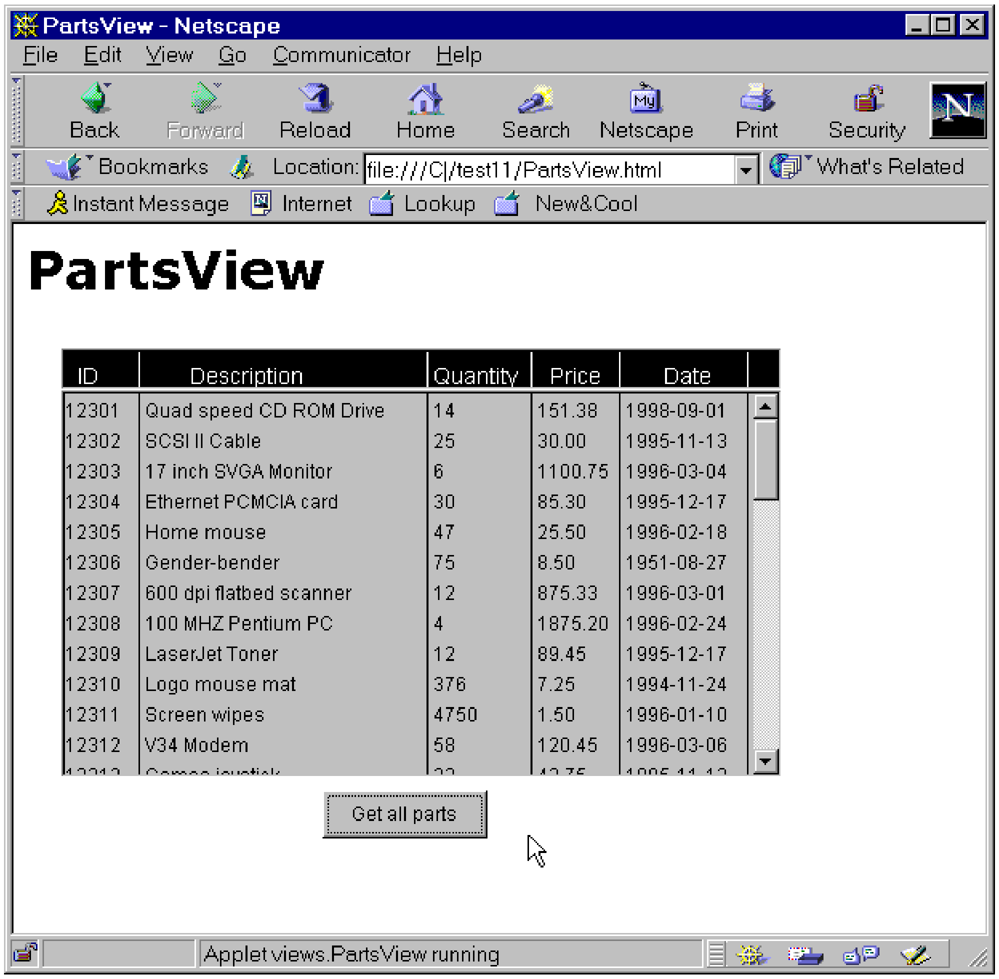
With applets, you do not need to invoke a program on the AS/400 system. Instead, the Java classes in the applet go more directly to the AS/400 system resource with which you want to work. The results of your request return directly to the applet, rather than returning as an HTML page. Because the applet is not limited to working with HTML as the browser is, the applet can display the results using any number of visual components, not only those supported by the browser in HTML.

An applet can also be used for the initial editing of user input before sending a request to the AS/400 system. If you wanted to edit user input when using Net.Data or CGI, you need to include scripting (JavaScript or VBScript) on your Web page. The alternative is to send the user input to the Net.Data macro or CGI program, which then performs edits and returns error notifications as required.

1.5.5 A Sample Applet

Figure 9 on page 33 shows a sample applet in the Netscape Navigator browser. At this point, the applet has sent a query request to the AS/400 system for the list of parts and formatted the returned part data into a multicolumn listbox.

When the applet starts, the Netscape Navigator browser displays the Java Security message box shown in Figure 10 on page 33. Even with the sandbox in place, the browser gives you the chance to deny the applet from starting on your computer.


PartsView - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security
Bookmarks Location: file:///C|/test11/PartsView.html What's Related
Instant Message Internet Lookup New&Cool

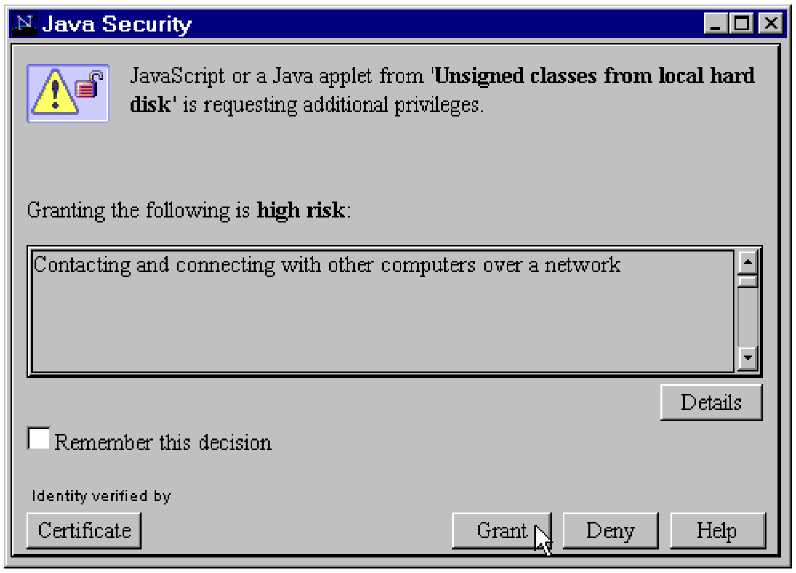
PartsView

ID	Description	Quantity	Price	Date
12301	Quad speed CD ROM Drive	14	151.38	1998-09-01
12302	SCSI II Cable	25	30.00	1995-11-13
12303	17 inch SVGA Monitor	6	1100.75	1996-03-04
12304	Ethernet PCMCIA card	30	85.30	1995-12-17
12305	Home mouse	47	25.50	1996-02-18
12306	Gender-bender	75	8.50	1951-08-27
12307	600 dpi flatbed scanner	12	875.33	1996-03-01
12308	100 MHZ Pentium PC	4	1875.20	1996-02-24
12309	LaserJet Toner	12	89.45	1995-12-17
12310	Logo mouse mat	376	7.25	1994-11-24
12311	Screen wipes	4750	1.50	1996-01-10
12312	V34 Modem	58	120.45	1996-03-06

Get all parts

Applet views.PartsView running

Figure 9. PartsView Applet Running in the Netscape Browser



JavaScript or a Java applet from '**Unsigned classes from local hard disk**' is requesting additional privileges.

Granting the following is **high risk**:

Contacting and connecting with other computers over a network

Details

Remember this decision

Identity verified by

Certificate Grant Deny Help

Figure 10. Java Security Message Displayed under Netscape

1.5.6 When to Use Applets

Applets are useful when you need to work with AS/400 system resources and you do not want to create Net.Data or CGI programs on the AS/400 system. You can also choose to work with applets because of the greater control you have in designing the user interface and interacting with the user at runtime.

Another reason to use applets is because you can use the industry-standard Java language, rather than the proprietary AS/400 system Net.Data or CGI programming languages. Although the Java classes to access the AS/400 system are proprietary, the classes are readily usable by any Java programmer with basic knowledge of the AS/400 system.

1.5.7 Applet Development

Rather than review the applet code and development process at this point, read Chapter 3, “Introduction to AS/400 Applets” on page 51 in this redbook. That chapter presents a complete overview of the applet development process and describes different options you have for hosting and deploying applets.

1.6 Java Servlets

Starting with OS/400 V4R3, an exciting new technique is available for AS/400 system Web serving—Java servlets. Based on specifications and Java classes developed by Sun Microsystems, Inc., Java servlets provide an alternative to Net.Data, CGI programming, and applets. Using servlets, you can parse requests from HTML forms and use simple `print` and `println` methods to send response HTML back to the browser.

Servlets run entirely on the AS/400 system as part of the IBM HTTP Server for AS/400. Because servlets run on the AS/400 system, they have ready access to the AS/400 system database and other system resources.

1.6.1 Why Use Servlets

There are several reasons why you should consider using servlets for your AS/400 system Web applications:

- **Few browser dependencies**

  One of the problems that is described in Section 1.5, “Java Applets” on page 28, is the differences between browsers. This includes differences in versions from the same companies. Since Java servlets are primarily tasked with generating and returning HTML as their output, servlets can be used with "least common denominator" browsers. With applets, you need several prerequisites in place in the browser to ensure that your applet runs correctly.

- **Uses industry-standard Java**

  The AS/400 system has traditionally been perceived as a proprietary platform, which limited its appeal as a choice for Web serving. Although using Net.Data and CGI programming techniques for Web serving is fine for AS/400 system users who have those skills, there was little to attract Java programmers to the AS/400 system as a Web serving platform. With Java servlet support, the AS/400 system is now in the mainstream as a powerful Web serving system. It is far easier to present the traditional AS/400 system strengths, such as its integrated database and security when there is a widely accepted programming language available for working with the system.

- **Handles form parsing, GET/POST processing, and STDOUT processing**

  In comparison to CGI programming, Java servlets are much easier to work with. Most of the work in CGI programming is concerned with getting the input data from the Web form, parsing the data into discrete field name and value pairs, and writing response HTML back to STDOUT. In contrast, Java servlets provide two simple input/output stream objects to get the data and write the response, and two simple methods to parse field name/value pairs.

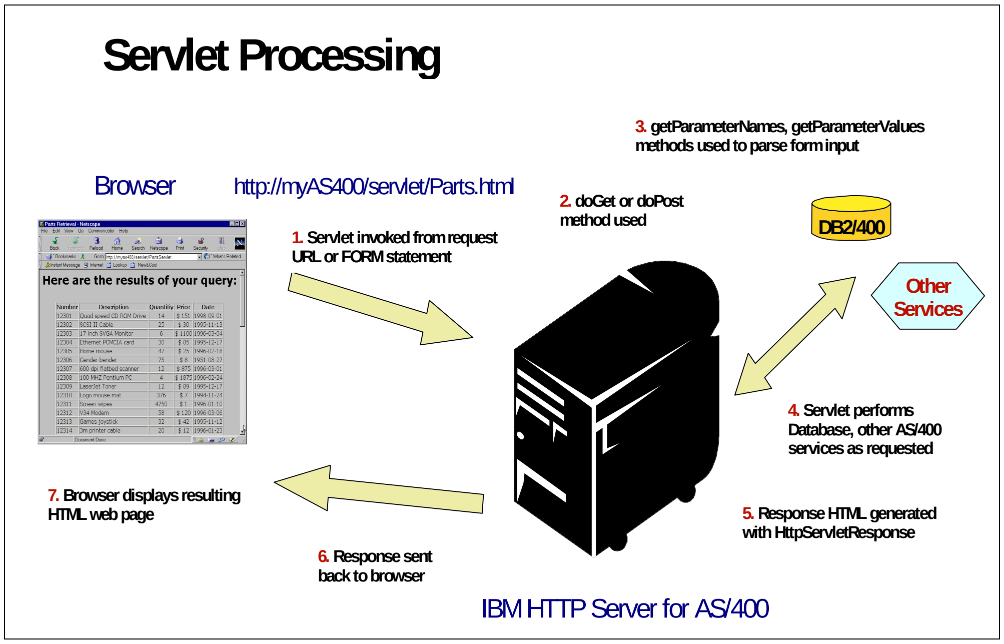
- **Runs in a multi-threaded pool as a prestarted thread**

  One side effect of having a popular Web site is that the response time of your AS/400 system may be adversely affected. When a request for Net.Data or CGI processing is received at the AS/400 system, a new job is started. Because Net.Data and CGI programs are typically short-lived, there is little that can be done to optimize their performance, since most of the work associated with the programs is in job initiation and termination. In contrast, Java servlets take advantage of the multi-threaded job capability available with the V4R3 version of the IBM HTTP Server for AS/400. In fact, your servlets can optionally be started when the IBM HTTP Server for AS/400 is started, so that they are available and waiting for incoming requests. Servlets also do not necessarily end when they are done servicing a request. They remain active and can service additional requests as they arrive at the AS/400 system. If you create an equivalent Net.Data or CGI application and a Java servlet, you typically see much better response time with the Java servlet.

1.6.2 Servlet Processing

Servlet processing is similar to CGI processing. The primary difference is that the servlet can be prestarted in the multi-threaded job pool so that there is no start-up overhead when it is invoked.

If a servlet is not currently active when it is invoked, you incur the start-up overhead on its first usage. After that point, the servlet is available for subsequent invocations.


Servlet Processing
Browser
http://myAS400/servlet/Parts.html
Parts Retrieval - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Go to: http://myas400/servlet/PartsServlet What's Related
Instant Message Internet Lookup New&Cool
Document: Done
1. Servlet invoked from request URL or FORM statement
2. doGet or doPost method used
3. getParameterNames, getParameterValues methods used to parse form input
DB2/400
Other Services
4. Servlet performs Database, other AS/400 services as requested
5. Response HTML generated with HttpServletResponse
6. Response sent back to browser
7. Browser displays resulting HTML web page
IBM HTTP Server for AS/400

Here are the results of your query:

Number	Description	Quantitiy	Price	Date
12301	Quad speed CD ROM Drive	14	$ 151	1998-09-01
12302	SCSI II Cable	25	$ 30	1995-11-13
12303	17 inch SVGA Monitor	6	$ 1100	1996-03-04
12304	Ethernet PCMCIA card	30	$ 85	1995-12-17
12305	Home mouse	47	$ 25	1996-02-18
12306	Gender-bender	75	$ 8	1951-08-27
12307	600 dpi flatbed scanner	12	$ 875	1996-03-01
12308	100 MHZ Pentium PC	4	$ 1875	1996-02-24
12309	LaserJet Toner	12	$ 89	1995-12-17
12310	Logo mouse mat	376	$ 7	1994-11-24
12311	Screen wipes	4750	$ 1	1996-01-10
12312	V34 Modem	58	$ 120	1996-03-06
12313	Games joystick	32	$ 42	1995-11-12
12314	3m printer cable	20	$ 12	1996-01-23

Figure 11. Servlet Processing

The following steps describe how a servlet is invoked, how it services an incoming request, and how it returns results to the browser. The numbers correspond to the steps shown in Figure 11.

1. **The Servlet is invoked from a URL or FORM statement.**

   The servlet is identified either in a URL that you type into the browser or click as a link. Or, it can be specified on an HTML `FORM` statement that is used when a `SUBMIT` button is clicked. The request is sent using the HTTP protocol to the IBM HTTP Server for AS/400, which identifies the servlet to invoke. If the servlet is not currently active, the IBM HTTP Server for AS/400 starts an instance of the servlet in the multi-threaded job pool reserved for Java servlets. If the servlet is currently active, the IBM HTTP Server for AS/400 passes control to the servlet.

2. **The Servlet uses the doGet or doPost method to read form data.**

   The Java servlet API includes the `doGet` and `doPost` methods that correspond to the HTML `METHOD="GET"` and `METHOD="POST"` techniques of sending data from the browser to the IBM HTTP Server for AS/400. You do not have to do anything in your servlet program to determine which method to use. The servlet chooses the correct method (`doGet` or `doPost`) depending on the `METHOD` used in the form.

   Regardless of the method used, the input data is available to the servlet in the `HttpServletRequest` input stream.

3. **The getParameter methods are used to parse field name/value pairs.**

   Now that the form data is available in the `HttpServletRequest` stream, it can be parsed into field name/value pairs that correspond to the data fields used on the HTML form. The Java servlets API includes the `getParameterNames` and `getParameterValues` methods to retrieve the list of field names and values from the input stream. After retrieving the name/value pairs, the values are available in enumerations within the servlet.

4. **The Servlet processes database and other requests**

   At this point, all of the data from the form is available to the servlet. The servlet can now run the functions that are required to service the request. For example, the servlet may run an SQL query against the AS/400 database, or use other Java classes in the AS/400 Toolkit for Java to work with other AS/400 system resources.

5. **The response HTML is generated.**

   The servlet can start generating response HTML to send to the browser at any point. Typically, the servlet generates HTML headers, followed by the actual form heading, then one or more lines of data, and finally a page footer.

   The Java servlets API provides the `HttpServletResponse` output stream to transport generated HTML statements from the servlet back to the browser. You create well-formed HTML statements as simple strings, using concatenation as necessary to build a string of HTML tags and the response data. To actually send the HTML, you simply use the `print` or `println` methods on the `HttpServletResponse` stream object.

6. **The IBM HTTP Server for AS/400 sends the response HTML to the browser.**

   As the HTML is written in the servlet, it is sent from the IBM HTTP Server for AS/400 to the browser. The STDOUT file is used, as is common for all server to browser communication.

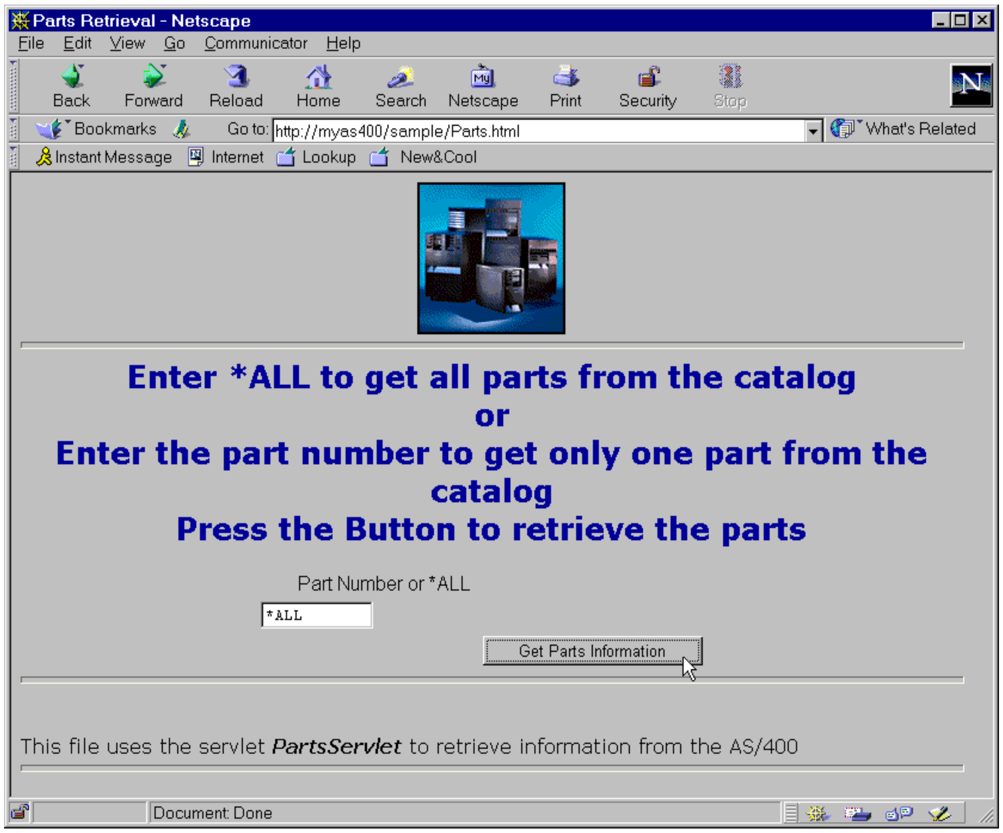
7. **The resulting Web page is displayed in the browser.**

   The browser now displays the resulting Web page. Because the page is composed of standard HTML elements, there are no special requirements or security considerations for the browser.

1.6.3 A Sample Servlet

The servlet sample developed for this redbook performs the same function as the Net.Data macro described in Section 1.3, “Net.Data” on page 6 and the RPG-CGI program described in Section 1.4, “Common Gateway Interface (CGI) Programming” on page 15.

Figure 12 on page 38 shows the initial prompting form that is displayed when you start the servlet application. You enter the part number selection and click the Get Parts Information button. When you click the button, the `FORM` statement in the HTML form invokes the servlet on the AS/400 system.


Parts Retrieval - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Go to: http://myas400/sample/Parts.html What's Related
Instant Message Internet Lookup New&Cool

Enter *ALL to get all parts from the catalog
or
Enter the part number to get only one part from the catalog
Press the Button to retrieve the parts

Part Number or *ALL

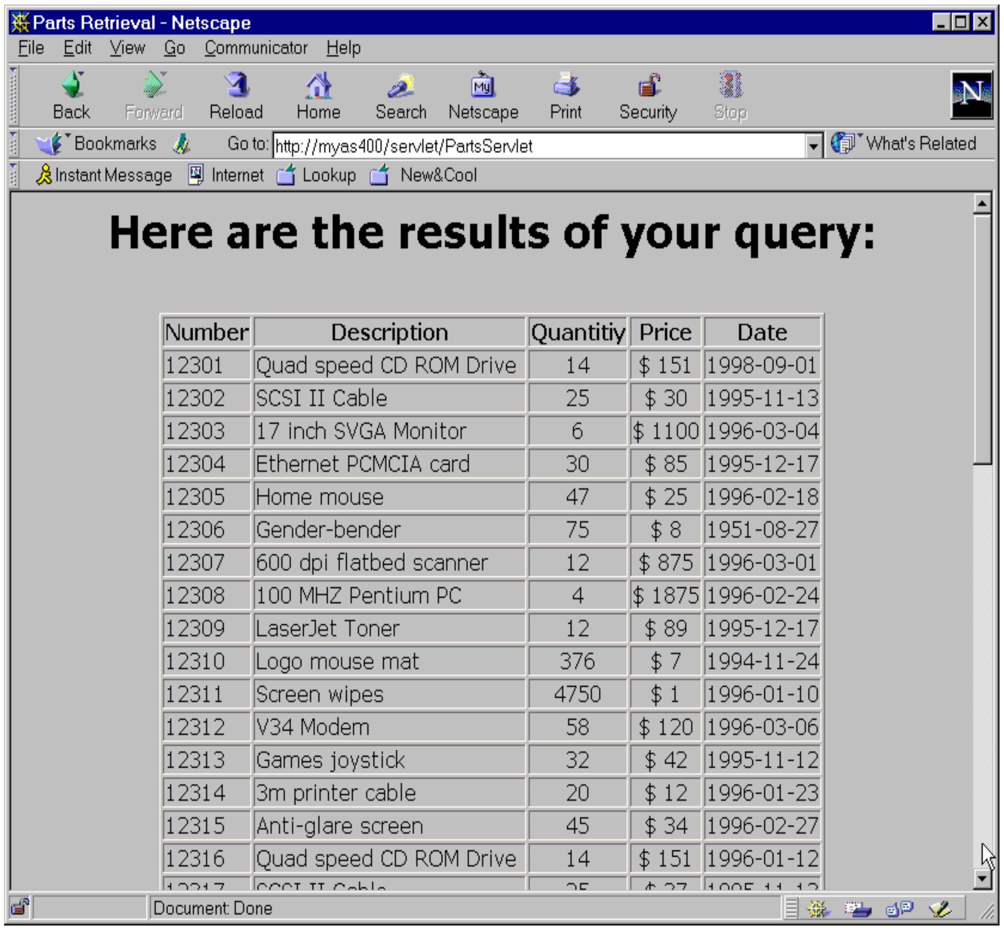
*ALL

Get Parts Information

This file uses the servlet *PartsServlet* to retrieve information from the AS/400

Figure 12. Servlet Example Prompting Page

The servlet uses your part number selection to run an SQL query on the Parts database. After running the SQL statement, the servlet iterates through the result set. For each part retrieved, the servlet generates the HTML to display the part data in an HTML table. Figure 13 on page 39 shows the resulting table that the servlet returns to the browser in response to the query.


Parts Retrieval - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Go to: http://myas400/servlet/PartsServlet What's Related
Instant Message Internet Lookup New&Cool
Document: Done

Here are the results of your query:

Number	Description	Quantitiy	Price	Date
12301	Quad speed CD ROM Drive	14	$ 151	1998-09-01
12302	SCSI II Cable	25	$ 30	1995-11-13
12303	17 inch SVGA Monitor	6	$ 1100	1996-03-04
12304	Ethernet PCMCIA card	30	$ 85	1995-12-17
12305	Home mouse	47	$ 25	1996-02-18
12306	Gender-bender	75	$ 8	1951-08-27
12307	600 dpi flatbed scanner	12	$ 875	1996-03-01
12308	100 MHZ Pentium PC	4	$ 1875	1996-02-24
12309	LaserJet Toner	12	$ 89	1995-12-17
12310	Logo mouse mat	376	$ 7	1994-11-24
12311	Screen wipes	4750	$ 1	1996-01-10
12312	V34 Modem	58	$ 120	1996-03-06
12313	Games joystick	32	$ 42	1995-11-12
12314	3m printer cable	20	$ 12	1996-01-23
12315	Anti-glare screen	45	$ 34	1996-02-27
12316	Quad speed CD ROM Drive	14	$ 151	1996-01-12

Figure 13. Servlet Example Output

1.6.4 When to Use Servlets

If you have not yet started creating AS/400 Web serving applications, seriously consider adopting Java servlets as the technique you will use, even if you do not yet use Java. In fact, learning Java by working with servlets is ideal, since servlets are basically batch processes that do not have to deal with user interface issues found in client-side programs.

There are only a few classes and methods that you need to work with to create functional servlets. As with Net.Data and CGI, you may find that most of the work involved in creating a servlet is actually spent creating HTML code.

If you anticipate having a lot of activity on your IBM HTTP Server for AS/400 for dynamic Web pages, consider using servlets as opposed to the other techniques. Servlets allow your Web site to scale-up much better than the other server-side techniques.

In summary, Java servlets are currently the best alternative for Web serving from the IBM HTTP Server for AS/400. They provide an easy to learn and use, industry-standard, and highly scalable architecture on which to build your AS/400 system Web presence.

1.6.5 Servlet Development

Rather than review the servlet code and development process at this point, read Chapter 4, “Introduction to AS/400 Servlets” on page 159 in this redbook. That chapter presents a complete overview of the servlet development process and describes the different options you have for hosting and deploying servlets.

Chapter 2. IBM HTTP Server for AS/400

Although IBM has included a no-charge HTTP server with OS/400 since V3R2 and V3R7, the IBM HTTP Server for AS/400 introduced with OS/400 V4R3 is a significantly different product than the previous versions. Some of the changes are immediately apparent, such as the product name and number, and the browser-based administration program. From the point of view of an AS/400 system Web programmer, the most important enhancements include:

- Support for the HTTP 1.1 protocol
- Support for proxy, cache, and local memory cache for selected Web pages
- Greatly improved CGI programming techniques, including Java CGI, PERL, REXX, and new APIs for traditional AS/400 system programming languages
- Persistent CGI, which enables a CGI program to remain active and span multiple Web pages
- Integration of Secure Sockets Layer (SSL) support, Certificates and Digital ID authentication
- Integration of the WebSphere Application Server for AS/400, which provides support for Java servlets on the AS/400 system

This chapter provides an overview of those enhancements. You can find information on how to install, configure, maintain, monitor, and program the IBM HTTP Server for AS/400 in the following publications:

- *HTTP Server for AS/400 Quick Beginnings*, GC41-5433
- *HTTP Server Webmaster's Guide*, GC41-5434

2.1 Product Packaging

Prior to OS/400 V4R3, the HTTP server provided with OS/400 is known as the Internet Connection Server (ICS). It is part of the TCP/IP Connectivity Utilities (5769-TC1) Licensed Program Product (LPP). A chargeable companion LPP, the Internet Connection Secure Server (ICSS, 5769-NC1 or 5769-NCE) is available for OS/400 V4R1 and V4R2 to provide SSL support.

Starting with V4R3, the HTTP server provided with OS/400 is called the IBM HTTP Server for AS/400. The HTTP Server is provided as a separate no-charge LPP (5769-DG1). SSL support is provided in the no-charge Cryptographic Access Provider LPP (5769-AC1, 5769-AC2 or 5769-AC3).

Another packaging change is the separation of the Digital Certificate Manager from the Web server itself. In V4R3, the Digital Certificate Manager is installed as option 34 of the base operating system (5769-SS1). By installing the Digital Certificate Manager as a separate component, other AS/400 system servers can use its services without requiring that the HTTP Server component be installed.

Table 4 on page 42 is a cross reference between pre-V4R3 and V4R3 components. When you install OS/400 V4R3 on your AS/400 system, you can either manually install the new LPPs for the HTTP Server, Cryptographic Services and Digital Certificate Manager. Or, you can use the "Prepare for install" option on the `GO LICPGM` menu to identify and install the new components from your installation media.

Table 4. Summary of OS/400 Pre-V4R3 and V4R3 HTTP Server Components

OS/400 Pre-V4R3	**OS/400 V4R3**
Internet Connection Server (ICS), part of TCP/IP Connectivity Utilities (5769-TC1)	IBM HTTP Server for AS/400 (5769-DG1)
Internet Connection Secure Server (ICSS), chargeable LPP 5769-NC1 – US/Canadian version 5769-NCE – Export version	Cryptographic Access Provider, no-charge LPP 5769-AC1 – 40 bit strength 5769-AC2 – 56 bit strength 5769-AC3 – 128 bit strength **Note**: The installation media includes only one of these providers, based on the country where the AS/400 system is located.
Digital Certificate Manager, part of ICS (5769-TC1)	OS/400 Digital Certificate Manager (5769-SS1, option 34)
All HTTP server tasks run in subsystem QSYSWRK.	All HTTP server tasks run in subsystem QHTTPSVR.

2.2 HTTP 1.1 Protocol

The IBM HTTP Server for AS/400 implements the HTTP version 1.1 protocol. This is the current version of the HTTP protocol.

Note

You can find more information about the HTTP 1.1 protocol and other Internet standards at the Web site for the Internet Engineering Task Force: http://www.ietf.org

Two of the more significant enhancements supported by the IBM HTTP Server for AS/400 are:

- Persistent connections
- Virtual hosts

2.2.1 Persistent Connections

When you enter a URL into your browser's address line or click on a link on a Web page, you open a connection between your browser and the HTTP server. Prior to the availability of persistent connections, each file referenced on the Web page was retrieved using a separate connection. This type of retrieval is tremendously costly for the HTTP server and the network since there is overhead required to establish and terminate each connection.

Persistent connections are the default behavior for an HTTP server that implements the HTTP 1.1 protocol. Persistent connections provide the following advantages:

- Because there is less opening and closing activity, CPU and memory utilization on the HTTP Server is reduced.
- Network congestion is minimized because of the fewer number of TCP/IP packets that are required to request the files.
- HTTP requests and responses can be pipelined on the connection. Using pipelining, a client can make several requests to the server without waiting for the responses to the requests.

2.2.2 Virtual Hosts

In previous versions of the AS/400 system HTTP server, the only way to host multiple Web sites on the AS/400 system and use the default HTTP port (80) is to use different communications adapters in the AS/400 system. If you want to host multiple Web sites through the same communications adapter, only one of the sites can use the default HTTP port. All other Web sites need a separate port assignment. To request those Web sites, the unique port assignment is included as part of the requesting URL.

Starting with the V4R3 IBM HTTP Server for AS/400, you can enable virtual hosting. This allows you to host any number of Web sites through one communications adapter. With virtual hosting, you do not need to assign a unique port to each Web site.

Virtual hosting is useful if you need to provide multiple "top-level" URLs for your Web sites or if you are providing Internet Service Provider (ISP) services to clients.

2.3 Proxy, Cache, and Local Memory Cache

The IBM HTTP Server for AS/400 can be configured as a non-caching or caching proxy server. When used as a non-caching proxy, the primary benefit of enabling proxy services is that the IP addresses used on your internal network are not sent out of your network. The proxy service forwards the request from your internal network using the IP address of the proxy server, not the address of the original requester. When the proxy server receives the response, it forwards the response to the original requester.

2.3.1 Proxy Caching

With caching enabled, the proxy server can act as a high-speed local store of previously accessed Web pages. For example, if you frequently access the same set of Web pages from one or more sites, it may be advantageous to activate the caching feature. The retrieved Web page is stored locally on your AS/400 system. Any subsequent accesses to the page occur at LAN speed, rather than Internet speed.

Web pages can be encoded with a "no-cache" attribute or a specific expiration date. You can also configure the IBM HTTP Server for AS/400 proxy service so that it periodically performs "garbage collection" to remove expired files from the cache. The cache is located in the QOpenSys file system, which provides support for case sensitivity in file names. You configure the maximum size of the cache (which uses AS/400 system disk storage), protocols and URLs to cache or not cache.

2.3.2 Proxy Logging

Another use of the proxy service (with or without caching) is to log client requests. Some of the data available includes:

- Client IP address
- Date and time
- URL requested
- Byte count
- Success code

2.3.3 Local Memory Cache

A proxy cache is traditionally most beneficial to clients on your network since it lets you store files that were retrieved from other Web sites. You can provide a caching service for files on your site using the local memory cache configuration options.

To use a local memory cache, you identify an amount of memory to allocate and a set of files to be cached. When the IBM HTTP Server for AS/400 is started, the files are read into the local memory cache, up to the limit of the amount of memory allocated or the limit of the number of files that you allow to be cached. When a request is received at your IBM HTTP Server for AS/400, the local memory cache is checked first to determine if it has a copy of the requested file. If so, the file is served from the cache, which is significantly faster than if the file is retrieved from disk storage.

2.4 CGI Programming

As described in Chapter 1, there are several techniques available to you to create Common Gateway Interface (CGI) programs for use with the IBM HTTP Server for AS/400. The V4R3 IBM HTTP Server for AS/400 provides the following enhancements for GCI programming:

- Java and REXX CGI
- Non-parsed headers CGI
- QzhbCgiParse API

2.4.1 Java and REXX CGI

In addition to traditional AS/400 system languages used for CGI programming (ILE RPG, ILE COBOL and ILE C), you can now use Java and REXX to create CGI programs. Java, in this case, means Java applications that use AS/400 system support for working with files `STDIN` and `STDOUT` or the CGI APIs, not Java servlets. Support for Java servlets is provided with the WebSphere server, which is an add-on to the IBM HTTP Server for AS/400.

Note

Perl version 5.003 is also available for the AS/400 system. Although IBM provides a version of Perl ported to the AS/400 system, IBM does not provide any support for the Perl environment or Perl script execution problems on the AS/400 system. You can download the Perl execution environment and related files in the Web Builder's Workshop area from the Web site at: http://www.as400.ibm.com/techstudio.

2.4.2 Non-parsed Headers CGI

Most CGI programs generate response headers and HTML and return the data to the requesting client through the IBM HTTP Server for AS/400. However, there may be occasions when you want to generate a response in your CGI program and return it directly to the requesting client.

If the name of your CGI program begins with `nph_` or `nph-`, its output is not converted by the IBM HTTP Server for AS/400. You are responsible in your non-parsed header CGI program for creating a complete HTTP response message, including the HTTP return code and status information.

2.4.3 QzhbCgiParse API

Prior to the V4R3 IBM HTTP Server for AS/400, CGI programs written in ILE RPG or ILE COBOL needed to use several APIs to work with HTML form data:

- `QtmhGetEnv` to get the value of the `QUERY_STRING` environment variable and other environment variables
- `QtmhRdStin` to read data from file `STDIN` for a `POST` request
- `QtmhCvtDb` to parse data from the `QUERY_STRING` environment variable or the data retrieved from `STDIN`

Although the `QtmhGetEnv` and `QtmhRdStin` APIs are easy to work with and relatively straightforward, the `QtmhCvtDb` API is somewhat cumbersome to use.

The IBM HTTP Server for AS/400 provides the `QzhbCgiParse` API, which combines the functionality of the three `Qtmh` APIs into one. Some of the features of this API include:

- Support for both `GET` and `POST` data
- Parses form data into field name/field value pairs
- Uses a list technique similar to other AS/400 system APIs. Rather than determine the format of the data to be retrieved in advance (that is, while coding the CGI program), you can simply call the API and walk through the list of retrieved values. If your HTML form changes, you do not necessarily need to recode and recompile your CGI program to accommodate the changes.

2.5 Persistent CGI

One of the most significant enhancements in the V4R3 IBM HTTP Server for AS/400 for CGI programs is the introduction of persistent CGI, or persistency. To understand the benefit of persistency, you can compare it with what happens in a non-persistent CGI application:

1. The requester makes entries on an HTML form and clicks a submit button on the form. The submit button has an associated URL that invokes a CGI program on the HTTP Server. A connection is created between the HTML form and the CGI program.
2. The CGI program is invoked and retrieves the form data (using either the `GET` or `POST` method). The CGI program processes the data, prepares a response, and returns the response to the requester using the `STDOUT` file.
3. After completing the output to `STDOUT`, the connection between the requester and the CGI program is terminated when the end-of-file indication is sent through `STDOUT`. At this point, the CGI program ends, since there is no way for any subsequent requests to reconnect to that instance of the CGI program.
4. Subsequent requests start at step 1 again. A new connection must be established between the HTML form and the HTTP Server.

Although this scheme is fine for HTML forms that can be processed in a single invocation of a CGI program, most Web transactions involve more than one HTML form or multiple interactions with the same form.

For example, a "shopping cart" application is a form that is continuously added to as the requester selects items from other Web pages. As items are added to the cart, the list of items must be maintained somewhere because in a non-persistent application, there is no program running on the HTTP Server that can retain the list. Some solutions for maintaining this state information include temporary files on the HTTP Server or "cookies" that are written to the requester's local storage.

2.5.1 How Persistent CGI Works

In contrast, a Web application that uses persistent CGI does not need to rely on temporary storage schemes to maintain state information. A persistent CGI application may work as described in the following series of events:

1. The requester fills in an HTML form and clicks the submit button. The requesting URL invokes a CGI program. The connection between the requester and the HTTP Server is now active.
2. The CGI program prepares output to send back to the requester. However, a special header record (`Accept-HTSession`) is returned to the requester along with a "handle" to identify the persistent CGI program that services the requester.
3. Output from the CGI program is sent to the requester in file `STDOUT`. As with non-persistent CGI, the connection between the requester and the CGI program is terminated after end-of-file. However, the CGI program itself remains active, since the HTTP Server can identify and use it for additional requests from the same requester.
4. Subsequent requests send the handle along with data from the form. The HTTP Server recognizes the incoming request as belonging to the particular instance of the CGI program. The new connection between the requester and the HTTP Server is linked to the already executing CGI program. Because the program never ended, any internal data structures that it had created on previous requests are still available.

2.5.2 Controlling Persistent CGI

Since there is no way to guarantee that a requester completes a transaction, the IBM HTTP Server for AS/400 includes directives that let you specify the amount of time a persistent CGI application can be inactive before being terminated. The time-out value can be specifed at both the server level and the application level so that you can allow some applications more time to complete.

No indication is sent to the requester when the timeout is reached. Also, any database and file processing is your responsibility. For database applications where changes occurred, you typically want to perform a `ROLLBACK` operation when a persistent CGI timeout occurs.

2.6 Cryptographic Support, Certificates, and Digital ID

Prior to the V4R3 IBM HTTP Server for AS/400, cryptographic support for the HTTP Server was provided in the Internet Connection Secure Server (ICSS) LPP (5769-NC1 and 5769-NCE). ICSS is a chargeable item and is required if you need to provide SSL support and server authentication.

2.6.1 Cryptographic Access Provider

Starting with V4R3, IBM includes the Cryptographic Access Provider as a no-charge LPP with OS/400. There are three versions of the Cryptographic Access Provider:

- 5769-AC1 provides 40-bit encryption
- 5769-AC2 provides 56-bit encryption
- 5769-AC3 provides 128-bit encryption

Only one version is shipped with your OS/400 installation media. The version shipped is based on the country where the AS/400 system is installed to comply with United States export laws for computer encryption products and local laws of the country.

By providing the Cryptographic Access Provider along with OS/400, it is now possible to use the support provided by this LPP even if the IBM HTTP Server for AS/400 is not installed.

2.6.2 Digital Certificate Manager

Another change introduced with OS/400 V4R3 is the Digital Certificate Manager option. This is installed as option 34 of the OS/400 base installation.

Digital Certificate Manager provides support for generating and maintaining digital certificates. Certificates are used for both server and client authentication. Although you can generate certificates for your HTTP Server to attest to its authenticity, you most likely need to apply for and receive a certificate from a well-known certificating authority if you intend to conduct e-commerce with your AS/400 system. Both the Netscape Navigator and Microsoft Internet Explorer browsers include a list of well-known certificating authorities that will be accepted by the browsers to authenticate Web pages from your HTTP Server.

Note

One of the most widely known and used certificating authorities is VeriSign, Inc. You can learn more about certificates and digital IDs at their Web site: http://www.verisign.com

2.6.3 Digital ID

Another feature introduced with the V4R3 IBM HTTP Server for AS/400 is using digital IDs to provide client certification. Although we traditionally think of digital IDs to provide verification of the server's authenticity, it is also useful for clients to provide the server with a guarantee of their authenticity.

Client authentication using digital IDs can be used as an alternative to prompting for a user ID and password.

2.7 WebSphere Application Server for AS/400

One of the most significant additions to the V4R3 IBM HTTP Server for AS/400 is the WebSphere Application Server for AS/400 ("WebSphere"). WebSphere is installed as part of the IBM HTTP Server for AS/400 (5769-DG1).

Attention

As of October 1998, there are several PTFs that you must obtain and apply to your AS/400 system prior to working with WebSphere. You can obtain the list of required PTFs at this Web site:
http://www.as400.ibm.com/tstudio/http/services/WASInfo.htm

You can also find additional information about configuring and using WebSphere at this site.

2.7.1 What WebSphere Provides

WebSphere provides a Java-based environment in which Java servlets are hosted and execute. Servlets are based on Sun Microsystems' Servlets API and include complete support for working with an incoming HTTP data stream and writing a response HTTP data stream.

WebSphere servlets run in a multi-threaded environment. That means that a servlet can potentially use additional threads so that it can service requests more quickly. Servlets can also be started when the WebSphere server is started, so there is no overhead associated with starting the servlet from the requester's point of view, as there is with CGI programs.

2.7.2 Accessing the WebSphere Server

The WebSphere server is configured using an applet specifically designed for it. You must invoke the applet from a Java-capable browser, such as Sun Microsystems' HotJava browser, Netscape Navigator 4.x, or Microsoft Internet Explorer 4.x. To invoke the applet, enter a URL similar to this:

```
http://myServer:9090
```

By default, port 9090 is used to identify the WebSphere configuration page.

2.8 Summary

Although it is possible to develop highly-functional Web serving applications on the AS/400 system prior to V4R3, it is advantageous to install OS/400 V4R3 and the associated LPPs for Internet enablement before starting any serious development efforts. It is important to understand that the underlying HTTP Server has undergone major changes, and that the V4R3 version of the IBM HTTP Server for AS/400 is the foundation for the future.

In several informal tests, we have not found any cases where configurations or CGI programs written for pre-V4R3 need to be significantly changed to work on the V4R3 IBM HTTP Server for AS/400. In one test, a complete upgrade to OS/400 V4R3 was performed over an OS/400 V4R2 system. When the upgrade was complete, the IBM HTTP Server for AS/400 started without incident and could use the existing configurations.

If you are just getting started with developing Internet applications for your AS/400 system, you may want to immediately investigate Java servlets, rather than create CGI programs using traditional AS/400 system programming languages. The new Java servlet support provided by the WebSphere Application Server for AS/400 provides better performance, scalability, and maintainability than the CGI techniques.

Chapter 3. Introduction to AS/400 Applets

In Chapter 1, “AS/400 Internet Application Development Overview” on page 1 of this redbook, you learned that Java applets are a special type of Java program that runs inside a browser. Because the applet is delivered to the browser in an HTML file, you can use applets to provide additional functionality for your Web pages. After the Web page containing the applet is downloaded, the browser starts running the applet. At that point, it is an active program, although it does have to comply with the security constraints imposed by the sandbox and the browser.

You can include classes from the IBM AS/400 Toolbox for Java in your applets. Using Toolbox classes, you can access resources on the AS/400 system. Some of the resources available to your applet using the Toolbox include:

- AS/400 system database
- Data queues
- Compiled AS/400 system programs
- Record-level access using the AS/400 system Distributed Data Management (DDM) server
- Printers and output queues
- Directories and files in the AS/400 system Integrated File System (IFS)

This chapter describes a simple applet that retrieves information from the AS/400 system database using SQL. Records from a sample Parts database are displayed in a multi-column listbox in the applet. The chapter includes these sections:

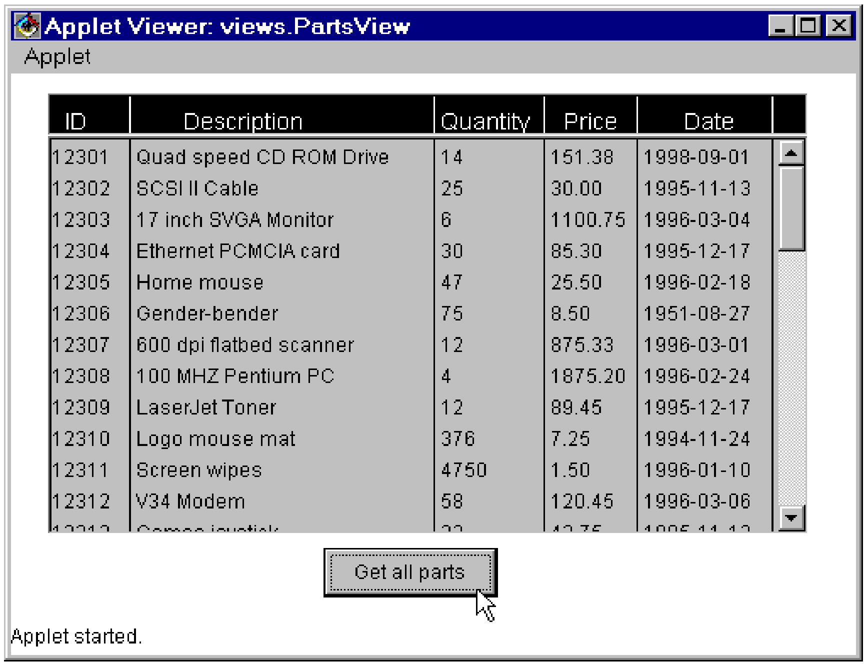
- An overview of the applet as constructed in the IBM VisualAge for Java (Enterprise) version 2.0 Integrated Development Environment (IDE)
- A description of how to import, prepare and test the applet in the VisualAge for Java IDE
- A review and description of the classes and Java source code used in the applet
- Instructions on how to deploy the applet to the local hard drive of a PC
- Instructions on how to deploy the applet from the IBM HTTP Server for AS/400
- Instructions on how to use the Java Plug-in to deploy applets

3.1 The PartsView Applet

Figure 14 on page 52 shows a sample of the PartsView applet running in the Applet Viewer that is included in the VisualAge for Java IDE. As you can see, the applet is simple and performs only one task: retrieve records from an AS/400 system database and display them in a multi-column listbox. Nevertheless, if you follow the steps described in this chapter to create and deploy the sample applet, you can spend your programming time creating more interesting applets.

Note

The example programs discussed in this chapter are available for download from the redbook Web site. Refer to Appendix A.1, “Downloading the Files from the Internet Web Site” on page 299 for details.



ID	Description	Quantity	Price	Date
12301	Quad speed CD ROM Drive	14	151.38	1998-09-01
12302	SCSI II Cable	25	30.00	1995-11-13
12303	17 inch SVGA Monitor	6	1100.75	1996-03-04
12304	Ethernet PCMCIA card	30	85.30	1995-12-17
12305	Home mouse	47	25.50	1996-02-18
12306	Gender-bender	75	8.50	1951-08-27
12307	600 dpi flatbed scanner	12	875.33	1996-03-01
12308	100 MHZ Pentium PC	4	1875.20	1996-02-24
12309	LaserJet Toner	12	89.45	1995-12-17
12310	Logo mouse mat	376	7.25	1994-11-24
12311	Screen wipes	4750	1.50	1996-01-10
12312	V34 Modem	58	120.45	1996-03-06

Figure 14. PartsView Applet

3.1.1 Importing the Source Code for the Applet to the Workbench

The source code for the applet is provided in both source files and in a VisualAge for Java repository file. To work with the applet in the VisualAge for Java IDE, you must import the project from the download repository file into your repository. Then, import the project from your repository into your workspace. The following instructions assume that you have downloaded the VisualAge for Java repository file from the redbooks Web site and that the repository file is in a directory named code5337 on your PC’s disk.

3.1.1.1 Starting the Import Process

Start the import process by right-clicking on the **All Projects** pane in the Workbench. Select the **Import** item on the pop-up menu, as shown in Figure 15.

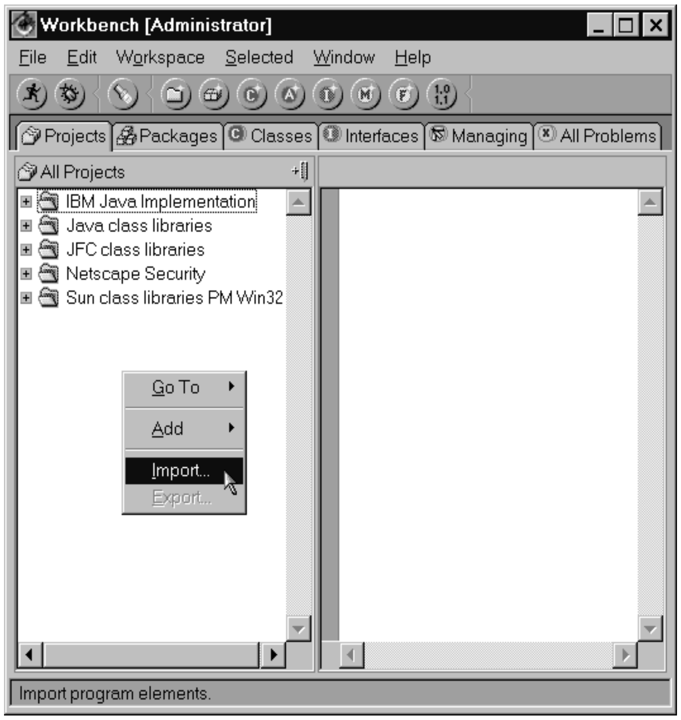
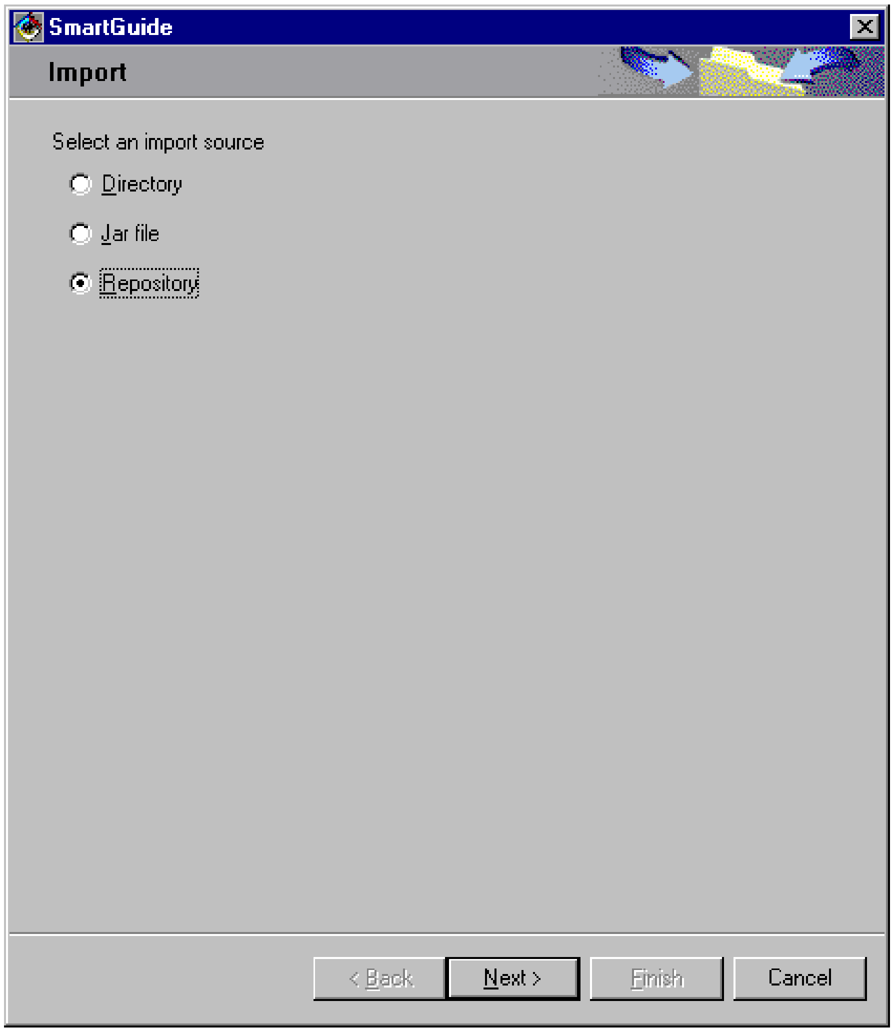

Workbench [Administrator]
File Edit Workspace Selected Window Help
Projects Packages Classes Interfaces Managing All Problems
All Projects
IBM Java Implementation
Java class libraries
JFC class libraries
Netscape Security
Sun class libraries PM Win32
Go To
Add
Import...
Export...
Import program elements.

Figure 15. Starting the Import Process

The Import SmartGuide window appears (Figure 16 on page 54). Select the **Repository** option, and click the **Next** button.



Select an import source

( ) Directory

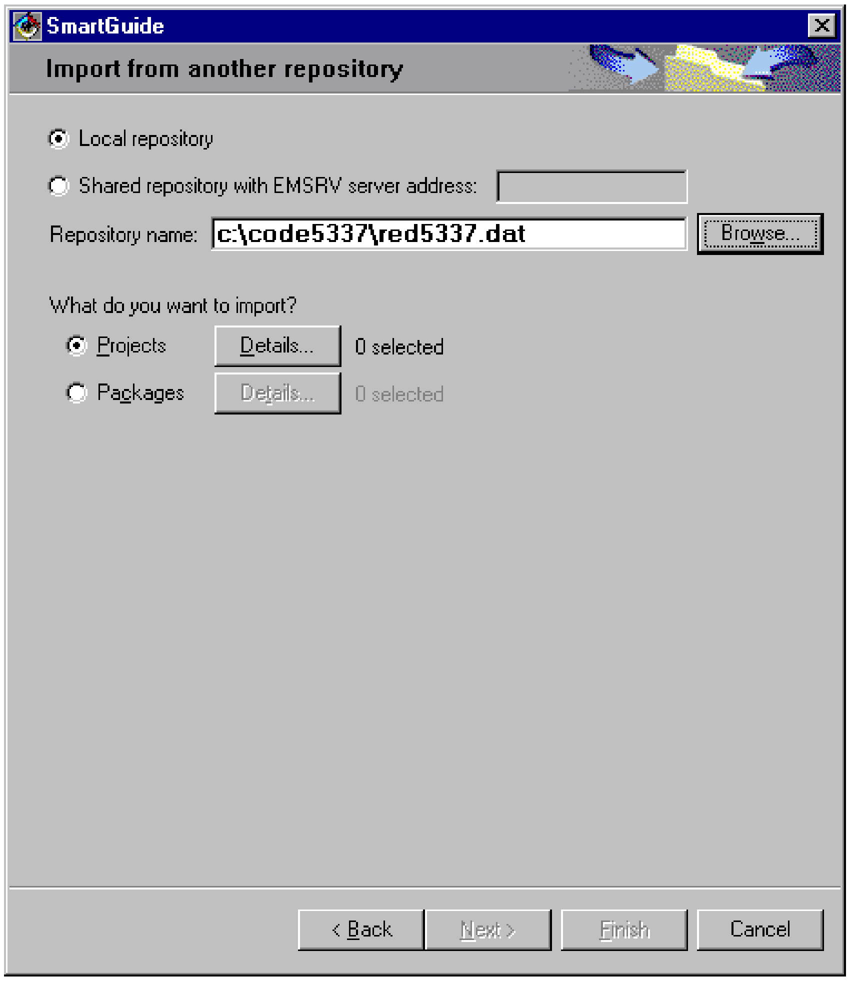
( ) Jar file

[✓ Repository]

Figure 16. Import SmartGuide

3.1.1.2 Working with the Import from a Directory Panel

As shown in Figure 17, the SmartGuide now shows the Import from another repository screen. Click on the **Browse** button and find the **red5337.dat** repository that you downloaded from the redbook Web site. In this example, it is found in the code5337 directory. Click on the **Details** button.


SmartGuide
Import from another repository
Local repository
Shared repository with EMSRV server address:
Browse...
What do you want to import?
Projects
Packages
Details...
Details...
< Back
Next >
Finish
Cancel

Repository name: c:\code5337\red5337.dat

Projects: 0 selected

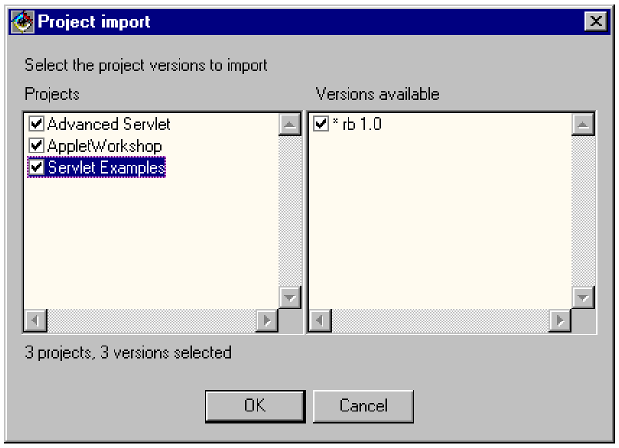
Packages: 0 selected

Figure 17. Import from another Repository SmartGuide

As shown in Figure 18 on page 56, the Project import dialog allows you to select which projects you want to import into your repository. The following projects are available:

- **Advanced Servlet**—Contains the examples discussed in Chapter 7, “Developing AS/400 Java Servlets” on page 221.
- **AppletWorkshop**—Contains the examples discussed in Chapter 6, “Developing AS/400 Java Applets” on page 193.
- **Servlet Examples**—Contains the applet examples discussed in this chapter and the servlet examples discussed in Chapter 4, “Introduction to AS/400 Servlets” on page 159.

  After you select the projects to import, click on **OK** and then click on the **Finish** button.



Project import

Select the project versions to import

Projects: ☑ Advanced Servlet, ☑ AppletWorkshop, ☑ Servlet Examples

Versions available: ☑ * rb 1.0

3 projects, 3 versions selected

Figure 18. Projects Import Dialog

After you click the **Finish** button, the selected projects are imported from the download repository into your repository. You can now add them into your workspace. To add a project, from the menu, follow this path: **Selected—>Add—>Project** (see Figure 19).

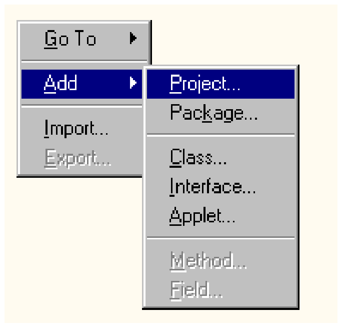
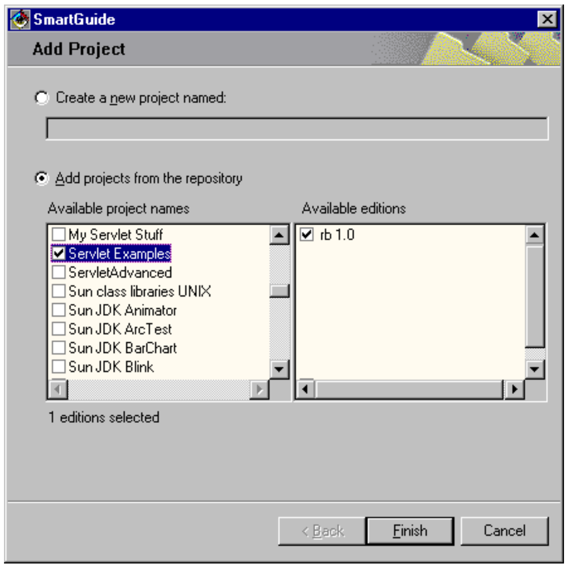


Figure 19. Add Project Menu

As shown in Figure 20 on page 57, click on the **Add Projects from the repository** radio button and then select the projects that you want to add to your workspace. In this case, we add the Servlet Examples Project. Click on the **Finish** button to add the project.


SmartGuide

Add Project

( ) Create a new project named:

(•) Add projects from the repository

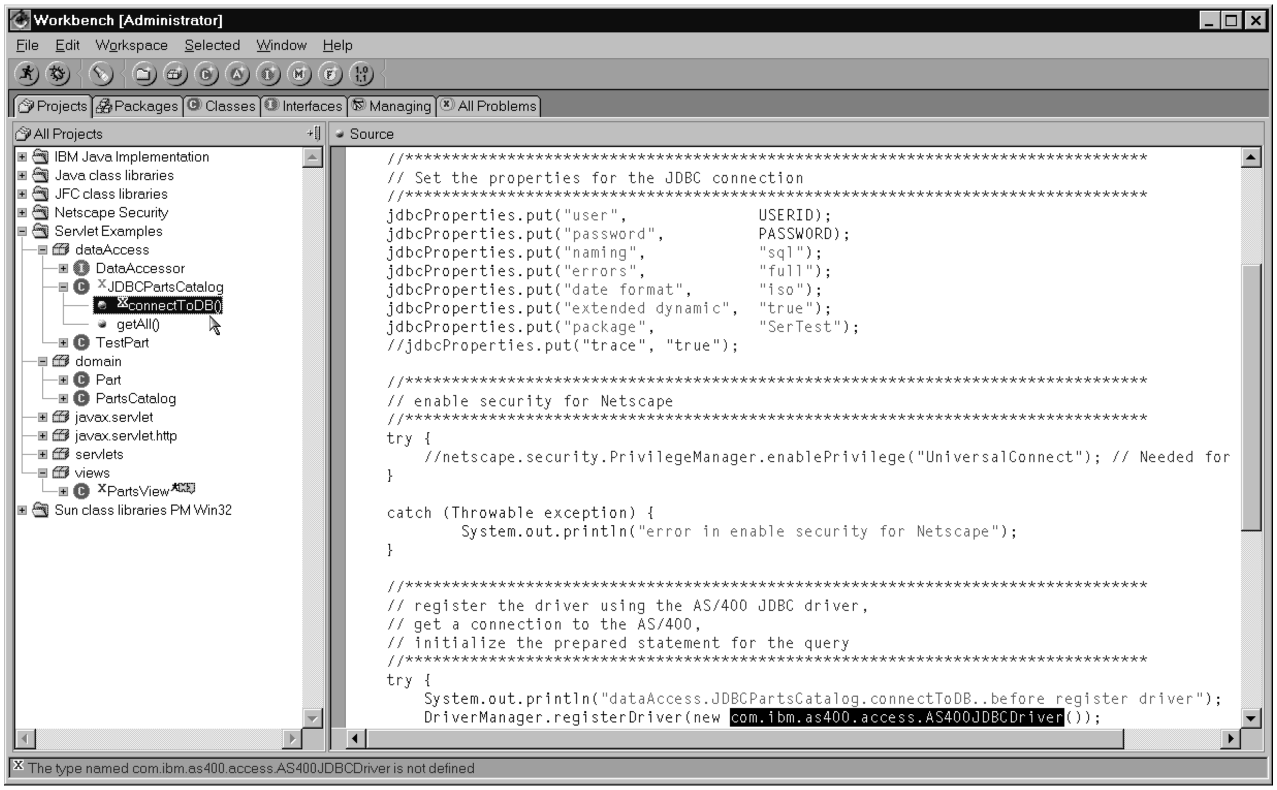
Available project names	Available editions
☐ My Servlet Stuff	☑ rb 1.0
☑ Servlet Examples	
☐ ServletAdvanced	
☐ Sun class libraries UNIX	
☐ Sun JDK Animator	
☐ Sun JDK ArcTest	
☐ Sun JDK BarChart	
☐ Sun JDK Blink	

1 editions selected

Figure 20. Add Projects SmartGuide

3.1.2 Resolving Problems in the Imported Applet

While the import process is running, you see a message indicating that a number of problems were found in the imported Java classes. When the import is complete, you see the project in the VisualAge for Java Workbench, with the packages expanded to indicate classes that have problems. The problems are indicated with an **X** to the left of the class or method name (Figure 21).


Workbench [Administrator]

File Edit Workspace Selected Window Help
Projects Packages Classes Interfaces Managing All Problems
All Projects
Source

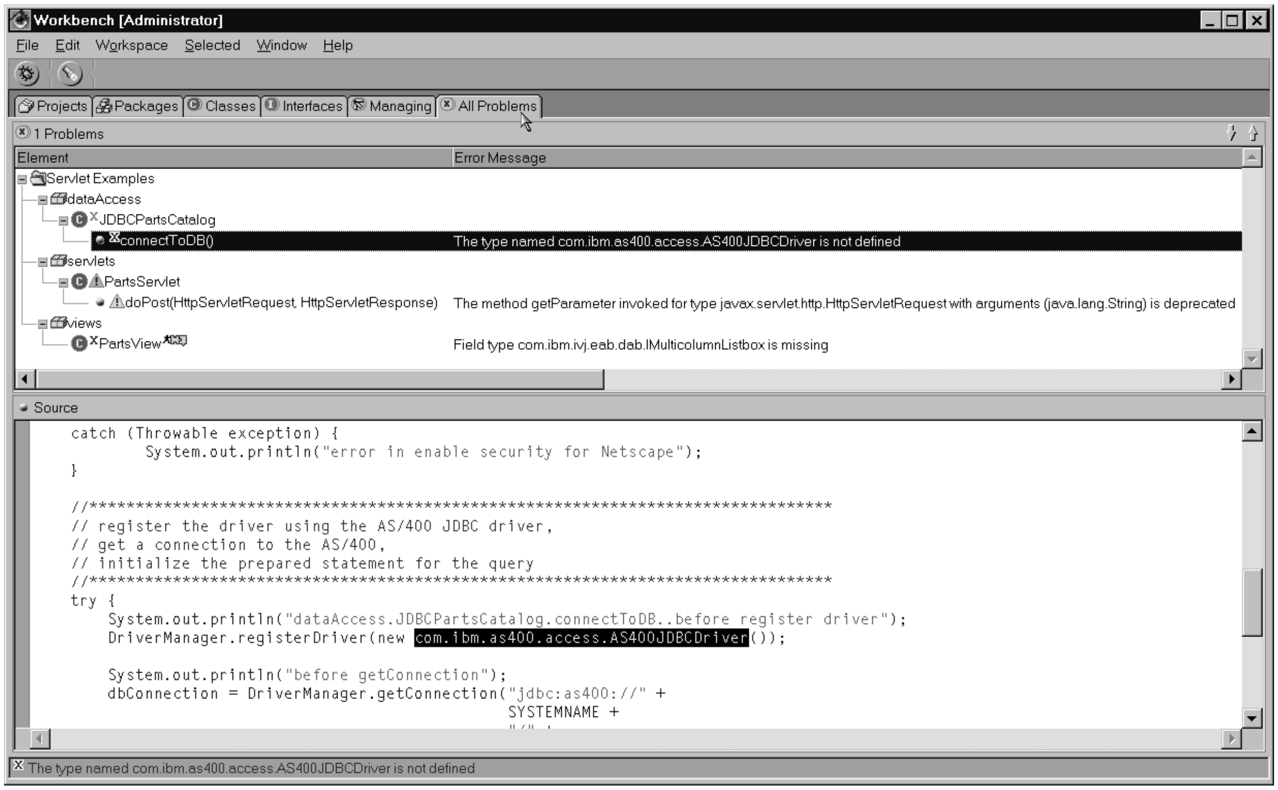
```
IBM Java Implementation
Java class libraries
JFC class libraries
Netscape Security
Servlet Examples
  dataAccess
    DataAccessor
    XJDBCPartsCatalog
      XconnectToDB()
      getAll()
    TestPart
  domain
    Part
    PartsCatalog
  javax.servlet
  javax.servlet.http
  servlets
  views
    XPartsView
Sun class libraries PM Win32
```

```
//****************************************************************************
// Set the properties for the JDBC connection
//****************************************************************************
jdbcProperties.put("user",               USERID);
jdbcProperties.put("password",           PASSWORD);
jdbcProperties.put("naming",             "sql");
jdbcProperties.put("errors",             "full");
jdbcProperties.put("date format",        "iso");
jdbcProperties.put("extended dynamic",   "true");
jdbcProperties.put("package",            "SerTest");
//jdbcProperties.put("trace", "true");

//****************************************************************************
// enable security for Netscape
//****************************************************************************
try {
    //netscape.security.PrivilegeManager.enablePrivilege("UniversalConnect"); // Needed for
}

catch (Throwable exception) {
        System.out.println("error in enable security for Netscape");
}

//****************************************************************************
// register the driver using the AS/400 JDBC driver,
// get a connection to the AS/400,
// initialize the prepared statement for the query
//****************************************************************************
try {
    System.out.println("dataAccess.JDBCPartsCatalog.connectToDB..before register driver");
    DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());
```

X The type named com.ibm.as400.access.AS400JDBCDriver is not defined

Figure 21. Problems Indicated in the Workbench

You can review all of the problems by clicking the **All Problems** tab. The Workbench view shown in Figure 22 appears.


Workbench [Administrator]
File Edit Workspace Selected Window Help
Projects Packages Classes Interfaces Managing All Problems
1 Problems
Source

Element	Error Message
Servlet Examples	
dataAccess	
JDBCPartsCatalog	
connectToDB()	The type named com.ibm.as400.access.AS400JDBCDriver is not defined
servlets	
PartsServlet	
doPost(HttpServletRequest, HttpServletResponse)	The method getParameter invoked for type javax.servlet.http.HttpServletRequest with arguments (java.lang.String) is deprecated
views	
PartsView	Field type com.ibm.ivj.eab.dab.IMulticolumnListbox is missing

```
catch (Throwable exception) {
        System.out.println("error in enable security for Netscape");
}

//*******************************************************************************
// register the driver using the AS/400 JDBC driver,
// get a connection to the AS/400,
// initialize the prepared statement for the query
//*******************************************************************************
try {
    System.out.println("dataAccess.JDBCPartsCatalog.connectToDB..before register driver");
    DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());

    System.out.println("before getConnection");
    dbConnection = DriverManager.getConnection("jdbc:as400://" +
                                               SYSTEMNAME +
```

The type named com.ibm.as400.access.AS400JDBCDriver is not defined

Figure 22. All Problems Tab

There are two types of problems in the imported Java source code:

- **Missing types**—This means that there are additional classes required in the project to provide the code for the missing types.
- **Deprecated methods**—A deprecated method is a method that has been superseded by a new method. This is a warning to help you identify methods that will be obsolete in the future. You can change the code to use the superseding method when it is convenient for you.

3.1.2.1 The Missing Types Problems

The missing types problems occur because the example Java source code uses two classes that are not available in the workspace:

- **Multi-column list box used in the applet** (see Figure 14 on page 52).

  This is in class `com.ibm.ivj.eab.dab`. This class is provided in the VisualAge for Java Enterprise Edition in the IBM Enterprise Data Access Libraries project.

- **JDBC class for the AS/400**

  This is in class `com.ibm.as400.access.AS400JDBCDriver`. This class is provided in the VisualAge for Java Enterprise Edition in the IBM Enterprise Toolkit for AS400. This class is also provided in the AS/400 Toolbox for Java that is installed on the AS/400 system as part of Licensed Program Product 5763-JC1.

3.1.2.2 Importing the Projects to Resolve the Missing Types

You need to import the two IBM provided projects into the VisualAge for Java workspace. You also need to import the Netscape Security project, since you will use that in the applet.

To start the import process, right-click in the **All Projects** panel in the Projects tab (Figure 23). Select the **Add** item on the pop-up menu, and select the **Project** item on the next pop-up menu, as shown in the following figure.

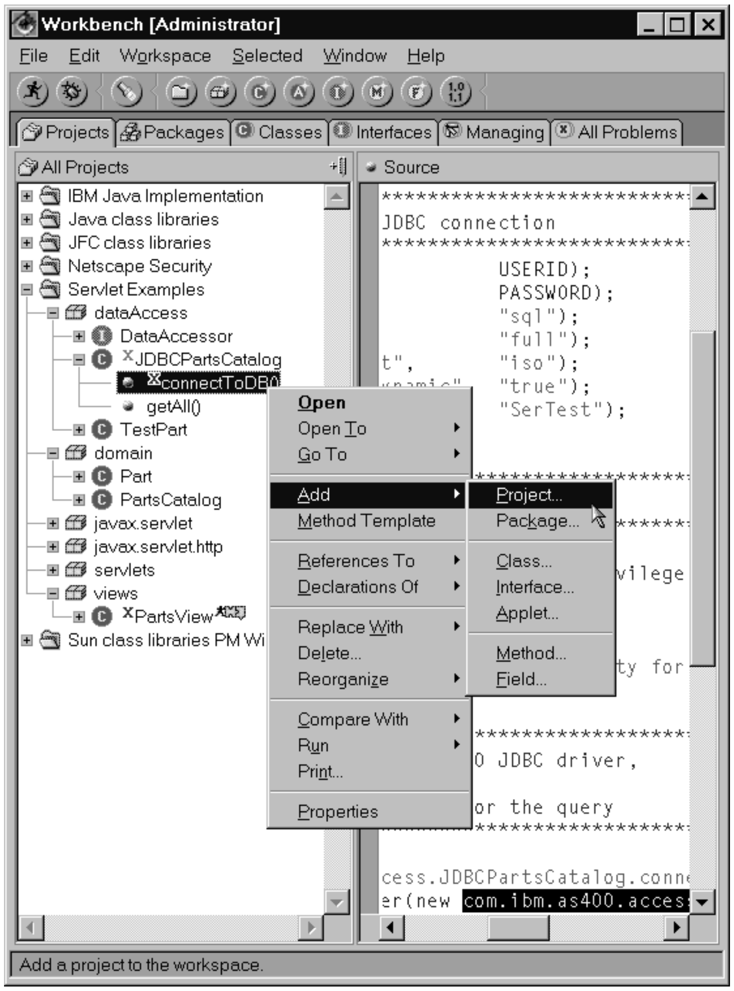
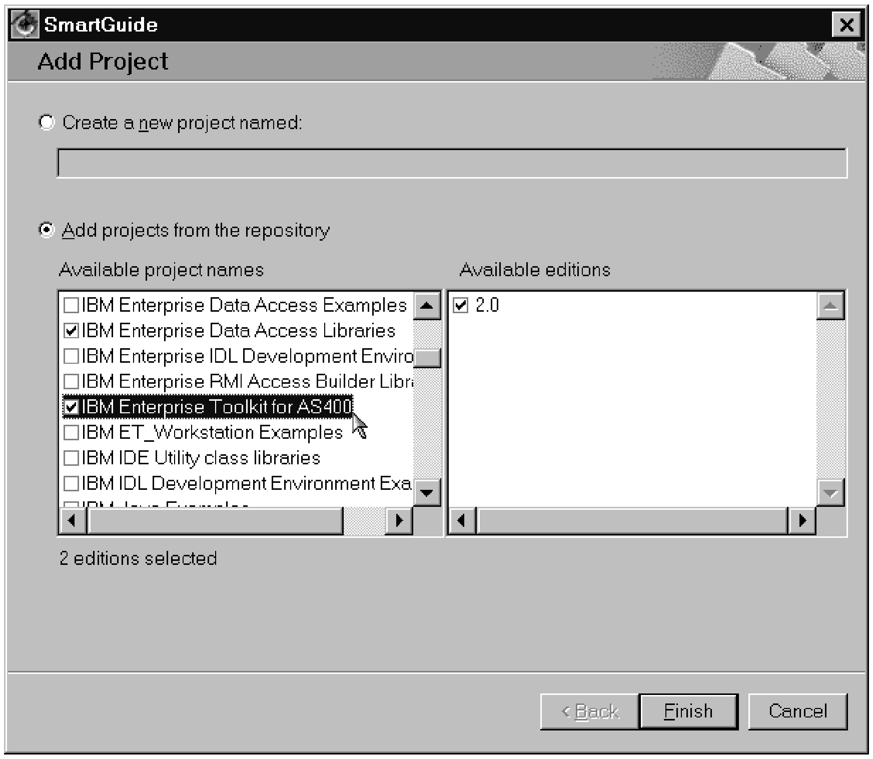


Figure 23. Add Project Menu

The Add Project SmartGuide appears (Figure 24 on page 60). Follow these steps to select the required projects for the applet:

1. Click the **Add projects from the repository** selection.
2. Scroll through the list of available project names. Check the following projects:
   – IBM Enterprise Data Access Libraries
   – IBM Enterprise Toolkit for AS400
   – Netscape Security
3. If there are multiple available editions for either project, select the highest numbered edition as the edition to add.
4. Click the **Finish** button to add the projects to the workspace.



SmartGuide

Add Project

( ) Create a new project named:

(●) Add projects from the repository

Available project names

- [ ] IBM Enterprise Data Access Examples
- [✓] IBM Enterprise Data Access Libraries
- [ ] IBM Enterprise IDL Development Enviro
- [ ] IBM Enterprise RMI Access Builder Libr
- [✓] IBM Enterprise Toolkit for AS400
- [ ] IBM ET_Workstation Examples
- [ ] IBM IDE Utility class libraries
- [ ] IBM IDL Development Environment Exa

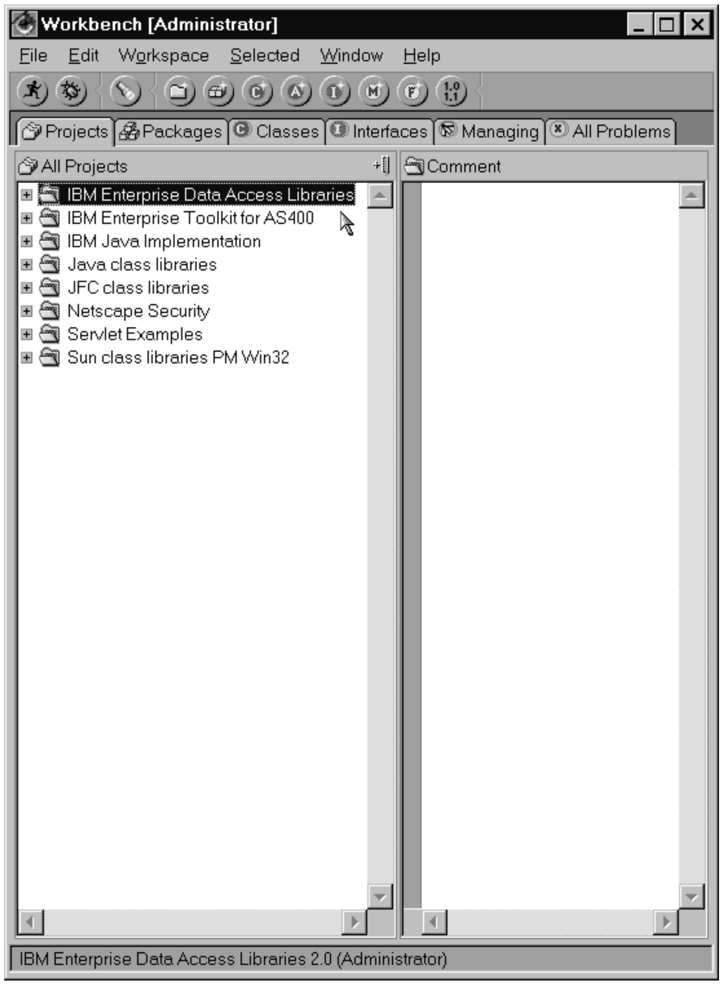
Available editions

- [✓] 2.0

2 editions selected

Figure 24. Add Project SmartGuide

As the projects are added into the workspace, the missing types problems are resolved. The All Projects pane should appear as shown in Figure 25 on page 61 when the project addition is complete. If you do not see the two projects in the list, or if you still have the missing types problems, you need to run the Add Project SmartGuide again to select the correct projects to add.



IBM Enterprise Data Access Libraries
IBM Enterprise Toolkit for AS400
IBM Java Implementation
Java class libraries
JFC class libraries
Netscape Security
Servlet Examples
Sun class libraries PM Win32

IBM Enterprise Data Access Libraries 2.0 (Administrator)

Figure 25. Workbench with Added Projects

3.1.3 Overview of Classes Used in the PartsView Applet

The PartsView applet was created in VisualAge for Java. Because the applet will be converted to run as a servlet on the AS/400 system, a project named ServletExamples was started to contain the packages and classes used in the applet. Figure 26 on page 62 shows a view of the ServletExamples project with the packages used for the applet expanded.

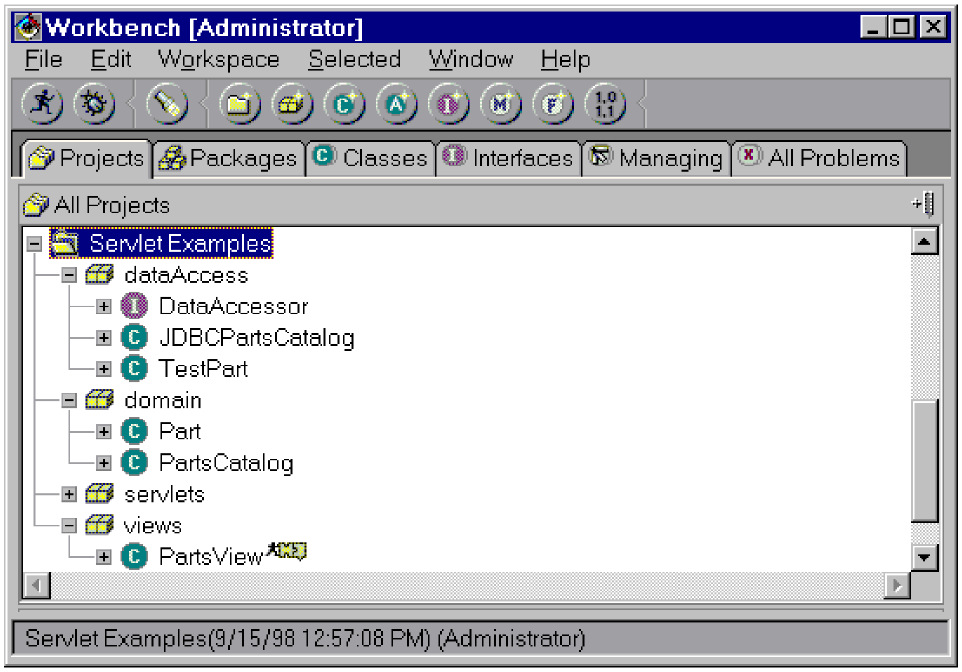

Workbench [Administrator]
File Edit Workspace Selected Window Help
Projects Packages Classes Interfaces Managing All Problems
All Projects
Servlet Examples
dataAccess
DataAccessor
JDBCPartsCatalog
TestPart
domain
Part
PartsCatalog
servlets
views
PartsView
Servlet Examples(9/15/98 12:57:08 PM) (Administrator)

Figure 26. ServletExamples Project

Table 5 briefly describes the packages, classes, and interfaces used in the applet.

Table 5. ServletExamples Packages, Classes, and Interfaces

ServletExamples Project		
Package	**Class/Interface**	**Description**
dataAccess	DataAccessor	Interface used to define methods that must be implemented by classes in this package.
	JDBCPartsCatalog	Used to connect to the AS/400 system database, returns a vector of parts.
	TestPart	Used to simulate a connection to a database, returns a vector of parts.
domain	Part	Represents a row of part data retrieved from the database.
	PartsCatalog	Determines which data source to get part data from, gets parts from the selected data source, and returns a vector of parts data.
views	PartsView	The visible part of the applet, includes the parts listbox and the button to start the query.

3.1.4 Working with the Applet in the Visual Composition Editor

The classes in the `dataAccess` and `domain` packages do not contain any user interface code. All of the user interface code for the applet is contained in the `views` package. The PartsView applet was designed in the Visual Composition editor in VisualAge for Java. Figure 27 on page 63 shows the applet with the multi-column listbox and push button in place. In this section of the chapter, we show how a PartsCatalog bean, which provides access to data, is added to the PartsView class.

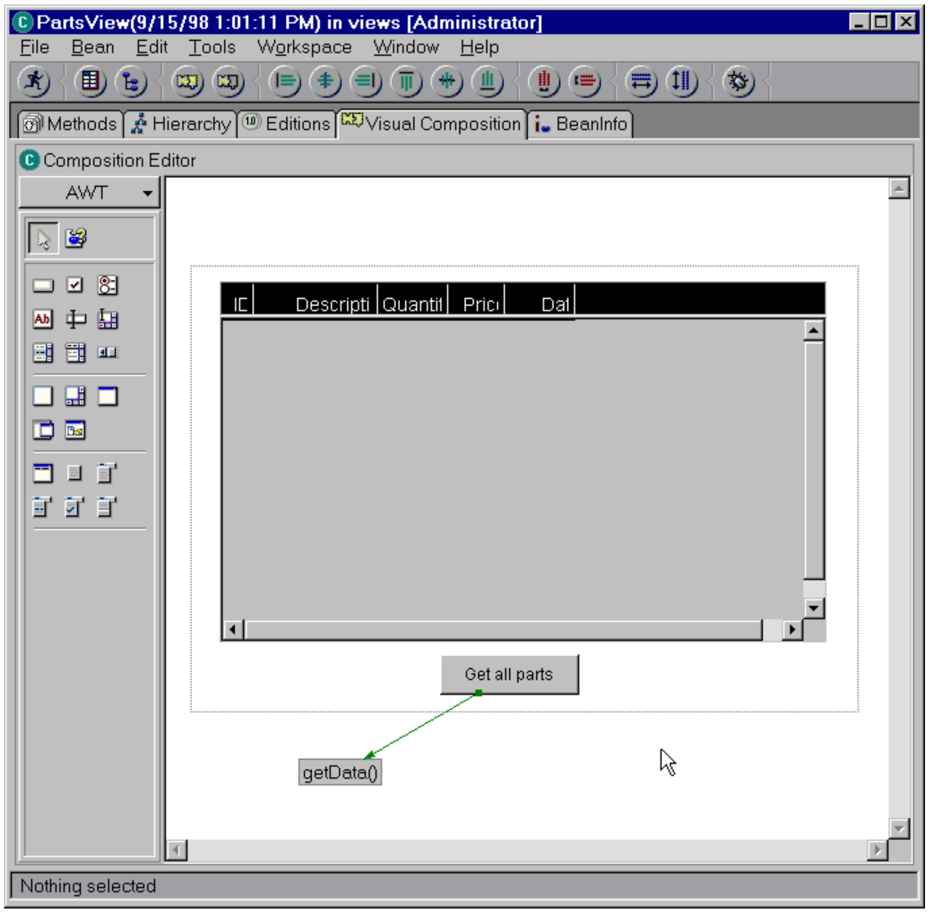

PartsView(9/15/98 1:01:11 PM) in views [Administrator]
File Bean Edit Tools Workspace Window Help
Methods Hierarchy Editions Visual Composition BeanInfo
Composition Editor
AWT
ID Description Quantity Price Date
Get all parts
getData()
Nothing selected

Figure 27. PartsView Applet in the Visual Composition Editor

3.1.4.1 Adding the PartsCatalog Bean to the Applet

The `PartsCatalog` bean contains the code that is responsible for obtaining records from the data source and making them available to the `PartsView` class. To complete the PartsView applet, the `PartsCatalog` bean(or class) must be added to the applet.

To add a bean to the applet, click the **Choose Bean** icon located in the palette, as shown in Figure 28 on page 64.

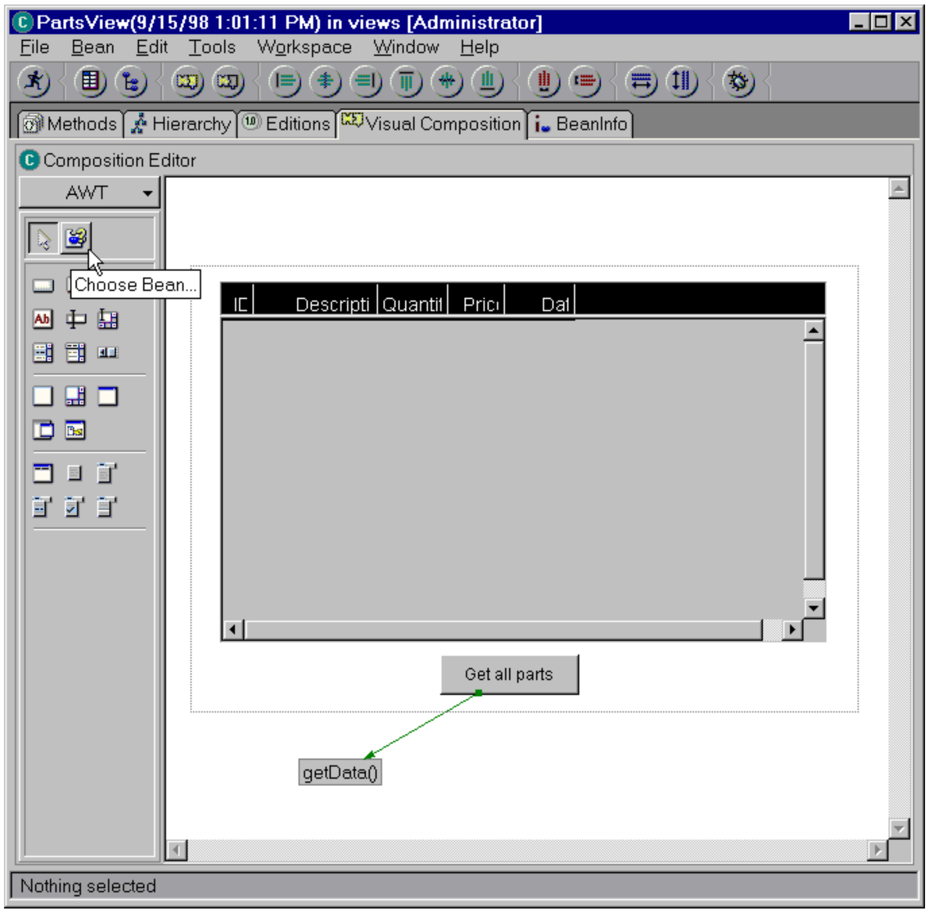
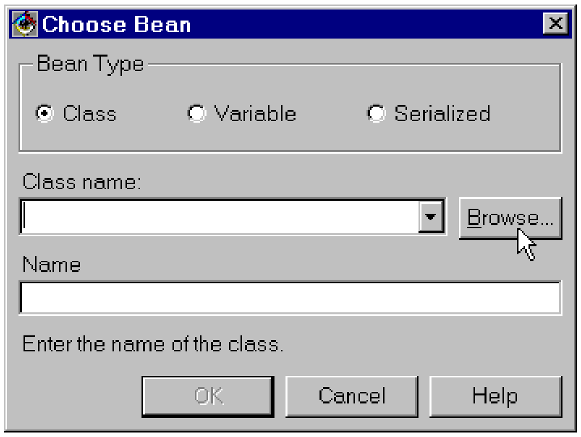

PartsView(9/15/98 1:01:11 PM) in views [Administrator]
File Bean Edit Tools Workspace Window Help
Methods Hierarchy Editions Visual Composition BeanInfo
Composition Editor
AWT
Choose Bean...
ID Descripti Quantit Pric Dat
Get all parts
getData()
Nothing selected

Figure 28. Clicking the Choose Bean Icon

VisualAge for Java now displays the Choose Bean dialog (Figure 29). Enter the package and class name, or click the **Browse** button to select the class.



Bean Type: [✓ Class] Variable Serialized

Enter the name of the class.

Figure 29. Choose Bean Dialog

When you click **Browse**, the Choose a valid class dialog is displayed (Figure 30 on page 65). The Class Names list in the middle of the dialog initially contains all classes in the VisualAge for Java workspace.

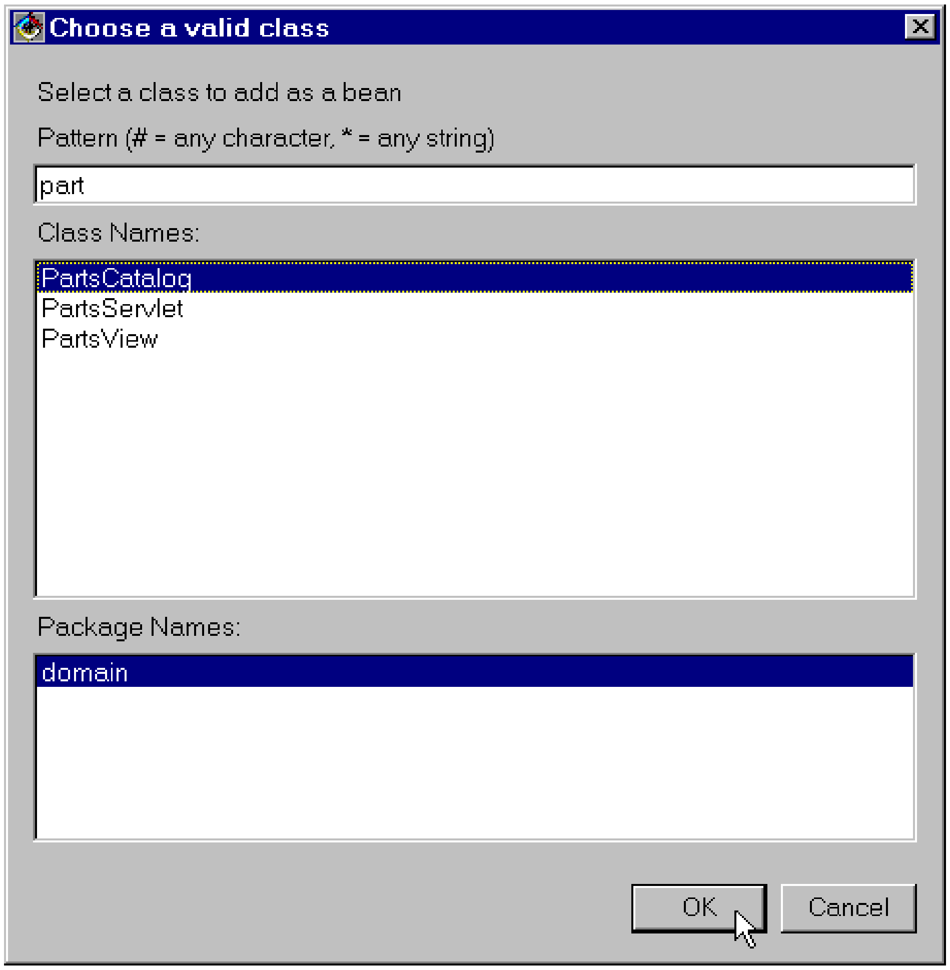
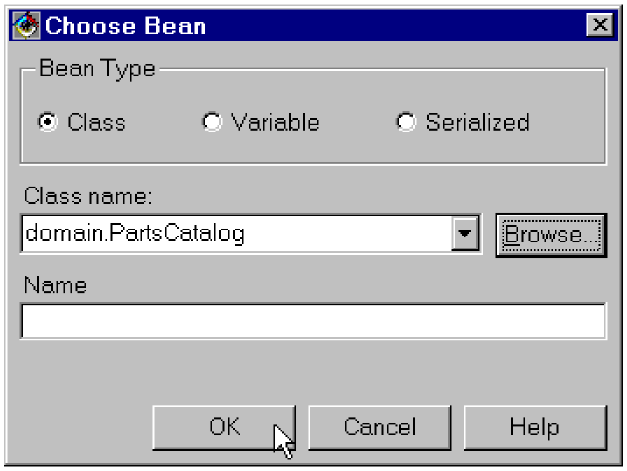


Figure 30. Choose Class Dialog

As you start typing the pattern, the list of classes is reduced so that it includes only those classes that start with the pattern. When you locate the class you need, click the class name to display the package in which the class is contained.

When you click **OK** in the Choose a valid class dialog, the selected class name returns to the Choose Bean dialog (Figure 31 on page 66).



Choose Bean

Bean Type: [✓ Class] Variable Serialized

Class name: domain.PartsCatalog

Name:

Figure 31. Choose Bean Dialog with PartsCatalog Selected

Click **OK** on the Choose Bean dialog to add the `PartsCatalog` bean to the workspace in the Visual Composition editor (Figure 32). Now that the `PartsCatalog` bean is part of the applet, you can use its methods to work with the data sources.

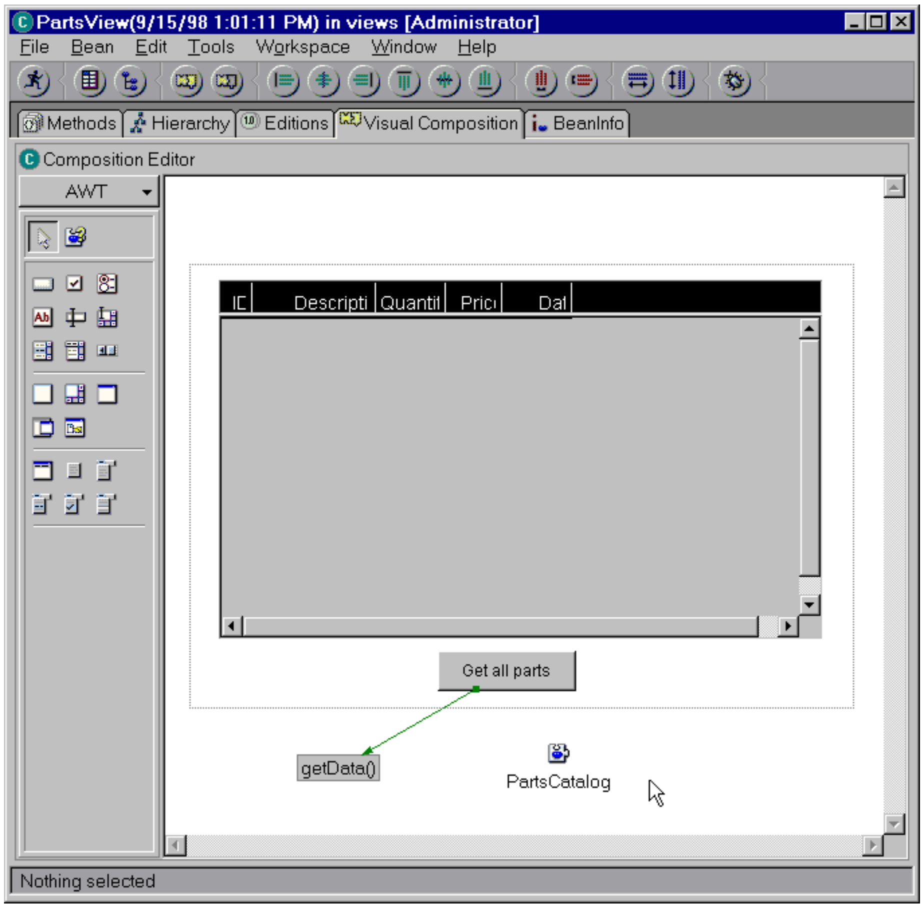


Figure 32. PartsCatalog Added to the PartsView Applet

3.1.4.2 Setting an Option for Export to Source Files

Although you can develop the entire applet within the VisualAge for Java IDE, there may be occasions where you need to export the Java source code from the applet so that you can work with the Java source code in another environment. For example, you may want to work with the Java source files in another editor or in a source-code version control system. Also, you may want to work with the Java source files in the VisualAge for Java IDE on another computer.

If you do plan to export and import Java source code files that were generated in the VisualAge for Java Visual Composition Editor (VCE), you need to set an option to indicate that an additional method is to be added to the bean (the bean in this case is the `PartsView` class). The `getBuilderData` method contains hexadecimal data that is used by the VCE to construct the view of the bean when the Java source code files are imported to the VisualAge for Java IDE and opened in the VCE.

To set the option, select **Windows—>Options** (Figure 33). The Window menu is available on most dialogs in VisualAge for Java, including the VCE shown in the following figure and the Workbench.

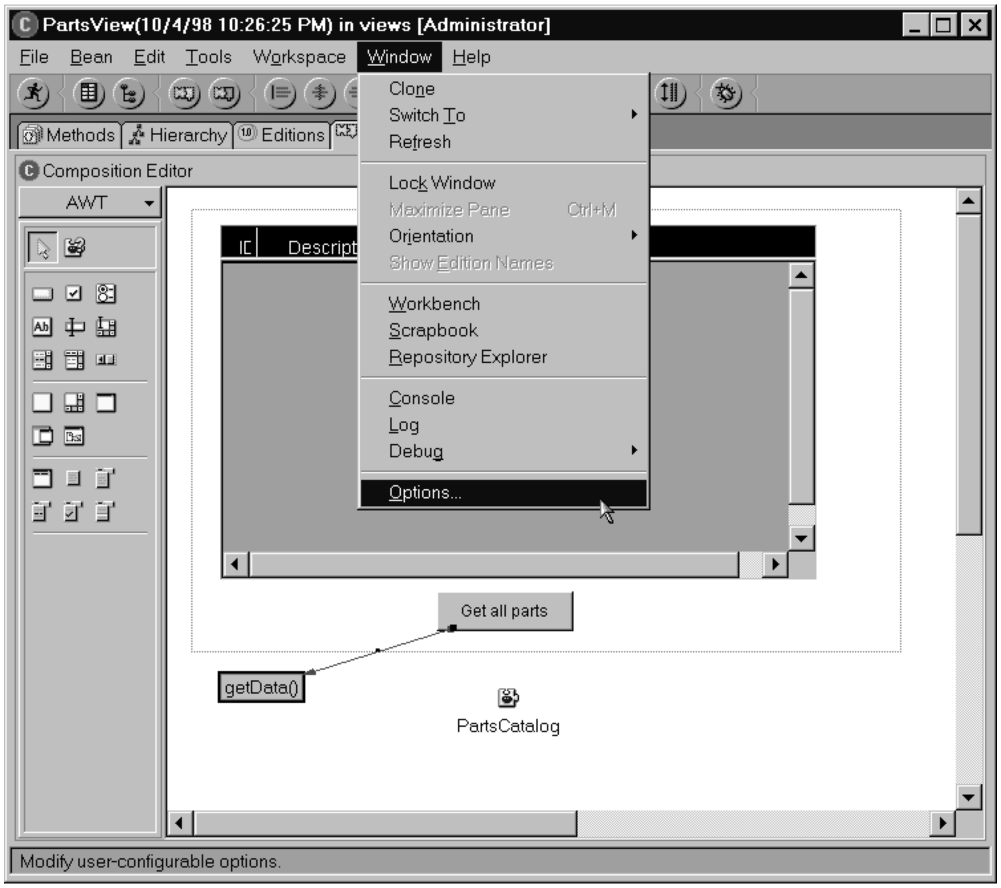
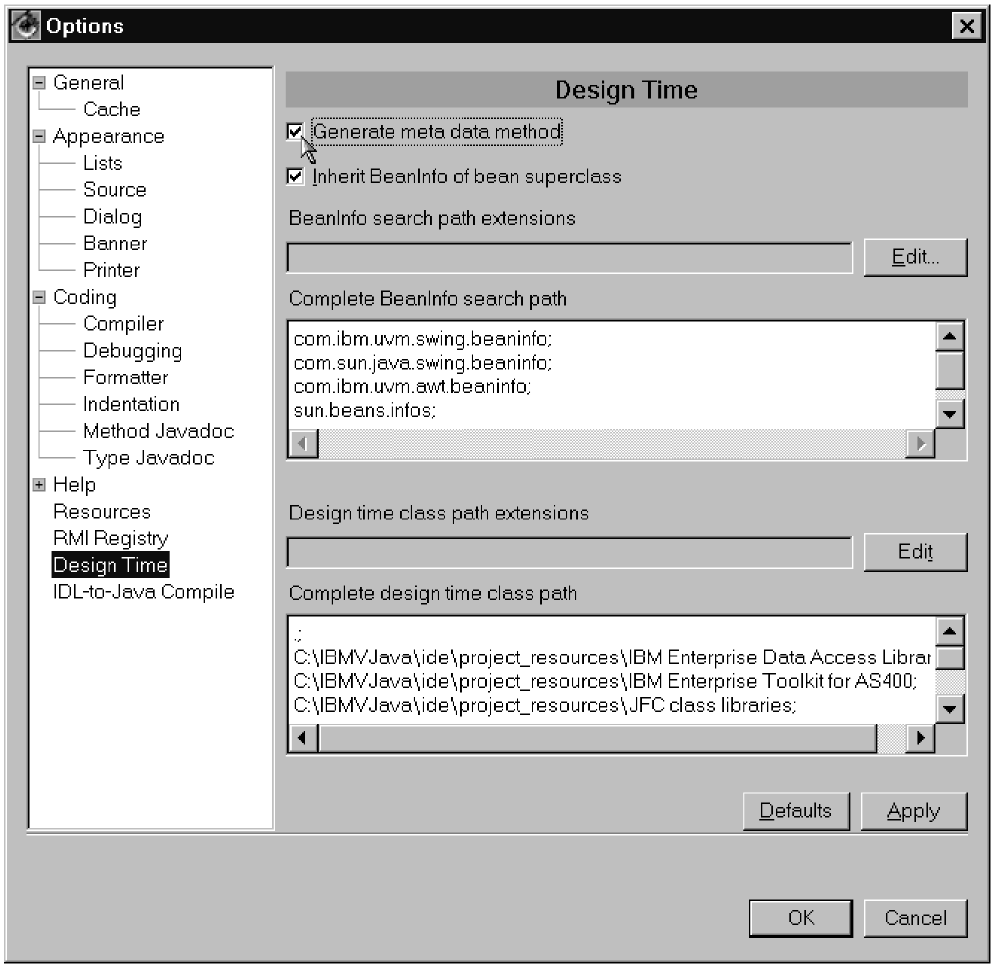


Figure 33. VisualAge for Java IDE Options Menu



[✓] Generate meta data method

[✓] Inherit BeanInfo of bean superclass

BeanInfo search path extensions

Complete BeanInfo search path

```
com.ibm.uvm.swing.beaninfo;
com.sun.java.swing.beaninfo;
com.ibm.uvm.awt.beaninfo;
sun.beans.infos;
```

Design time class path extensions

Complete design time class path

```
.;
C:\IBMVJava\ide\project_resources\IBM Enterprise Data Access Librar
C:\IBMVJava\ide\project_resources\IBM Enterprise Toolkit for AS400;
C:\IBMVJava\ide\project_resources\JFC class libraries;
```

Figure 34. Design Time Options Dialog

On the Options dialog, click the **Design Time** item in the left column. Click the **Generate meta data method** option at the top of the Design Time panel (Figure 34). By enabling that option, you instruct VisualAge for Java to automatically generate a `getBuilderData` method for each bean that you create in the VCE.

Tip

You may want to leave the **Generate meta data method** checked for all beans that you create in the VisualAge for Java Visual Composition Editor. This creates an extra method in each of the beans that you create. However, it provides the only available technique to export/import Java source files from VisualAge for Java so you can view the bean again in the Visual Composition Editor.

Since you may not be able to predict in advance if you need to export or import the Java source files, you may find it easier to check this option and leave it checked.

3.1.4.3 Saving the Bean

Before running the bean, save it. Saving the bean incorporates all of the settings and options from the Visual Composition Editor. To save the bean, click **Bean—>Save Bean** (Figure 35).

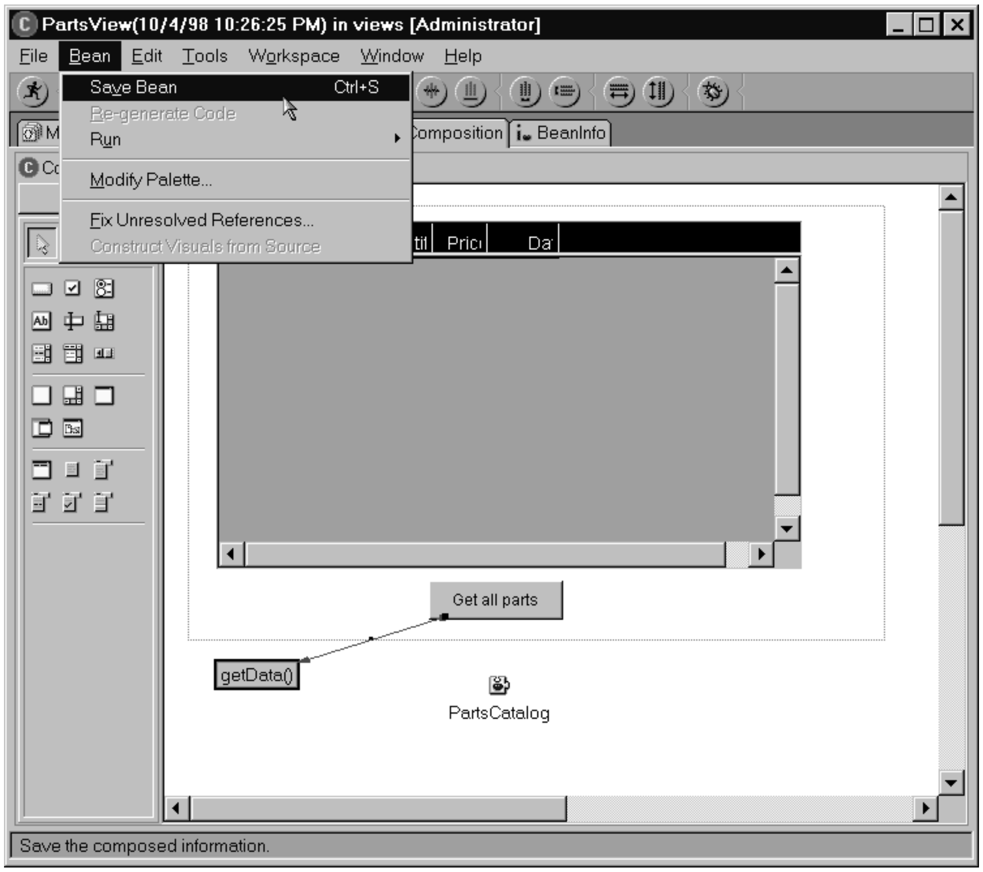

PartsView(10/4/98 10:26:25 PM) in views [Administrator]
File Bean Edit Tools Workspace Window Help
Save Bean Ctrl+S
Re-generate Code
Run
Modify Palette...
Fix Unresolved References...
Construct Visuals from Source
Composition
BeanInfo
Get all parts
getData()
PartsCatalog
Save the composed information.

Figure 35. Saving a Bean

3.1.5 Testing the Applet in the VisualAge for Java Applet Viewer

VisualAge for Java includes a built-in Applet Viewer that you can use to view and test the applet before launching it from within a browser. As you develop an applet, you may want to test it frequently so that you can see if it is working correctly.

Note: Although it is convenient to test the applet from within the VisualAge IDE, there is **no guarantee** that **the applet will run in a browser** because it runs successfully in the Applet Viewer. Even so, the Applet Viewer is useful for initial testing and debugging.

Note

The applet can display data retrieved from the AS/400 system or it can display data retrieved from local test data. The version that is available from the redbooks Web site is set to access an AS/400 system.

Before you can test the applet and access your AS/400 system, you must update the JDBCPartsCatalog class variables to include a valid system name, user ID and password. See Section 3.2.5.2, “JDBCPartsCatalog Class Code” on page 91 for details about the JDBCPartsCatalog class.

If you want to test using the local test data, you need to change the defaultDataAccessor method so that the qualifiedClassName variable is set to "**dataAccess.TestPart**." See Section 3.2.8.4, “The defaultDataAccessor Method in the PartsCatalog Class” on page 101, for details.

3.1.5.1 Setting Applet Attributes and Class Path

Before testing the Applet, you need to apply these additional settings:

- **Attributes**—To specify the height and width of the Applet (in pixels) when it is displayed.
- **Class Path**—To indicate which additional projects are used in the Applet.

You set both of those values in the Properties for PartsView dialog. To get to that dialog, click the **Bean—>Run—>Check Class Path** in the Visual Composition Editor (Figure 36).

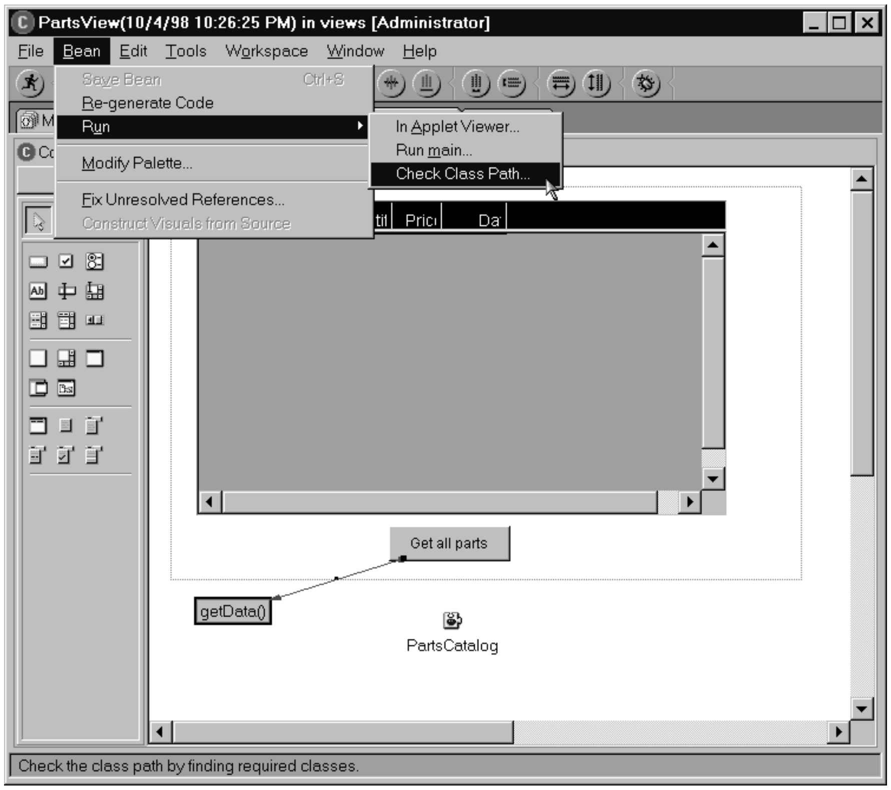
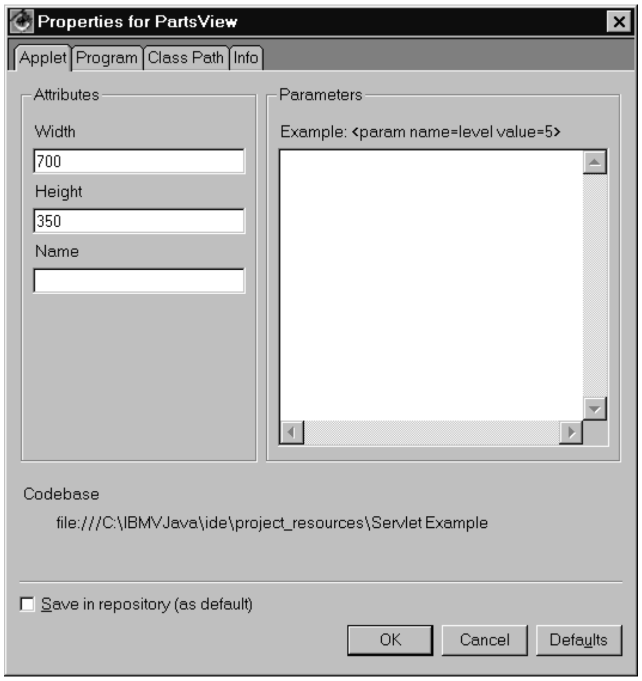


Figure 36. Checking the Class Path

Setting Applet Attributes

Click the **Applet** tab on the Properties for PartsView dialog (Figure 37). Change the value for **Width** to **700** and **Height** to **350**.



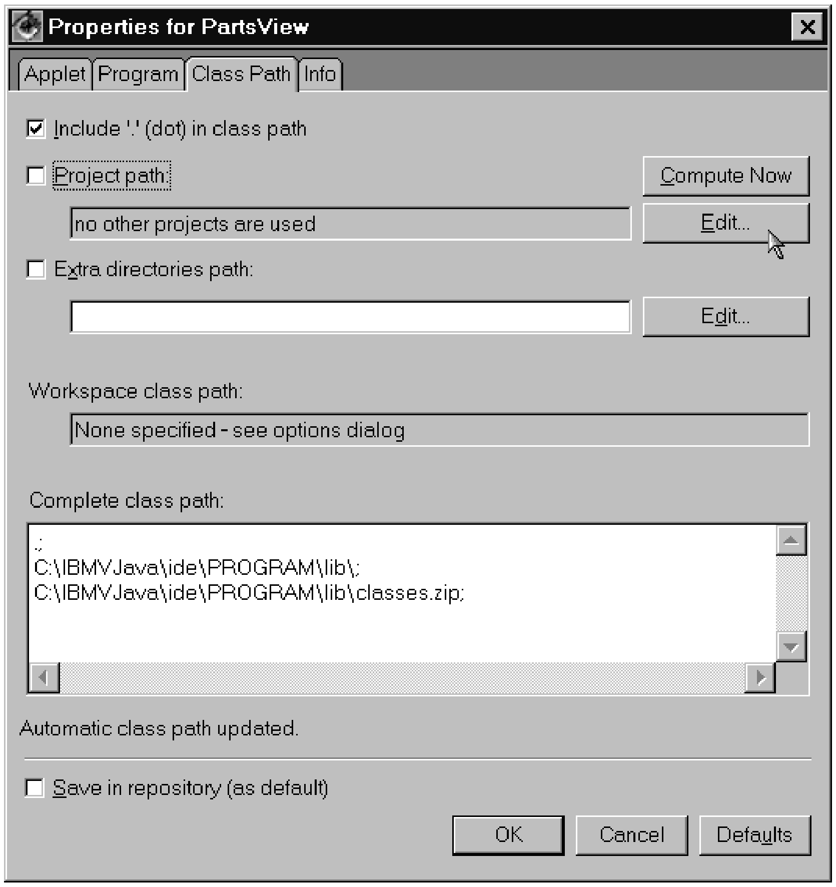
Width: 700
Height: 350
Name:
Example: <param name=level value=5>
Codebase
file:///C:\IBMVJava\ide\project_resources\Servlet Example
☐ Save in repository (as default)

Figure 37. Setting the Width and Height Attributes

Setting the Class Path

Click the Class Path tab. The tab initially appears as shown in Figure 38 on page 72. You use this part of the Properties dialog to indicate the following information:

- If the current directory is to be included in the Class Path
- The names of any other projects that are in the VisualAge for Java workspace and are to be included in the Class Path
- Any additional directories where **.jar** or **.zip** files are located that contain classes used in the project



[✓] Include '.' (dot) in class path

[ ] Project path:

no other projects are used

[ ] Extra directories path:

Workspace class path:

None specified - see options dialog

Complete class path:

```
.;
C:\IBMVJava\ide\PROGRAM\lib\;
C:\IBMVJava\ide\PROGRAM\lib\classes.zip;
```

Automatic class path updated.

[ ] Save in repository (as default)

Figure 38. Setting the Class Path

The VisualAge for Java version 2.0 Help system states the following about using class paths in the IDE:

> "In VisualAge for Java, each runnable class is responsible for its own class path. The class path is necessary when running a class in order for the class loader to properly find the classes that your class references and the classes that they in turn reference."

Note

A *runnable class* is a class that can be launched in the VisualAge for Java IDE using the Run menu item. Runnable classes are indicated in the Workbench with a small "running-man" icon next to the class name. You can see an example of this icon for the PartsView class in Figure 26 on page 62.

Click the **Edit** button next to the *Project path* field to go to the Class Path dialog (Figure 39 on page 73). On that dialog, select the following projects to include in the Applet's class path:

- IBM Enterprise Data Access Libraries
- IBM Enterprise Toolkit for AS400
- Netscape Security

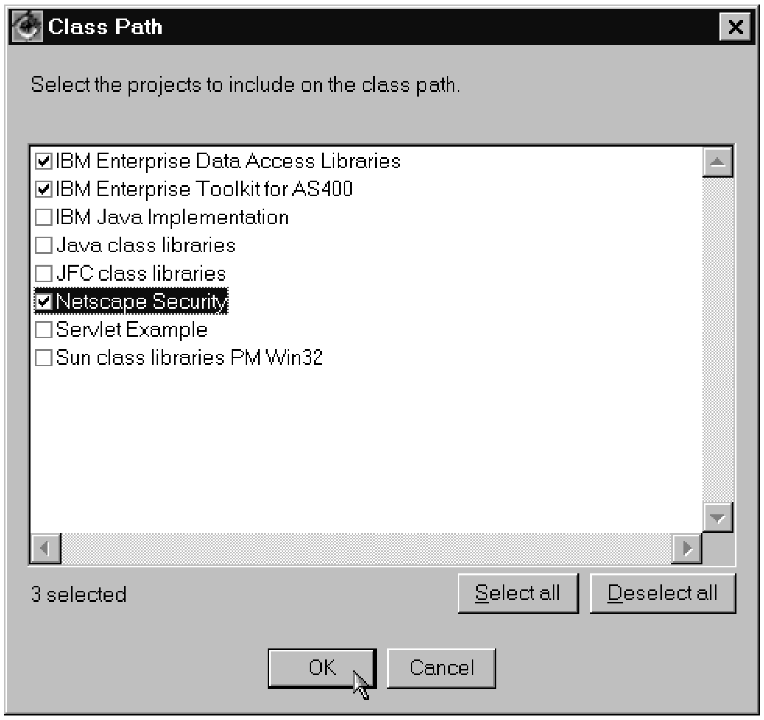
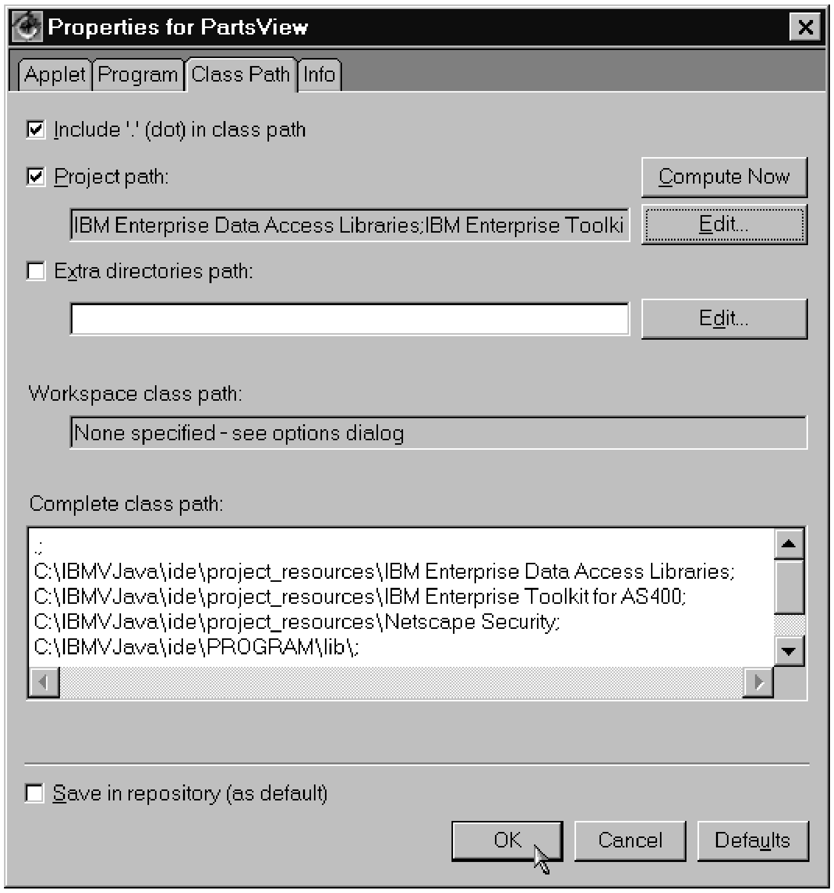

Class Path
Select the projects to include on the class path.
☑ IBM Enterprise Data Access Libraries
☑ IBM Enterprise Toolkit for AS400
☐ IBM Java Implementation
☐ Java class libraries
☐ JFC class libraries
☑ Netscape Security
☐ Servlet Example
☐ Sun class libraries PM Win32
3 selected
Select all
Deselect all
OK
Cancel

Figure 39. Select the Projects to be Included in the Class Path

Click the **OK** button to return to the Class Path tab. You now see the additional projects that you selected in both the **Project Path** and the **Complete class path** (Figure 40 on page 74).

Click **OK** on the Class Path tab to close the Properties for PartsView dialog.



☑ Include '.' (dot) in class path

☑ Project path: IBM Enterprise Data Access Libraries;IBM Enterprise Toolki

☐ Extra directories path:

Workspace class path: None specified - see options dialog

Complete class path:

```
.;
C:\IBMVJava\ide\project_resources\IBM Enterprise Data Access Libraries;
C:\IBMVJava\ide\project_resources\IBM Enterprise Toolkit for AS400;
C:\IBMVJava\ide\project_resources\Netscape Security;
C:\IBMVJava\ide\PROGRAM\lib\;
```

☐ Save in repository (as default)

Figure 40. Project Path and Complete Class Path

Attention

Although the Class Path tab includes a **Compute Now** button (see Figure 40), it did not correctly determine all of the additional projects that need to be included in the PartsView applet's class path. The **Compute Now** function only returned the IBM Enterprise Data Access Libraries project to the list.

If you are using classes from multiple additional projects, click the **Edit** button and manually select the additional projects.

3.1.5.2 Launching the Applet Viewer

To launch the Applet Viewer, click **Bean—>Run—>In Applet Viewer** (Figure 41 on page 75). That menu item starts the Applet Viewer, at which point you can see the applet and start working with it (Figure 42 on page 75).

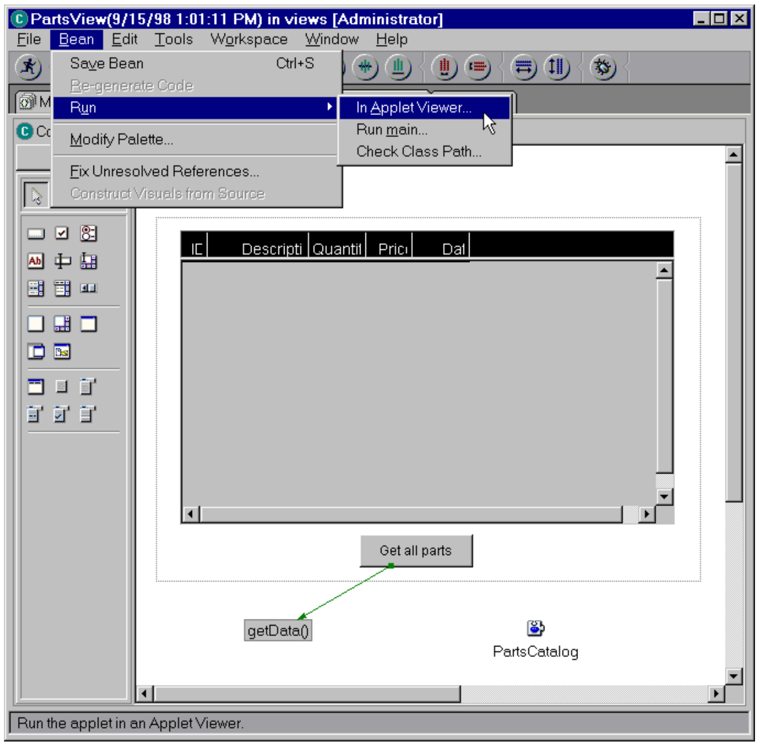

PartsView(9/15/98 1:01:11 PM) in views [Administrator]
File Bean Edit Tools Workspace Window Help
Save Bean Ctrl+S
Re-generate Code
Run
Modify Palette...
Fix Unresolved References...
Construct Visuals from Source
In Applet Viewer...
Run main...
Check Class Path...
ID Description Quantity Price Date
Get all parts
getData()
PartsCatalog
Run the applet in an Applet Viewer.

Figure 41. Running the PartsView Applet

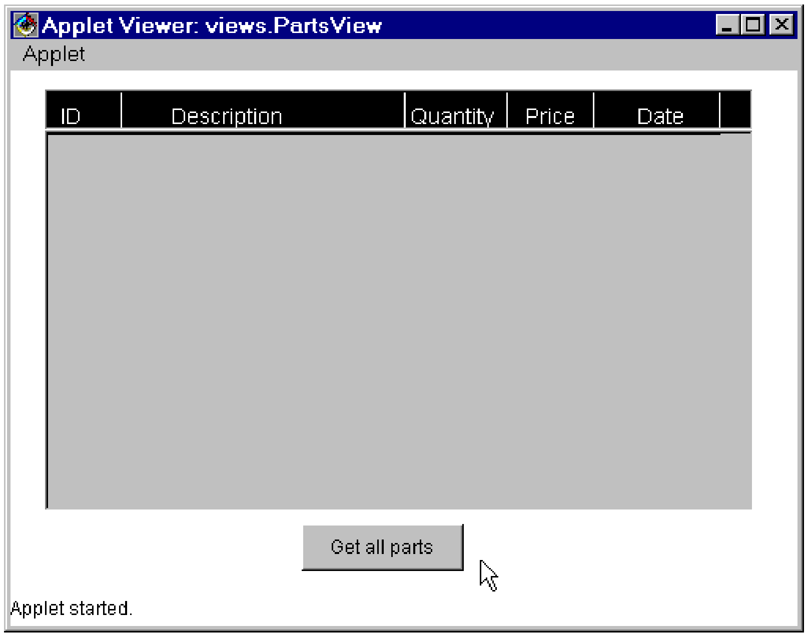

Applet Viewer: views.PartsView
Applet
ID Description Quantity Price Date
Get all parts
Applet started.

Figure 42. PartsView Applet in the Applet Viewer

3.1.5.3 Setting AppletViewer properties

Before you can get data from the AS/400 system, you may need to change the AppletViewer properties to allow the applet to access resources in the network. In the AppletViewer, click **Applet—>Properties** (Figure 43 on page 76).

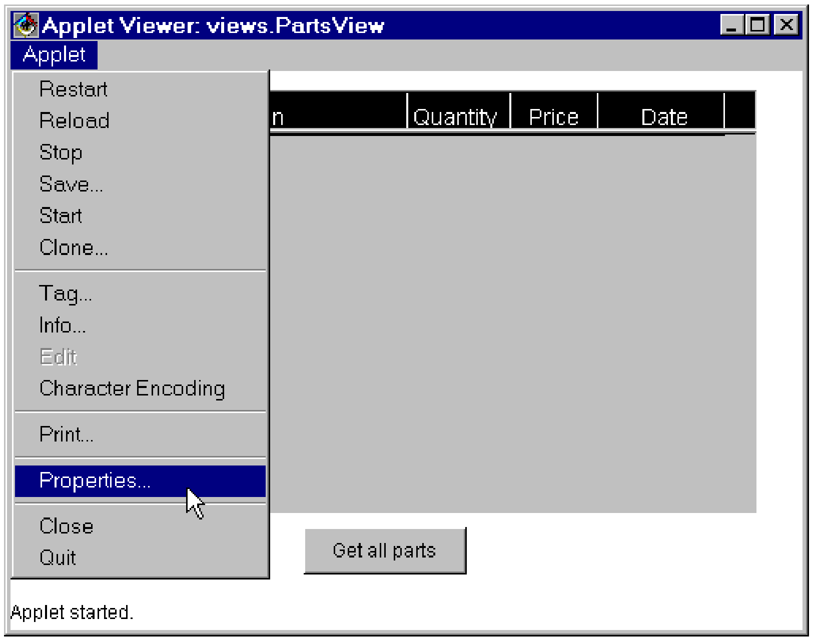
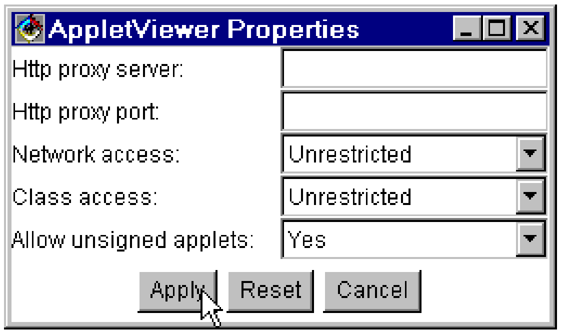


Figure 43. Applet Viewer Properties Dialog

The AppletViewer Properties dialog shown in Figure 44 appears. Set the AppletViewer properties as follows:

- **Network access:** Unrestricted
- **Class access:** Unrestricted
- **Allow unsigned applets:** Yes

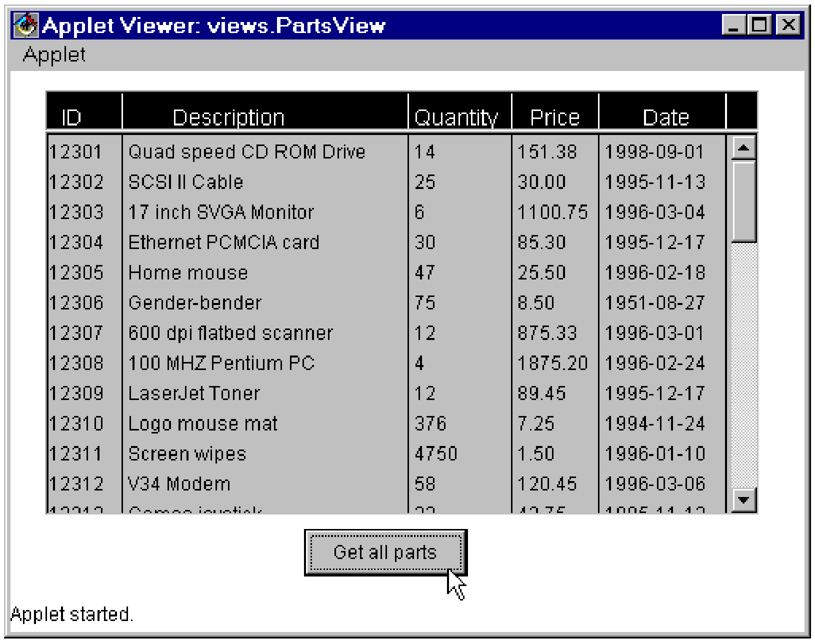

AppletViewer Properties

Http proxy server:	
Http proxy port:	
Network access:	Unrestricted
Class access:	Unrestricted
Allow unsigned applets:	Yes

Figure 44. Applet Viewer Properties

3.1.5.4 Running the Applet

After changing the AppletViewer properties, you can run the applet. When you click the Get all parts button, a Java Database Connectivity (JDBC) query request is sent to the AS/400 system. The AS/400 system responds by creating a result set of Parts data that is returned to the applet. The applet fills the multi-column listbox with the returned Parts data, as shown in Figure 45.

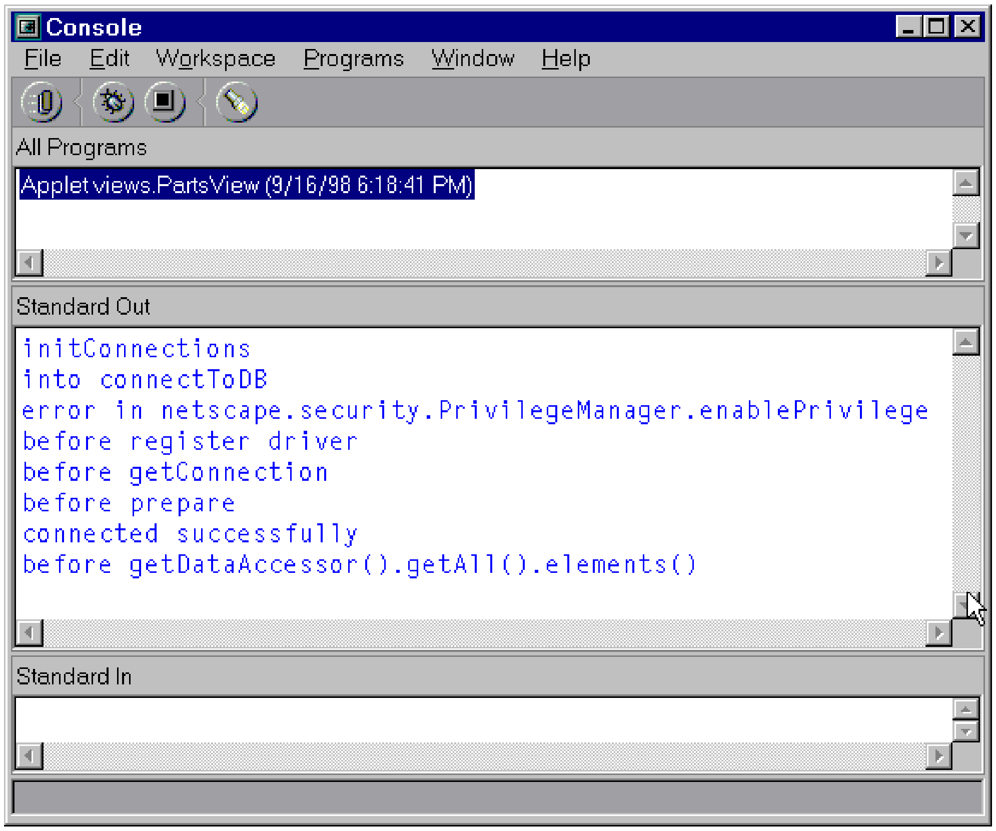


ID	Description	Quantity	Price	Date
12301	Quad speed CD ROM Drive	14	151.38	1998-09-01
12302	SCSI II Cable	25	30.00	1995-11-13
12303	17 inch SVGA Monitor	6	1100.75	1996-03-04
12304	Ethernet PCMCIA card	30	85.30	1995-12-17
12305	Home mouse	47	25.50	1996-02-18
12306	Gender-bender	75	8.50	1951-08-27
12307	600 dpi flatbed scanner	12	875.33	1996-03-01
12308	100 MHZ Pentium PC	4	1875.20	1996-02-24
12309	LaserJet Toner	12	89.45	1995-12-17
12310	Logo mouse mat	376	7.25	1994-11-24
12311	Screen wipes	4750	1.50	1996-01-10
12312	V34 Modem	58	120.45	1996-03-06

Applet started.

Figure 45. Displaying the AS/400 Parts Data Using the PartsView Applet

The sample applet also sends status messages to the Java Console so that you can track the execution of the applet (Figure 46 on page 78). If the applet generates any unhandled exceptions, the Java stack trace is also written to the Console. Using the Console and the VisualAge for Java Debugger, you can locate and correct any coding errors in the applet.



```
Applet views.PartsView (9/16/98 6:18:41 PM)
```

```
initConnections
into connectToDB
error in netscape.security.PrivilegeManager.enablePrivilege
before register driver
before getConnection
before prepare
connected successfully
before getDataAccessor().getAll().elements()
```

Figure 46. The Java Console Messages

3.2 Detailed Review of Java Classes Used in the Applet

Now that you know the steps required to create the applet and test it in the VisualAge for Java Applet Viewer, you review the code in the Java classes. As described in Section 3.1.1, "Importing the Source Code for the Applet to the Workbench" on page 52, there are three packages, five classes, and one interface used in the applet. Before reviewing the code, it is helpful to understand the overall design of the applet and the testing and debugging features that are included in the applet.

3.2.1 Design of the Applet

Although the application was initially designed, tested, and used as an applet, the intention was to create a body of reusable code that can be used when the application is migrated to a servlet. By adopting the three-tier architecture shown in Figure 47, the application can be easily changed.

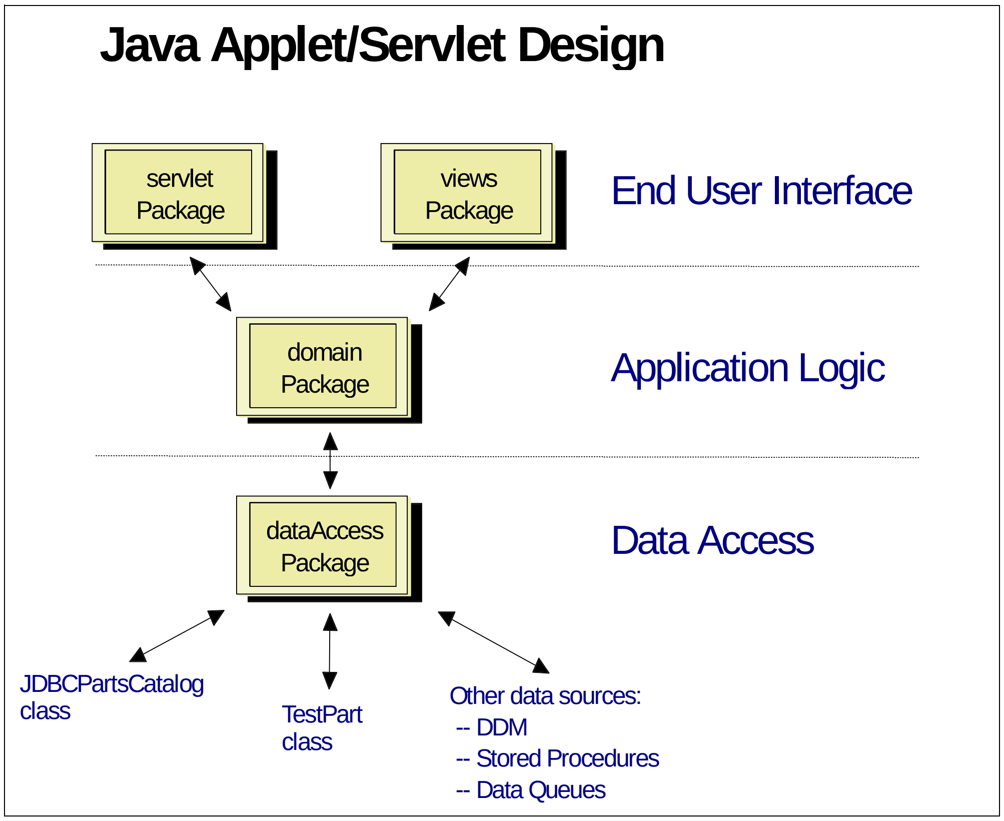


Figure 47. The Applet/Servet Three-Tier Design

In fact, it is quite easy to change the application at any of the three tiers. In addition to changing the end-user interface code to support an applet or a servlet presentation, the data source can be changed in the data access layer without affecting the other two layers. Also, if any of the application logic needs to be changed, the changes can be made at that layer without affecting either the data access or end user interface layers.

The application is well positioned for change if a different data access technique is used. For example, the IBM Toolkit for Java supports record-level access using the AS/400 system Distributed Data Management (DDM) server, program call, and data queues. If you decide to change from the Java Database Connectivity (JDBC) technique, all you need to do is code your new data access class so that it returns the same type and format of data to the application logic layer.

The applet uses the `JDBCPartsCatalog` and `TestPart` classes. The `TestPart` class is used to simulate a connection to a data source.

3.2.2 Testing and Debugging Features in the PartsView Applet

The PartsView applet includes code to help develop, test, and debug the applet. You may want to incorporate some of the same techniques in applets that you create.

3.2.2.1 Internal Test Data

The primary test feature is the `DataAccessor` interface in the `dataAccess` package. Using the `DataAccessor` interface, we created an internal (to the applet) data source that provides test data to the other classes. The reason for the internal data source is that it is relatively difficult to debug network and database connection issues when you develop an applet. Because you can test with and view the internal data, you can verify that the main parts of the applet work correctly before trying to connect to the AS/400 system database. Also, because there is no overhead associated with connecting to the applet's internal data, you can test the applet more quickly in the applet viewer. This way, you do not have to wait for the database connection to be established each time you start a test.

The most obvious benefit of the test data is that it exposes errors in the rest of your applet at an early stage in its development. If your applet cannot correctly obtain and display the internal test data, it obviously cannot work with data from the AS/400 system database.

Deferring the network connection until after all of the other parts of applet are tested is one of the most productive techniques you can adopt. Although the applet includes "extra code", not only for the test data but also to select the connection, the small amount of time and effort required to include the test code is quickly repaid.

3.2.2.2 Logging to the Console

The applet includes several calls to the `System.out.println` method to write messages to the Java console. The messages are written before major sections of the applet are executed. Although it may be argued that console logging is unnecessary, since you can use the Debugger to follow the applet's flow, the console log is useful in helping you isolate a failing section of code. If an expected message does not appear in the console, you know that you need to start debugging in the code after the last displayed message and before the expected message.

3.2.2.3 Exception Handling

The applet uses `try` / `catch` blocks for sections of code where there is a known possibility of an error occuring. In some classes, the `catch` exception handling code writes a message to the console with the `System.out.println` method. In the `PartsView` class, the `handleException` method is invoked. That method uses the `exception.printStackTrace` method to write the Java stack trace to the Java console.

VisualAge for Java automatically includes the `handleException` method in the `PartsView` class since that class was created in the Visual Composition editor. However, the code inside the method is commented out. You need to remove the comments from the code before testing the applet to get the benefit of the stack trace logging.

3.2.3 The PartsView Class

The `PartsView` class is used to display the user interface components of the applet. This class contains code to instantiate the components, respond to the click event on the button, and put records retrieved from the data source into the multicolumn listbox.

Figure 48 shows the `views` package, which contains the `PartsView` class in the VisualAge for Java workbench.

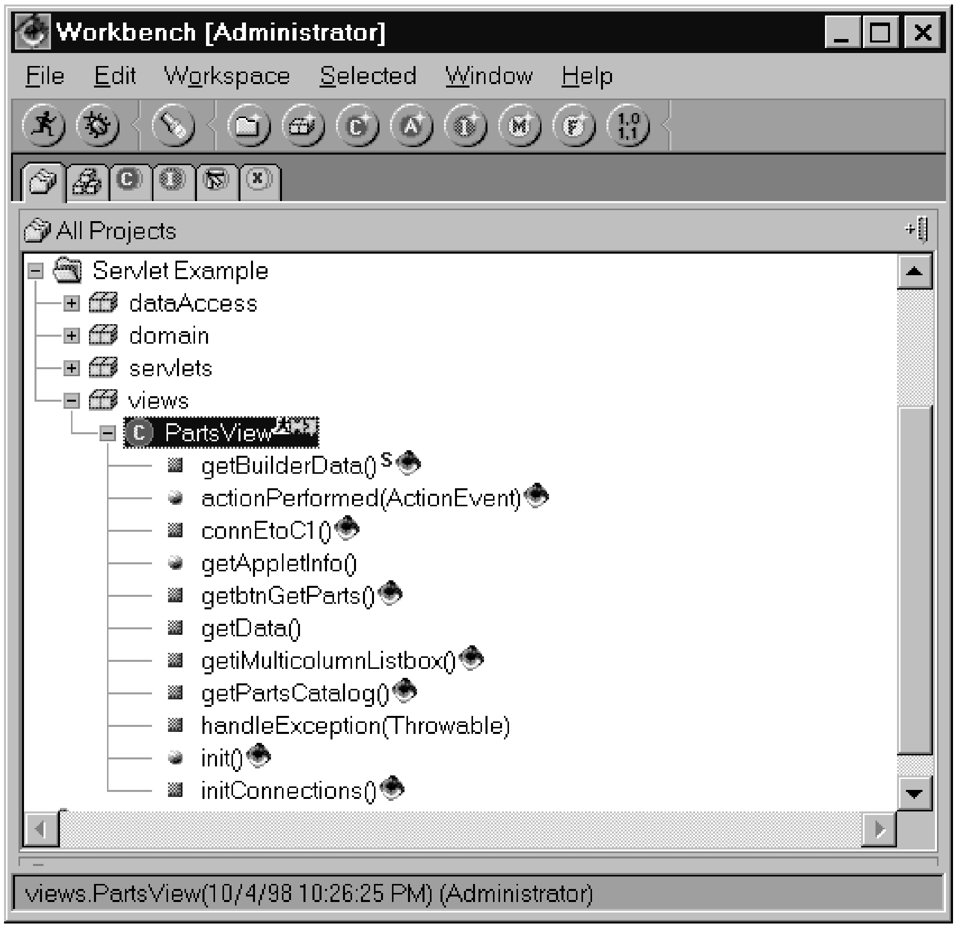


Figure 48. The Views Package

3.2.3.1 Methods Used in the PartsView Class

Table 6 summarizes the methods used in the `PartsView` class.

Table 6. Methods Used in the PartsView Class

Method	Description
`getBuilderData`	Added by the VisualAge for Java Visual Composition Editor (VCE). This method is used to instruct the VCE how to display the bean if the PartsView class is imported from a Java source file. The code for this method is not shown in the following sections, since it is simply a large comment block of hexadecimal characters.
`actionPerformed`	Invoked when an event occurs in the applet (Java 1.1 event model event handler).
`connEtoC1`	Event handler for the button click event (method name generated in Visual Composition editor).
`getAppletInfo`	Returns text describing the class.
`getbtnGetParts`	Adds the button to the applet.
`getData`	Get parts from the data source; add to the multicolumn listbox.
`getIMulticolumListbox`	Adds the mutlicolumn listbox to the applet.
`getPartsCatalog`	Adds the PartsCatalog bean to the applet.
`handleException`	Generic exception handler; logs Java stack to console.
`init`	Invoked when the applet is loaded; adds listbox, and button; initializes connections (database and event handler).
`initConnections`	Connects to the data source; adds event listener to the button.

3.2.3.2 PartsView Class Code

The code shown in Figure 49 is used to define the `PartsView` class. Note that the class extends the `Applet` class.

```
package views;

//*****************************************************************
// PartsView class - visual interface for the applet
//*****************************************************************
import java.applet.*;
import java.awt.*;

public class PartsView extends Applet implements java.awt.event.ActionListener {

    private com.ibm.ivj.eab.dab.IMulticolumnListbox  ivjIMulticolumnListbox = null;
    private domain.PartsCatalog                      ivjPartsCatalog        = null;
    private Button                                   ivjbtnGetParts         = null;

}
```

Figure 49. The PartsView Class

3.2.3.3 The actionPerformed Method in the PartsView Class

The code shown in Figure 50 is used to define the `actionPerformed` method. This method is invoked whenever there is an action event on the applet. The source of the action is compared with the user interface components that are registered as event listeners. For this applet, the only event listener is for the button. When the button is clicked, the `connEtoC1` method is invoked.

```
//*****************************************************************
// actionPerformed method - handle events for the ActionListener
// interface
//*****************************************************************
public void actionPerformed(java.awt.event.ActionEvent e) {

    if ((e.getSource() == getbtnGetParts()) ) {
        connEtoC1();
    }
}
```

Figure 50. The actionPerformed Method in the PartsView Class

3.2.3.4 The connEtoC1 Method in the PartsView Class

The code shown in Figure 51 is used to define the `connEtoC1` method. This method is invoked when the button on the applet is clicked (the method is invoked from within the `actionPerformed` method in this class). The `connEtoC1` method invokes the `getData` method in this class, which retrieves records from the data source that is opened for this execution of the applet in the `initConnections` method in this class.

The method name `connEtoC1` was automatically generated by the VisualAge for Java Visual Composition editor when the connection from the button to the `getData` method was drawn.

```
//*****************************************************************
// connEtoC1 method - used to handle the button click event
//*****************************************************************
private void connEtoC1() {

    try {
        this.getData();
    }

    catch (java.lang.Throwable ivjExc) {
        handleException(ivjExc);
    }
}
```

Figure 51. The connEtoC1 Method in the PartsView Class

3.2.3.5 The getAppletInfo Method in the PartsView Class

The code shown in Figure 52 is used to define the `getAppletInfo` method. This method is automatically generated by the VisualAge for Java Visual Composition editor. Its purpose is to return a brief text identifier that describes the applet. This method is not otherwise used in this project.

```
//***************************************************************
// getAppletInfo method - returns string description of the applet
//***************************************************************
public String getAppletInfo() {

    return "servlets.examples.views.PartsView created using VisualAge for Java.";
}
```

Figure 52. The getAppletInfo Method in the PartsView Class

3.2.3.6 The getbtnGetParts Method in the PartsView Class

The code shown in Figure 53 is used to define the `getbtnGetParts` method. This method is invoked in the `init` method in this class when the applet starts. If the `ivjbtnGetParts` object was not previously instantiated (the button on the applet has not yet been created), the code inside the `try` block is executed. That code instantiates a `java.awt.Button` object and sets its properties.

If there is an existing instance of the `ivjbtnGetParts` button, the code in the `if` block is skipped, and the reference to the existing button object is returned.

```
//***************************************************************
// getbtnGetParts method - instantiate/return the button used on
// the applet
//***************************************************************
private Button getbtnGetParts() {

    if (ivjbtnGetParts == null) {
        try {
            ivjbtnGetParts = new java.awt.Button();
            ivjbtnGetParts.setName("btnGetParts");
            ivjbtnGetParts.setBounds(185, 288, 103, 30);
            ivjbtnGetParts.setActionCommand("Get all parts");
            ivjbtnGetParts.setLabel("Get all parts");

        }

        catch (java.lang.Throwable ivjExc) {
            handleException(ivjExc);
        }
    }

    return ivjbtnGetParts;
}
```

Figure 53. The getbtnGetParts Method in the PartsView Class

3.2.3.7 The getData Method in the PartsView Class

The code shown in Figure 54 is used to define the `getData` method. This method is invoked in the `connEtoC1` method in this class when the button is clicked on the applet.

The method creates a `Vector` and an `Enumeration` object. The `data` vector is used to hold the rows retrieved from the data source. The `parts` enumeration is used to access each of the elements within the vector so that each part can be accessed individually.

The `while` block is used to iterate over the `parts` enumeration. For each element in the enumeration, object `aPart` is instantiated, which is an instance of the `Part` class. The `getAttributeString` method for the `aPart` object is invoked, which returns the list of column values for the part. The values are added as a row to the multicolumn listbox.

```
//***************************************************************
// getData method - get parts from data source, add to listbox
//***************************************************************
private void getData ( ) {

    //***************************************************************
    // get all parts from the data source to vector "data",
    // fill enumeration "parts" with all data elements
    //***************************************************************
    java.util.Vector data = getPartsCatalog().getAll();
    java.util.Enumeration parts = data.elements();

    //***************************************************************
    // add each part to the MultiColumnListBox,
    // use the part number as key
    //***************************************************************
    while (parts.hasMoreElements()) {

        domain.Part aPart = (domain.Part)parts.nextElement();
        getIMulticolumnListbox().addRow(aPart.getAttributeString(),
                                        aPart.getNumber());
    }

return;
}
```

Figure 54. The getData Method in the PartsView Class

3.2.3.8 The getIMulticolumnListbox Method in the PartsView Class

The code shown in Figure 55 is used to define the `getIMulticolumnListbox` method. This method is invoked in the `init` method in this class when the applet starts. If the `ivjIMulticolumnListbox` object was not previously instantiated (the multicolumn listbox on the applet is not yet created), the code inside the `try` block is executed. That code instantiates a `com.ibm.ivj.eab.dab.IMulticolumnLibstbox` object and sets its properties.

If there is an existing instance of the `ivjIMulticolumnListbox` listbox, the code in the `if` block is skipped, and the reference to the existing listbox object is returned.

```
//****************************************************************
// getIMulticolumnListbox method - instantiate/return the listbox
// used on the applet
//****************************************************************
private com.ibm.ivj.eab.dab.IMulticolumnListbox getIMulticolumnListbox() {

    if (ivjIMulticolumnListbox == null) {

        try {
            ivjIMulticolumnListbox = new com.ibm.ivj.eab.dab.IMulticolumnListbox();
            ivjIMulticolumnListbox.setName("IMulticolumnListbox");
            ivjIMulticolumnListbox.setDataBackground(java.awt.Color.lightGray);
            ivjIMulticolumnListbox.setHeadingForeground(java.awt.Color.white);
            String ivjLocal0columns [] = {
                "  ID",
                "       Description",
                "Quantity",
                "  Price",
                "      Date"};
            ivjIMulticolumnListbox.setColumns(ivjLocal0columns);
            int ivjLocal0columnWidths [] = {
                50,
                180,
                65,
                55,
                80};
            ivjIMulticolumnListbox.setColumnWidths(ivjLocal0columnWidths);
            ivjIMulticolumnListbox.setBounds(22, 12, 450, 268);
            ivjIMulticolumnListbox.setHeadingBackground(java.awt.Color.black);
            ivjIMulticolumnListbox.setHeadingFont(new java.awt.Font("dialog", 0, 14));

        }

        catch (java.lang.Throwable ivjExc) {
            handleException(ivjExc);
        }
    }

    return ivjIMulticolumnListbox;
}
```

Figure 55. The getIMulticolumnListbox Method in the PartsView Class

3.2.3.9 The getPartsCatalog Method in the PartsView Class

The code shown in Figure 56 is used to define the `getPartsCatalog` method. This method is invoked in the `initConnections` method when the applet starts and in the `getData` method in this class when the button is clicked on the applet.

The method returns a reference to the `ivjPartsCatalog` object, which points to the data source being used. The data source is either the internal test data source for the applet or the connection to the AS/400 system.

```
//***************************************************************
// getPartsCatalog method - instantiate/return the PartsCatalog
// used on the applet
//***************************************************************
private domain.PartsCatalog getPartsCatalog() {

    if (ivjPartsCatalog == null) {

        try {
            ivjPartsCatalog = new domain.PartsCatalog();

        }

        catch (java.lang.Throwable ivjExc) {
            handleException(ivjExc);
        }
    }

    return ivjPartsCatalog;
}
```

Figure 56. The getPartsCatalog method in the PartsView Class

3.2.3.10 The handleException Method in the PartsView Class

The code shown in Figure 57 is used to define the `handleException` method. This method is invoked within several `catch` blocks in this class. The method logs a message to the Java console followed by the Java stack trace.

This method is added to the applet automatically in the VisualAge for Java Visual Composition editor. The two lines of code that log to the Java console are commented out. To enable the logging, you must remove the comments before testing the applet.

```
//***************************************************************
// handleException method - handle exceptions that are otherwise
// unhandled
//***************************************************************
private void handleException(Throwable exception) {

     System.out.println("--------- UNCAUGHT EXCEPTION ---------");
     exception.printStackTrace(System.out);
}
```

Figure 57. The handleException Method in the PartsView Class

3.2.3.11 The init Method in the PartsView Class

The code shown in Figure 58 is used to define the `init` method. This method is automatically invoked for the applet when it is started in either the VisualAge for Java AppletViewer or in a browser.

The `super.init()` method call is used to initialize the `Applet` class that the `PartsView` class extends. Inside the `try` block, the user interface for the applet is constructed. Finally, the `initConnections` method is called to initialize the connection to the selected data source and to add an action listener to the button.

```
//*****************************************************************
// init method - called when the applet is invoked to initialize
// the applet
//*****************************************************************
public void init() {

    super.init();

    try {
        setName("PartsView");
        setLayout(null);
        setSize(495, 331);
        add(getIMulticolumnListbox(), getIMulticolumnListbox().getName());
        add(getbtnGetParts(), getbtnGetParts().getName());
        initConnections();

    }

    catch (java.lang.Throwable ivjExc) {
        handleException(ivjExc);
    }
}
```

Figure 58. The init Method in the PartsView Class

3.2.3.12 The initConnections Method in the PartsView Class

The code shown in Figure 59 is used to define the `initConnections` method. This method is used to instantiate a `PartsCatalog` object and open the connection to the selected data source. It also adds an action listener to the button on the applet.

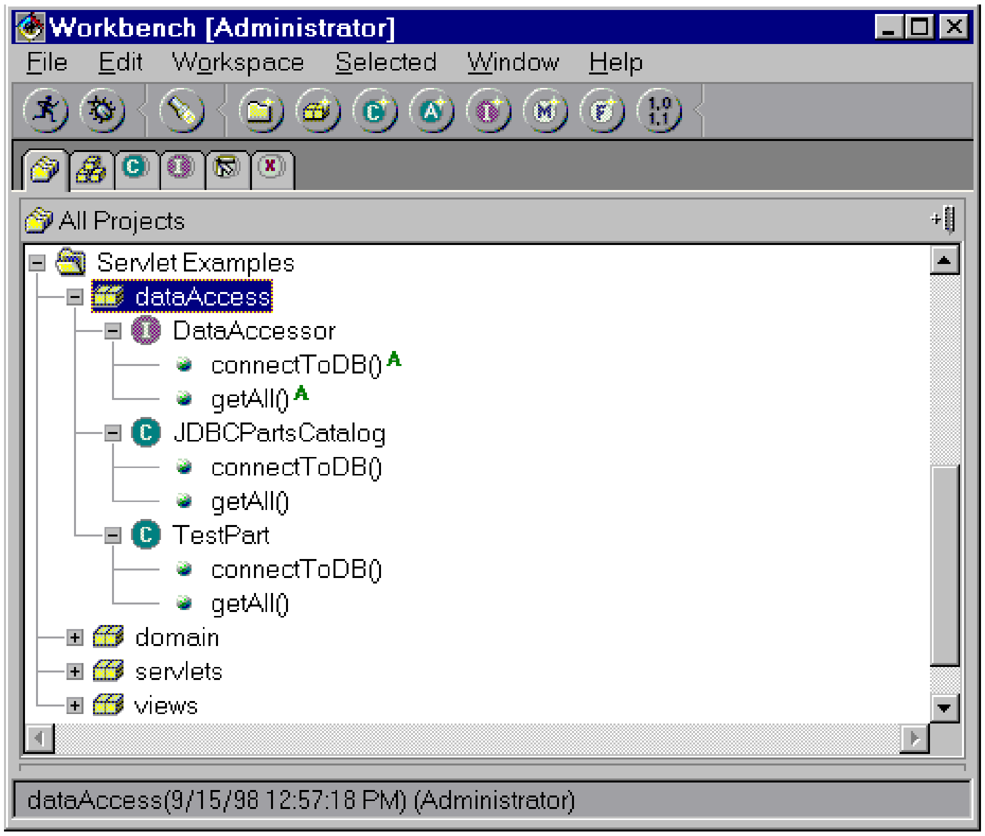
```
//*****************************************************************
// initConnections method - initialize connection to data source,
// add action listener to button
//*****************************************************************
private void initConnections() {

    System.out.println("initConnections");
    getPartsCatalog().connectToDB();

    getbtnGetParts().addActionListener(this);
}
```

Figure 59. The initConnections Method in the PartsView Class

3.2.4 The DataAccessor Interface

The DataAccessor interface is used to define methods that must be included in classes that implement the DataAccessor interface. The interface does not contain any code to indicate how the methods must be implemented. It simply lists the required methods and their parameters.

Figure 60 shows the dataAccess package in the VisualAge for Java workbench. That package contains the DataAccessor interface, the JDBCPartsCatalog class, and the TestPart class.



dataAccess(9/15/98 12:57:18 PM) (Administrator)

Figure 60. The dataAccess Package

3.2.4.1 Methods defined in the DataAccessor interface

Table 7 summarizes the methods defined in the DataAccessor interface.

Table 7. Methods Defined in the DataAccessor Interface

Method	Description
connectToDB	Defines the connectToDB method that must be included in classes that implement the DataAccessor interface. This method is used to create a connection to a data source.
getAll	Defines the getAll method that must be included in classes that implement the DataAccessor interface. This method is used to return all data from the selected data source in a Vector.

3.2.4.2 DataAccessor Interface Code

The code shown in Figure 61 is used to define the DataAccessor interface.

```
package dataAccess;

import java.util.*;

//*****************************************************************
// define methods that must be implemented for accessing
// a data source
//*****************************************************************
public interface DataAccessor {

//*****************************************************************
// connectToDB method - connect to a data source
//*****************************************************************
public void connectToDB ( );

//*****************************************************************
// getAll method - get all data from the data source
//*****************************************************************
public Vector getAll( );

}
```

Figure 61. The DataAccessor Interface in the dataAccess Package

3.2.5 The JDBCPartsCatalog Class

The JDBCPartsCatalog class is used to connect to the AS/400 system using the Java Database Connectivity (JDBC) driver. This class also returns all records from a query run over the connection to the applet. Figure 60 on page 89 shows the dataAccess package, which contains the JDBCPartsCatalog class in the VisualAge for Java workbench.

3.2.5.1 Methods Used in the JDBCPartsCatalog Class

Table 8 summarizes the methods used in the JDBCPartsCatalog class. Note that these methods are required because the JDBCPartsCatalog class implements the DataAccessor interface.

Table 8. Methods Used in the JDBCPartsCatalog Class

Method	Description
connectToDB	This method is used to create a connection to the AS/400 system database using JDBC. The method performs the following functions: – Sets properties for the JDBC connection – Enables security for the Netscape Navigator browser – Registers the AS/400 JDBC driver with the Driver Manager – Gets a connection to the AS/400 system – Creates the prepared statement object that defines the SQL query statement to be run
getAll	This method is used to execute the query and return the query results in a Vector object.

3.2.5.2 JDBCPartsCatalog Class Code

The code shown in Figure 62 on page 91 is used to define the `JDBCPartsCatalog` class. Note that the class implements the `dataAccess.DataAccessor` interface. Therefore, it is required to implement the two methods defined in the interface.

The class uses `static final String` variables to hold the values required to make the connection to the AS/400 system. If you run the sample applet, you need to change the SYSTEMNAME, USERID, and PASSWORD variables to the required values for your AS/400 system. In a production applet, you would allow the user to enter the values, rather than hard-code them into the code for the applet.

The class also defines the `java.sql.Connection` and `java.sql.PreparedStatement` objects that will be used in the query.

```
//*****************************************************************
// Class JDBCPartsCatalog - work with a JDBC data source
//*****************************************************************
import java.util.*;
import java.sql.*;
import java.net.*;

public class JDBCPartsCatalog implements dataAccess.DataAccessor {

    private static final String SYSTEMNAME = "MYAS400";     // add your system name
    private static final String LIBRARY = "APILIB";         // add your library
    private static final String USERID = "MYUSERID";        // add your userid
    private static final String PASSWORD = "MYPASSWORD";    // add your password

    private java.sql.Connection dbConnection;
    private java.sql.PreparedStatement psAllRecord;
}
```

Figure 62. The JDBCPartsCatalog Class

3.2.5.3 The connectToDB Method in the JDBCPartsCatalog Class

The code shown in Figure 63 on page 92 is used to define the `connectToDB` method. This method is invoked from the `domain.PartsCatalog` `connectToDB` method when the applet needs to make the connection to the AS/400 system.

This class includes three main sections:

- Instantiation and setting of a `Properties` object used for the JDBC connection
- Enabling security for the Netscape Navigator browser
- Instantiating and setting the `Connection` and `PreparedStatement` objects for the query

```
//*****************************************************************
// connectToDB method - connect to JDBC data source
//*****************************************************************
public void connectToDB ( ) {

    System.out.println("into connectToDB");

    //*****************************************************************
    // Create a properties object for JDBC connection
    //*****************************************************************
    Properties jdbcProperties = new Properties();

    //*****************************************************************
    // Set the properties for the JDBC connection
    //*****************************************************************
    jdbcProperties.put("user",              USERID);
    jdbcProperties.put("password",          PASSWORD);
    jdbcProperties.put("naming",            "sql");
    jdbcProperties.put("errors",            "full");
    jdbcProperties.put("date format",       "iso");
    jdbcProperties.put("extended dynamic",  "true");
    jdbcProperties.put("package",           "SerTest");

    //*****************************************************************
    // enable security for Netscape
    //*****************************************************************
    try {
        netscape.security.PrivilegeManager.enablePrivilege("UniversalConnect");
    }

    catch (Throwable exception) {
        System.out.println("error in enable security for Netscape");
    }

    //*****************************************************************
    // register the driver using the AS/400 JDBC driver,
    // get a connection to the AS/400,
    // initialize the prepared statement for the query
    //*****************************************************************
    try {
        System.out.println("before register driver");
        DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());

        System.out.println("before getConnection");
        dbConnection = DriverManager.getConnection("jdbc:as400://" +
                                                   SYSTEMNAME +
                                                   "/" +
                                                   LIBRARY,
                                                   jdbcProperties);

        System.out.println("before prepare");
        psAllRecord = dbConnection.prepareStatement("Select * from Parts");
    }

    catch (SQLException ex) {
        System.out.println("connect failed");
        ex.printStackTrace();
        return;
    }

    System.out.println("connected successfully");
    return;
}
```

Figure 63. The connectToDB Method in the JDBCPartsCatalog Class

The Properties Object for the JDBC Connection

A properties object named `jdbcProperties` is created to contain the property settings that are associated with the JDBC connection to the AS/400 system. The advantage of setting the JDBC connection properties is that you can assign a value to each setting individually, rather than code all of the properties in a lengthy string. It is much easier to review and change the individual settings.

Note

You can find documentation about the JDBC properties and values in the VisualAge for Java Help system. To get to the Help page for JDBC properties, perform the following steps:

1. Click ***Help*** on the VisualAge for Java workbench menu.
2. Click **Tools—>AS/400 Toolbox for Java**. A Web page, which is the Help index, opens.
3. On the Web page, click the **Access Classes** link.
4. Click the **JDBC** link.
5. Click the **Connection** link.
6. Scroll down on the Connections page, and click the **JDBC Properties** link.

Enable Security for the Netscape Navigator Browser

When you try to make a connection to the AS/400 system from the applet, the Netscape browser throws a security exception. To enable the Netscape Navigator browser to connect from the applet to the AS/400 system, you need to include the line of code shown in the method.

The applet was tested in the Netscape Navigator browser, version 4.07 for Windows 95/NT.

Instantiate the Connection and PreparedStatement Objects

The code in this section of the method is used to get a `Connection` object to the AS/400 system and to initialize a `PreparedStatement` object with the SQL statement that will perform the query.

The `java.sql.DriverManager.registerDriver` method is used to load the AS/400 JDBC driver for the applet. The `DriverManager.getConnection` method is used to identify the AS/400 system name and library that the applet will use and also to associate the JDBC `Properties` object with the `Connection` object.

After instantiating the `Connection` object, the `Connection` object's `prepareStatement` method is used to instantiate and initialize the `PreparedStatement` object that contains the SQL statement.

3.2.5.4 The getAll Method in the JDBCPartsCatalog Class

The code shown in Figure 64 on page 94 is used to define the `getAll` method. This method is invoked from the `domain.PartsCatalog getAll` method when the applet needs to get database records from the AS/400 system.

The code defines a `Vector` and a `ResultSet` object. The `ResultSet` object is filled with the result set of the query executed on the `PreparedStatement` object from the connectToDB method (see “Instantiate the Connection and PreparedStatement Objects” on page 93).

After executing the query, the code loops through the result set. Column data in each row retrieved from the result set is put into a String array. The String array is added to the Vector object.

The Vector object is returned from this method. It is up to the invoking class to retrieve the results from the Vector object.

```
//*****************************************************************
// getAll method - execute query, get/return all rows from
// data source
//*****************************************************************
public Vector getAll() {

    Vector aDataVector = new Vector();
    java.sql.ResultSet aResultSet = null;

    //*****************************************************************
    // excute the query, put all row data into vector
    //*****************************************************************
    try {
        aResultSet = psAllRecord.executeQuery();

        //*****************************************************************
        // loop through all rows retrieved by the SQL query
        //*****************************************************************
        while (aResultSet.next()) {

            String[] data = new String[5];

            data[0] = aResultSet.getBigDecimal(1,0).toString();   // number
            data[1] = aResultSet.getString(2);                    // description
            data[2] = aResultSet.getBigDecimal(3,0).toString();   // quantity
            data[3] = aResultSet.getBigDecimal(4,2).toString();   // price
            data[4] = aResultSet.getDate(5).toString();           // date

            //*****************************************************************
            // add the data element (String[]) to the vector
            //*****************************************************************
            aDataVector.addElement(data);
        }
    }

    catch (SQLException ex) {
        ex.printStackTrace();
    }

    return aDataVector;
}
```

Figure 64. The getAll Method in the JDBCPartsCatalog Class

3.2.6 The TestPart Class

The TestPart class is used to simulate a connection to the AS/400 system. The class also returns sample records that are hard-coded into the class. This class is used early in the development and test cycle for the applet to help you debug the user interface layer of the applet and the application logic layer.

Figure 60 on page 89 shows the dataAccess package, which contains the TestPart class in the VisualAge for Java workbench.

3.2.6.1 Methods Used in the TestPart Class

Table 9 summarizes the methods used in the TestPart class. Note that these methods are required because the TestPart class implements the DataAccessor interface.

Table 9. Methods Used in the TestPart Class

Method	Description
connectToDB	This method is used to simulate a connection to the AS/400 system.
getAll	This method is used to simulate the execution of a query. Test data is returned in a Vector object.

3.2.6.2 TestPart class Code

The code shown in Figure 65 is used to define the TestPart class. Note that the class implements the dataAccess.DataAccessor interface, and is, therefore, required to implement the two methods defined in the interface.

```
//****************************************************************
// TestPart class - provide simulated data from a data source
//****************************************************************
import java.util.*;

public class TestPart implements dataAccess.DataAccessor {
}
```

Figure 65. The TestPart Class

3.2.6.3 The connectToDB Method in the TestPart Class

The code shown in Figure 66 is used to define the connectToDB method. This method is invoked from the domain.PartsCatalog connectToDB method when the applet needs to make the connection to the AS/400 system.

Because this class is used for testing, the method immediately returns. As far as the invoking class is concerned, there is now a valid connection to a data source.

```
//****************************************************************
// connectToDB method - simulate connection to data source
//****************************************************************
public void connectToDB() {

    return;
}
```

Figure 66. The connectToDB Method in the TestPart Class

3.2.6.4 The getAll Method in the TestPart Class

The code shown in Figure 67 on page 96 is used to define the getAll method. This method is invoked from the domain.PartsCatalog getAll method when the applet needs to get database records from the data source.

The code defines a Vector and a String array object. Elements of the string array are filled with data to simulate column values in a database row. After filling each column value, the simulated row is added as an element of the Vector.

The class returns the Vector to the invoking class. To the invoking class, it appears as if a query was performed against a data source.

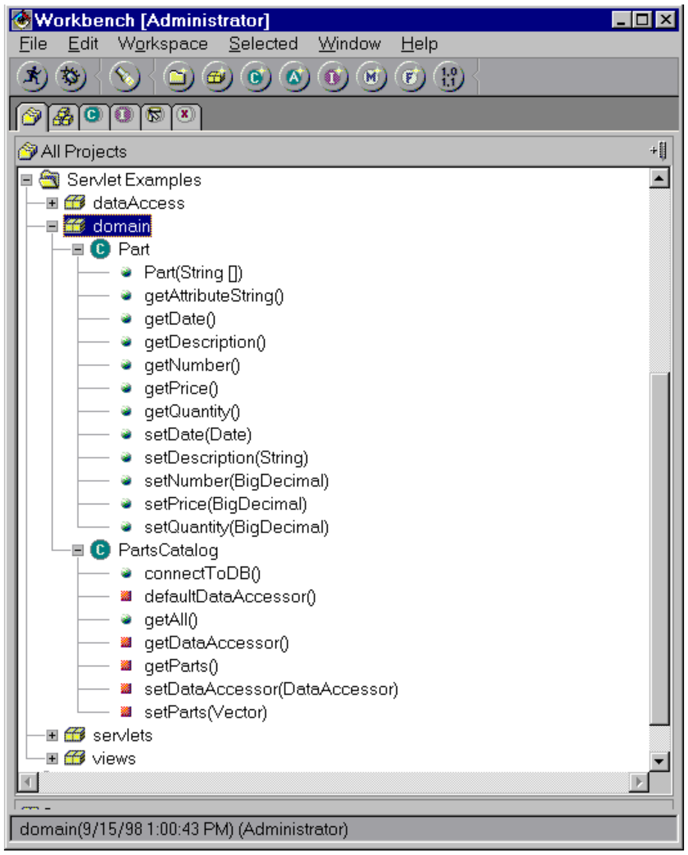
```
//****************************************************************
// getAll method - simulate returning results from a query
//****************************************************************
public Vector getAll() {

    Vector dataVector = new Vector();
    String[] data = new String[5];

    data[0] = "00001";
    data[1] = "Parts Description";
    data[2] = "25";
    data[3] = "552";
    data[4] = "1997-03-02";
    dataVector.addElement(data);

    data = new String[5];

    data[0] = "00002";
    data[1] = "Parts Description2";
    data[2] = "99";
    data[3] = "123";
    data[4] = "1998-03-02";
    dataVector.addElement(data);

    return dataVector;
}
```

Figure 67. The getAll Method in the TestPart Class

3.2.7 The Part Class

The `Part` class is used to define the data and methods that are used to represent a part in the database. The `Part` class is used in the `PartsView` class, `getData` method so that the method can work with the `Vector` of parts returned from the `JDBCPartsCatlog` or `TestPart` classes.

Figure 68 on page 97 shows the `domain` package, which contains the `Part` class and the `PartsCatalog` class in the VisualAge for Java workbench.



```
Servlet Examples
  dataAccess
  domain
    Part
      Part(String [])
      getAttributeString()
      getDate()
      getDescription()
      getNumber()
      getPrice()
      getQuantity()
      setDate(Date)
      setDescription(String)
      setNumber(BigDecimal)
      setPrice(BigDecimal)
      setQuantity(BigDecimal)
    PartsCatalog
      connectToDB()
      defaultDataAccessor()
      getAll()
      getDataAccessor()
      getParts()
      setDataAccessor(DataAccessor)
      setParts(Vector)
  servlets
  views
domain(9/15/98 1:00:43 PM) (Administrator)
```

Figure 68. The Domain Package

3.2.7.1 Methods Used in the Part Class

Table 10 summarizes the methods used in the `Part` class.

Table 10. Methods Used in the Part Class

Method	Description
`Part`	Constructor for the class.
`getAttributeString`	Returns a string array containing part data.
`getDate`	Returns the value of the data associated with the part.
`getDescription`	Returns the value of the description associated with the part.
`getNumber`	Returns the value of the part number associated with the part.
`getPrice`	Returns the value of the price associated with the part.
`getQuantity`	Returns the value of the quantity associated with the part.
`setDate`	Sets the value of the date associated with the part.
`setDescription`	Sets the value of the description associated with the part.
`setNumber`	Sets the value of the part number associated with the part.
`setPrice`	Sets the value of the price associated with the part.
`setQuantity`	Sets the value of the quantity associated with the part.

3.2.7.2 Part Class Code

The code shown in Figure 69 is used to define the `Part` class. The class defines five private variables that are used to hold the data for the part record retrieved from the data source.

```
//*************************************************************
// Part class - all data about a part from the data source
//*************************************************************
import java.math.*;
import java.util.*;
import java.sql.*;

public class Part {

    private BigDecimal    iNumber      = null;
    private String        iDescription = null;
    private BigDecimal    iQuantity    = null;
    private BigDecimal    iPrice       = null;
    private java.sql.Date iDate        = null;

}
```

Figure 69. The Part Class

3.2.7.3 Part Constructor in the Part Class

The code shown in Figure 70 is used to define the constructor used for the `Part` class. When the class is instantiated, a `String` array is passed to the constructor. The `String` array contains elements for each of the data fields contained in the `Part` class. The constructor extracts the data field values from the `String` array and uses the `set` methods in the class to store the values.

```
//*************************************************************
// Part constructor - invoked when a new object based on the
// Part class is instantiated
//*************************************************************
public Part ( String[] data) {

    setNumber(new BigDecimal(data[0]));
    setDescription(data[1]);
    setQuantity(new BigDecimal(data[2]));
    setPrice(new BigDecimal(data[3]));
    setDate(java.sql.Date.valueOf(data[4]));
}
```

Figure 70. The Part Constructor in the Part Class

3.2.7.4 The getAttribute Method in the Part Class

The code shown in Figure 71 on page 99 is used to define the `getAttribute` method. This method is used to return all of the field values associated with the instance of the `Part` class in a `String` array. By using this class, you do not have to invoke each of the `get` methods in your higher-level classes that use the `Part` class when you need to work with the field values.

```
//***************************************************************
// getAttributeString method - return a string array of part data
//***************************************************************
public String[] getAttributeString ( ) {

    String[] returnString = new String[5];

    returnString[0] = getNumber().toString();
    returnString[1] = getDescription();
    returnString[2] = getQuantity().toString();
    returnString[3] = getPrice().toString();
    returnString[4] = getDate().toString();

    return returnString;
}
```

Figure 71. The getAttributeString Method in the Part Class

3.2.7.5 The get Methods in the Part Class

The code shown in Figure 72 is used to define the five `get` methods used in the `Part` class. These methods are used to obtain the current value of a field associated with this instance of the `Part` class.

```
public java.sql.Date getDate() {

        return iDate;
}

public String getDescription() {

        return iDescription;
}

public BigDecimal getNumber() {

        return iNumber;
}

public BigDecimal getPrice() {

        return iPrice;
}

public BigDecimal getQuantity() {

        return iQuantity;
}

public void setDate(java.sql.Date aDate) {

         iDate = aDate;
}
```

Figure 72. The get Methods in the Part Class

3.2.7.6 The set Methods in the Part Class

The code shown in Figure 73 on page 100 is used to define the five `set` methods used in the `Part` class. These methods are used to store the current value of a field for this instance of the `Part` class.

```
public void setDate(java.sql.Date aDate) {

        iDate = aDate;
}

public void setDescription(String aDescription) {

        iDescription = aDescription;
}

public void setNumber(BigDecimal aNumber) {

        iNumber = aNumber;
}

public void setPrice(BigDecimal aPrice) {

        iPrice = aPrice;
}

public void setQuantity(BigDecimal aQuantity) {

        iQuantity = aQuantity;
}
```

Figure 73. The set Methods in the Part Class

3.2.8 The PartsCatalog Class

The `PartsCatalog` class is used for the following purposes:

- To select the data accessor that will be used, which indicates which data source will provide part data to the applet
- To connect to the selected data source
- To retrieve all of the parts from the data source and return a `Vector` object containing the part data

Figure 68 on page 97 shows the `domain` package which contains the PartsCatalog class in the VisualAge for Java workbench.

3.2.8.1 Methods Used in the PartsCatalog Class

Table 11 summarizes the methods used in the `PartsCatalog` class.

Table 11. Methods Used in the PartsCatalog Class

Method	Description
`connectToDB`	Makes the connection to the selected data source.
`defaultDataAccessor`	Determines which data source to use and returns a `Class` that points to the data source.
`getAll`	Initiates the query of the data source and returns all data in a `Vector`.
`getDataAccessor`	Returns a `Class` that points to the selected data source.
`getParts`	Returns a `Vector` containing the parts data.
`setDataAccessor`	Sets the value of the data accessor associated with this class.
`setParts`	Sets the value of the parts `Vector` associated with this class.

3.2.8.2 PartsCatalog Class Code

The code shown in Figure 74 is used to define the `PartsCatalog` class. The class defines a `Vector`, which is used to contain parts data retrieved from the data source, and a `DataAccessor`, which points to the currently selected data source.

```
//**************************************************************
// PartsCatalog class - performs required operations to get
// parts data from the data source and return a vector of
// parts data
//**************************************************************
import java.util.*;

public class PartsCatalog {

    private Vector parts = new Vector();
    private dataAccess.DataAccessor dataAccessor;
}
```

Figure 74. The PartsCatalog Class

3.2.8.3 The connectToDB Method in the PartsCatalog Class

The code shown in Figure 75 is used to define the `connectToDB` method. This method is invoked from the applet in the `initConnections` method (see Section 3.2.3.12, “The initConnections Method in the PartsView Class” on page 88).

The method first determines which data source to use by invoking the `getDataAccessor` method in this class. After determining which data source to use, the `connectToDB` method in the data source’s class is invoked:

- If the data accessor is the `JDBCPartsCatalog`, the `connectToDB` method in the `JDBCPartsCatalog` is invoked (see Section 3.2.5.3, “The connectToDB Method in the JDBCPartsCatalog Class” on page 91).
- If the data accessor is the `TestPart`, the `connectToDB` method in the `TestPart` class is invoked (see Section 3.2.6.3, “The connectToDB Method in the TestPart Class” on page 95).

```
//**************************************************************
// connectToDB method - get the data accessor (select data source
// to use), open connection to the data source
//**************************************************************
public void connectToDB( ) {

    getDataAccessor().connectToDB();
    return;
}
```

Figure 75. The connectToDB Method in the PartsCatalog Class

3.2.8.4 The defaultDataAccessor Method in the PartsCatalog Class

The code shown in Figure 76 on page 102 is used to define the `defaultDataAccessor` method. This method is invoked from the `getDataAccessor` method in this class.

The method includes two lines of code that are used to select the data accessor to use. When you initially test the applet, comment out the line that assigns the `dataAccess.JDBCPartsCatalog` class and remove the comment from the `dataAccess.TestPart` line. When you want to test the applet with the connection to the AS/400 system, remove the comment from the `dataAccess.JDBCPartsCatalog` line and comment out the `dataAccess.TestPart` line.

In a production applet that needs to work with multiple data sources, you can make the selection of the data accessor a user selectable option.

After assigning the class name, the `Class.forName` method is used to instantiate the selected data accessor class, which is returned to the invoking method.

```
//*************************************************************
// defaultDataAccessor method - set the name of the class
// containing the data source to use
//
// Select one of the qualifiedClassName choices, comment the
// other
//*************************************************************
private Class defaultDataAccessor ( ) {

    String qualifiedClassName = "dataAccess.JDBCPartsCatalog";
    //String qualifiedClassName = "dataAccess.TestPart";

    try {
        return Class.forName(qualifiedClassName);
    }

    catch (ClassNotFoundException cnfe) {
        System.out.println("No such class... " + cnfe);
    }

    return null;
}
```

Figure 76. The defaultDataAccessor Method in the PartsCatalog Class

3.2.8.5 The getAll Method in the PartsCatalog Class

The code shown in Figure 77 on page 103 is used to define the `getAll` method. This method is used to get the part data from the selected data source and return the data to the invoking method in a `Vector`. This method is invoked from the applet's `getData` method (see Section 3.2.3.7, "The getData Method in the PartsView Class" on page 85).

The method instantiates a `Vector` used to return the part data. It instantiates and fills an `Enumeration` with the part data by invoking the `getAll` method for the selected data accessor (see Section 3.2.5.4, "The getAll Method in the JDBCPartsCatalog Class" on page 93 and Section 3.2.6.4, "The getAll Method in the TestPart Class" on page 95).

After filling the `Enumeration`, the method iterates through it. Each element in the `Enumeration` is cast to a `Part` object, which is added to the `Vector`. The `Vector` is then returned from the method to the invoking method.

```
//*****************************************************************
// getAll method - get all parts from data source, return
// vector of parts data
//*****************************************************************
public Vector getAll( ) {

    Vector aPartsVector = new Vector();

    //*****************************************************************
    // get all elements from the database.
    // use the data accessor provided in the method getDataAccessor()
    //*****************************************************************
    System.out.println("before getDataAccessor().getAll().elements()");
    Enumeration parts = getDataAccessor().getAll().elements();

    //*****************************************************************
    // for each element in the parts vector create a new instance of
    // Part and add it to the vector aPartsVector
    //*****************************************************************
    while (parts.hasMoreElements()) {
        Part aPart = new Part((String[])parts.nextElement());
        aPartsVector.addElement(aPart);
    }

    setParts(aPartsVector);
    return aPartsVector;
}
```

Figure 77. The getAll Method in the PartsCatalog Class

3.2.8.6 The getDataAccessor Method in the PartsCatalog Class

The code shown in Figure 78 on page 104 is used to define the `getDataAccessor` method. This method is invoked in the `connectToDB` method in this class to determine which data source is to be used (see Section 3.2.8.3, “The connectToDB Method in the PartsCatalog Class” on page 101).

The method invokes the `defaultDataAccessor` method in this class, which returns a `Class` pointing to the selected data accessor (see Section 3.2.8.4, “The defaultDataAccessor Method in the PartsCatalog Class” on page 101). That class value is used by the `setDataAccessor` method in this class to store the value of the selected data accessor (see Section 3.2.8.8, “The set Methods in the PartsCatalog Class” on page 105).

```
//***************************************************************
// getDataAccessor method - return the name of the currently
// selected data source
//***************************************************************
private dataAccess.DataAccessor getDataAccessor() {

    //***************************************************************
    // if the variable dataAccessor is null, create a new instance of
    // the data accessor provided in the method defaultDataAccessor()
    //***************************************************************
    try {
        if (dataAccessor == null) {
            setDataAccessor((dataAccess.DataAccessor)
                             defaultDataAccessor().newInstance());
        }
    }

    catch (InstantiationException ie) {
        System.out.println("Error instantiating DataAccessor class");
    }

    catch (IllegalAccessException ie) {
        System.out.println("Illegal access to DataAccessor class");
    }

    return dataAccessor;
}
```

Figure 78. The getDataAccessor Method in the PartsCatalog Class

3.2.8.7 The getParts Method in the PartsCatalog Class

The code shown in Figure 79 is used to define the `getParts` method. The method is used to return the `Vector` of parts data for this instance of the `PartsCatalog` class.

```
//***************************************************************
// getParts method - return the parts vector
//***************************************************************
private Vector getParts( ) {

    return parts;
}
```

Figure 79. The getParts Method in the PartsCatalog Class

3.2.8.8 The set Methods in the PartsCatalog Class

The code shown in Figure 80 is used to define the `set` methods used in the class. The methods are used to store the values of the parts `Vector` and `DataAccessor` object that are local to the class.

```
//***************************************************************
// setDataAccessor method - set the dataAccessor to the value
// of the currently selected data source
//***************************************************************
private void setDataAccessor(dataAccess.DataAccessor aAccessor) {

        dataAccessor = aAccessor;
        return;
}


//***************************************************************
// setParts method - set the parts vector the parts returned
// from the data source
//***************************************************************
private void setParts(Vector aPartVector) {

        parts = aPartVector;
        return;
}
```

Figure 80. The set Methods in the PartsCatalog Class

3.3 Running the Applet in a Browser

Now that you tested the applet in the AppletViewer included with VisualAge for Java and reviewed all of the classes used in the applet, it is time to run the applet in a browser. The applet is loaded into a browser from an HTML file. The HTML code includes the APPLET tag, which identifies the applet to run and its location in the network.

You have several options for packaging and serving the applet. You need to carefully assess how your applet will be used before deciding how to deploy the applet. For example, the deployment considerations are different if you are developing the applet for use on your enterprise’s intranet, as opposed to making the applet publicy available over the Internet.

3.3.1 Test Environment

While writing this redbook, we tested the applet in the following environment:

- AS/400 system at OS/400 V4R2 and OS/400 V4R3
- AS/400 Toolbox for Java, V3R2 level (5763-JC1)
- Windows NT 4 with Service Pack 3
- Microsoft Windows NT 4 TCP/IP stack with Token-Ring connection to the AS/400 system
- VisualAge for Java Enterprise edition, version 2.0
- Netscape Navigator 4.07
- Microsoft Internet Explorer 4.01 with Service Pack 1 (SP1)

Other test environments should work as well. For example, the Netscape Navigator browser can be used on a Windows 95 or Windows 98 PC.

3.3.2 Serving the Applet from the PC Drive

For your first test, try serving the applet from the PC drive. This test is recommended because there are no dependencies on the network for serving the applet. The steps used in this test are:

1. In the VisualAge for Java workbench, export all of the required classes to a Java Archive (Jar) file on the PC drive.
2. Create or work with the generated HTML file that contains the APPLET tag and add the ARCHIVE tag.
3. Start the browser. This includes opening the Java Console and the HTML file.
4. Respond to the browser's Java Security alert.
5. Work with the applet in the browser.

Each of these steps is explained in detail in the following sections.

3.3.2.1 Exporting Classes to the PC Drive

To run the applet in the browser, you need to get the Java classes used by the applet into a Jar file. The process is called *exporting*. It is supported by several options in the VisualAge for Java environment.

To begin the process, select the three packages that are used in the applet in the VisualAge for Java workbench, as shown in Figure 81 on page 107. To select multiple packages, press and hold the `Ctrl` key, while clicking the required packages.

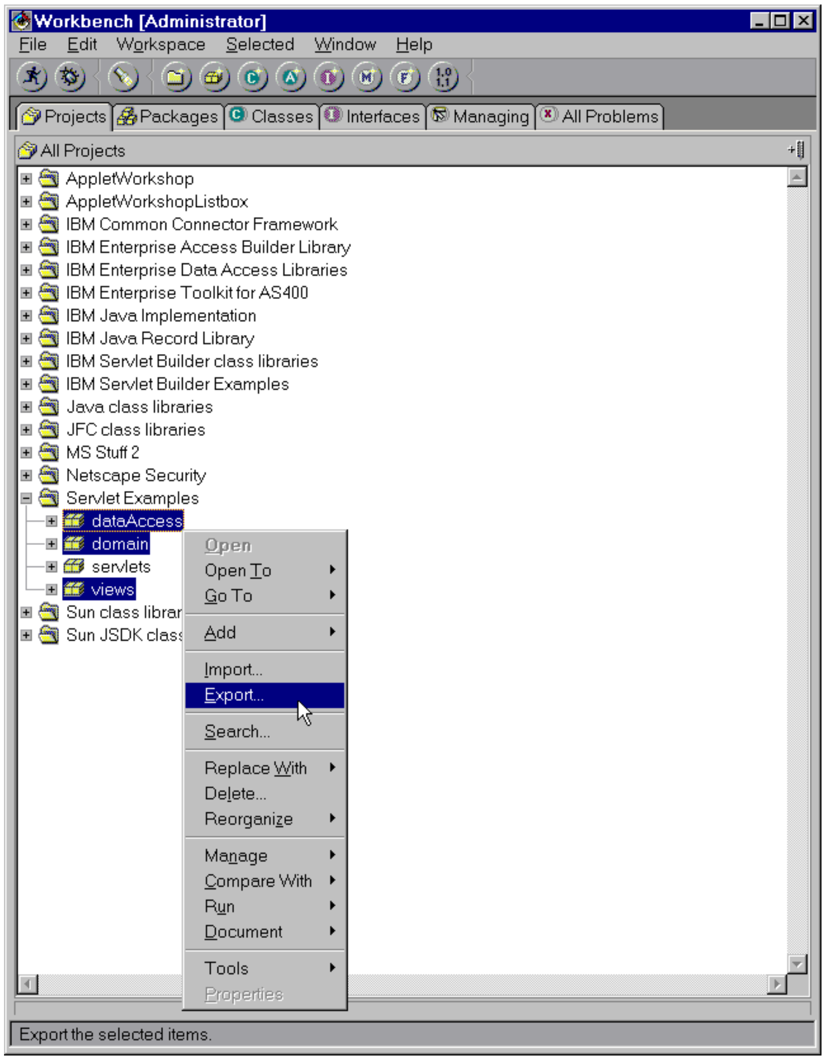


Figure 81. Exporting the Applet Packages

After making your selections, right-click to display the pop-up menu. Click **Export** on the menu. The Export SmartGuide shown in Figure 82 on page 108 appears.

To export the classes to the PC drive, select **Jar file** as the export destination. After selecting the export option, click **Next** to continue.

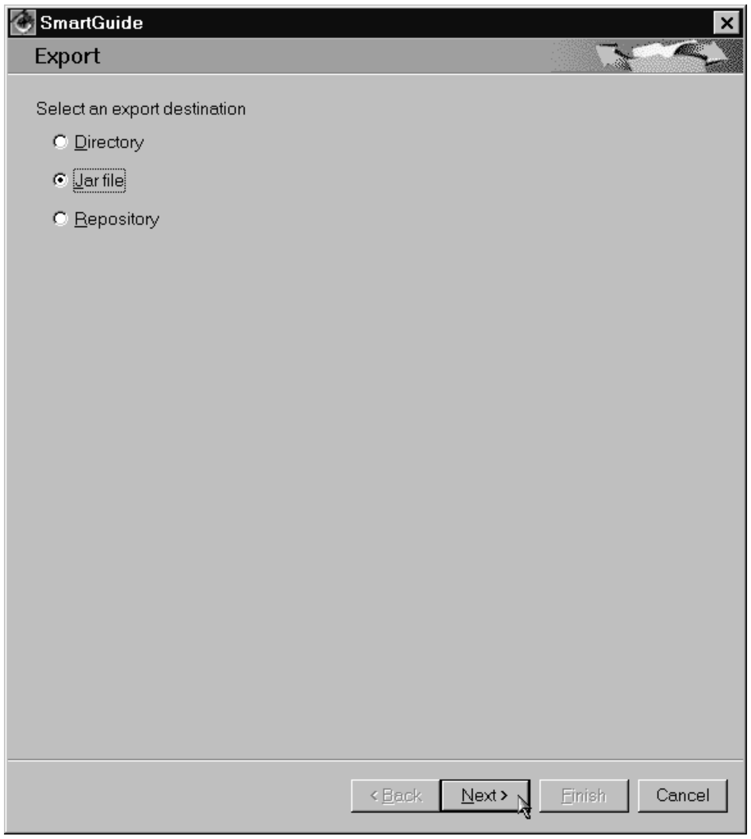


Figure 82. Export SmartGuide

3.3.2.2 Specifying Export Options

The SmartGuide now displays the Export to a jar file dialog (see Figure 83 on page 109). The following selections were made in the dialog:

- The directory to export to is `c:\AppletTest`. The drive and directory are not important. You can export to any directory you want on your PC.
- The class, resource, beans, and html options are selected by default. The six selected classes are the classes defined in the `dataAccess`, `domain`, and `views` packages used in the project. The HTML file is generated by VisualAge for Java so that you can test the applet.

Click the **Finish** button to complete the export. VisualAge for Java writes the jar file into the AppletTest directory and also generates file `PartsView.html`. The name of the HTML file is based on the runnable class in the project.

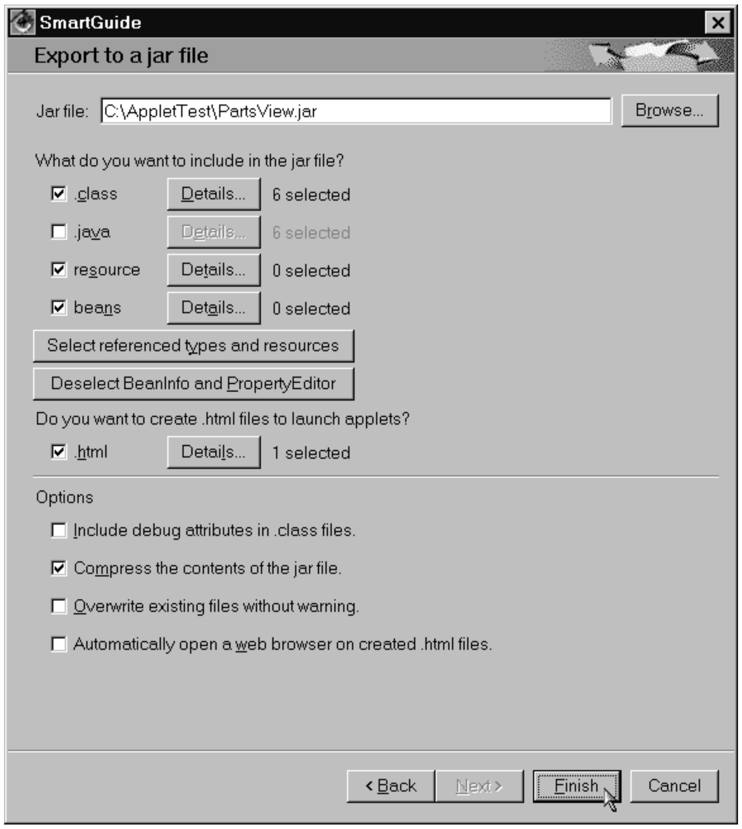

SmartGuide
Export to a jar file
Jar file: C:\AppletTest\PartsView.jar
Browse...
What do you want to include in the jar file?
☑ .class Details... 6 selected
☐ .java Details... 6 selected
☑ resource Details... 0 selected
☑ beans Details... 0 selected
Select referenced types and resources
Deselect BeanInfo and PropertyEditor
Do you want to create .html files to launch applets?
☑ .html Details... 1 selected
Options
☐ Include debug attributes in .class files.
☑ Compress the contents of the jar file.
☐ Overwrite existing files without warning.
☐ Automatically open a web browser on created .html files.
< Back
Next >
Finish
Cancel

Figure 83. The Export to a Jar File Dialog

3.3.2.3 Copying the jt400.jar File to the AppletTest Directory

The PartsView applet uses several Java classes that are provided in the AS/400 Toolbox for Java. When you exported the Java classes in the project, you only exported the classes that are specific to the applet.

For the applet to work, it needs access to the classes in the AS/400 Toolbox for Java. There are several techniques you can use to provide access to those classes. In this example, you copy the Jar file that contains the Toolbox to the directory where the applet's jar file is located.

The AS/400 Toolbox for Java is located in the AS/400 system Integrated File System (IFS). Use a Client Access for Windows 95/NT connection or the AS/400 system NetServer to get to the following IFS directory on your AS/400 system:

```
QIBM\ProdData\HTTP\Public\jt400\lib
```

That directory contains files `jt400.jar` and `jt400.zip`. Copy the `jt400.jar` file to the `c:\AppletTest` directory.

3.3.2.4 Copying the dab.jar File to the AppletTest Directory

The PartsView applet uses several Java classes that are provided in the IBM Enterprise Data Access Libraries project. They are available in a jar file that you download from the redbook Web site. They are in a jar file named `dab.jar`. The classes in this jar file support the multi-column listbox. Copy the `dab.jar` file to the AppletTest directory.

3.3.2.5 Adding the ARCHIVE Parameter to the HTML file

Attention

When using Netscape Navigator, you need version 4.07 or later to use multiple jar files. In our tests with earlier Netscape Navigator browsers, we could not use multiple jar files.

Now that you have the `jt400.jar` and `dab.jar` file in the `AppletTest` directory, you need to modify the generated HTML file so that the applet can use the classes in `jt400.jar`. and `dab.jar`. Perform these steps:

1. Use NotePad or another editor to open file `c:\AppletTest\PartsView.html`.
2. Change the line `ARCHIVE=PartsView.jar` so that it includes the reference to `jt400.jar` and `dab.jar`, as shown in the following code.
3. Save the changed HTML file to the `c:\AppletTest` directory.

```
<HTML>
<HEAD>
<TITLE>PartsView</TITLE>
</HEAD>
<BODY>
<H1>PartsView</H1>
<APPLET CODE=views.PartsView.class
    ARCHIVE=PartsView.jar,jt400.jar,dab.jar
    WIDTH=700
    HEIGHT=350>
</APPLET>
</BODY>
</HTML>
```

Note

Be sure there are no blank spaces in the `ARCHIVE` tag. If you leave a blank after the comma that separates the two jar files, the applet will fail.

3.3.2.6 Starting the Browser; Opening the Java Console

You can now start the Netscape 4.07 or later browser. Adjust the size of the browser so that it does not occupy the entire screen space on your monitor.

You can follow the progress of the applet as it executes by opening the Java Console. Use the **Window—>Java Console** menu item to open the Java console (see Figure 84).

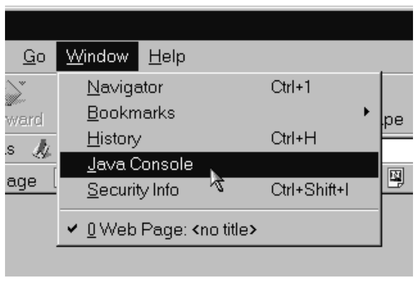
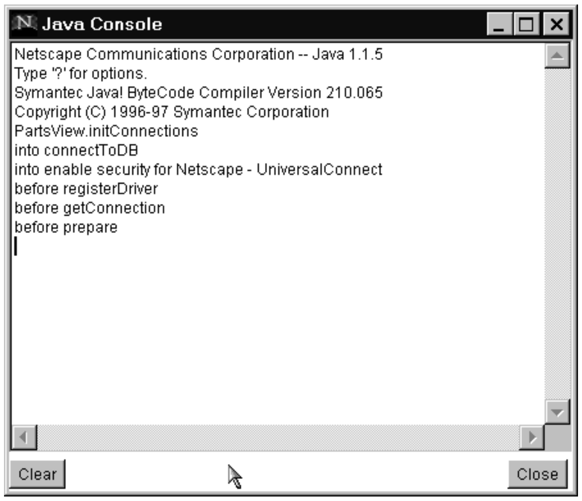


Figure 84. Opening the Netscape Java Console

A sample of the Java Console at runtime is shown in Figure 85 on page 111.



```
Netscape Communications Corporation -- Java 1.1.5
Type '?' for options.
Symantec Java! ByteCode Compiler Version 210.065
Copyright (C) 1996-97 Symantec Corporation
PartsView.initConnections
into connectToDB
into enable security for Netscape - UniversalConnect
before registerDriver
before getConnection
before prepare
```

Figure 85. Netscape Java Console when the PartsView Applet is Running

3.3.2.7 Opening the PartsView.html file

You need to load the `PartsView.html` file to start the applet. When you load the HTML file, the `APPLET` tag in the file is processed:

- The `CODE` parameter on the `APPLET` tag is used to identify the runnable class in the applet.
- The `ARCHIVE` parameter on the `APPLET` tag is used to identify the jar files that contain the applet classes and the AS/400 Toolbox for Java classes used in the applet.

Click on **File—>Open Page** to open the HTML file (see Figure 86). Use the file dialogs to navigate to the `c:\AppletTest` directory and select file `PartsView.html` within that directory.

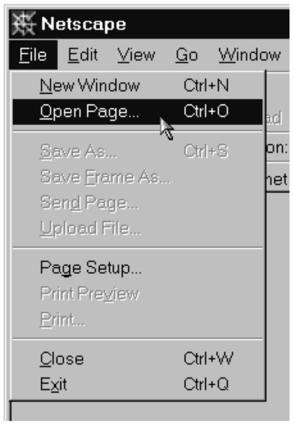


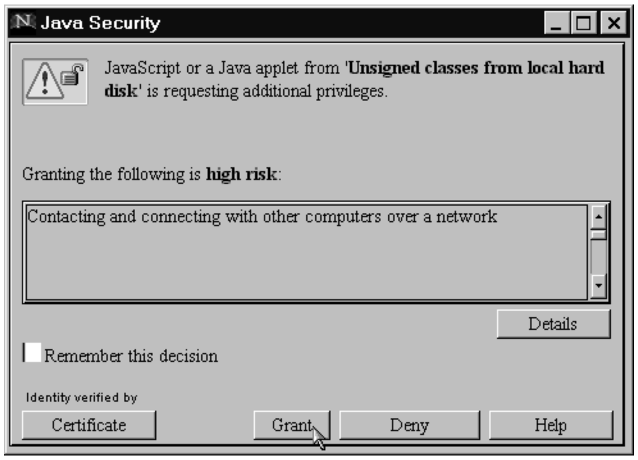
Figure 86. Opening the PartsView.html File in the Browser

Soon after opening the HTML file, you should see the status messages from the applet displayed in the Java console. It may take several seconds before you see the first message. The delay is because the Java Virtual Machine in the browser is analyzing the Java bytecodes from the applet and Toolbox classes.

3.3.2.8 Responding to the Netscape Security Prompt

Because the applet includes the `enablePrivilege` code in the `connectToDB` method (see Figure 75 on page 101), the browser displays the Java Security dialog (see Figure 87). You can accept or reject the applet's attempt to connect to the AS/400 system using the dialog.

To continue processing, click the **Grant** button.



JavaScript or a Java applet from '**Unsigned classes from local hard disk**' is requesting additional privileges.

Granting the following is **high risk**:

Contacting and connecting with other computers over a network

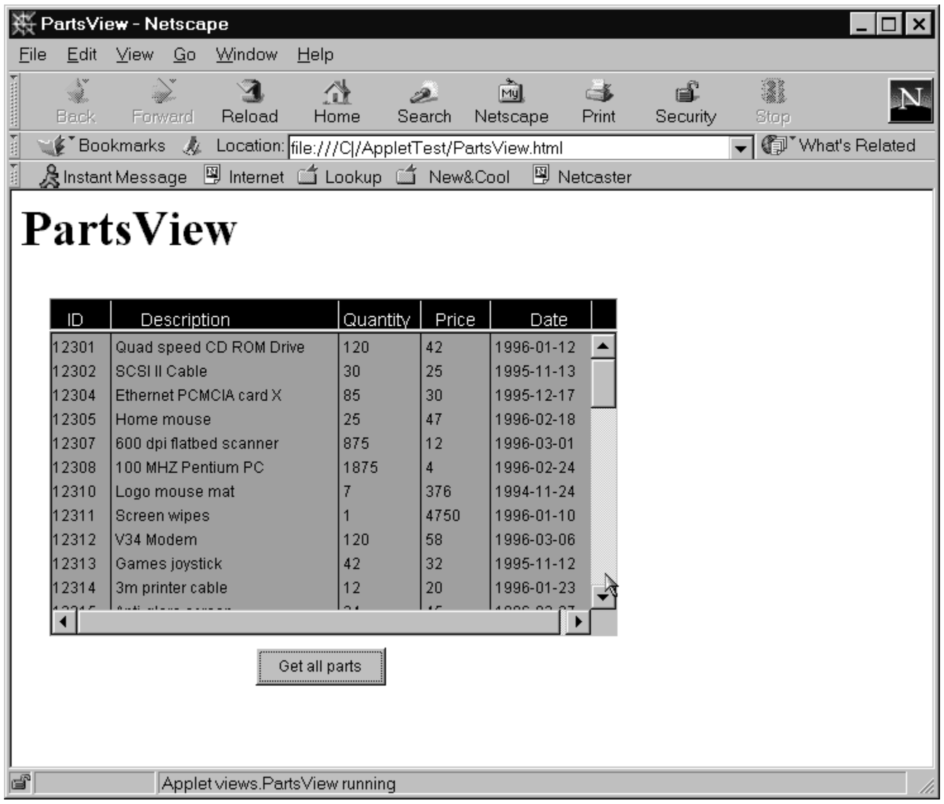
Remember this decision

Figure 87. Java Security Dialog

3.3.2.9 Working with the Applet

After granting authorization to the applet to connect to the AS/400 system, the applet runs the JDBC code to retrieve records from the AS/400 database. The records are displayed in the multi-column listbox in the applet.

Figure 88 on page 113 shows how the PartsView applet appears in the Netscape browser.


PartsView - Netscape
File Edit View Go Window Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: file:///C|/AppletTest/PartsView.html What's Related
Instant Message Internet Lookup New&Cool Netcaster
Get all parts
Applet views.PartsView running

PartsView

ID	Description	Quantity	Price	Date
12301	Quad speed CD ROM Drive	120	42	1996-01-12
12302	SCSI II Cable	30	25	1995-11-13
12304	Ethernet PCMCIA card X	85	30	1995-12-17
12305	Home mouse	25	47	1996-02-18
12307	600 dpi flatbed scanner	875	12	1996-03-01
12308	100 MHZ Pentium PC	1875	4	1996-02-24
12310	Logo mouse mat	7	376	1994-11-24
12311	Screen wipes	1	4750	1996-01-10
12312	V34 Modem	120	58	1996-03-06
12313	Games joystick	42	32	1995-11-12
12314	3m printer cable	12	20	1996-01-23

Figure 88. PartsView Applet in the Netscape Browser

3.3.2.10 Testing the Applet Using Microsoft Internet Explorer 4.01

If you have Microsoft Internet Explorer version 4.01 available, you should also test the applet using that browser. When you create an applet for general-purpose use, you need to test it extensively in all of the browsers that your applet's users may have. As you see, there are many deployment considerations you have to accommodate, based on the selection of browser used to work with the applet.

Note

The level of Microsoft Internet Explorer used in the tests described in this redbook is version 4.01 with Service Pack 1. If you are uncertain what version of Microsoft Internet Explorer you have, click on **Help—>About Internet Explorer** in the browser.

If you need to update your version of Microsoft Internet Explorer, open a connection to the Internet from your PC, select **Help—>Product Updates** in the browser. This takes you to a Microsoft update Web site where you can install the Service Pack and additional features of Microsoft Internet Explorer.

Before you open the HTML file to start the applet in Microsoft Internet Explorer, open the Java console. Click on **View—>Java Console**. You can now click **File—>Open** to open the HTML file in the browser.

Applet Failure

When you start the applet in Microsoft Internet Explorer, it fails with a `SecurityExceptionEx` exception. You get the failure first at the `enablePrivilege` code for the Netscape browser. You can ignore that failure. The more critical failure for the applet is the failure when the JDBC connection is attempted.

Netscape and Microsoft use different techniques to implement browser security for Java applets. With the Netscape browser, you can simply include the `enablePrivilege` code to indicate to the Java Virtual Machine (JVM) the types of privileges that your applet needs. For each requested privilege, a Java Security dialog (similar to Figure 87 on page 112) is displayed.

Microsoft implements applet security by using a *signed cabinet file*. The cabinet file is used instead of the jar file and is "signed" with a digital certificate. Digital Certificates are explained in greater detail in Section 8.2, “Digital Certificates and Certificate Authority” on page 267 of this redbook. When Microsoft Internet Explorer starts an applet in a signed cabinet file, the browser presents one security dialog where you can review all of the requested privileges.

The reason for the applet failure is to protect your PC from harmful actions that may be coded into applets that you download from the Internet. If the applet is allowed to freely access your PC, it can perform any number of actions that can damage data on your PC and compromise security.

3.3.3 Creating a Signed Cabinet File for Microsoft Internet Explorer 4.01

There are a number of steps you need to follow to create a signed cabinet file that can be used in Microsoft Internet Explorer. You need a signed cabinet file for either of the following conditions:

- The applet is installed on your PC and you do not set the `CLASSPATH` environment variable (see Section 3.3.4, “Using the CLASSPATH Environment Variable” on page 125). Since you have to explicitly set the `CLASSPATH` environment variable, the assumption is that if you did change it, you did intend to allow applets to run on your PC and have access to the classes identified in the `CLASSPATH`.
- The applet is in a cabinet file that is served from the IBM HTTP Server for AS/400. Since the cabinet file can be downloaded and installed on your PC as part of its invocation from a Web page, there needs to be some mechanism to let you accept or reject the applet from running. If you attempt to run an applet in an unsigned cabinet file, Microsoft Internet Explorer rejects the request.

3.3.3.1 Signed Cabinet Files at Runtime

When you open a Web page that loads an applet in a signed cabinet file, the Microsoft Internet Explorer JVM presents a Security Warning panel, similar to Figure 92 on page 119. You can review information about the applet and the signer in the tabs on the panel. Note the following points:

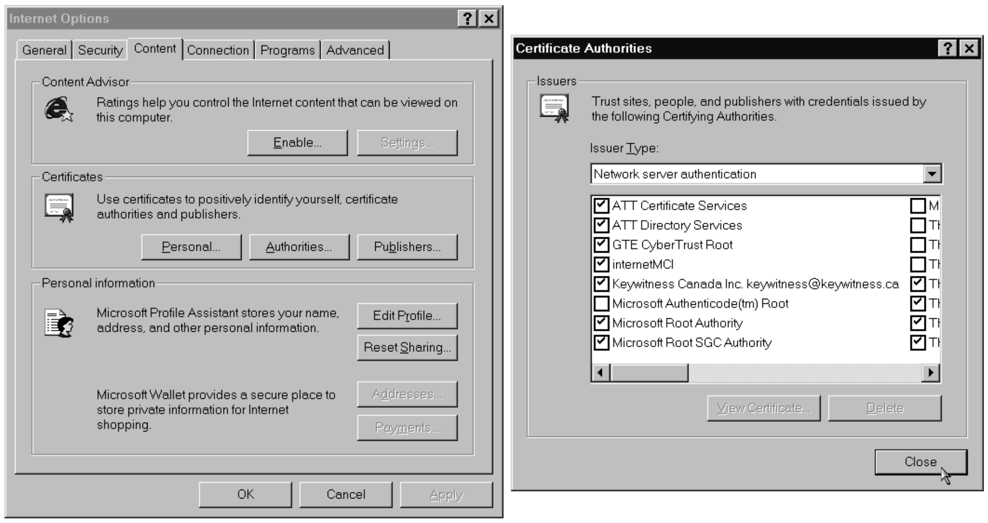
- If you recognize and trust the signer of the applet, you may choose to let it execute.
- If you do not recognize the signer, you should carefully review the permissions that the applet is requesting. Let the applet execute only if you are certain that it will not harm your system.

3.3.3.2 Digital Certificate Concepts

The signed cabinet technique is based on *digital certificates*. A *digital certificate* is an encrypted string of bytes that can be attached to a cabinet file.

If you intend to conduct e-business over the Internet, you need to obtain a certificate from a Certification Authority (CA). You apply to the CA of your choice and submit required information about your enterprise or organization. The CA examines the credentials you present and, for a fee, issues a digital certificate that is unique to you. The certificate can be used to identify your Web server and also to "sign" applets.

When a signed applet is loaded into the browser, the signature is compared with a list of known CAs that are configured as part of the browser. Figure 89 shows a list of CAs that are provided with Microsoft Internet Explorer. Click on **Views—>Internet Options** to open the dialog. Click the **Content** tab, then the **Authorities** button).



Ratings help you control the Internet content that can be viewed on this computer.

Use certificates to positively identify yourself, certificate authorities and publishers.

Microsoft Profile Assistant stores your name, address, and other personal information.

Microsoft Wallet provides a secure place to store private information for Internet shopping.

Trust sites, people, and publishers with credentials issued by the following Certifying Authorities.

Issuer Type: Network server authentication

☑ ATT Certificate Services	☐ M
☑ ATT Directory Services	☐ Th
☑ GTE CyberTrust Root	☐ Th
☑ internetMCI	☐ Th
☑ Keywitness Canada Inc. keywitness@keywitness.ca	☑ Th
☐ Microsoft Authenticode(tm) Root	☑ Th
☑ Microsoft Root Authority	☑ Th
☑ Microsoft Root SGC Authority	☑ Th

Figure 89. Microsoft Internet Explorer List of Certificate Authorities

If you do not intend to conduct public e-business or if you simply need to test signed applets, you can use tools provided with the Microsoft Java Software Development Kit (SDK) to generate test certificates. It is unlikely that you can use these "free" certificates for e-business and avoid paying a fee to a Certification Authority. As shown in Figure 92 on page 119, the Security Warning panel displays information about the certificate attached to a signed applet. If the Security Warning indicates that the applet was signed with a certificate issued from an unknown CA, it is unlikely that anyone will choose to allow the applet to execute in their browser.

3.3.3.3 Obtaining the Microsoft Java Software Development Kit

The programming tools you need to create signed cabinet files are contained in the Microsoft Java SDK. You can freely obtain the SDK from Microsoft's Web site at the following address: http://www.microsoft.com/java

Follow the links on that page to the Downloads section. At the time this redbook was written, Microsoft had three versions of the Java SDK available for download:

- 3.1
- 2.02
- 1.5.1

Unless you have specific reasons for working with the earlier versions, you should download version 3.1. The instructions in this redbook assume that you are working with the Microsoft Java SDK version 3.1.

The SDK is provided as a self-extracting EXE file. Simply run the EXE file to create the setup procedure. The setup procedure prompts you for the additional steps required to install the SDK on your PC.

3.3.3.4 Locating the Certificate Tools in the SDK

The programs you use to create a test digital certificate and sign your applet are in subdirectories of the SDK. Assuming that you install the Microsoft Java SDK to a directory named SDK, the tools are in the directory:

```
c:\sdk\Bin\PackSign
```

You should add that directory to your PC's `PATH` environment variable so that you can access the programs from any other directory on your PC. The programs you will use are:

- `makecert`—Used to create (make) a test digital certificate
- `cert2spc`—Used to convert the test certificate into a Software Publisher Certificate (SPC)
- `signcode`—Used to sign your applet with the SPC file
- `chkjava`—Used to display information about the signed applet

You need to consult Microsoft's documentation for a complete description of all of the options available with those programs. You can review documentation for the tools at the Microsoft Java SDK Web site or optionally download the documentation. The instructions in this redbook simply show the parameters to use for each of the programs.

3.3.3.5 Creating the Test Certificate

If you have the `makecert` and `cert2spc` tools available, you can create the test certificate that you will attach to the applet. Follow these steps to create the test certificate:

1. Open a MS-DOS Prompt window (Windows 95/98) or a Command Prompt window (Windows NT).
2. Set the `PATH` environment variable so that it includes the path to the SDK `PackSign` directory. You can add the SDK directory to the existing path by entering this command in the window:

   ```
   PATH=c:\sdk\Bin\PackSign;%PATH%
   ```

3. Change the directory in the window to `c:\AppletTest`.
4. Enter the `makecert` command and parameters as shown. This command creates a test certificate with a name and a test key.

   ```
   makecert -sk TestKey -n "CN=ITSO-Applet" TestCert.cer
   ```

5. Enter the `cert2spc` command and parameters as shown in the following example. This command converts the test certificate from makecert into a Software Publisher Certificate, which is required to sign the cabinet file that will contain the applet:

   ```
   cert2spc TestCert.cer TestCert.spc
   ```

6. Leave the window open. You return to the window to create the cabinet file and sign the cabinet file after exporting the applet in VisualAge for Java.

3.3.3.6 Adding Trace Statements to the connectToDB Method

To create the cabinet file, you need to export the applet packages from VisualAge for Java. However, there is a problem in the export process, in that not all of the referenced classes used in the applet are exported. This problem does not become evident until you try to run the applet from the signed cabinet file in the browser.

To help locate the missing classes, you can add trace statements to the Java source code in the applet's `connectToDB` method. The trace statements write to the Java Console when the applet is invoked. Using the trace statements, it is relatively easy to determine which of the referenced classes do not get exported. Without the trace statements, there is little debugging information available to help identify and resolve the problem.

Go to the `dataAccess.connectToDB` method in VisualAge for Java. Add the Trace statements as shown in Figure 90.

Note: Several lines of code that are already in the method are shown for reference to help you locate where to add the Trace statements.

```
//***********************************************************************
*****
// connectToDB method - connect to JDBC data source
//***********************************************************************
*****
public void connectToDB ( )   {

      System.out.println("into connectToDB");

      com.ibm.as400.access.Trace.setTraceErrorOn(true);
      com.ibm.as400.access.Trace.setTraceWarningOn(true);
      com.ibm.as400.access.Trace.setTraceDiagnosticOn(true);
      com.ibm.as400.access.Trace.setTraceInformationOn(true);
      com.ibm.as400.access.Trace.setTraceOn(true);


      //*****************************************************************
      ***********
      // Create a properties object for JDBC connection

      //*****************************************************************
      ***********
      Properties jdbcProperties = new Properties();
```

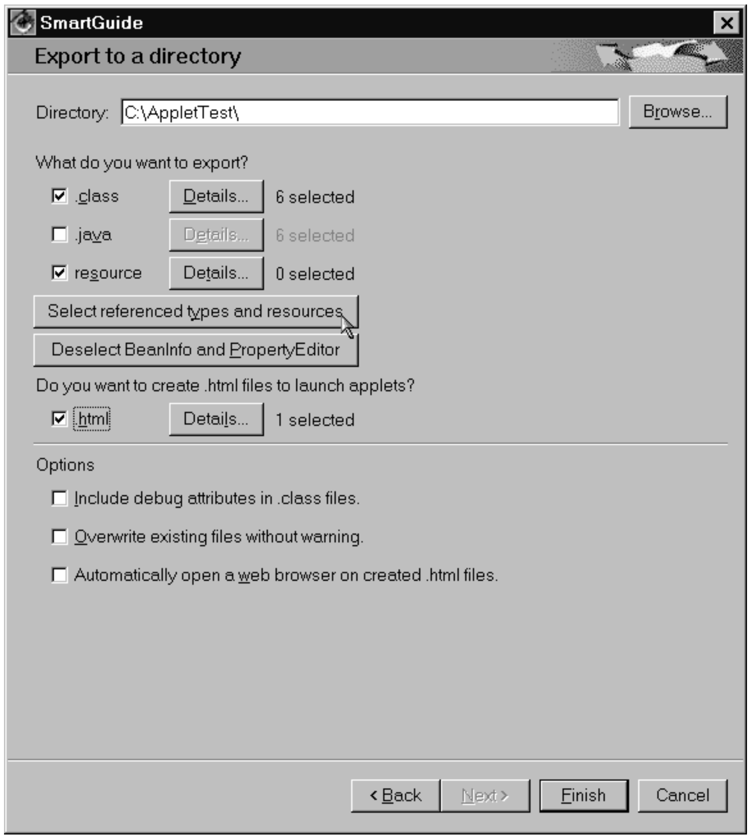
Figure 90. Enable Tracing of the Applet at Runtime

3.3.3.7 Exporting the Applet in VisualAge for Java

In Section 3.3.2.1, “Exporting Classes to the PC Drive” on page 106, you saw how to use the Export option in VisualAge for Java to export the packages used in the applet to a jar file. Follow the instructions in that section to start the export process again. Instead of selecting Jar file as the export destination (see Figure 82 on page 108), select **Directory** as the export destination.

Figure 91 shows the Export to a directory dialog. Follow these steps to complete the export of the applet to a directory:

1. Enter `c:\AppletTest\` for the directory.
2. Click the `.html` checkbox so that an HTML file is generated for the applet.
3. Click the **Select referenced types and resource** button so that VisualAge for Java gets the list of additional classes and resources used.
4. After generating the references, click the **Finish** button to write the classes and resources to the directory.


SmartGuide

Export to a directory

Directory: C:\AppletTest\

What do you want to export?

- [✓] .class — 6 selected
- [ ] .java — 6 selected
- [✓] resource — 0 selected

Do you want to create .html files to launch applets?

- [✓] .html — 1 selected

Options

- [ ] Include debug attributes in .class files.
- [ ] Overwrite existing files without warning.
- [ ] Automatically open a web browser on created .html files.

Browse...
Details...
Select referenced types and resources
Deselect BeanInfo and PropertyEditor
< Back
Next >
Finish
Cancel

Figure 91. Export to a Directory Dialog

3.3.3.8 Creating a Signed Cabinet File Based on the Applet Directories

Now that the classes and resources used in the applet are available in the `c:\AppletTest` directory and subdirectories, you can create a cabinet file of those files. A cabinet file is similar to a jar file, in that it is a compressed collection of other files and is intended to be transported in a network.

The reason why you export to a directory, rather than to a jar file, is so that the cabinet creation program has the correct files and directory structure to add to the cabinet. When Microsoft Internet Explorer opens the cabinet file, it locates classes and resources based on the directory information stored in the cabinet file. If you create a cabinet file based on a jar file, the browser cannot open the files correctly in the cabinet file.

Go back to the MS-DOS or Command Prompt window and follow these steps to create, sign, and test the cabinet file:

1. Verify that the current directory is `c:\AppletTest`.
2. Enter the `cabarc` command to create the cabinet file. Be sure you enter the `/p` and `/r` parameters, to preserve directory names and to recurse directories:

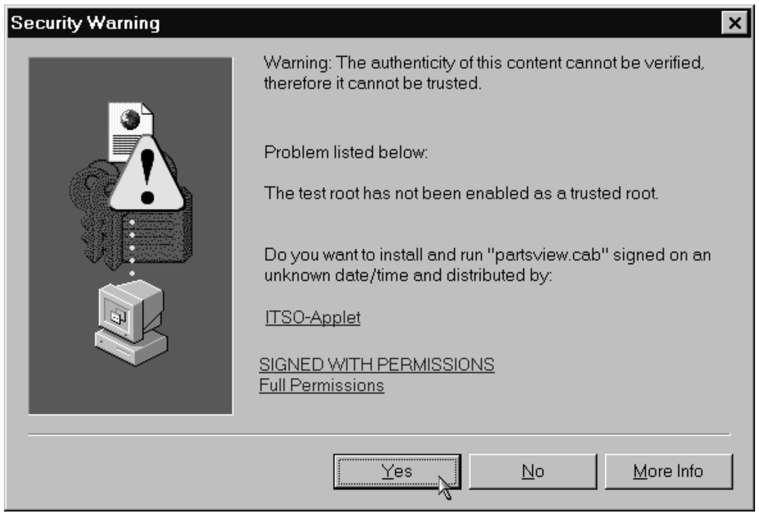
   ```
   cabarc /p /r n PartsView.cab *.*
   ```

3. Verify that file `PartsView.cab` is in the `c:\AppletTest` directory. If not, repeat the steps in this section.
4. Enter the `signcode` command to sign the cabinet file with the test certificate you generated in Section 3.3.3.5, “Creating the Test Certificate” on page 116. The `-jp low` parameter is used to assign a security level of "low" to the applet:

   ```
   signcode -j javasign.dll -jp low -spc TestCert.spc -k TestKey PartsView.cab
   ```

5. Enter the `chkjava` command to check the signature on the cabinet file. When you enter the command, the display in Figure 92 appears. You should spend a few minutes in the panel to examine all of the options used to identify and describe the signed applet.

   ```
   chkjava PartsView.cab
   ```



Warning: The authenticity of this content cannot be verified, therefore it cannot be trusted.

Problem listed below:

The test root has not been enabled as a trusted root.

Do you want to install and run "partsview.cab" signed on an unknown date/time and distributed by:

ITSO-Applet

SIGNED WITH PERMISSIONS
Full Permissions

Figure 92. The chkjava Command Security Warning Panel

3.3.3.9 Modifying the HTML File to Point to the Cabinet File

Open the `PartsView.html` file that VisualAge for Java generated in the `c:\AppletTest` directory. Add the `PARAM` statement to the HTML file so that it points to the cabinet file. Save the file in the `c:\AppletTest` directory.

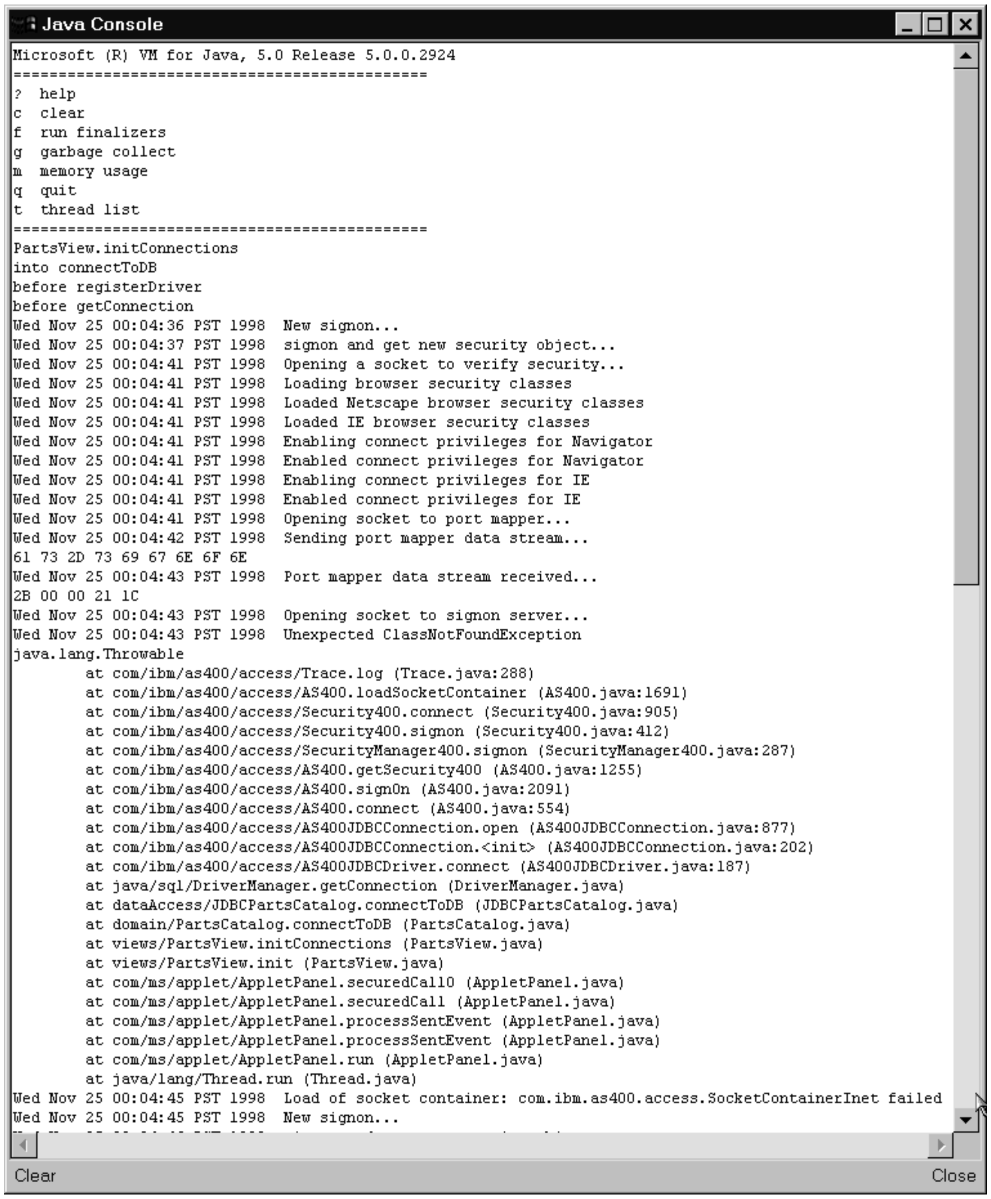
```
<HTML>
<HEAD>
<TITLE>PartsView</TITLE>
</HEAD>
<BODY>
<H1>PartsView</H1>
```

```
<APPLET CODE=views.PartsView.class
      WIDTH=700
      HEIGHT=350>
<PARAMNAME=cabbase
      VALUE=PartsView.cab>
</APPLET>
</BODY>
</HTML>
```

3.3.3.10 Testing the Cabinet File in Microsoft Internet Explorer

You can now test the signed cabinet file in Microsoft Internet Explorer. Follow these steps to run the test:

1. Start Microsoft Internet Explorer version 4.01 (SP1). Adjust the browser so that it does not take up the entire display space on your monitor.
2. Click on **View—>Java Console** to open the Java Console.
3. Click on **File—>Open** to open the file `c:\AppletTest\PartsView.html`
4. The Security Warning panel appears (see Figure 92 on page 119). Click **Yes** to allow the applet to run.
5. The applet fails because of the classes that are missing. Figure 93 on page 121 shows a portion of the Java Console with the trace messages written during execution of the applet. The next to last line in the figure identifies the missing class as `com.ibm.as400.access.SocketContainerInet`.
6. Close the Microsoft Internet Explorer browser. You need to add classes to the cabinet file to make the applet work correctly.



```
Microsoft (R) VM for Java, 5.0 Release 5.0.0.2924
==============================================
?  help
c  clear
f  run finalizers
g  garbage collect
m  memory usage
q  quit
t  thread list
==============================================
PartsView.initConnections
into connectToDB
before registerDriver
before getConnection
Wed Nov 25 00:04:36 PST 1998  New signon...
Wed Nov 25 00:04:37 PST 1998  signon and get new security object...
Wed Nov 25 00:04:41 PST 1998  Opening a socket to verify security...
Wed Nov 25 00:04:41 PST 1998  Loading browser security classes
Wed Nov 25 00:04:41 PST 1998  Loaded Netscape browser security classes
Wed Nov 25 00:04:41 PST 1998  Loaded IE browser security classes
Wed Nov 25 00:04:41 PST 1998  Enabling connect privileges for Navigator
Wed Nov 25 00:04:41 PST 1998  Enabled connect privileges for Navigator
Wed Nov 25 00:04:41 PST 1998  Enabling connect privileges for IE
Wed Nov 25 00:04:41 PST 1998  Enabled connect privileges for IE
Wed Nov 25 00:04:41 PST 1998  Opening socket to port mapper...
Wed Nov 25 00:04:42 PST 1998  Sending port mapper data stream...
61 73 2D 73 69 67 6E 6F 6E
Wed Nov 25 00:04:43 PST 1998  Port mapper data stream received...
2B 00 00 21 1C
Wed Nov 25 00:04:43 PST 1998  Opening socket to signon server...
Wed Nov 25 00:04:43 PST 1998  Unexpected ClassNotFoundException
java.lang.Throwable
        at com/ibm/as400/access/Trace.log (Trace.java:288)
        at com/ibm/as400/access/AS400.loadSocketContainer (AS400.java:1691)
        at com/ibm/as400/access/Security400.connect (Security400.java:905)
        at com/ibm/as400/access/Security400.signon (Security400.java:412)
        at com/ibm/as400/access/SecurityManager400.signon (SecurityManager400.java:287)
        at com/ibm/as400/access/AS400.getSecurity400 (AS400.java:1255)
        at com/ibm/as400/access/AS400.signOn (AS400.java:2091)
        at com/ibm/as400/access/AS400.connect (AS400.java:554)
        at com/ibm/as400/access/AS400JDBCConnection.open (AS400JDBCConnection.java:877)
        at com/ibm/as400/access/AS400JDBCConnection.<init> (AS400JDBCConnection.java:202)
        at com/ibm/as400/access/AS400JDBCDriver.connect (AS400JDBCDriver.java:187)
        at java/sql/DriverManager.getConnection (DriverManager.java)
        at dataAccess/JDBCPartsCatalog.connectToDB (JDBCPartsCatalog.java)
        at domain/PartsCatalog.connectToDB (PartsCatalog.java)
        at views/PartsView.initConnections (PartsView.java)
        at views/PartsView.init (PartsView.java)
        at com/ms/applet/AppletPanel.securedCall0 (AppletPanel.java)
        at com/ms/applet/AppletPanel.securedCall (AppletPanel.java)
        at com/ms/applet/AppletPanel.processSentEvent (AppletPanel.java)
        at com/ms/applet/AppletPanel.processSentEvent (AppletPanel.java)
        at com/ms/applet/AppletPanel.run (AppletPanel.java)
        at java/lang/Thread.run (Thread.java)
Wed Nov 25 00:04:45 PST 1998  Load of socket container: com.ibm.as400.access.SocketContainerInet failed
Wed Nov 25 00:04:45 PST 1998  New signon...
```

Figure 93. Java Console with the Trace Messages

3.3.3.11 Correcting the Missing Classes Problem

There are actually two missing classes in the applet:

- `com.ibm.as400.access.NLSImplRemote`
- `com.ibm.as400.access.SocketContainerInet`

To add the missing classes to the cabinet file, perform the following tasks:

1. Delete file `c:\AppletTest\PartsView.cab`.
2. Delete the following directories:

   ```
   c:\AppletTest\com
   c:\AppletTest\dataAccess
   c:\AppletTest\domain
   c:\AppletTest\netscape
   c:\AppletTest\views
   ```

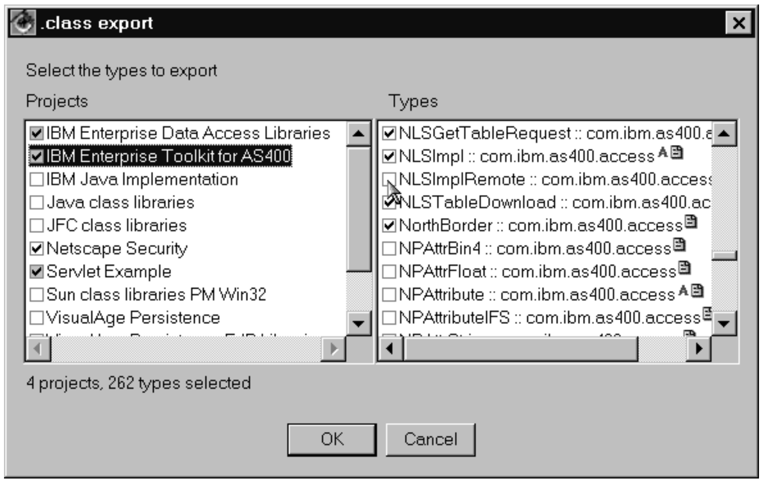
   You should leave these files in the `c:\AppletTest` directory:

   ```
   PartsView.html
   TestCert.cer
   TestCert.spc
   ```

   Note: Although you can simply export the two missing classes to the directory, you will start the entire export process over in these steps so that you can remove some of the unnecessary classes and resources that VisualAge for Java exported. By removing the unnecessary classes and resources, you reduce the size of the cabinet file. As with any files that are transmitted in a network, the smaller the file size is, the better the performance is.
3. Re-export the applet packages in VisualAge for Java.
4. On the Export to a directory panel (see Figure 91 on page 118), remove the check from the **.html** option.
5. Click the **Select referenced types and resource** button.

After VisualAge for Java generates its list of referenced classes and resources, manually add the two classes listed above:

1. Click the **Details** button that is to the right of the .class check box (see Figure 91 on page 118).
2. In the .class export dialog (Figure 94 on page 123), click the **IBM Enterprise Toolkit for AS/400** entry in the Projects list (on the left).
3. Scroll through the **Types** list (on the right) to the **NLSImplRemote** entry. Click in the box next to the name to add that class to the list of classes to be exported.
4. Continue scrolling through the Types list to the **SocketContainerInet** entry. Select that entry also.
5. Click the **OK** button to return to the Export to a directory dialog.



☑ IBM Enterprise Data Access Libraries
☑ IBM Enterprise Toolkit for AS400
☐ IBM Java Implementation
☐ Java class libraries
☐ JFC class libraries
☑ Netscape Security
☑ Servlet Example
☐ Sun class libraries PM Win32
☐ VisualAge Persistence

☑ NLSGetTableRequest :: com.ibm.as400.a
☑ NLSImpl :: com.ibm.as400.access
☐ NLSImplRemote :: com.ibm.as400.access
☑ NLSTableDownload :: com.ibm.as400.ac
☑ NorthBorder :: com.ibm.as400.access
☐ NPAttrBin4 :: com.ibm.as400.access
☐ NPAttrFloat :: com.ibm.as400.access
☐ NPAttribute :: com.ibm.as400.access
☐ NPAttributeIFS :: com.ibm.as400.access

4 projects, 262 types selected

Figure 94. Selecting the Missing Classes in the .class Export Dialog

3.3.3.12 Removing Unnecessary Classes and Resources

This step is optional, although as explained in the previous section, it helps reduce the size of the cabinet file.

When VisualAge for Java generates its list of referenced types and resources, it includes many hundreds of types and resources that are not used in the applet. Although it is difficult to identify every unneeded type or resource, there are some large groups that can be removed from the export list.

Follow these steps to remove some of the unnecessary types and resources:

1. Click the **Details** button next to the resource check box in the Export to a directory panel (see Figure 91 on page 118).
2. In the **Resource Export** dialog, expand the **IBM Enterprise Data Access Libraries** listing to its lowest level (`com/ibm/ivj/eab/dab`). See Figure 95 on page 124.
3. The list of resources in the `dab` branch are listed in the right list.
4. You can remove the check mark from any of the entries with the `.gif` extension. If there are several sequential `.gif` entries, you do not have to click each entry individually to remove the check mark. Instead, position the mouse pointer over the first `.gif` that you want to uncheck, left-click and hold, then move the mouse down through the list of `.gif` entries. The check mark is removed or added for each of the check boxes that the mouse moves over.
5. Remove the check for the `.gif` entries only. Leave the `.properties` entries checked.
6. Now expand the top-level listing for IBM Enterprise Toolkit for AS/400. The expansion of that item is shown in Figure 96 on page 124.
7. Uncheck the `vaccess` entry (`com/ibm/as400/vaccess`).
8. Uncheck the `ivj` entry (`com/ibm/ivj`). Unchecking `ivj` also unchecks `et400` and `examples`, which are under `ivj`.
9. Click the **OK** button to close the dialog.
10. When you return to the Export to a directory panel, you should notice that the count of types and resources to be exported is much less than before.

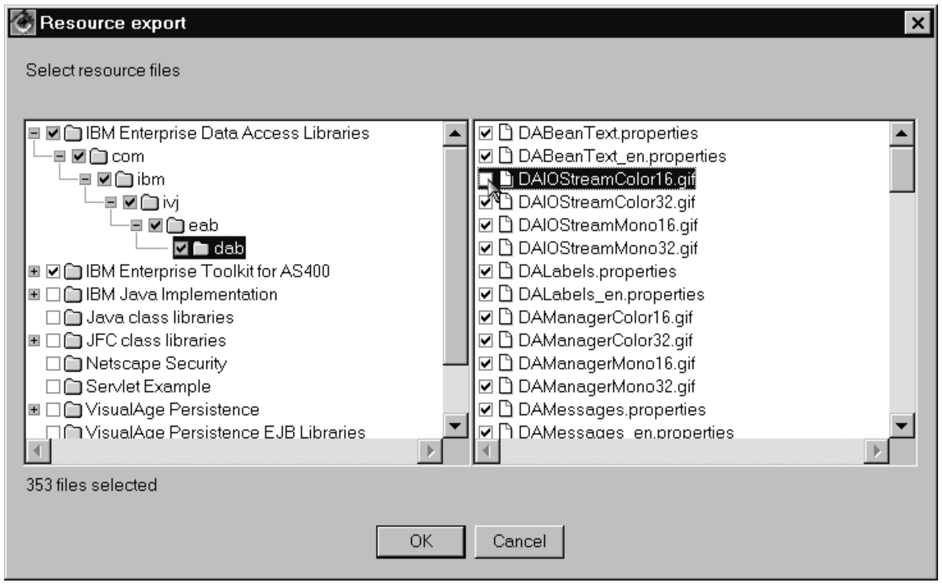


Figure 95. Removing Unwanted Entries from the Export List

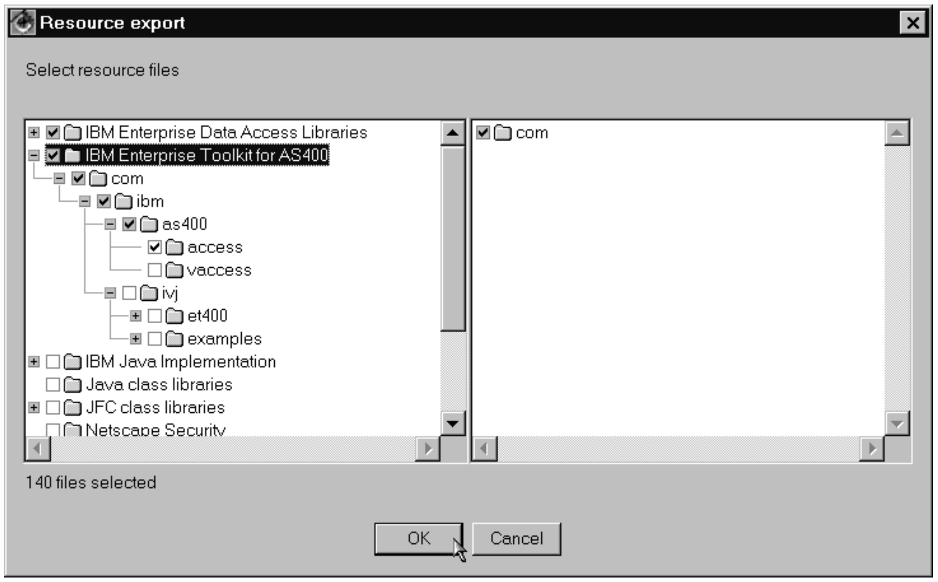


Figure 96. Resource Export Dialog

3.3.3.13 Completing the Export; Making the New Cabinet File

Now that you have manually added the two addition required classes and removed some of the unneeded resources, you can complete the export to the directories. Click the **Finish** button on the Export to a directory panel (see Figure 91 on page 118).

After the export is complete, go back to the MS-DOS or Command Prompt window. You need to run the steps shown in Section 3.3.3.8, “Creating a Signed Cabinet File Based on the Applet Directories” on page 118 again to create and sign the cabinet file.

After creating the signed cabinet file, start Microsoft Internet Explorer again and load the cabinet file, as described in Section 3.3.3.10, “Testing the Cabinet File in Microsoft Internet Explorer” on page 120.

3.3.4 Using the CLASSPATH Environment Variable

As an alternative to the Netscape `enablePrivilege` or Microsoft signed cabinet file, you can set your PC’s `CLASSPATH` environment variable to point to the location of the AS/400 Toolbox for Java. When the applet starts, it refers to the `CLASSPATH` to determine where the AS/400 Toolbox for Java is located. The AS/400 Toolbox for Java can be located on any of the following locations:

- Your PC
- A server in your network
- The AS/400 system IFS

In this example, you copy the AS/400 Toolbox for Java to your PC and set the `CLASSPATH` to point to the location on your PC.

Because you must manually set the `CLASSPATH` on your PC, the browser assumes that you are explicitly allowing applets hosted inside the browser to have access to all of the classes in the AS/400 Toolbox for Java. By using the `CLASSPATH`, you do not need to add a reference to the AS/400 Toolbox for Java to the `ARCHIVE` parameter of the `APPLET` tag in the HTML file.

3.3.4.1 Creating the JT400 Directory

In the tests, we created directory `c:\jt400` on the PC. This directory contains the AS/400 Toolbox for Java. By using this directory, every applet that is installed on the PC can use the common copy of the AS/400 Toolbox for Java contained in this directory.

3.3.4.2 Copying the jt400.zip File to the JT400 Directory

In Section 3.3.2.3, “Copying the jt400.jar File to the AppletTest Directory” on page 109, you copied file `jt400.jar` from the AS/400 system IFS to the applet test directory on your PC. You now need to copy the `jt400.zip` file from the IFS directory to the `c:\jt400` directory on your PC.

Use a Client Access for Windows 95/NT connection or the AS/400 system NetServer to get to the following IFS directory on your AS/400 system:

```
QIBM\ProdData\HTTP\Public\jt400\lib
```

This directory contains the `jt400.jar` and `jt400.zip` file. Copy the `jt400.zip` file to the `c:\jt400` directory.

Note

In Section 3.3.2.3, “Copying the jt400.jar File to the AppletTest Directory” on page 109, you copied file `jt400.jar`. For the CLASSPATH setting, you need to copy the `jt400.zip` file (the files contain equivalent Java classes).

If you use file `jt400.jar` instead of `jt400.zip` in the CLASSPATH, Netscape Navigator fails when it attempts to retrieve classes from the jar file.

3.3.4.3 Add/Modify CLASSPATH in AUTOEXEC.BAT (Windows 95/98)

If you are working with Windows 95 or Windows 98, you need to add or modify the CLASSPATH statement in file AUTOEXEC.BAT. Consider the following points:

- If you have VisualAge for Java installed on your PC, you already have a CLASSPATH statement in AUTOEXEC.BAT that was written when you installed VisualAge for Java. You need to modify that statement.
- If you do not have VisualAge for Java installed on your PC, add the CLASSPATH statement to file AUTOEXEC.BAT.

Use NotePad or another editor to open file `c:\autoexec.bat`.

- If there is already a CLASSPATH statement in the file, change it so that the first directory referenced is `jt400`. Leave all of the other paths as they were, following the new reference to `jt400`:

  ```
  CLASSPATH=c:\jt400\jt400.zip;...
  ```

- If there is no CLASSPATH statement in the file, add the following statement. It does not matter where it is located in the file:

  ```
  CLASSPATH=c:\jt400\jt400.zip;
  ```

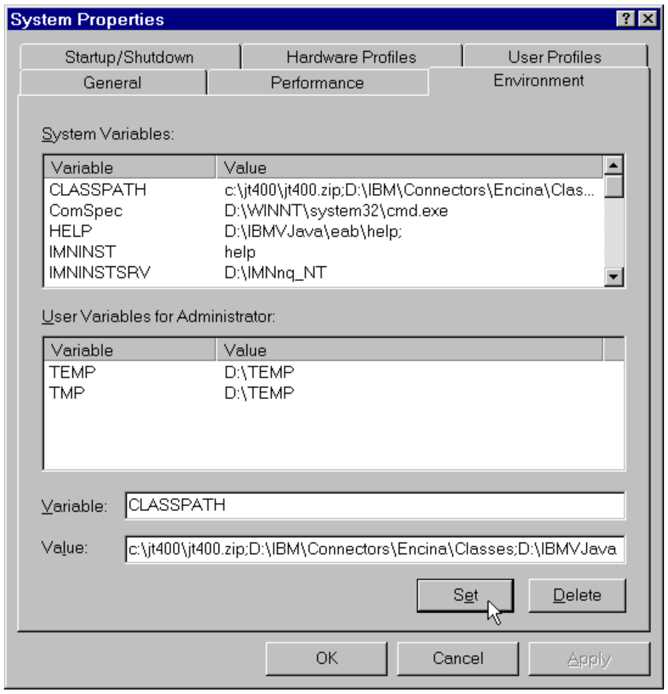
Save file `c:\autoexec.bat` after making the change. Reboot the PC so that the CLASSPATH change take affect, and set the environment variable.

3.3.4.4 Add/Modify CLASSPATH Environment Variable (Windows NT)

If you are working with Windows NT 4.0, set the CLASSPATH in the System Properties dialog. Follow these steps to set the CLASSPATH for a Windows NT 4.0 PC:

1. Right-click on the **My Computer** icon on the Windows NT 4.0 desktop.
2. Click the **Properties** item in the pop-up menu.
3. In the **System Properties** dialog, click the **Environment** tab.
4. If there is an existing CLASSPATH variable in the System Variables list, click to select and modify it. Add the new path `c:\jt400\jt400.zip` to the front of the CLASSPATH list (see Figure 97 on page 127).
5. If there is no CLASSPATH variable in the System Variables list, enter the variable name CLASSPATH and the value as `c:\jt400\jt400.zip;`
6. Click the **Set** button to set the value.
7. Click the **Apply** button.
8. Click the **OK** button to close the System Properties dialog.

You do not need to reboot Windows NT 4.0 for the change to take affect.



System Variables:

Variable	Value
CLASSPATH	c:\jt400\jt400.zip;D:\IBM\Connectors\Encina\Clas...
ComSpec	D:\WINNT\system32\cmd.exe
HELP	D:\IBMVJava\eab\help;
IMNINST	help
IMNINSTSRV	D:\IMNnq_NT

User Variables for Administrator:

Variable	Value
TEMP	D:\TEMP
TMP	D:\TEMP

Variable: CLASSPATH

Value: c:\jt400\jt400.zip;D:\IBM\Connectors\Encina\Classes;D:\IBMVJava

Figure 97. Setting the Windows NT 4.0 CLASSPATH Environment Variable

3.3.4.5 Commenting Out the EnablePrivilege Statement

Now that you have a CLASSPATH statement in affect, you can comment out the enablePrivilege statement in the connectToDB method. You can review the method in Figure 75 on page 101.

This step is optional. If you leave the enablePrivilege statement in the method, you get an exception in Microsoft Internet Explorer. However, the exception does not terminate the applet.

3.3.4.6 Changing the ARCHIVE Tag in the HTML File

Because the CLASSPATH now points to the location of the AS/400 Toolbox for Java, you no longer need the reference to the Toolbox in the ARCHIVE tag. To change the ARCHIVE tag, perform the following steps:

1. Use NotePad or another editor to open file c:\AppletTest\PartsView.html.
2. Change the line ARCHIVE=PartsView.jar so that it no longer includes the reference to jt400.jar. The only jar files that needs to be specified are PartsView.jar and dab.jar.
3. Save the changed HTML file to the c:\AppletTest directory.

```
<HTML>
<HEAD>
<TITLE>PartsView</TITLE>
</HEAD>
<BODY>
<H1>PartsView</H1>
        <APPLET CODE=views.PartsView.class
        ARCHIVE=PartsView.jar,dab.jar
        WIDTH=700
HEIGHT=350>
</APPLET>
</BODY>
</HTML>
```

3.3.4.7 Testing the Applet in Both Browsers

You should now test the applet in Netscape Navigator and Microsoft Internet Explorer. Remember to open the Java Console for both browsers before opening the HTML file.

The Netscape Navigator browser presents the Java Security dialog (see Figure 87 on page 112), even if you have commented out the `enablePrivilege` statement. Click the **Grant** button to continue the applet.

Microsoft Internet Explorer should now be able to run the applet. If you did not comment out the `enablePrivilege` statement, you see several security exception messages. You can ignore those, since the applet continues running.

3.3.5 Considerations for Using CLASSPATH

Using the `CLASSPATH` environment variable seems to be an "easy fix" to the problem of the applet failure in Microsoft Internet Explorer. Some other advantages of using `CLASSPATH` are that you do not have to add the `jt400.jar` reference to the `ARCHIVE` parameter of the `APPLET` tag. You can use the same `jt400.zip` file for all of your applets (you do not need to copy `jt400.jar` into each applet's directory).

However, you need to manually set the `CLASSPATH` for each PC where you want to use this technique. This is obviously not feasible if your applet is used by the general public. You may be able to use this technique in a corporate intranet environment.

You may also need to install file `jt400.zip` on each PC that uses the `CLASSPATH`. Again, this is probably not possible for publicly accessed applets, but it may be possible in your own environment.

If you are using applets in an intranet, you do not necessarily need to install `jt400.zip` on each PC. As an alternative, you can install `jt400.zip` on a server or point to the version of the file in the AS/400 system IFS. Using this technique, you do not have multiple versions of `jt400.zip` on different PCs. By pointing to a common version of the file, all users have access to the same version. It is much easier to maintain the common version rather than all of the copies.

The disadvantage of pointing the `CLASSPATH` to a server or the AS/400 system IFS is network performance. If you make extensive use of applets, you encounter significant network traffic as each applet loads classes from `jt400.zip` across the network. For infrequently used applets, the network traffic factor may not be of much concern. It would make sense in that case to locate `jt400.zip` on a server or the AS/400 system IFS.

In the next series of tests, you serve the applet from the IBM HTTP Server for AS/400. When you serve the applet from the Web server, you usually serve the AS/400 Toolbox for Java classes from the Web server.

3.3.6 Serving Applets from the HTTP Server for AS/400

Up to this point, you have worked with the applet on your PC. Even though the applet uses JDBC to work the AS/400 system database, there is nothing in the deployment of the applet that requires the services of the HTTP Server for AS/400.

In this part, you see what is required to configure your HTTP Server for AS/400 to serve applets from the AS/400 system. Serving applets from the AS/400 system is actually easier to implement than serving applets from your PC. You do not need to create signed cabinet files for Microsoft Internet Explorer, nor do you need to set the `CLASSPATH` environment variable. There are a few configuration steps you need to take to allow the applet to be served from the HTTP Server for AS/400.

Your main deployment consideration is performance. As you may have noticed, the jar and cabinet files that you created on the PC are hundreds or thousands of kilobytes in size. If you are using applets in an intranet, the applet file sizes are of some concern. They are not extraordinary in terms of files that are sent through a network.

If you are using the applet for public access to your HTTP Server for AS/400, for example, to provide support for dial-in users to access your AS/400 system, the applet files may be too large to be practical.

This part of the redbook describes two scenarios for serving applets from the HTTP Server for AS/400.

3.3.6.1 Applet and Supporting Files on the AS/400 System

For this method, the entire applet and all supporting jar files are served from the AS/400 system. This is the easiest technique to use to deploy the applet, but also the most expensive in terms of network performance and response time in the browser.

3.3.6.2 Applet on the AS/400 System; jt400.zip on the PC

The code for the applet is served from a jar file on the AS/400 system, and the `jt400.zip` file is located on the PC. This method requires that the `CLASSPATH` environment variable be set on the PC to point to the local copy of the `jt400.zip` file. You also need to develop a plan to update `jt400.zip` as changes are made to it by IBM.

This method is probably impractical for public deployment because of the requirement to distribute the `jt400.zip` file and set the `CLASSPATH`. It may be the best solution for intranet or extranet applets. An *extranet* is an extension of an intranet, which usually includes other companies or organizations with which you have an on-going relationship.

When used appropriately, this method offers a good compromise between ease of deployment and performance. As you will see, the size of the jar file for the applet itself is quite small. By moving `jt400.zip` to the PC, you avoid the necessity of transmitting it to the browser every time the applet is invoked.

3.3.6.3 Configuring the AS/400 System for Applet Serving

To test serving applets from the AS/400 system, there are several steps you need to take to configure the AS/400 system. After following the steps, you will use VisualAge for Java to export the applet to the AS/400 system.

Creating a Directory to Contain the Applet

Although you can use any of the existing directories in the AS/400 system Integrate File System (IFS), you may find it easier to create a new directory for testing. For our tests, we created directory apptest as an IFS root directory (not contained in any other directories).

You can use the OS/400 command Create Directory (`CRTDIR`) to create the directory. Enter the following command on a 5250 display:

```
CRTDIR DIR('apptest')
```

Configuring the HTTP Server for AS/400 to Serve from Apptest

The applet jar files and the HTML file that invokes the applet are located in directory `apptest`. To load the HTML file in a browser using the HTTP protocol, you need to configure the HTTP Server for AS/400 so that it can serve Web pages from the `apptest` directory. For details about how to configure the IBM HTTP Server for AS/400, see Section 9.3, “IBM HTTP Server for AS/400” on page 293.

3.3.6.4 Copying the jt400.jar File to the apptest Directory

If you have a Client Access for Windows 95/NT connection to the AS/400 system, use the Windows 95/NT Explorer program. You can also choose to use the AS/400 NetServer in the Windows 95/NT Explorer to copy file `jt400.jar` to the apptest directory.

Copy file `jt400.jar` from this directory on your AS/400 system IFS:

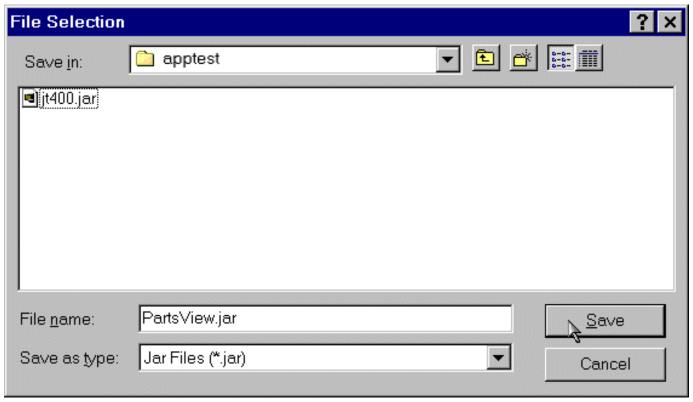
```
\QIBM\ProdData\HTTP\Public\jt400\lib\jt400.jar
```

Copy file `dab.jar` to the apptest directory located in the AS/400 integrated file system.

3.3.6.5 Exporting the Applet from VisualAge for Java

You can now go into VisualAge for Java and export the applet as a jar file to the AS/400 `apptest` directory. Follow the instructions in Section 3.3.2.1, “Exporting Classes to the PC Drive” on page 106. When you get to Section 3.3.2.2, “Specifying Export Options” on page 108, click the **Browse** button on the Export to a jar file dialog (see Figure 83 on page 109) and navigate to the `apptest` directory on your AS/400 system.

Be certain that the name of the jar file to export to (`PartsView.jar`) includes the `.jar` extension, as shown in Figure 98 on page 131. The export process does not append the extension, even though it suggests the extension in the File Selection dialog.



File name: PartsView.jar

Save as type: Jar Files (*.jar)

Figure 98. Setting the Name of the jar File

On the Export to a jar file dialog (see Figure 83 on page 109), be sure the .html option is checked, so that VisualAge for Java generates the HTML file required to serve the applet. Do not click the *Select referenced types and resources* button, since you will use the `jt400.jar` file to provide the required AS/400 Toolbox for Java classes and resources.

Click the **Finish** button on the Export to a jar file dialog. When the export is done, use the Windows 95/NT Explorer to verify that file `PartsView.jar` and `PartsView.html` are in the `apptest` directory on your AS/400 system.

3.3.6.6 Adding the ARCHIVE Parameter to the HTML File

You need to modify the generated HTML file to add the `ARCHIVE` parameter to the `APPLET` tag. The `ARCHIVE` parameter includes the references to the `PartsView.jar` `dab.jar` and `jt400.jar` files.

Follow the instructions in Section 3.3.2.5, “Adding the ARCHIVE Parameter to the HTML file” on page 110. Be certain that you edit file `PartsView.html` from the `apptest` directory on the AS/400 system, and not the `.html` file on your PC.

3.3.6.7 Loading the HTML file in the Browser

You can now start either the Netscape Navigator or the Microsoft Internet Explorer browser. After starting the browser, open the Java Console so that you can follow the progress of the applet.

Instead of using the browser `File-->Open` command to load the HTML file, enter a URL to load the Web page. Enter the URL as shown in the following example. Substitute the name of your AS/400 system as known in your TCP/IP network, or enter your AS/400 system IP address in place of system name `AS400ABC`:

```
http://AS400ABC/apptest/PartsView.html
```

Note

The `apptest` string in the URL is case-sensitive. You **must** enter apptest exactly as specified in the `PASS` directive (see Section 3.3.6.3, “Configuring the AS/400 System for Applet Serving” on page 130).

If you want to allow various cases for the directory (for example, `apptest` and `Apptest`), you need to enter additional `PASS` directives for each variation.

If the browser responds with a 403 error message, check the entire URL carefully. The 403 message is usually sent when you mistype any portion of the URL.

As the applet is loaded from the AS/400 HTTP Server, you may notice that the start-up time is longer than when you loaded the applet from your PC. This is because the browser has to load the `jt400.jar` file from the AS/400 HTTP Server in addition to the start-up time associated with the browser’s JVM.

3.3.6.8 Using jt400.zip on the PC, PartsView.jar on the AS/400 System

Use the following steps to implement the applet serving as described in Section 3.3.6.2, “Applet on the AS/400 System; jt400.zip on the PC” on page 129. If you have followed all of the other steps in this chapter, you have essentially already used this method. The only differences are that the `PartsView.jar` file is located on the AS/400 system and the HTTP Server for AS/400 is used to provide the HTML file to the browser.

Follow these steps to implement this technique:

1. Verify or create directory `JT400` on your PC (see Section 3.3.4.1, “Creating the JT400 Directory” on page 125).
2. Copy the `jt400.zip` file to the `JT400` directory on your PC (see Section 3.3.4.2, “Copying the jt400.zip File to the JT400 Directory” on page 125).
3. Set the `CLASSPATH` environment variable and reboot your PC (Windows 95/98) (see Section 3.3.4.3, “Add/Modify CLASSPATH in AUTOEXEC.BAT (Windows 95/98)” on page 126 and Section 3.3.4.4, “Add/Modify CLASSPATH Environment Variable (Windows NT)” on page 126).
4. Modify the HTML file in the `apptest` directory on the AS/400 system (see Section 3.3.4.6, “Changing the ARCHIVE Tag in the HTML File” on page 127).

You can now load the HTML file using the URL shown in Section 3.3.6.7, “Loading the HTML file in the Browser” on page 131. This time, you should notice that the applet loads about as quickly as when you deployed it on the PC. This is because the `jt400.zip` file is now available to the browser locally. The download time for the `PartsView.jar` file is trivial. In our tests, that file was only 8KB in size.

This technique provides a good combination of download time with ease of maintenance of the applet itself. If you need to make corrections or add features to the applet, you only need to update the applet’s jar file on the AS/400 system, not on all of the PCs. However, you must balance this against the requirement to initially load (and subsequently maintain) the `jt400.zip` file on each PC, and also the requirement to set the PCs `CLASSPATH` environment variable.

3.4 Working with the Sun Java Plug-in

Throughout this chapter, you may have noticed that it is far easier to work with the Java source code for the applet than it is to understand all of the issues surrounding the actual deployment of the applet. Getting your applet to run in different browsers or different versions of the same browser can be incredibly difficult. Not only do the browsers work with different file types (jars or cabinets), but they are also sensitive to the AS/400 Toolbox for Java files (jar or zip). Also, there are inconsistencies in the Java runtime environments implemented in the browsers that you must be aware of and for which you may have to write "work-around" code.

The easiest solution for applet deployment is to mandate that all users of your applet use a specific browser, at a specific release level with whatever required service packs, patches, or plug-ins you require. That obviously rules out deployment of your applet over the Internet, since users who choose not to conform to your requirements may simply take their browser (and their business) elsewhere.

In an intranet environment, you may be able to constrain a small user community to a certain browser environment. If you do not "lock down" your user’s configurations, you will soon have "browser creep", with different versions of browsers in your intranet.

Sun Microsystems, Inc., who has a great interest in the advancement of Java as a corporate platform, has developed what may be your best choice for applet deployment. You can freely obtain the Sun Java Plug-in, which provides a common Java runtime environment for the Internet Explorer and Netscape Navigator series of browsers. Although there is additional setup work that you need to do to implement the Java Plug-in, you will may find it easier to use the Java Plug-in, rather than attempt to resolve all of the problems you encounter trying to support the different browsers.

3.4.1 Java Plug-in Basics

The Java Plug-in is provided in a file that you download from one of Sun Microsystems, Inc. Web sites. Upon installing the plug-in on a PC, you have the following options are available:

- A Java Plug-in Properties control panel lets you indicate how the plug-in is to behave when invoked. You can specify which Java runtime environment is to be used, if you have more than one installed on your PC.
- For Microsoft Internet Explorer, an ActiveX object provides a link to the selected Java runtime environment.
- For Netscape Navigator, a plug-in provides a link to the selected Java runtime environment.

You control the invocation of the Java Plug-in in the HTML for the Web page that contains the applet. Because the required changes to the HTML can be rather complicated, Sun Microsystems, Inc. also provides a no-charge HTML Converter that you can use to convert your existing HTML file to a version that invokes the Java Plug-in for either Microsoft Internet Explorer or Netscape Navigator.

Table 12 shows the operating systems and browsers supported by the Java Plug-in.

Table 12. Operating Systems and Browser Versions Supported by the Java Plug-in

	Internet Explorer		**Netscape Navigator**	
Operating System	**3.02**	**4.x**	**3.x**	**4.x**
Windows 95	Yes	Yes	Yes	Yes
Windows 98		Yes	Yes	Yes
Windows NT 4.0	Yes	Yes	Yes	Yes
Solaris/SPARC			Yes	Yes
Solaris/x86			Yes	

3.4.2 Working with the Java Plug-in—A Step-by-Step Approach

The best way to learn about the capabilities and deployment considerations for the Java Plug-in is to work through a complete example, using both Internet Explorer and Netscape Navigator. In the following sections, you learn how to:

- Obtain the required programs for the Java Plug-in from Sun Microsystems, Inc.
- Convert the HTML file to invoke the Java Plug-in
- Install the Java Plug-in on the AS/400 system so that the Java Plug-in can be installed as part of the applet invocation
- Customize the Java Plug-in install process for Microsoft Internet Explorer and Netscape Navigator
- Test and verify the operation of the Java Plug-in in both browsers

3.4.2.1 Downloading the Java Plug-in HTML Converter

To understand why you should get the Java Plug-in HTML Converter ("the Converter" from this point on), review both the original `PartsView.html` file and a converted version of `PartsView.html` that invokes the Java Plug-in when the applet is loaded.

The following example shows the original `PartsView.html` code before it is converted to invoke the Java Plug-in.

```
<HTML>
<HEAD>
<TITLE>PartsView</TITLE>
</HEAD>
<BODY>
<H1>PartsView</H1>
<APPLET CODE=views.PartsView.class
   ARCHIVE=PartsView.jar
   WIDTH=700
   HEIGHT=350>
</APPLET>
</BODY>
</HTML>
```

Figure 99. PartsView.html Code before Conversion for the Java Plug-in

The following example shows the converted `PartsView.html` file with the code required to invoke the Java Plug-in in either Microsoft Internet Explorer or Netscape Navigator.

```
<HTML>
<HEAD>
   <TITLE>PartsView</TITLE>
</HEAD>

<BODY>
   <H1>PartsView</H1>

   <!--"CONVERTED_APPLET"-->
   <!-- CONVERTER VERSION 1.0 -->

   <OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
      WIDTH=700
      HEIGHT=350
      codebase="http://java.sun.com/products/plugin/1.1.1/jinstall-111-win32.c
      ab#Version=1,1,1,0">
   <PARAM NAME=CODE    VALUE=views.PartsView.class >
   <PARAM NAME=ARCHIVE VALUE=PartsView.jar >
   <PARAM NAME="type"  VALUE="application/x-java-applet;version=1.1">

   <COMMENT>
   <EMBED type="application/x-java-applet;version=1.1"
      java_CODE=views.PartsView.class
      java_ARCHIVE=PartsView.jar
      WIDTH=700
      HEIGHT=350
      pluginspage="http://java.sun.com/products/plugin/1.1.1/plugin-install.ht
      ml">
   <NOEMBED>
   </COMMENT>
   </NOEMBED>
   </EMBED>
   </OBJECT>

   <!--
   <APPLET CODE=views.PartsView.class
      ARCHIVE=PartsView.jar
      WIDTH=700
      HEIGHT=350>
   </APPLET>
   -->
   <!--"END_CONVERTED_APPLET"-->

</BODY>
</HTML>
```

Figure 100. PartsView.html File after Conversion for Use with the Java Plug-in

The Converter takes as input your original HTML file (for example, the file shown in Figure 99 on page 134) and outputs the version shown in Figure 100. Although you can certainly create the code shown in the converted file, it should be obvious that using the Converter is preferred to hand-coding the Java Plug-in enabled HTML file.

The Java Plug-in HTML Converter is available at the following Web site:
http://java.sun.com/products/plugin/converter.html

The file to download for Windows 95/NT is `htmlconv111-win32.exe`. You can download the file to a temporary directory on your PC. The file that we downloaded for testing was 3367KB in size.

Note

Please note that the URLs shown for the Java Plug-in are provided by Sun Microsystems, Inc. or its Javasoft subsidiary, and are not controlled by IBM. As with any URL, the URL, Web pages available, or downloads available from a particular page are subject to change.

If any of the URLs given for the Java Plug-in do not take you to the referenced Web page, try a less-specific version of the URL. For example, the following URL may not work:

```
http://java.sun.com/products/plugin/converter.html
```

If not, enter this URL instead:

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

You may find links on the less-specific Web page that take you to the referenced Web page.

If these options fail, start at the top-level URLs and look for links to the lower level pages from there. Two top-level URLs that you can use are:

```
http://java.sun.com
http://www.javasoft.com
```

3.4.2.2 Installing the Java Plug-in HTML Converter

After downloading the Java Plug-in HTML Converter file, you must run it to launch the installation process. When you run it, a conventional installation process is started, which leads you through a series of prompts.

The only significant prompt is the selection of a Java VM for the use of the Converter (Figure 101 on page 137). Unless you have specific reasons for selecting another Java VM, select the default option and let the Converter install a JavaSoft Java runtime environment (JRE) as shown in Figure 101 on page 137.

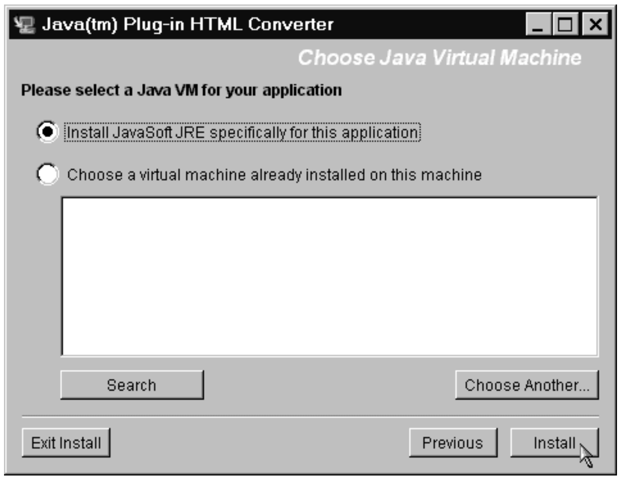

Java(tm) Plug-in HTML Converter
Choose Java Virtual Machine
Please select a Java VM for your application
Install JavaSoft JRE specifically for this application
Choose a virtual machine already installed on this machine
Search
Choose Another...
Exit Install
Previous
Install

Figure 101. Java Plug-in HTML Converter Options

When you install the Java Plug-in HTML Converter, you are also given several choices to start the Converter. For example, you can add an item to a new or existing program group or create a shortcut on the WIndows desktop. It does not matter which of the choices you select, as long as you can locate the Converter so that you can start it.

3.4.2.3 Running the Java Plug-in HTML Converter for PartsView.html

You have already seen the sample input to the Converter (Figure 99 on page 134) and the sample output of the Converter (Figure 100 on page 135). In this part, you review how to run the Converter and the options that are available to you.

When you start the Converter, you work with the dialog shown in Figure 102 on page 138. In the upper half of the dialog, you have the following choices:

- If you have multiple HTML files to convert, select the **All Files in Folder** option. Then, specify the folder, file extensions, and if you want to include subfolders.
- If you have just one HTML file to convert, select the **One File** option. Then, specify the complete path and file name of that HTML file.

For the test, select **One File** and specify the path to file `PartsView.html` in the `apptest` directory on your AS/400 system.

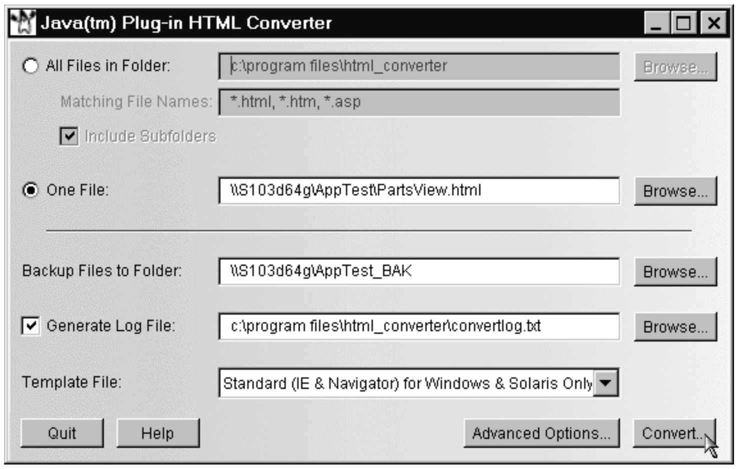


Figure 102. Java Plug-in HTML Converter Dialog

Note

In our tests, the AS/400 system name **S103D64G** is a test system. In all cases where you see that system name S103D64G, **substitute** the name of **your AS/400 system**.

As soon as you specify the HTML file to be converted, the Converter proposes a directory to use for a backup copy of the HTML file (the **Backup Files to Folder** option in the lower half of the dialog). You can change the proposed directory if you want, but you should not make the backup directory the same as the directory where the original HTML file is located.

The backup process works as follows:

- When you start the conversion, the backup directory is created if it does not already exist.
- If there *is not* a file in the backup directory with the same name as the HTML file being converted, the HTML file is copied from the originating directory to the backup directory. When the conversion is completed, the converted file is in the originating directory.
- If there *is* a file in the backup directory with the same name as the HTML file being converted, the HTML file in the originating directory *is not* copied to the backup directory. When the conversion is completed, the converted file is in the originating directory as `temp.conv`. The original HTML file remains in the originating directory and is not altered.

In other words, your original HTML file is always maintained. If you specify a valid backup directory, the original HTML file is copied there if possible. If you do not specify a valid backup directory or if the HTML file cannot be copied (because of duplication), the original HTML file is kept in the originating directory. You then have to manually rename the `temp.conv` file, which is the output of the Converter.

Note

We noticed in our testing that the automatic backup to a directory does not seem to work if the originating directory is mapped to a network drive. For example, the network path `\\s103d64g\apptest` was mapped to drive letter `G:`

The automatically proposed backup directory was `G:_BAK`. Upon completing the conversion, directory `_BAK` was located in the `apptest` directory on the AS/400 system. However, the original HTML file was not copied into the `_BAK` directory. Instead, the original HTML file remained in the `apptest` directory and the converted file was available as `temp.conv`.

Suggestion: Specify the full network path to be used for the backup directory. Do not use a mapped network drive.

The **Generate Log File** option is of most used when you convert multiple HTML files. Figure 103 shows a sample of the log file data. The summary information is also displayed in a summary panel when you run the Converter.

```
Applet Conversion November 28, 1998 9:14:51 PM PST
\\S103d64g\apptest\PartsView.html  Processing...Done  Applets Found:  1
All Done  Files Processed:  1  Applets Found:  1
```

Figure 103. Sample Output of the Generate Log File

The **Template File** option lets you select which browsers you want to generate Java Plug-in compatible code for. The Converter uses template files to create the required additional HTML. The templates provided with the Converter work with the following targets:

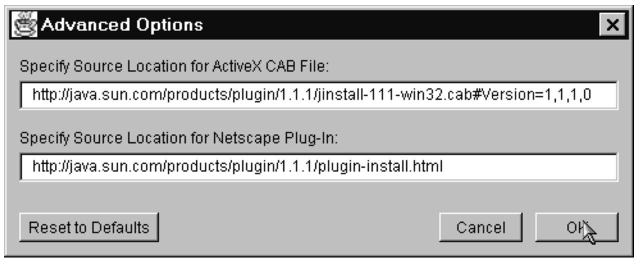
- **Standard (IE & Navigator) for Windows and Solaris only**—Generates HTML that will be processed by Microsoft Internet Explorer and Netscape Navigator browsers on Windows and Sun Solaris platforms.
- **Extended (Standard + All Browsers/Platforms)**—Generates HTML that will be processed by the standard browsers and other browsers. The generated HTML includes JavaScript that uses browser detection to determine which browser opened the HTML file. However, because a user may disable JavaScript in their browser, the generated HTML from this template may fail to load the Java Plug-in. The HTML generated by the Standard template (above) does not include any JavaScript for browser detection. Standard is the preferred template unless you know that you need to support other browsers or platforms.
- **Internet Explorer for Windows and Solaris only**—Generates the HTML that will be processed by Microsoft Internet Explorer. The "Solaris" reference seems to be an error in the selection list in the Converter, since Sun Microsystems, Inc. does not support the Java Plug-in for Microsoft Internet Explorer on the Solaris platform, as shown in Table 12 on page 134. See the description of the next template.
- **Navigator for Windows only**—Generates the HTML that will be processed by Netscape Navigator. This template includes the Solaris platform that was mistakenly coded in the previous template.

- **Other Template**—You can also create a customized template. When you install the Converter, there is a `readme.txt` file in the program directory where the Converter program is located. That file contains information about the variables you can use in a template file and additional information about creating a custom template.

The Advanced Options Button

If you click the Advanced Options button on the Java Plug-in HTML Converter dialog (Figure 102 on page 138), the Advanced Options dialog is displayed (Figure 104). You use this dialog to specify what action Microsoft Internet Explorer or Netscape Navigator should take when an HTML file that requires the Java Plug-in is loaded.

Leave the settings on this dialog as they are for now. You return to this dialog later to change the settings for use in your intranet environment.



Specify Source Location for ActiveX CAB File:
http://java.sun.com/products/plugin/1.1.1/jinstall-111-win32.cab#Version=1,1,1,0
Specify Source Location for Netscape Plug-In:
http://java.sun.com/products/plugin/1.1.1/plugin-install.html

Figure 104. Advanced Options Dialog

Running the Converter

If you have not already done so, run the Converter now to convert the `PartsView.html` file. You should end up with these versions:

- A backup version of `PartsView.html` in the directory specified in the **Backup Files to Folder** directory.
- A converted version of `PartsView.html` in the `apptest` directory on the AS/400 system.

3.4.2.4 Opening the Converted PartsView.html in Internet Explorer

This part assumes that you have Microsoft Internet Explorer version 3.02, 4.0, or 4.01 installed on your computer. If you do not, go to the next step and open the file in the Netscape Navigator browser.

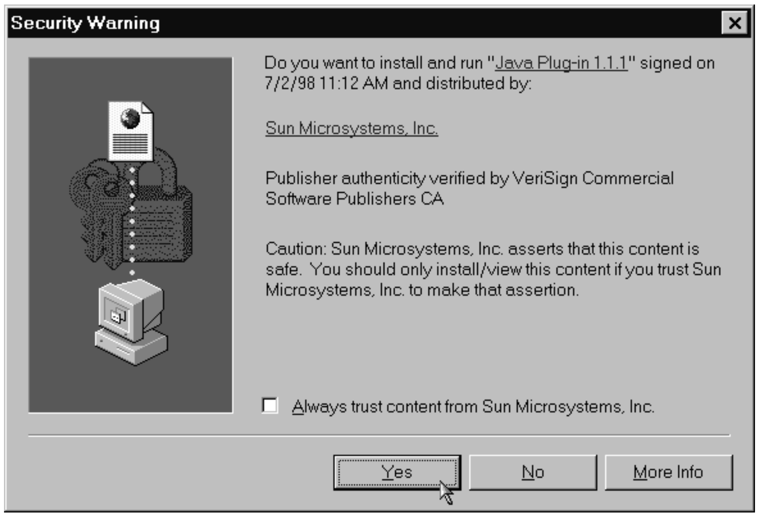
Load the converted `PartsView.html` file by entering one of the following URLs that point to your AS/400 system:

```
http://AS400_name/apptest/PartsView.html
```

or

```
http://AS400_ipaddress/apptest/PartsView.html
```

Instead of loading the applet directly, you see the Security Warning panel shown in Figure 105 on page 141.


Security Warning

Do you want to install and run "Java Plug-in 1.1.1" signed on 7/2/98 11:12 AM and distributed by:

Sun Microsystems, Inc.

Publisher authenticity verified by VeriSign Commercial Software Publishers CA

Caution: Sun Microsystems, Inc. asserts that this content is safe. You should only install/view this content if you trust Sun Microsystems, Inc. to make that assertion.

Always trust content from Sun Microsystems, Inc.

Figure 105. Microsoft Internet Explorer Security Warning

This panel appears because Microsoft Internet Explorer processes the following lines of code in the converted HTML file (see Figure 100 on page 135 for the complete listing of the converted HTML file):

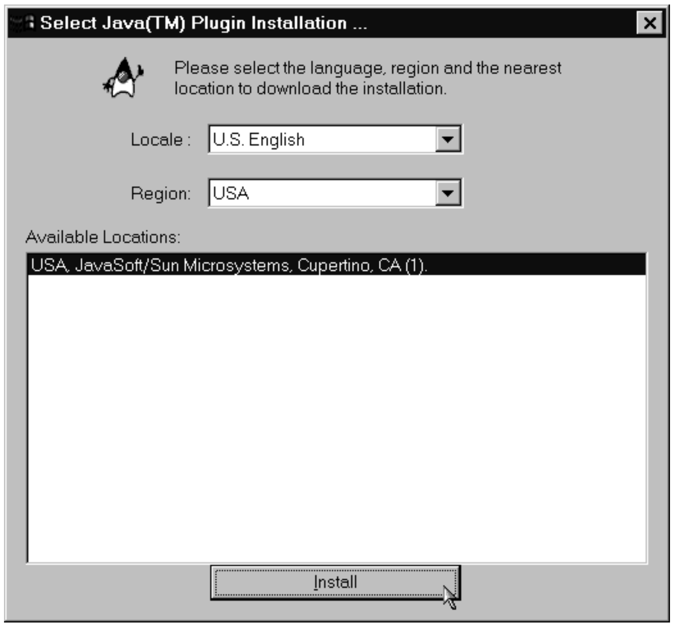
```
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
   WIDTH=700
   HEIGHT=350
   codebase="http://java.sun.com/products/plugin/1.1.1/jinstall-111-win32.cab#
   Version=1,1,1,0">
<PARAM NAME=CODE    VALUE=views.PartsView.class >
<PARAM NAME=ARCHIVE VALUE=PartsView.jar >
<PARAM NAME="type"  VALUE="application/x-java-applet;version=1.1">
```

Figure 106. Code in the Converted HTML File

The `OBJECT` tag is used to identify an ActiveX control that Microsoft Internet Explorer uses to provide Java Plug-in support. The ActiveX control is uniquely identified by the `classid` parameter. When you load the converted HTML file, Microsoft Internet Explorer uses the `classid` to determine if that particular ActiveX control is already loaded and available on the PC (the Windows 95/NT Registry contains a list of installed ActiveX controls).

The first time you load the converted HTML file, the ActiveX control is not on the PC, so the `codebase` parameter is used to indicate where Microsoft Internet Explorer can find a copy of the control. The URL in the `codebase` parameter is set in the Converter Advanced Options (see Figure 104 on page 140). The control is available on a Sun Microsystems, Inc. Web site.

Upon contacting that site and starting the download of the control, the Security Warning panel is displayed (Figure 105 on page 141). This is the default behavior of Microsoft Internet Explorer, since downloading an ActiveX control is a potentially harmful activity (ActiveX controls are not subject to the same security constraints as Java applets). After downloading the ActiveX control, it prompts you to download the Java Plug-in (Figure 107 on page 142).


Select Java(TM) Plugin Installation ...

Please select the language, region and the nearest location to download the installation.

Locale : U.S. English

Region: USA

Available Locations:

USA, JavaSoft/Sun Microsystems, Cupertino, CA (1).

Figure 107. Select Java Plug-in Installation Dialog

There are two important points you should note about the Java Plug-in installation process in Microsoft Internet Explorer, using the defaults from the Converter:

- The ActiveX control is not the Java Plug-in itself. Its only purpose is to create a connection to a Web site where you can download the Java Plug-in from.
- The Java Plug-in file that is downloaded is 4779KB in size. If you are using a dial-up connection to the Internet, it may take quite a long time to download the Java Plug-in.

The good news is that you only need to download and install the Java Plug-in once for each PC. The next time you load the HTML file, the ActiveX control is located on your PC, so the download process is not required.

The bad news is that most people using a dial-up connection to the Internet may cancel the download of the Java Plug-in. The other consideration is that if you are using this technique in an intranet environment, it is wasteful for every PC to download the Java Plug-in from the Sun Microsystems, Inc. Web site.

In Section 3.4.2.6, “Downloading the Java Plug-in for an Intranet Environment” on page 145, you see how to install the Java Plug-in using an intranet environment. For now, you can cancel the download and close the Microsoft Internet Explorer browser.

3.4.2.5 Opening the Converted PartsView.html in Netscape Navigator

This part assumes that you have Netscape Navigator version 3.x or 4.x installed on your computer. If you do not, go to the next step to set up the Java Plug-in to run in an intranet environment.

You load the converted `PartsView.html` file by entering one of the following URLs that point to your AS/400 system:

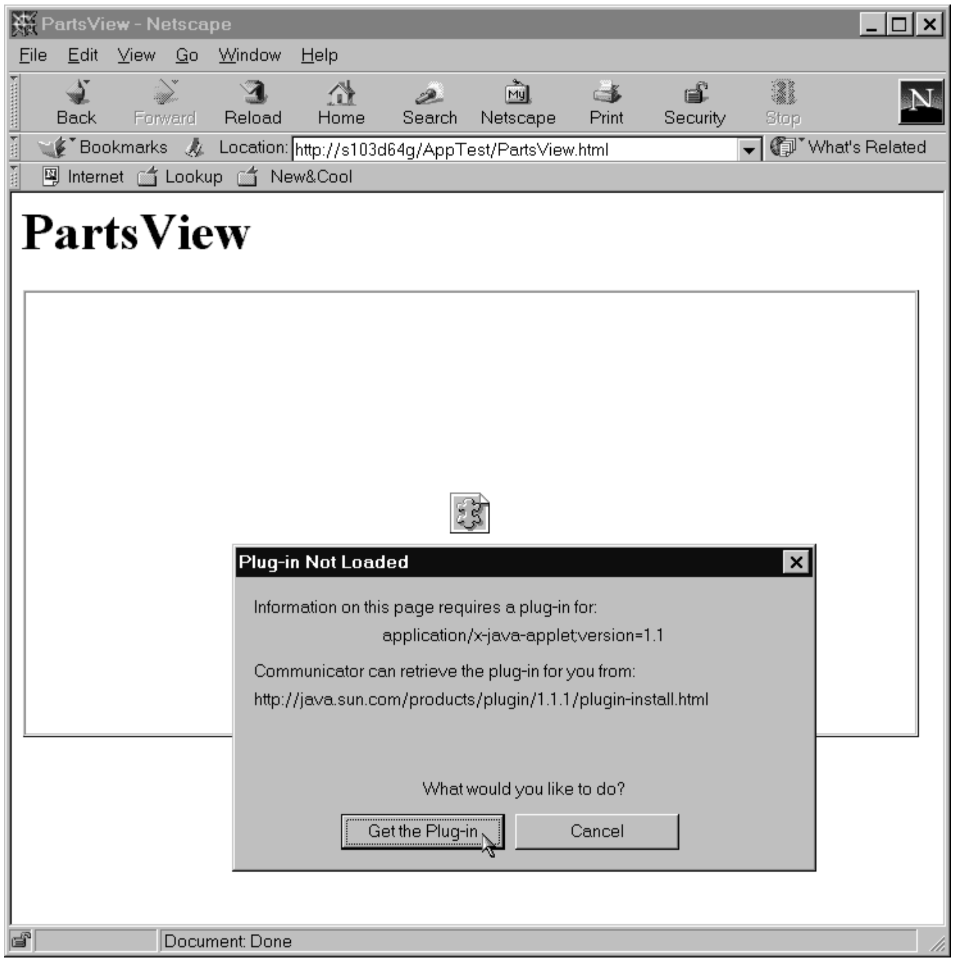
- `http://AS400_name/apptest/PartsView.html`
- `http://AS400_ipaddress/apptest/PartsView.html`

When you load the HTML file, the PartsView page appears in the browser with an icon in the middle of the page. If you click on the icon, you see the Plug-in Not Loaded panel, as shown in Figure 109. That panel appears because Netscape Navigator processes the following lines of code in the converted HTML file (see Figure 100 on page 135 for the complete listing of the converted HTML file):

```
<EMBED type="application/x-java-applet;version=1.1"
   java_CODE=views.PartsView.class
   java_ARCHIVE=PartsView.jar,dab.jar
   WIDTH=700
   HEIGHT=350
   pluginspage="http://java.sun.com/products/plugin/1.1.1/plugin-install.html"
   >
```

Figure 108. Code in the Converted HTML File that is Processed by Netscape Navigator

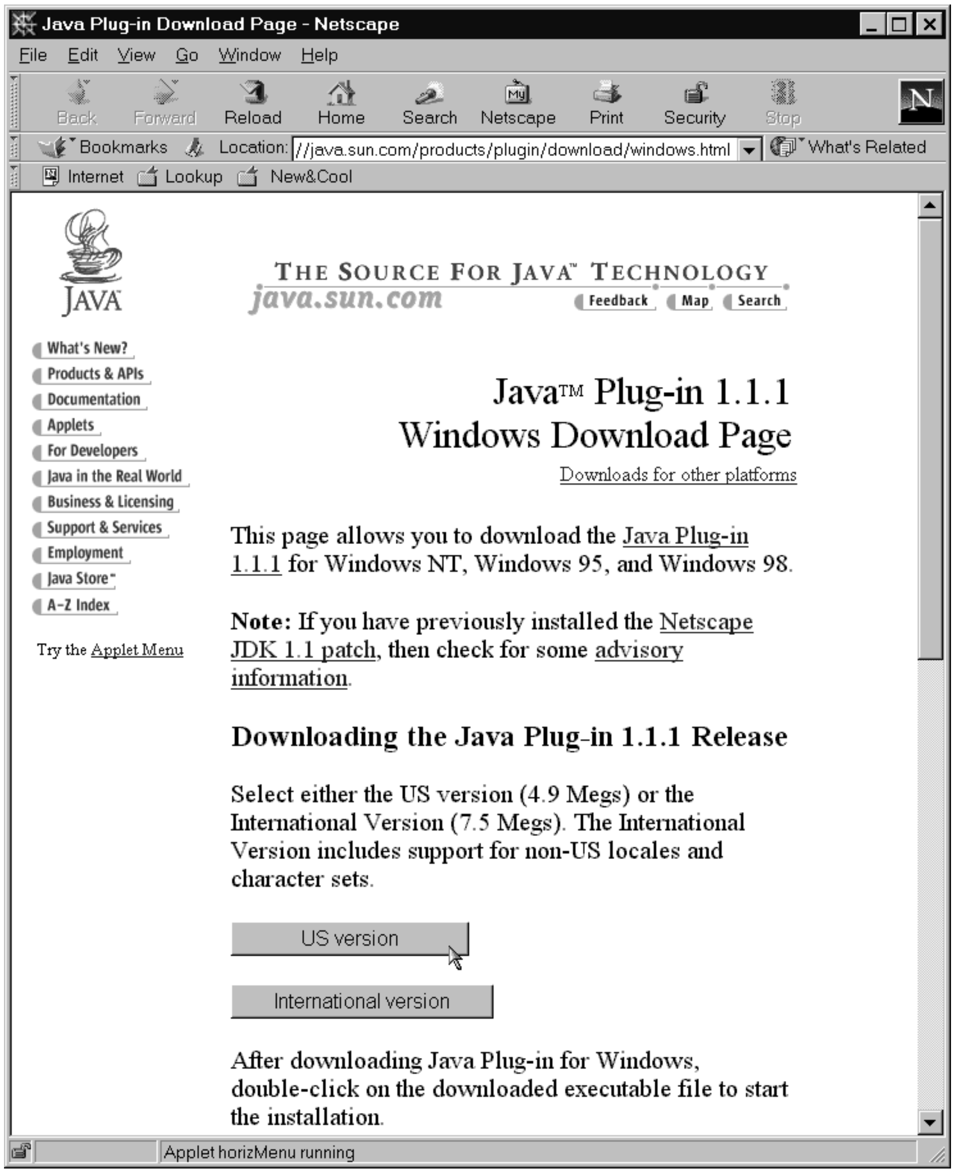
The `EMBED` tag is used to identify the plug-in that is used to process the Java applet. If the plug-in is not installed, the `pluginspage` parameter is used to indicate where Netscape Navigator can go to retrieve the plug-in. The URL in the `pluginspage` parameter is set in the Converter Advanced Options (see Figure 104 on page 140). The page is available on a Sun Microsystems, Inc. Web site.


PartsView - Netscape
File Edit View Go Window Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://s103d64g/AppTest/PartsView.html
What's Related
Internet Lookup New&Cool
PartsView
Plug-in Not Loaded
What would you like to do?
Get the Plug-in
Cancel
Document: Done

Information on this page requires a plug-in for:
application/x-java-applet;version=1.1
Communicator can retrieve the plug-in for you from:
http://java.sun.com/products/plugin/1.1.1/plugin-install.html

Figure 109. Netscape Navigator Plug-in Not Loaded Panel

When you click the Get the Plug-in button in the Plug-in Not Loaded panel (Figure 109 on page 143), another browser window is open to display the Web page identified in the `pluginspage` parameter. Figure 110 shows a view of part of the page. You can download the Java Plug-in from the page.



Java™ Plug-in 1.1.1
Windows Download Page

Downloads for other platforms

This page allows you to download the Java Plug-in 1.1.1 for Windows NT, Windows 95, and Windows 98.

Note: If you have previously installed the Netscape JDK 1.1 patch, then check for some advisory information.

Downloading the Java Plug-in 1.1.1 Release

Select either the US version (4.9 Megs) or the International Version (7.5 Megs). The International Version includes support for non-US locales and character sets.

After downloading Java Plug-in for Windows, double-click on the downloaded executable file to start the installation.

Figure 110. Java Plug-in Download Page

The file that you download from the Java Plug-in page is the same file that is downloaded for Microsoft Internet Explorer. The primary difference between the Microsoft Internet Explorer download and the Netscape Navigator download is that Microsoft Internet Explorer automatically installs the Java Plug-in after the download completes. With Netscape Navigator, you need to close the browser, navigate to the directory where you stored the downloaded file, run the installation program, and restart the browser.

As with the Microsoft Internet Explorer test (see Section 3.4.2.4, “Opening the Converted PartsView.html in Internet Explorer” on page 140), you do not download the Java Plug-in at this time. Close the download window (Figure 110 on page 144) and the applet window (Figure 109 on page 143).

3.4.2.6 Downloading the Java Plug-in for an Intranet Environment

As you saw in the previous two sections, the default behavior of the converted HTML file is to connect to the Internet and download the Java Plug-in the first time it is required on each PC. Because of the size of the file (4779KB), it is far better to download the Java Plug-in once to a server, and install it on each PC in your intranet as required from the server. There are two primary advantages to installing the Java Plug-in from your intranet:

- The download time for each PC is considerably less than if those PCs retrieved the Java Plug-in from the Internet. Also, if you block downloads through your firewall, you can still install the Java Plug-in on your PCs that are behind the firewall.
- You control the version of the Java Plug-in that is installed on the PCs. If Sun Microsystems, Inc. posts a new version of the Java Plug-in, you can download and test it on your test PCs, rather than potentially have different versions of the Java Plug-in downloaded to PCs in your organization.

To download the Java Plug-in to a server, go to the following Web site:

`http://java.sun.com/products/plugin/1.1.1/index-1.1.1.html`

You are prompted to select a directory to which download the file. The name of the file is:

`plugin-111-win32.exe`

The file size is approximately 4779KB (at the time we tested this process).

Download the file to the `apptest` directory on your AS/400 system. You serve the Java Plug-in from your AS/400 system, along with the applet.

3.4.2.7 Changing the Converter to the Local Version of the Java Plug-in

Now that you have a version of the Java Plug-in on your AS/400 system, you need to change the Java Plug-in HTML Converter so that the converted HTML that it outputs points to your copy of the Java Plug-in.

Go back into the Converter (Figure 102 on page 138) and click the **Advanced Options** button. Specify the following information for the two entries in the Advanced Options dialog:

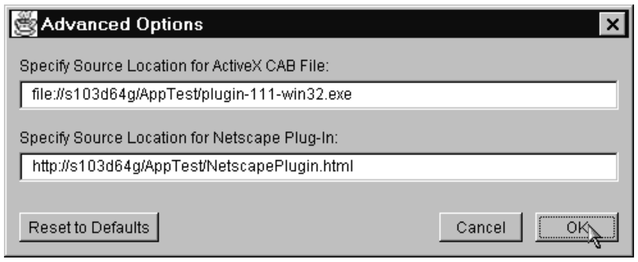
- **Source location for ActiveX CAB file**. This is used in the `codebase` parameter for the `OBJECT` tag (Figure 106 on page 141):

  `file://AS400_name/apptest/plugin-111-win32.exe`

- **Source location for Netscape plug-in**. This is used in the `pluginspage` parameter for the `EMBED` tag (Figure 108 on page 143):

  `http://AS400_name/apptest/NetscapePlugin.html`

When you are finished, your Advanced Options dialog should appear as shown in Figure 111 on page 146 (with your AS/400 system name in place of `s103d64g`).



Specify Source Location for ActiveX CAB File:
file://s103d64g/AppTest/plugin-111-win32.exe

Specify Source Location for Netscape Plug-In:
http://s103d64g/AppTest/NetscapePlugin.html

Figure 111. Revised Advanced Options Settings for the HTML Converter

The ActiveX CAB File

For Microsoft Internet Explorer, you can specify a file URL to point to the actual Java Plug-in file that you downloaded from Sun Microsystems, Inc. Microsoft Internet Explorer can directly open the Java Plug-in file and install it using the file URL. You can see what happens with the revised HTML file in Section 3.4.2.10, “Testing the Revised HTML File in Microsoft Internet Explorer” on page 148.

The Netscape Plug-in

Netscape Navigator cannot directly load the Java Plug-in file that you downloaded. You need to point the browser to an HTML file that contains a link to the Java Plug-in file. You create the HTML file referenced in this setting in the next section.

3.4.2.8 Creating the NetscapePlugin.html File

Enter the code shown in Figure 112 on page 147 using NotePad or another editor, then save the file as `NetscapePlugin.html` in the `Applet` directory on the AS/400 system. This HTML is referenced in the Advanced Options dialog in the HTML Converter (Figure 111 on page 146).

The most important line in this HTML file is the line with the `<a href>` tag. The code in that line points to a file URL that identifies the Java Plug-in on the AS/400 system. Change the AS/400 system name to the name of your AS/400 system. You can change any of the other lines that you like. If you do, remember to follow the steps listed in this version of the HTML file.

```
<html>

<head>
   <title>Install Java Plug-In 1.1.1 for Netscape Navigator</title>
</head>

<body bgcolor="white">

   <h1>Install Java Plug-In 1.1.1</h1>

   Click <a href="file://S103D64G/apptest/plugin-111-win32.exe">here</a>
   to download the Java Plug-In installer program.
   <br>
   <br>

   Follow these steps to download and install the Java Plug-In:
   <br>

   <ol>
```

```
      <li>Download file plugin-111-win32.exe to a temporary directory on your
      PC.
      <li>Exit your browser.
      <li>Run plugin-111-win32.exe in the temporary directory.
      <li>Restart the applet in your browser.
   </ol>
</body>
</html>
```

Figure 112. Netscape Plug-in HTML File

3.4.2.9 Regenerating the Converted HTML

Now that you downloaded the Java Plug-in to your AS/400 system and changed the Converter to point to that version of the plug-in, you need to regenerate the HTML file. Follow these steps to create an updated version of the HTML file:

1. Using the Windows 95/NT Explorer, go to directory `APPTEST_BAK` on your AS/400 system (or the directory that you specified for the **Backup Files in Folder** directory in Figure 102 on page 138).
2. Right-click on file `PartsView.html` in the backup directory. Select the **Cut** option from the pop-up menu.
3. Right-click on directory `apptest` on your AS/400 system and select **Paste** from the pop-up menu. Select the option to overwrite file `PartsView.html` with the backup version of the file.
4. Run the Converter again to create a new version of PartsView.html (see Section 3.4.2.3, “Running the Java Plug-in HTML Converter for PartsView.html” on page 137).
5. Examine the generated `PartsView.html` file in directory `apptest`. You should specifically check the `codebase` parameter in the `OBJECT` tag and the `pluginspage` parameter in the `EMBED` tag to verify that the values are the same as those you entered in the Advanced Options dialog (Figure 111 on page 146). If any of the parameters are incorrect, review the steps in Section 3.4.2.7, “Changing the Converter to the Local Version of the Java Plug-in” on page 145 and the steps in this section.

Note

Be especially careful to specify and verify the file name for the ActiveX CAB file correctly. The file name is:

```
plugin-111-win32.exe
```

In one of our tests, we mistakenly entered the file name in the Advanced Options dialog as:

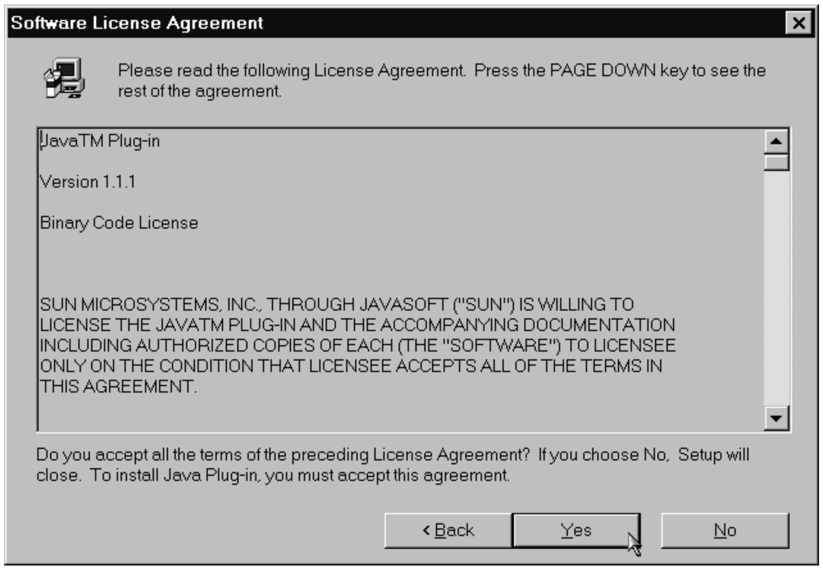
```
plugin-win32.exe
```

When Microsoft Internet Explorer opened the Web page with the applet, it simply did nothing. It did not load the plug-in because of the incorrect file name, and it did not display any type of error or warning message.

3.4.2.10 Testing the Revised HTML File in Microsoft Internet Explorer

Enter the URL to load the applet from the AS/400 system, as shown in Section 3.4.2.4, “Opening the Converted PartsView.html in Internet Explorer” on page 140. This time, when the browser realizes that the Java Plug-in is not installed, it goes to the file URL and starts to download the Java Plug-in from the `apptest` directory on the AS/400 system. Because the ActiveX control is not being downloaded from the Sun Microsystems, Inc. Web site, you do not see the Security Warning panel (Figure 105 on page 141).

After downloading the Java Plug-in (over 4MB), the browser launches the installation part of the program. The installation program is a conventional InstallShield installation. The first panel displayed is the Software License Agreement (Figure 113).



Please read the following License Agreement. Press the PAGE DOWN key to see the rest of the agreement.

JavaTM Plug-in

Version 1.1.1

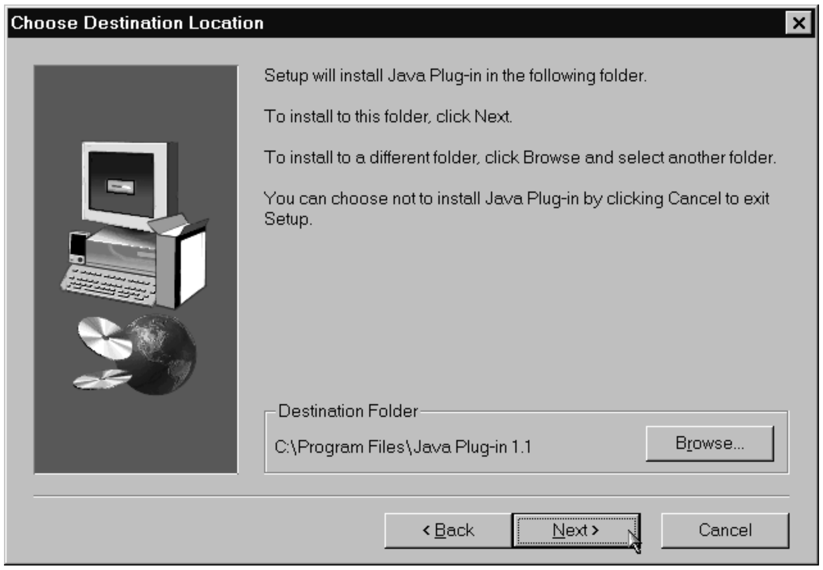
Binary Code License

SUN MICROSYSTEMS, INC., THROUGH JAVASOFT ("SUN") IS WILLING TO LICENSE THE JAVATM PLUG-IN AND THE ACCOMPANYING DOCUMENTATION INCLUDING AUTHORIZED COPIES OF EACH (THE "SOFTWARE") TO LICENSEE ONLY ON THE CONDITION THAT LICENSEE ACCEPTS ALL OF THE TERMS IN THIS AGREEMENT.

Do you accept all the terms of the preceding License Agreement? If you choose No, Setup will close. To install Java Plug-in, you must accept this agreement.

Figure 113. Java Plug-in Software License Agreement

The panel in Figure 114 on page 149 is the Choose Destination Location panel. You can accept the default destination or click the Browse button to select or specify another destination.



Setup will install Java Plug-in in the following folder.

To install to this folder, click Next.

To install to a different folder, click Browse and select another folder.

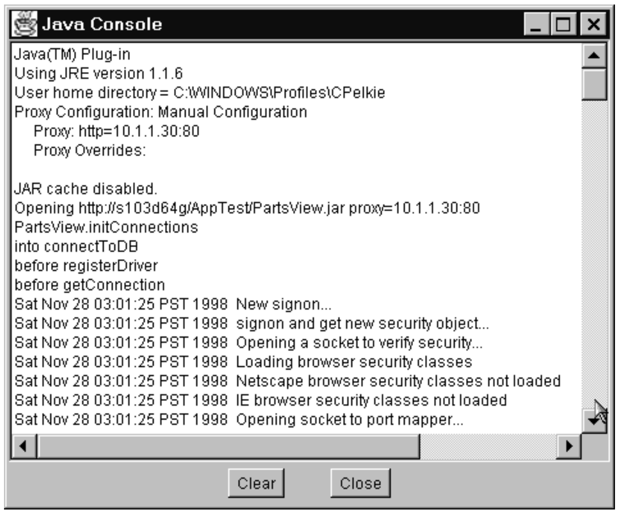
You can choose not to install Java Plug-in by clicking Cancel to exit Setup.

Destination Folder
C:\Program Files\Java Plug-in 1.1

Figure 114. Java Plug-in Choose Destination Location Dialog

After specifying the destination location, the Java Plug-in is installed. The browser continues loading the applet, which invokes the Java runtime environment from the Java Plug-in.

You can tell if the Java runtime environment is provided by the Java Plug-in in the Java Console (Figure 115). As shown in Figure 115, the first two lines in the Java Console identify the Java Plug-in and the version of the Java runtime environment in use (1.1.6).



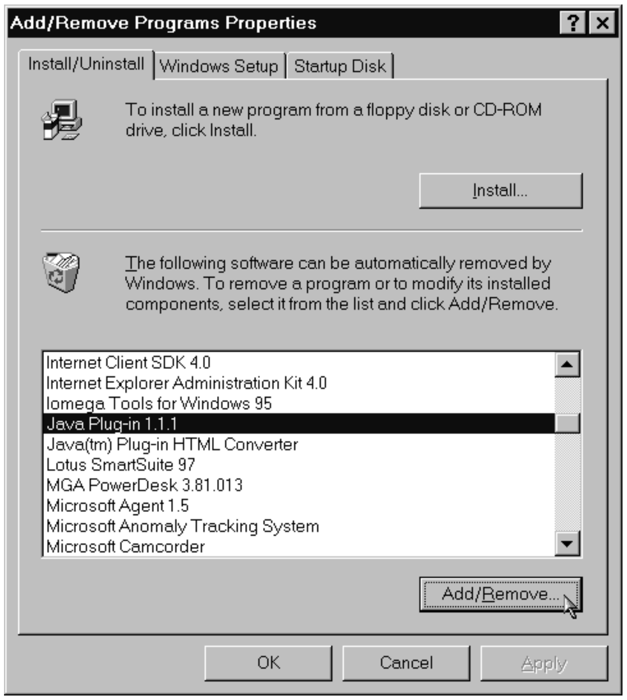
```
Java(TM) Plug-in
Using JRE version 1.1.6
User home directory = C:\WINDOWS\Profiles\CPelkie
Proxy Configuration: Manual Configuration
    Proxy: http=10.1.1.30:80
    Proxy Overrides:

JAR cache disabled.
Opening http://s103d64g/AppTest/PartsView.jar proxy=10.1.1.30:80
PartsView.initConnections
into connectToDB
before registerDriver
before getConnection
Sat Nov 28 03:01:25 PST 1998  New signon...
Sat Nov 28 03:01:25 PST 1998  signon and get new security object...
Sat Nov 28 03:01:25 PST 1998  Opening a socket to verify security...
Sat Nov 28 03:01:25 PST 1998  Loading browser security classes
Sat Nov 28 03:01:25 PST 1998  Netscape browser security classes not loaded
Sat Nov 28 03:01:25 PST 1998  IE browser security classes not loaded
Sat Nov 28 03:01:25 PST 1998  Opening socket to port mapper...
```

Figure 115. Java Console

3.4.2.11 Uninstalling the Java Plug-in for the Netscape Navigator Test

If you have Netscape Navigator 3.x or 4.x on your PC, test its ability to install the Java Plug-in using the `NetscapePlugins.html` page that you created (see Section 3.4.2.8, “Creating the NetscapePlugin.html File” on page 146). To test the installation process, you need to uninstall the Java Plug-in from your PC, assuming that you completed the test with Microsoft Internet Explorer (see Section 3.4.2.10, “Testing the Revised HTML File in Microsoft Internet Explorer” on page 148). The reason why you need to uninstall the Java Plug-in is because Netscape Navigator considers the Java Plug-in to be installed, even if it was initially installed by Microsoft Internet Explorer. If the Java Plug-in is already installed, Netscape Navigator does not try to run its version of the installation process, but simply loads the applet.

To uninstall the Java Plug-in, go to the Windows 95/NT Control Panel and start the **Add/Remove Programs** program. Scroll to the entry for the Java Plug-in 1.1.1 in the **Install/Uninstall** tab (Figure 116). Click the **Add/Remove** button to remove the Java Plug-in from you PC. Close the **Add/Remove Programs** program. You can now run the test with Netscape Navigator. It attempts to install the Java Plug-in.



To install a new program from a floppy disk or CD-ROM drive, click Install.

The following software can be automatically removed by Windows. To remove a program or to modify its installed components, select it from the list and click Add/Remove.

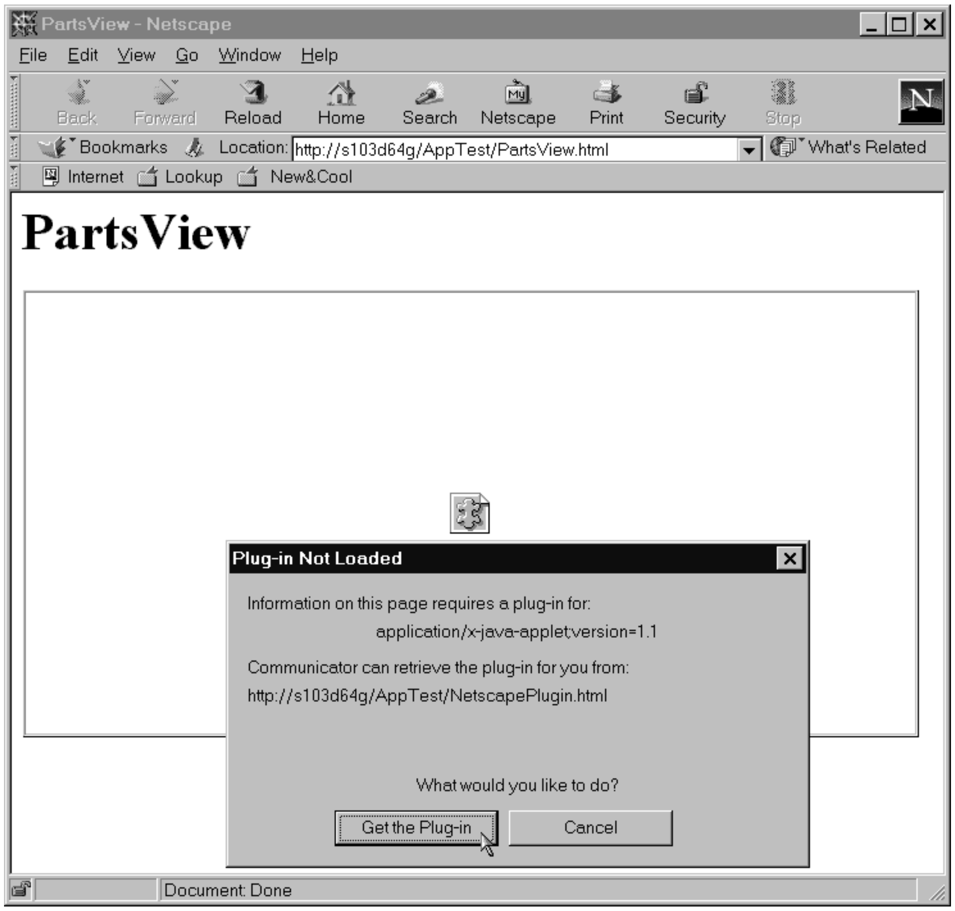
Internet Client SDK 4.0
Internet Explorer Administration Kit 4.0
Iomega Tools for Windows 95
Java Plug-in 1.1.1
Java(tm) Plug-in HTML Converter
Lotus SmartSuite 97
MGA PowerDesk 3.81.013
Microsoft Agent 1.5
Microsoft Anomaly Tracking System
Microsoft Camcorder

Figure 116. Windows 95/NT Add/Remove Programs Dialog

3.4.2.12 Testing the Revised HTML File in Netscape Navigator

Enter the URL to load the applet from the AS/400 system, as shown in Section 3.4.2.5, “Opening the Converted PartsView.html in Netscape Navigator” on page 142. Because the Java Plug-in is not installed, Netscape Navigator displays the icon in place of the applet. Click the icon to display the Plug-in Not Loaded panel, as shown in Figure 117 on page 151.

This time, the Plug-in Not Loaded panel should refer to the `NetscapePlugin.html` file that you created in Section 3.4.2.8, “Creating the NetscapePlugin.html File” on page 146. You indicated that you wanted to use that HTML file, rather than the Sun Microsystems, Inc. Web page (see Figure 109 on page 143) in the Advanced Options dialog (see Figure 111 on page 146).


PartsView - Netscape
File Edit View Go Window Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://s103d64g/AppTest/PartsView.html What's Related
Internet Lookup New&Cool
PartsView
Plug-in Not Loaded
Get the Plug-in
Cancel
Document: Done

Information on this page requires a plug-in for:
application/x-java-applet;version=1.1

Communicator can retrieve the plug-in for you from:
http://s103d64g/AppTest/NetscapePlugin.html

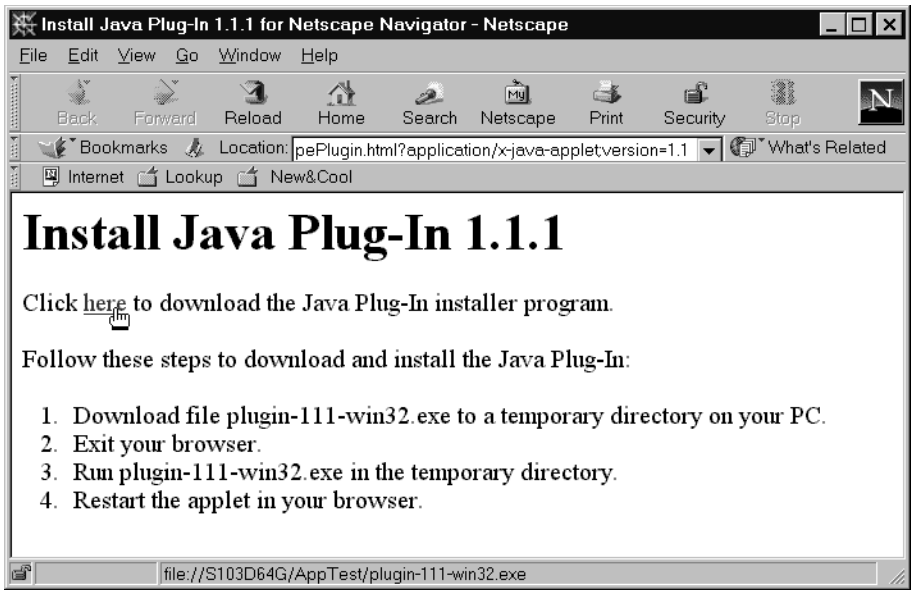
What would you like to do?

Figure 117. Plug-in Not Loaded Panel

When you click the Get the Plug-in button in the Plug-in Not Loaded panel, the Web page that you created is displayed (Figure 118 on page 152). Click the link to start downloading the Java Plug-in file from the AS/400 system to your PC.

Unlike Microsoft Internet Explorer, Netscape Navigator does not automatically start the Java Plug-in installation procedure. Instead, you need to follow the steps as shown on the `NetscapePlugin.html` page. You need to manually start the Java Plug-in installation procedure. You work with the same panels that are used in the Microsoft Internet Explorer install (see Figure 113 on page 148 and Figure 114 on page 149).

After completing the Java Plug-in installation procedure, you can restart the Netscape Navigator browser and reload the `PartsView.html` file to start the applet. After starting, you should see the same Java Console as is used in the Microsoft Internet Explorer version of the applet (Figure 115 on page 149).


Install Java Plug-In 1.1.1 for Netscape Navigator - Netscape
File Edit View Go Window Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: pePlugin.html?application/x-java-applet;version=1.1 What's Related
Internet Lookup New&Cool

Install Java Plug-In 1.1.1

Click here to download the Java Plug-In installer program.

Follow these steps to download and install the Java Plug-In:

1. Download file plugin-111-win32.exe to a temporary directory on your PC.
2. Exit your browser.
3. Run plugin-111-win32.exe in the temporary directory.
4. Restart the applet in your browser.

file://S103D64G/AppTest/plugin-111-win32.exe

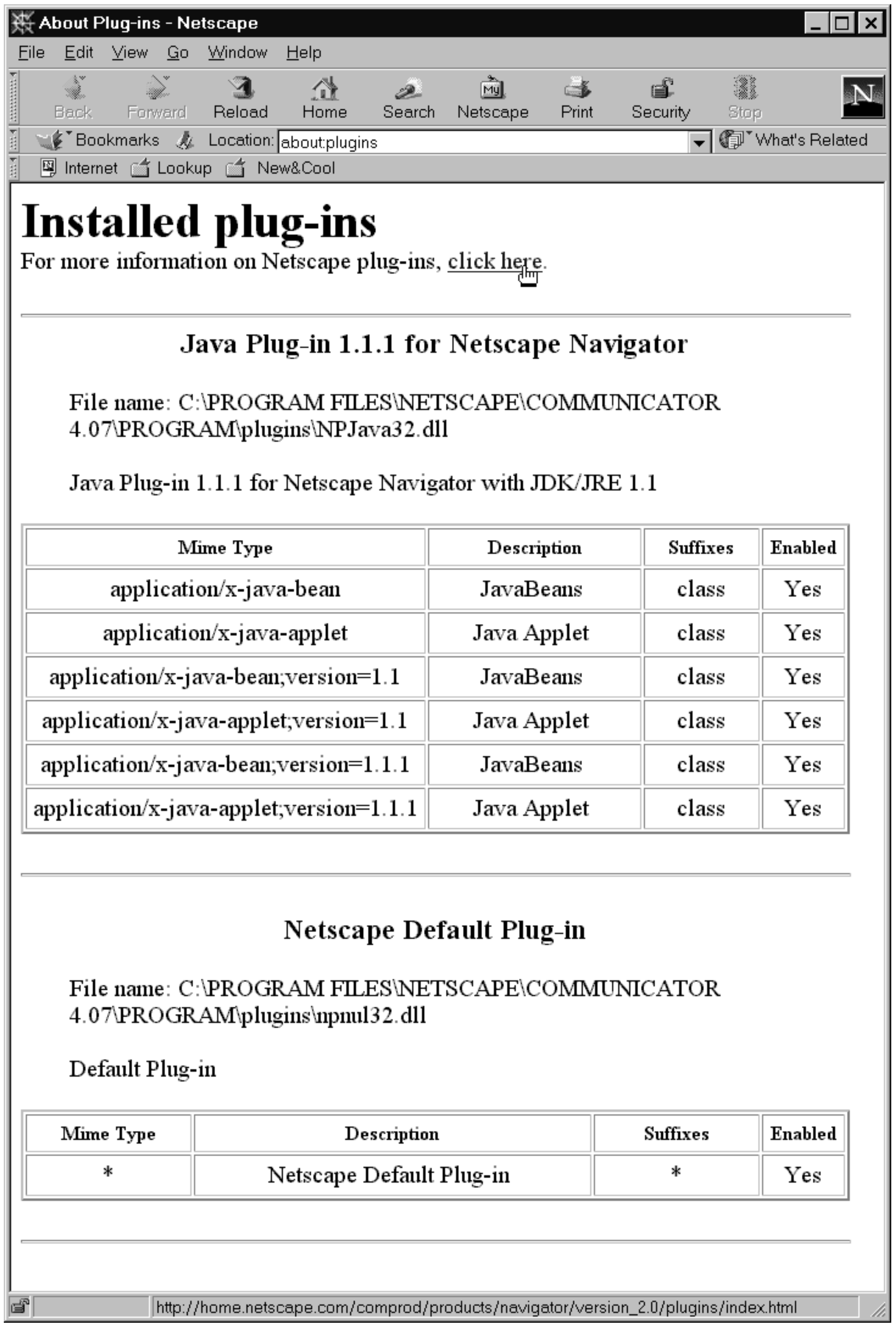
Figure 118. Netscape Plug-in.html File

3.4.2.13 Verifying Netscape Navigator Plug-ins

Netscape Navigator includes a feature you can use to easily verify the presence or absence of any plug-ins used with the browser. In the browser's Location entry space, enter:

```
about:plugins
```

The browser responds with a page similar to Figure 119 on page 153, which lists all of the plug-ins that are currently available to the browser. Microsoft Internet Explorer does not provide a feature similar to this.


About Plug-ins - Netscape
File Edit View Go Window Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: about:plugins What's Related
Internet Lookup New&Cool

Installed plug-ins

For more information on Netscape plug-ins, click here.

Java Plug-in 1.1.1 for Netscape Navigator

File name: C:\PROGRAM FILES\NETSCAPE\COMMUNICATOR 4.07\PROGRAM\plugins\NPJava32.dll

Java Plug-in 1.1.1 for Netscape Navigator with JDK/JRE 1.1

Mime Type	Description	Suffixes	Enabled
application/x-java-bean	JavaBeans	class	Yes
application/x-java-applet	Java Applet	class	Yes
application/x-java-bean;version=1.1	JavaBeans	class	Yes
application/x-java-applet;version=1.1	Java Applet	class	Yes
application/x-java-bean;version=1.1.1	JavaBeans	class	Yes
application/x-java-applet;version=1.1.1	Java Applet	class	Yes

Netscape Default Plug-in

File name: C:\PROGRAM FILES\NETSCAPE\COMMUNICATOR 4.07\PROGRAM\plugins\npnul32.dll

Default Plug-in

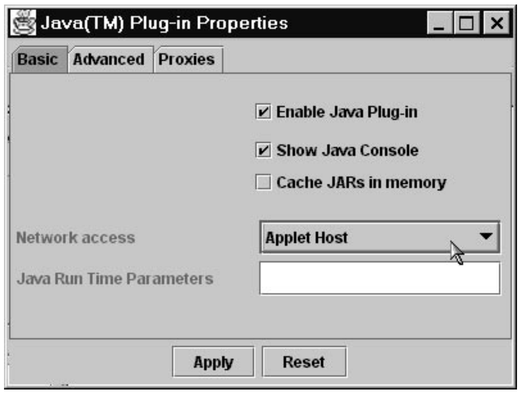
Mime Type	Description	Suffixes	Enabled
*	Netscape Default Plug-in	*	Yes

http://home.netscape.com/comprod/products/navigator/version_2.0/plugins/index.html

Figure 119. About:plugins Feature

3.4.2.14 Java Plug-in Control Panel

Regardless of which browser is used to install the Java Plug-in on your PC, the Java Plug-in Control Panel is also installed. The Control Panel is available from the **Start—>Programs** menu in Windows 95/NT.


Java(TM) Plug-in Properties

Basic | Advanced | Proxies

[✓] Enable Java Plug-in
[✓] Show Java Console
[ ] Cache JARs in memory

Network access: Applet Host

Java Run Time Parameters:

Figure 120. Basic Tab of the Java Plug-in Control Panel

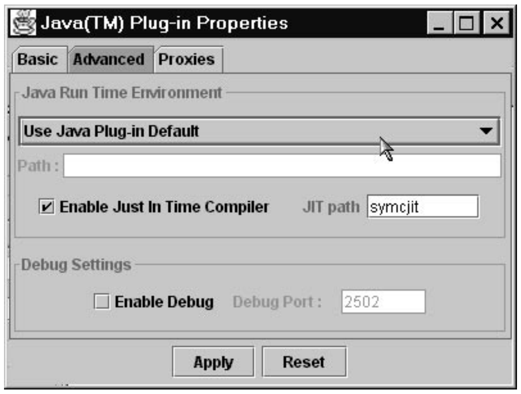
Figure 120 shows the Basic panel of the Control Panel. The options on this panel are:

- **Enable Java Plug-in**—If checked, the Java Plug-in is used to provide the Java runtime environment for applets. If not checked, the Java runtime environment is provided by the browser. You can use this option to test differences in browser Java support compared with the Java Plug-in, rather than uninstall the Java Plug-in. The default for this option is checked.
- **Show Java Console**—If checked, the Java Console is displayed when an applet runs (see Figure 115 on page 149). As shown throughout this chapter, the Java Console is one of the best tools you have for debugging applets. The default setting for this option is unchecked, which is appropriate for users other than the applet programmers.
- **Cache JARs in memory**—If checked, applet classes are cached and reused when the applet is reloaded. This improves memory usage and performance. However, you should uncheck this option when you are developing and testing an applet so that the most recent classes are always loaded.
- **Network access**—This option lets you choose the level of permission you grant an applet in your network. The options are:
  - **None**—The applet cannot access any resources in the network, not even the host server it was loaded from.
  - **Applet Host**—The applet can connect back to the server it was loaded from. This is the default setting for this option.
  - **Unrestricted**—The applet can access any resources in the network. This is considered to be a security hazard.

  Note: This option can only be changed if the Java runtime environment in use is a 1.1.x version. If you use the 1.2 runtime environment, you must use the new security architecture to select the level of network access the applet is allowed.
- **Java Run Time Parameters**—This option is used to enter startup parameters, similar to those you can provide for the `java.exe` command line.

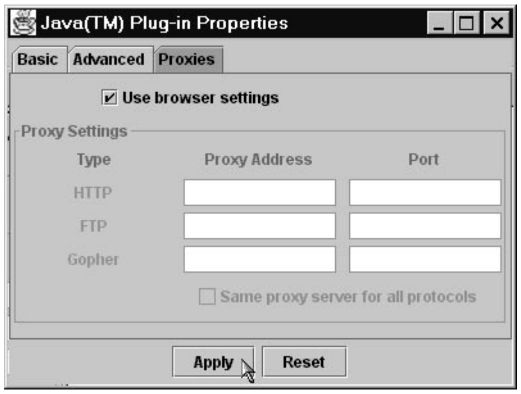
The Control Panel has two additional tabs:

- **Advanced** (Figure 121)—The main feature on this tab is the Java Run Time Environment selection. When you install the Java Plug-in, a version of the Java runtime environment is installed with the plug-in. If you want to use a different runtime environment, you can select any of the versions that are installed on your PC.
- **Proxies** (Figure 122)—If you need to use different proxy settings than those in your browser, you can uncheck the **Use browser settings** check box and enter the proxy information required.



Use Java Plug-in Default
Path :
☑ Enable Just In Time Compiler
JIT path symcjit
Debug Settings
☐ Enable Debug
Debug Port : 2502

Figure 121. Advanced Tab in the Java Plug-in Control Panel



☑ Use browser settings
Proxy Settings

Type	Proxy Address	Port
HTTP		
FTP		
Gopher		

☐ Same proxy server for all protocols

Figure 122. Proxies Tab in the Java Plug-in Control Panel

3.4.3 Summary of the Java Plug-in

Although it may seem like a lot of work to use the Java Plug-in, we found in our testing that the Java Plug-in actually makes it much easier to develop, test, and deploy an applet. For example, some of the early Netscape Navigator 4.x browsers (4.01 and 4.04) do not provide stable Java runtime environments. Rather than code the work-arounds required for those versions of the browsers, try to accommodate the later versions (4.05, 4.06, 4.07 and most recently 4.5), and provide support for Microsoft Internet Explorer (3.02 and 4.01), it is far easier to install the Java Plug-in and learn how to work with its Java runtime environment. There are some browser Java runtime environments that are simply not worth the effort to develop for and support.

Although it may be argued that imposing the installation of the Java Plug-in on users is an extra burden, keep in mind that the installation is a one-time event. Given that the installation process requires minimal user intervention and no entries other than a few mouse clicks, even the most hesitant user should be able to successfully negotiate the installation process.

The primary advantage of deploying the Java Plug-in to provide applet support is that you can debug any problems with much greater confidence and accuracy than if you have multiple browser versions. Based on our testing and the problems we encountered trying to get applets to work successfully with different browsers, the only realistic alternative to the Java Plug-in is to adopt one browser at one specific release level and support it. You then need to "lock-down" the users so that they cannot upgrade or alter their browser configuration. Using the Java Plug-in alternative, it does not matter which browser or which version is used. When the browser hosts an applet, the same Java runtime environment is called upon.

3.5 Conclusion

As you have seen throughout this chapter, there are a number of factors that are involved in developing and deploying applets. In our tests, one of the main factors in the success or failure of the applet was the version of the browser being used. Generally speaking, the more recent versions of Netscape Navigator and Microsoft Internet Explorer are more "applet friendly".

Because of the browser dependencies, it is difficult to guarantee that an applet using the AS/400 Toolbox for Java classes is suitable for use by the general public. The `jt400.jar` file is approximately 2MB in size. The `jt400.zip` file is approximately 4MB. Those sizes alone rule out the use of applets served over a dial-up connection, except by the most dedicated or desperate of users.

It may be possible to reduce the size of the required Toolbox jar file by careful study of the classes and resources required to be exported. However, you may find that, in practice, the resulting jar file is still several hundred kilobytes in size. The classes and resources required go far beyond those that you directly reference in your Java code. As is, you need to rely on VisualAge for Java to discover classes and resources. Otherwise, if you try to select just the classes and resources you think are required, you may need to iteratively debug the applet many times.

There is a tool named JarMaker available as a beta release on the AS/400 Toolbox for Java homepage. In your browser, enter the URL **www.as400.ibm.com/toolbox**. Click on **Downloads** to find it.

The JarMaker class speeds download through its ability to create a smaller jar file from a larger one. You can use this tool to help reduce the size of jar files that want to serve over a network.

The test described in Section 3.3.6.8, “Using jt400.zip on the PC, PartsView.jar on the AS/400 System” on page 132, seems to provide the best combination of performance and ease of deployment. That being said, that test requires the installation of the `jt400.zip` file on the PC and a modification to the CLASSPATH environment variable. Those requirements may not be overly burdensome for PCs that you can directly control.

Although applets provide access to GUI elements in the AWT or with Swing that you may really want to use, you may find that deployment issues render applets impractical. You can still take advantage of the AS/400 Toolbox for Java by using servlets, which are described in the following chapters in this redbook.

Chapter 4. Introduction to AS/400 Servlets

The objective of this chapter is to provide an introduction to Java servlets. It explains:

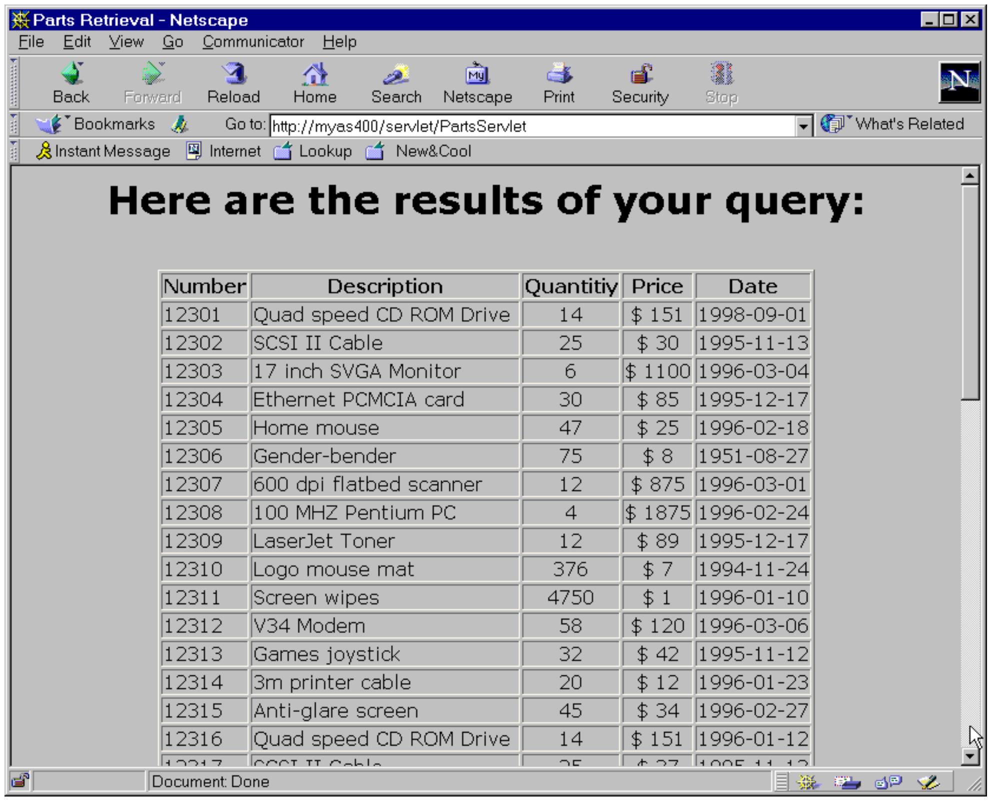
- An introduction to Java servlet support
- How to use servlet support within VisualAge for Java
- How to change the application discussed in Chapter 3, “Introduction to AS/400 Applets” on page 51, from an applet to a servlet
- How to configure an HTTP server to run servlets
- How to run servlets using HTML files

In Chapter 3, “Introduction to AS/400 Applets” on page 51, we discuss an applet that displays data retrieved from an AS/400 database. In this chapter, you see how to add servlet functionality by simply replacing the front end. Instead of using a Graphical User Interface to display the data, we use HTML.

The objective of this chapter is to build a servlet that accesses a database file on the AS/400 system. The output from the servlet is shown in Figure 123.

Note

The example programs discussed in this chapter are available for you to download from the redbook Web site. Refer to Appendix A.1, “Downloading the Files from the Internet Web Site” on page 299, for details.



Parts Retrieval - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks
Go to: http://myas400/servlet/PartsServlet
What's Related
Instant Message Internet Lookup New&Cool
Document: Done

Here are the results of your query:

Number	Description	Quantitiy	Price	Date
12301	Quad speed CD ROM Drive	14	$ 151	1998-09-01
12302	SCSI II Cable	25	$ 30	1995-11-13
12303	17 inch SVGA Monitor	6	$ 1100	1996-03-04
12304	Ethernet PCMCIA card	30	$ 85	1995-12-17
12305	Home mouse	47	$ 25	1996-02-18
12306	Gender-bender	75	$ 8	1951-08-27
12307	600 dpi flatbed scanner	12	$ 875	1996-03-01
12308	100 MHZ Pentium PC	4	$ 1875	1996-02-24
12309	LaserJet Toner	12	$ 89	1995-12-17
12310	Logo mouse mat	376	$ 7	1994-11-24
12311	Screen wipes	4750	$ 1	1996-01-10
12312	V34 Modem	58	$ 120	1996-03-06
12313	Games joystick	32	$ 42	1995-11-12
12314	3m printer cable	20	$ 12	1996-01-23
12315	Anti-glare screen	45	$ 34	1996-02-27
12316	Quad speed CD ROM Drive	14	$ 151	1996-01-12

Figure 123. Servlet Application

4.1 Introduction to the Servlet Support

Before starting with the development of the servlet application, this section explains briefly the basic idea behind servlets. Figure 124 shows an overview of the servlet architecture.

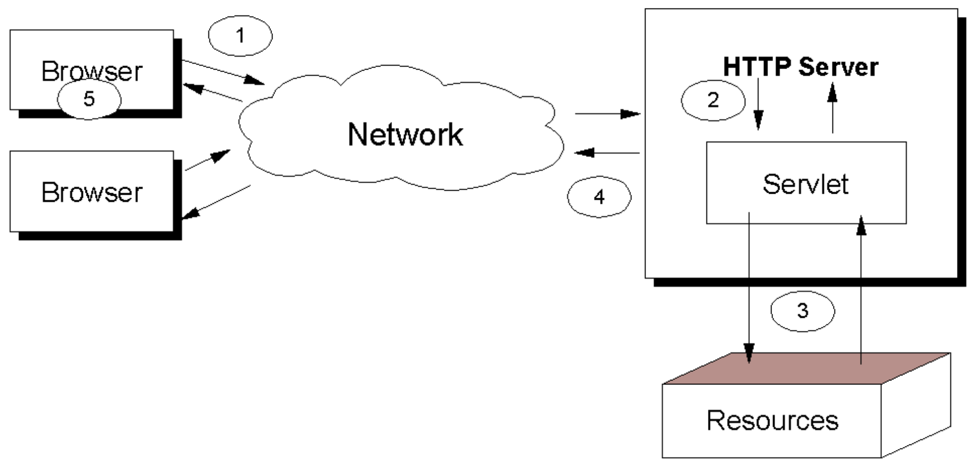


Figure 124. Servlet Architecture

Servlets are modules that run inside request or response-oriented servers, such as Java-enabled Web servers, and extend them in some manner. For example, a servlet may be responsible for taking data in an HTML order-entry form and apply the business logic used to update a company's order database. Servlets relate to servers as applets relate to browsers.

Servlets can provide services or extensions to HTTP servers, performing functions equivalent to CGI programs, server side includes and server side APIs (NSAPI and ISAPI). Although servlets also provide services outside of the HTTP environment, it is the HTTP servlet is of most interest to the AS/400 system at this point. Servlets can also be used as a powerful substitution for CGI-Scripts.

As shown in Figure 124, communication between a browser and a servlet application follows this sequence:

1. The client (browser) sends a request to the HTTP server.
2. The HTTP server forwards the request to the servlet.
3. The servlet receives the request and generates a response by accessing resources and passes the response back to the HTTP server. The response usually contains HTML or other data that can be processed by the client.
4. The HTTP server sends the response to the calling client (browser).
5. The browser renders the data.

Servlet support is provided in two packages:

javax.servlet.*	Provides basic servlet classes and interfaces
javax.servlet.http.*	Provides classes and interfaces for use with HTTP

The classes and interfaces are organized as shown in Figure 125 on page 161.

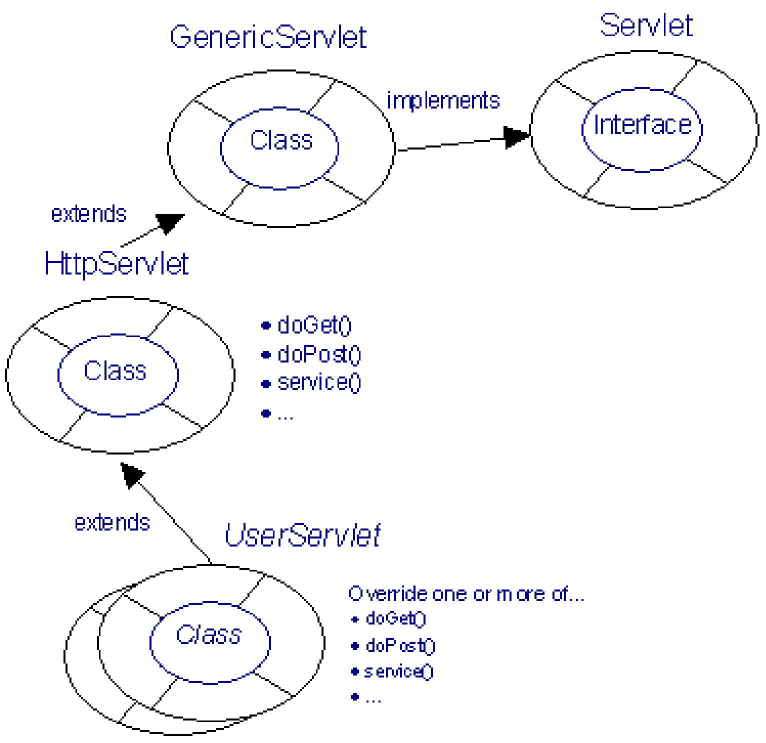


Figure 125. Servlet Hierarchy

Servlet support for Java is provided as part of JDK 1.2. However, the support can be used in JDK 1.1. Two packages are required, javax.servlet and javax.servlet.http. They can be obtained at the Javasoft Web site. In VisualAge for Java 2.0, these packages are available in the **Sun JSDK class libraries** project.

When we start writing servlets, we usually use HTTP for communication between the browser and the server. For that reason, we normally extend the class HttpServlet, which extends the GenericServlet class. To react to a client request, we must extend HttpServlet and override one or more of the following methods:

- doGet()—To support HTTP GET requests
- doPost()—To support HTTP POST requests
- doDelete()—To support HTTP DELETE requests
- doPut()—To support HTTP PUT requests

4.1.1 Why Use Servlets

Java applets have many restrictions. Servlets offer alternatives for some of these restrictions. Servlets do not offer a "better" solution than applets. They simply provides an alternative solution. The "better" solution depends on the user environment and requirements.

You normally store applets and their associated classes at a central location, and download them as needed. As a result, the classes are downloaded as needed by the browser. Depending on the size and number of classes, and the speed of the communications line, the download time can be expensive. There are various solutions available and proposed. Nearly all implement a "download the first time, check, and synchronize changes" scheme. For an infrequent user, this may result in synchronization and downloading for each access. An analogy for this is the infrequent AOL or Prodigy user that receives software updates every time they signs on.

Servlets run on the server, which eliminates the class download time. Since an HTTP servlet can output HTML, relatively small amounts of data can be transferred between the Web server and the Web browser. A high speed link between the Web server and the data server should provide reasonable performance for users that are connected using modems or over the Internet.

Applets running in a browser cannot access system resources and can only open a socket to the server that served the applet. This restriction is relaxed in the newer browsers, which allows the user to grant signed applets to perform some of the restricted tasks. However, this is up to the individual user, not the administrator. The user may not have the necessary knowledge to know if this authority should be granted. To simplify the decision process for the end users, the administrator may choose to store the applets, the depending classes, and the pages on the same system as the data. In an enterprise where the data is stored on multiple servers, the administrator maintains applets, classes, and HTML pages on each server.

Not all browsers run applets in the same way. For example, security is handled differently in Netscape Navigator than it is handled in Microsoft Internet Explorer. Different versions of the same browser may also handle applets differently. This can make implementing an applet solution difficult. Most browsers are capable of rendering HTML. Running a servlet on a server and using HTML to control input and output can help solve these browser compatibility problems.

Servlets offer these important advantages:

- Servlets have full access to local resources.
- They are easy to develop.
- Servlets are portable.
- Servlets are multi-threaded.
- Since servlets run on a server, no downloading is necessary.
- Servlets can be dynamically loaded and unloaded without shutting down the HTTP server.
- Once installed, servlets can be compiled using a JIT or native compiler.

4.1.2 Servlets versus CGI.BIN

Servlets provide functionality similar to cgi-bin programs. Servlets offer a number of advantages of cgi-bin. These include:

- Servlets are written in Java and are easy to develop.
- Cgi-bin programs are platform dependent. For example if you write a cgi-bin program in RPG, it can only run on an AS/400 system. Since servlets are written in Java, they can run on any platform that supports Java.
- Java programs support threads. You can take advantage of this to write multi-threaded servlets to increase functionality and efficiency.

For a more complete discussion of cgi.bin programs on the AS/400 system, see Section 1.4, “Common Gateway Interface (CGI) Programming” on page 15.

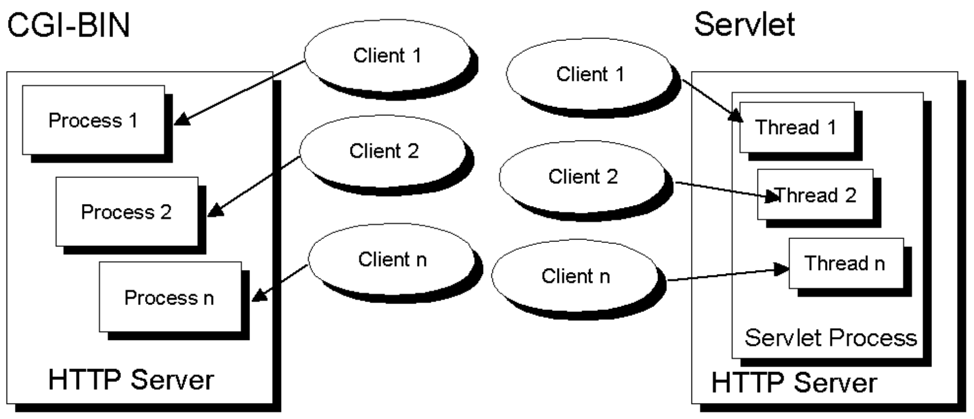


Figure 126. Servlets versus CGI.BIN

4.2 How to Use Servlets

This section introduces you to the basics of how servlets work. For detailed information about servlets, refer to the Javasoft Web site at: www.javasoft.com

Servers load and run servlets, which then accept requests from clients and return data to them. They can also remove servlets. Servlets follow the lifecycle described in the following text.

When a server loads a servlet, it runs the servlet's **init** method. Even though most servlets run in multi-threaded servers, there are no concurrency issues during servlet initialization. The server calls the init method once when it loads the servlet, and does not call it again unless it reloads the servlet. The server cannot reload a servlet until after it removes the servlet by calling the **destroy** method. Initialization is allowed to complete before client requests are handled (before the service method is called) or the servlet is destroyed.

After the server loads and initializes the servlet, the servlet can handle client requests. It processes them in its **service** method. Each client's request has its call to the service method run in its own servlet thread. The method receives the client's request, and sends the client its response.

Servlets can run multiple service methods at a time. It is important, therefore, that service methods be written in a thread-safe manner. For example, if a service method updates a field in the servlet object, that access should be synchronized. If, for some reason, a server does not run multiple service methods concurrently, the servlet should implement the SingleThreadModel interface. This interface guarantees that no two threads run the servlet's service methods concurrently.

During a servlet's lifecycle, it is important to write thread-safe code for destroying the servlet. It is also important for servicing client requests, unless the servlet implements the SingleThreadModel interface. In general, you need three methods to run a servlet:

- init()
- service()
- destroy()

For HTTP servlets, these methods can also be used:

- **doGet()**

  This method is called by the service method to handle HTTP GET operations. This operation allows the client to "get" a response from an HTTP server. It is the default method executed when running an HTTP servlet. This method is generally used for query type requests, where no changes to stored data are made.

- **doPost()**

  This method is called by the service method to handle HTTP POST operations. This operation includes data in the request body that should be acted upon by the servlet.

4.2.1 Communication with an HTTP Server

Servlets implement the javax.servlet.Servlet interface. While servlet writers can develop servlets by implementing the interface directly, this is usually not required. Because most servlets extend Web servers that use the HTTP protocol to interact with clients, the most common way to develop servlets is by extending the javax.servlet.http.HttpServlet class.

The HttpServlet class implements the Servlet interface by extending the GenericServlet base class, and provides a framework for handling the HTTP protocol. Its service method supports standard HTTP/1.1 requests by dispatching each request to a method designed to handle it. By default, servlets written by specializing the HttpServlet class can have multiple threads concurrently running its service method.

Several methods are provided to allow HTTP servlets to interface with a client:

- **doGet**—For handling GET, conditional GET and HEAD requests
- **doPost**—For handling POST requests
- **doPut**—For handling PUT requests
- **doDelete**—For handling Delete requests

When you write a servlet, you override these method with application specific code. Each method has two parameters that are passed in:

- ServletRequest—Encapsulates the request to the servlet
- ServletResponse—Encapsulates the response from the servlet

By using the ServletRequest interface or its subclass HttpServletRequest, servlets can access protocol specific header information such as the scheme of the URL used in the request or the value of the specified parameters. After retrieving the data from the HttpServletRequest, the servlet performs the requested task and sends the information back to the client using the ServletResponse object. It allows the servlet to set the MIME content type and a Writer, through which the servlet can pass the information back to the client.

Parameters passed by the client can be accessed using these methods of the ServletRequest class:

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

4.2.2 Invoking a Servlet

There are three different methods to invoke a servlet:

- Using the servlet's URL:

```
http://<yourServer>/servlet/ServletToCall
```

Figure 127. Calling a Servlet Directly

- Using the HTML <form> tag:

```
<form method=GET action=/servlet/ServletToCall> ... </form>

<form method=POST action=/servlet/ServletToCall> ... </form>
```

Figure 128. Calling a Servlet Using HTML Files

- Using the HTML <servlet> </servlet> tags:

```
<servlet name=ServletToCall code=ServletToCall.class>
..
</servlet>
```

Figure 129. Calling a Servlet Using SHTML Files

In this case, you have to use the file extension .shtml instead of .html or .htm.

4.3 A Simple Servlet

The servlet example in Figure 130 on page 166 displays "Hello world" in a browser.

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


public class HelloWorldServlet extends HttpServlet{

public void doGet (HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
    {
        PrintWriter        out;

        res.setContentType("text/html");
        out = res.getWriter();

        out.println("<html>");
        out.println("<head><title>Hello World</title></head>");
        out.println("<body>");
        out.println("<h1>Hello World</h1>");
        out.println("</body></html>");
    }
}
```

Figure 130. HelloWorldServlet

To call the servlet directly, you can specify the servlet name directly in the URL as shown in the following example:

```
http://<YourServer>/servlet/HelloWorldServlet
```

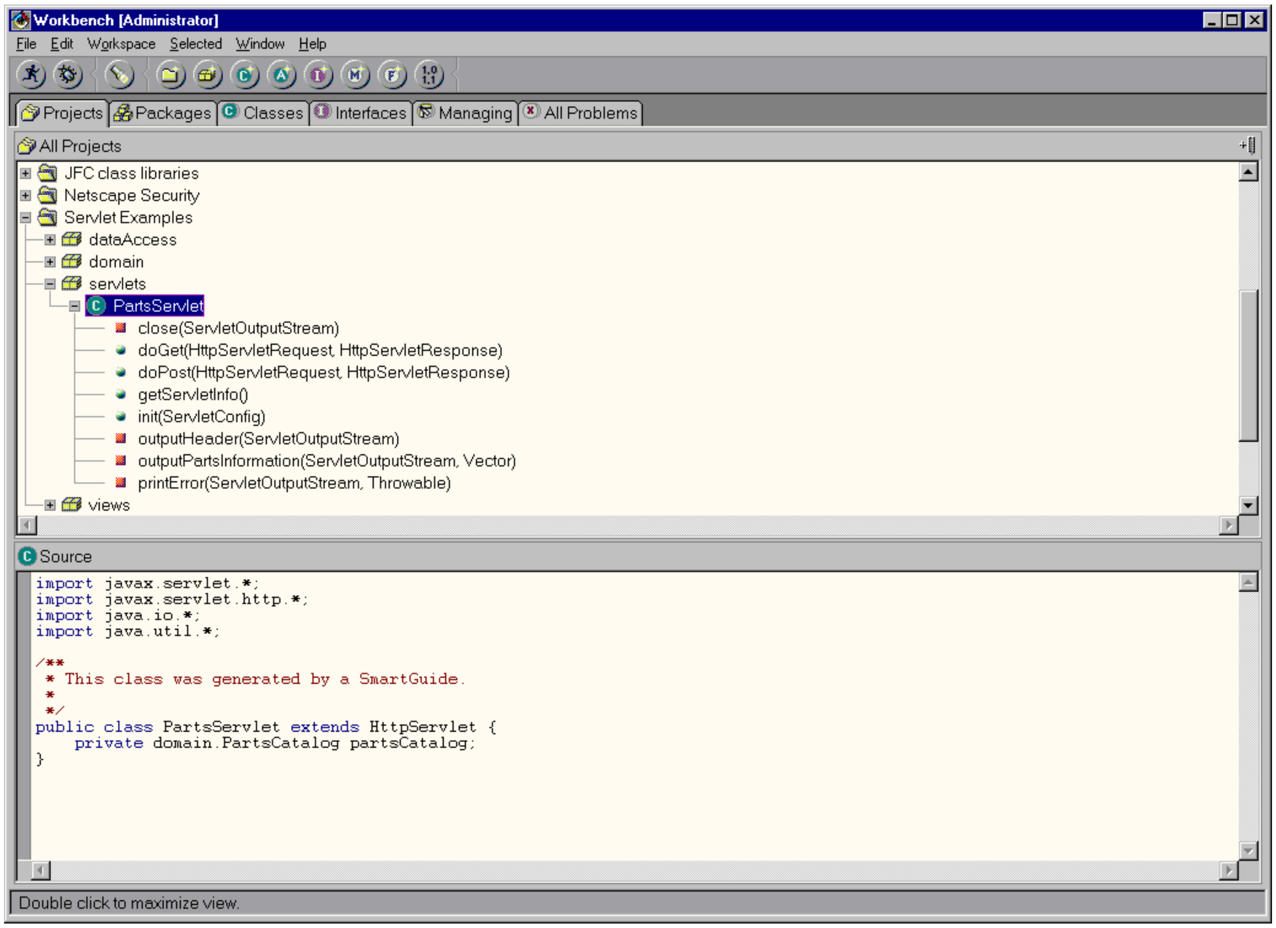
The HelloWorldServlet demonstrates some of the basic concepts, as shown in the following sequence, that are used when creating HTTP servlets:

1. We import support classes from the javax.servlet and javax.servlet.http packages.
2. The HelloWorldServlet extends the HttpServlet class, which implements the Servlet interface.
3. The doGet method is called when a client makes a GET request (the default HTTP request method). If the servlet is called directly, the doGet method is called.
4. We override the doGet method to handle application processing.
5. Two parameters are passed in to the doGet method:
   - HttpServletRequest req—Encapsulates the input
   - HttpServletResponse res—Encapsulates the output
6. Because text data is returned to the client, the reply is sent using a print writer object obtained from the HttpServletResponse object:
   a. We declare a PrintWriter object named out.
   b. We set the response content to "text/html".
   c. We use the HttpServletResponse getWriter method to create the print writer named out.
   d. We use the println method to write HTML tags.
7. The HMTL tags are rendered by the browser.

4.4 Developing the Servlet Application

This section explains the conversion of the applet (discussed in Chapter 3, “Introduction to AS/400 Applets” on page 51) to a servlet.

The PartsServlet servlet was created in VisualAge for Java. Figure 131 shows a view of the ServletExamples project with the servlets package expanded.


Workbench [Administrator]
File Edit Workspace Selected Window Help
Projects
Packages
Classes
Interfaces
Managing
All Problems
All Projects
JFC class libraries
Netscape Security
Servlet Examples
dataAccess
domain
servlets
PartsServlet
close(ServletOutputStream)
doGet(HttpServletRequest, HttpServletResponse)
doPost(HttpServletRequest, HttpServletResponse)
getServletInfo()
init(ServletConfig)
outputHeader(ServletOutputStream)
outputPartsInformation(ServletOutputStream, Vector)
printError(ServletOutputStream, Throwable)
views
Source
Double click to maximize view.

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

/**
 * This class was generated by a SmartGuide.
 *
 */
public class PartsServlet extends HttpServlet {
    private domain.PartsCatalog partsCatalog;
}
```

Figure 131. ServletExamples Project

Table 13 briefly describes the packages, classes and interfaces used.

Table 13. Summary of Packages, Classes, and Interfaces Used for the PartsServlet

ServletExamples Project		
Package	**Class/Interface**	**Description**
dataAccess	DataAccessor	Interface used to define methods that must be implemented by classes in this package.
	JDBCPartsCatalog	Used to connect to AS/400 system database. Returns a vector of parts.
	TestPart	Used to simulate a connection to a database. Returns a vector of parts.
domain	Part	Represents a row of part data retrieved from the database.
	PartsCatalog	Determines which data source to get part data from. Gets parts from the selected data source and returns a vector of parts data.

ServletExamples Project		
Package	**Class/Interface**	**Description**
views	PartsView	The visible part of the applet. Includes the parts listbox and the button to start the query.
servlets	PartsServlet	Provides the servlet support.

Although the application was initially designed, tested, and used as an applet, the intention was to create a body of reusable code that can be used when the application is migrated to a servlet. By adopting the three-tier architecture shown in Figure 132, the application can be easily changed.

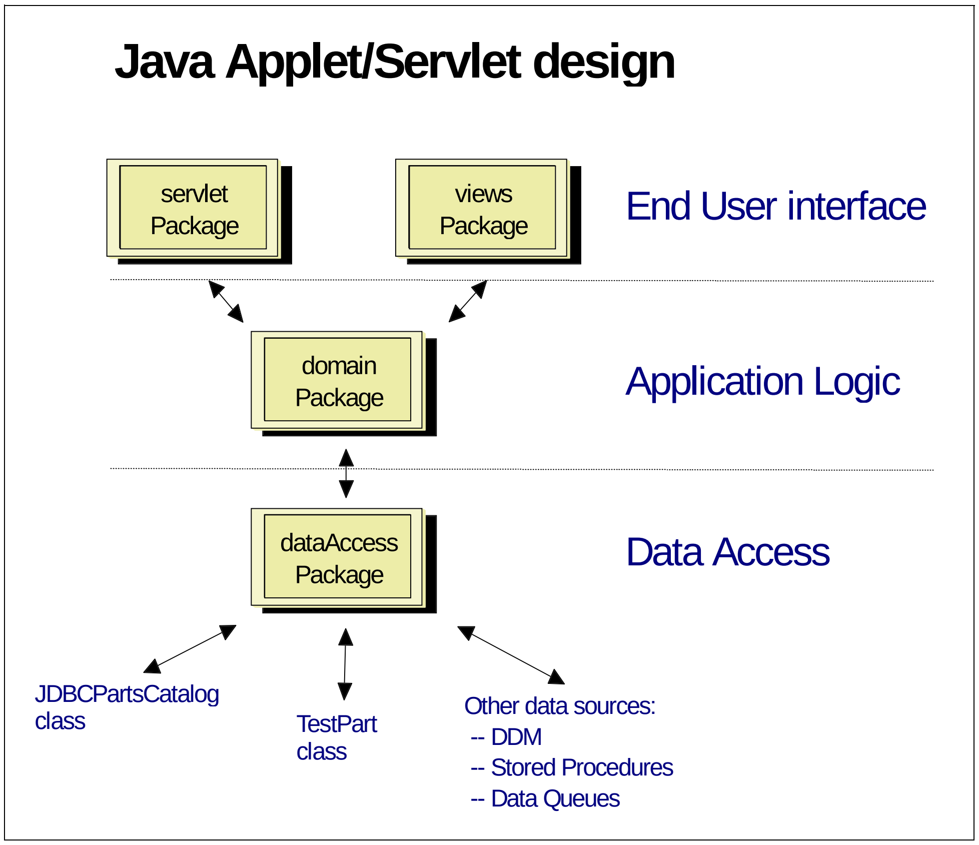


Figure 132. Java Applet/Servlet Design

In fact, it is quite easy to change the application at any of the three tiers. In addition to changing the end user interface code to support an applet or a servlet presentation, the data source can be changed in the data access layer without affecting the other two layers. Also, if any of the application logic needs to be changed, the changes can be made at that layer without affecting either the data access or end user interface layers.

The application is well positioned for change if a different data access technique is used. For example, the IBM AS/400 Toolbox for Java supports record-level access using the AS/400 system Distributed Data Management (DDM) server, program call, and data queues. If you decide to change from the Java Database Connectivity (JDBC) technique, you only need to code your new data access class so that it returns the same type and format of data to the application logic layer. The applet uses the `JDBCPartsCatalog` and `TestPart` classes. The `TestPart` class is used to simulate a connection to a data source.

For detailed information about the dataAccess package, the views package, and the domain package, refer to Chapter 3, “Introduction to AS/400 Applets” on page 51. This chapter covers the servlet package.

4.5 Migrating the Applet to a Servlet

The servlet application is contained in a class named PartsServlet, which resides in the servlet package.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
public class PartsServlet extends HttpServlet {
private domain.PartsCatalog partsCatalog;
}
```

Figure 133. PartsServlet Class

Note the following points in Figure 133:

- A private instance variable named **partsCatalog** is in the class definition. Because of our object-oriented design, we can use the same domain package and dataAccess package that we used for the applet. Here, we declare a PartsCatalog object from the domain package.
- We also import the **javax.servlet** and **javax.servlet.http** packages. To use these packages inside VisualAge for Java, we restore the **Sun JSDK class libraries** project from the repository.

```
public void init (ServletConfig config) throws ServletException {
super.init(config);
log("PartsServlet: init()...");
partsCatalog = new domain.PartsCatalog();
partsCatalog.connectToDB();
log("PartsServlet: init() executed.");
return;
```

Figure 134. Servlet init Method

The init method is called when the servlet is started. Note the following steps:

1. We instantiate the PartsCatalog object named **partsCatalog**.
2. We call the **connectToDB** method. This loads the JDBC driver into the JVM and performs any required JDBC initialization.
3. We use the log method to write messages to the log. This may be useful for debugging an application.

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
log("PartsServlet: doGet()...");
// set the MIME type to text/html
response.setContentType("text/html");
// create the output stream
ServletOutputStream  out = response.getOutputStream();
Vector parts = null;
try {
      log("PartsServlet: doGet(), try{} block ...");
      // write the HTML header to the output stream
      outputHeader(out);

     parts = partsCatalog.getAll();
     log("after get all...");
     // write the HTML table to the output stream
     outputPartsInformation(out, parts);
     close(out);
     log("PartsServlet: doGet(), try{} block executed.");
     return;
}
catch(Throwable e){
     printError(out, e);
}
}
```

Figure 135. The doGet Method

The doGet method is used to retrieve all the records from the Parts database file and display them in a browser.

```
private void outputHeader(ServletOutputStream printStream) throws IOException {
log("PartsServlet: outputHeader()...");
printStream.print("<HTML><HEAD><TITLE>Parts Retrieval</TITLE>");
printStream.println("</HEAD><BODY>");
printStream.print("<BODY BGCOLOR=#C0C0C0>");
log("PartsServlet: outputHeader() executed.");
} // end outputHeader()
```

Figure 136. The outputHeader Method

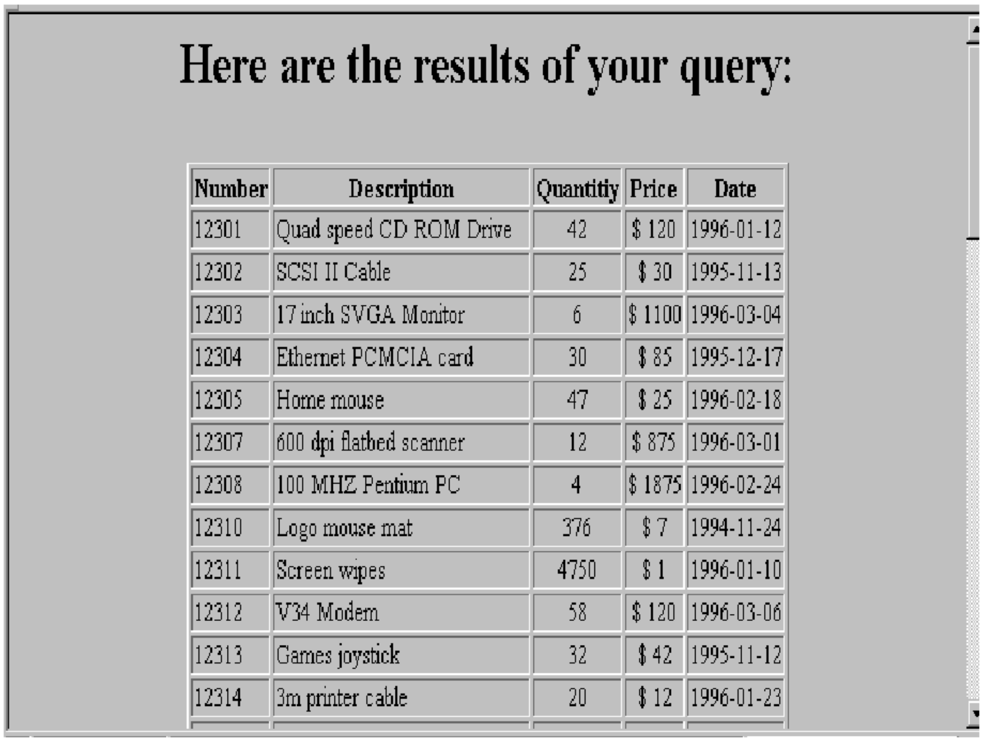
The **outputHeader()** method is called to create the HTML tags, which, when sent to the browser, produce the heading.

We call the partsCatalog object’s **getAll** method to retrieve all the records from the database. The getAll method returns a Vector.

```
private void outputPartsInformation(ServletOutputStream printStream, Vector partsVector) throws
IOException {
log("PartsServlet: outputPartsInformation()...");
Enumeration parts = partsVector.elements();
printStream.println("<TABLE BORDER>");
printStream.println("<P><CENTER><B><FONT SIZE=+3>Here are the results of your query:</FONT></B></P>");
printStream.println("<TR>");
 printStream.print("<TH>Number</TH><TH>Description</TH><TH>Quantitiy</TH><TH>Price</TH><TH>Date</TH>");
printStream.println("</TR>");
while (parts.hasMoreElements()) {
domain.Part aPart = ((domain.Part)parts.nextElement());
printStream.print("<TD>" + aPart.getNumber() + "</TD>");
printStream.print("<TD>" + aPart.getDescription() + "</TD>");
printStream.print("<TD><CENTER>" + aPart.getQuantity() + "</CENTER></TD>");
printStream.print("<TD><CENTER>$ " + aPart.getPrice() + "</CENTER></TD>");
printStream.print("<TD><CENTER>" + aPart.getDate() + "</CENTER></TD>");
printStream.println("</TR>");
}; // end while
printStream.println("</TABLE>");
printStream.println("</FONT></BODY></HTML>");
log("PartsServlet: outputPartsInformation() executed.");
} // end outputPartsInformation()
```

Figure 137. The outputPartsInformation Method

We call the **outputPartsInformation** to build an HTML table which contains all the records. We then send the HTML table to the browser for display. The output is shown in Figure 138.



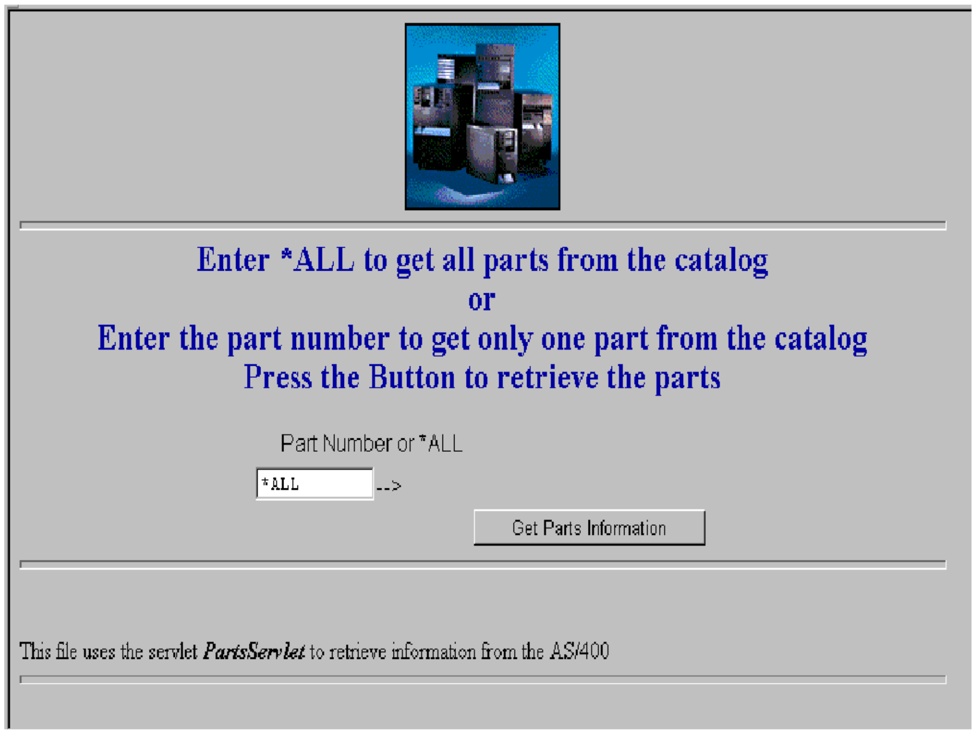
Here are the results of your query:

Number	Description	Quantitiy	Price	Date
12301	Quad speed CD ROM Drive	42	$ 120	1996-01-12
12302	SCSI II Cable	25	$ 30	1995-11-13
12303	17 inch SVGA Monitor	6	$ 1100	1996-03-04
12304	Ethernet PCMCIA card	30	$ 85	1995-12-17
12305	Home mouse	47	$ 25	1996-02-18
12307	600 dpi flatbed scanner	12	$ 875	1996-03-01
12308	100 MHZ Pentium PC	4	$ 1875	1996-02-24
12310	Logo mouse mat	376	$ 7	1994-11-24
12311	Screen wipes	4750	$ 1	1996-01-10
12312	V34 Modem	58	$ 120	1996-03-06
12313	Games joystick	32	$ 42	1995-11-12
12314	3m printer cable	20	$ 12	1996-01-23

Figure 138. The doGet Method Output

4.5.1 Enhancing the Servlet

This section shows how to enhance the servlet to allow user input and to use an HTML file to control the servlet. The enhanced servlet is shown in Figure 139.



Enter *ALL to get all parts from the catalog

or

Enter the part number to get only one part from the catalog

Press the Button to retrieve the parts

Part Number or *ALL

*ALL -->

Get Parts Information

This file uses the servlet ***PartsServlet*** to retrieve information from the AS/400

Figure 139. Enhanced Servlet

To produce the enhanced servlet, perform the following steps:

1. Add the doPost method to the PartsServlet class.
2. Create an HTML file to control running the servlet.

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

  // set the MIME type to text/html
  response.setContentType("text/html");
  // create the output stream
  ServletOutputStream  out = response.getOutputStream();
  try {
      // get the value from the input field named part (see HTML file)
      // and check if value is valid
      String[] parameter = request.getParameterValues("partno");
      if (parameter[0].trim().length() == 0) parameter[0] = "*ALL";
      Vector parts = null;
      // write the HTML header to the output stream
      outputHeader(out);
      // retrieve all data from the database
      parts = partsCatalog.getAll();
      // process the request according to the value of the input field
      if (!parameter[0].toUpperCase().equals("*ALL")) {
         Vector selectedParts = new Vector();
         Enumeration partsE = parts.elements();
         while (partsE.hasMoreElements()){
            domain.Part aPart = (domain.Part)partsE.nextElement();
            if ((aPart.getNumber()).equals((new java.math.BigDecimal(parameter[0])))){
              selectedParts.addElement(aPart);
            };
          };
          parts = selectedParts;
    };
    // write the HTML table to the output stream
    outputPartsInformation(out, parts);
    close(out);
    return;
    }
  catch(Throwable e){
    printError(out, e);
  }
} // end of doPost()
```

Figure 140. The doPost Method

In Figure 140, we show the doPost method. We use the outputHeader, getAll, and outputPartsInformation methods in exactly the same way as in the doGet method.

We use the HttpServletRequest class getParameterValues method to read the value from the **partno** field. We need to define the **partno** field in the HTML file. If the parameter is *all, we return all parts. Otherwise, we return only the part record requested.

Notice that we use the getAll method to retrieve all the records from the database. If only one record is requested from the database, we find it in the Vector containing all the records and return only that record. Since the database is small, this does not cause a performance problem. If we actually implemented this in a production environment, we would add a getPart method, which would retrieve only one part record.

```
<HTML>
<BODY BGCOLOR="#C0C0C0">
<FORM action="/servlet/PartsServlet" method="POST">
<CENTER><IMG SRC="as400.gif" BORDER=2 X-SAS-UseImageWidth X-SAS-UseImageHeight HEIGHT=120
WIDTH=120></CENTER>
<CENTER>
<HR SIZE="5"></CENTER>
<CENTER><B><FONT COLOR="#0000AF"><FONT SIZE=+2>Enter *ALL to get all parts
from the catalog</FONT></FONT></B></CENTER>
<CENTER><B><FONT COLOR="#0000AF"><FONT SIZE=+2>or</FONT></FONT></B></CENTER>
<CENTER><B><FONT COLOR="#0000AF"><FONT SIZE=+2>Enter the part number to
get only one part from the catalog</FONT></FONT></B></CENTER>
<CENTER><B><FONT COLOR="#0000AF"><FONT SIZE=+2>Press the Button to retrieve
the parts</FONT></FONT></B></CENTER>
<CENTER>&nbsp;</CENTER>
<CENTER><TABLE BORDER=0 WIDTH="50%" HEIGHT="1" >
<TR>
<TD WIDTH="169" HEIGHT="14">
<DIV ALIGN=right><FONT FACE="Arial,Helvetica">Part Number or *ALL</FONT>&nbsp;</DIV>
</TD>
</TR>
<TD WIDTH="118" HEIGHT="14">
<!-- Add the input field after this line -->
<INPUT TYPE="text" NAME="partno" VALUE="*ALL" SIZE=10 MAXLENGTH=10>--></TD>
</TR>
<TR>
<TD WIDTH="169"></TD>
<TD WIDTH="118"><INPUT TYPE="submit" NAME="Submit" VALUE="Get Parts Information"></TD>
</TR>
</TABLE></CENTER>
<CENTER>
<HR SIZE="5"></CENTER>
</FORM>
<BR>This file uses the servlet <B><I>PartsServlet</I></B> to retrieve information
from the AS/400
<BR>
<HR SIZE=5 WIDTH="100%">
</BODY></HTML>
```

Figure 141. Servlet HTML File (Parts.html)

Figure 141 shows the source for the HTML file that we use to run the servlet. We use the FORM action tag to specify which servlet to run and the method to use. In this case, we use the **POST** method. We use the INPUT TYPE tag to define an input field named **partno**. The name we use here must match the parameter name that we use in the doPost method.

4.6 Executing the Servlet

After creating the servlet, we are ready to deploy the servlet on an HTTP server. The application discussed in this chapter uses servlets to access AS/400 resources. To run it, we need a server that supports Java servlets. We tested the application using two servers:

- The Domino Go Webserver running on a Windows/NT platform
- The IBM HTTP Server for AS/400 running on an AS/400 system

4.6.1 Running under the Domino Go Webserver

The Domino Go Webserver is part of the recently announced IBM WebSphere software family. It is a scalable, high-performance Web server that is available, in addition to OS/390, on many workstation platforms (AIX, Solaris, HP-UX, OS/2 Warp, Windows NT, and Windows 95).

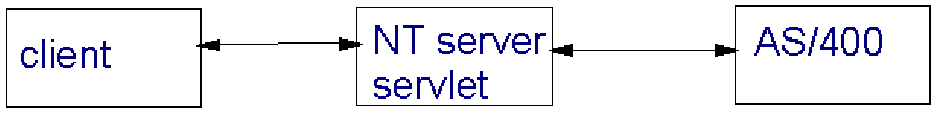


Figure 142. Three-Tier Servlet Architecture

The Domino Go Webserver includes:

- State-of-the-art security
- Site indexing capabilities
- Advanced server statistics reporting
- Relational database connectivity with Net.Data
- Support for Java servlets on all platforms
- Support for JDK 1.1
- Support for the JIT (Just-in-Time) compiler
- HTTP 1.1
- Web-site content rating

We use the combination of Domino Go Webserver and ServletExpress to provide a three-tier implementation for the servlet:

- **Client**—Provides the end-user interface
- **NT server**—Provides the HTTP server
- **AS/400**—Provides the database server

For the Domino Go Webserver and ServletExpress, we export the classes to the **\ServletExpress\servlets** directory on the NT server.

To view the configuration steps for Domino Go Webserver, see Section 9.1, “Domino Go Webserver” on page 287.

Note

You can run both the client and Domino Go Webserver on the same hardware platform. This allows you to test the three-tier implementation using two hardware platforms.

To export the Java classes, follow these steps:

1. In the Workbench, select the packages **dataAccess**, **domain**, and **servlets**.
2. From the File menu, select **Export**. In the dialog that follows, select only the **Class Files** option.
3. Ensure that the **Directory radio button** is selected.
4. Press the **Next** button.
5. In this dialog, specify the path to the **x:\ServletExpress\servlets** directory. Where x = a drive on the NT server.
6. Export only the class files.
7. Press the **Finish** button.

4.6.1.1 Configure ServletExpress

Now you are ready to configure ServletExpress. ServletExpress allows you manage the servlets running under the control of the Web server. You do not have to use ServletExpress to run servlets under the Domino Go HTTP server if they are stored in the default directory. For the PartsServlet class, it is stored in a package named servlets. When you export it, it is exported to a subdirectory named servlets in the default directory. You need to configure ServletExpress so it can find the PartsServlet class. ServletExpress also makes it easy for you to load and unload servlets without stopping the Web server. To configure ServletExpress, see Section 9.2, “ServletExpress” on page 289.

4.6.1.2 Running the PartsServlet

After these configuration steps, you are ready to test your servlet. Point your Web browser to the following URL: http://server:xxxx/servlet/PartsServlet

Press the **Enter** key. Replace server with the name of your server and xxxx with the port you are using (by default, Domino Go uses port 0080).

If everything is configured properly, you should see the table with all the parts in it as shown in Figure 138 on page 171.

But what if not everything works as expected? Are there any means to debug or trace the program? Maybe you noticed in some of the PartsServlet's methods that the log() method is called. When you call this method, the arguments are logged on the server. This is similar to what you are normally doing when using System.out.println().

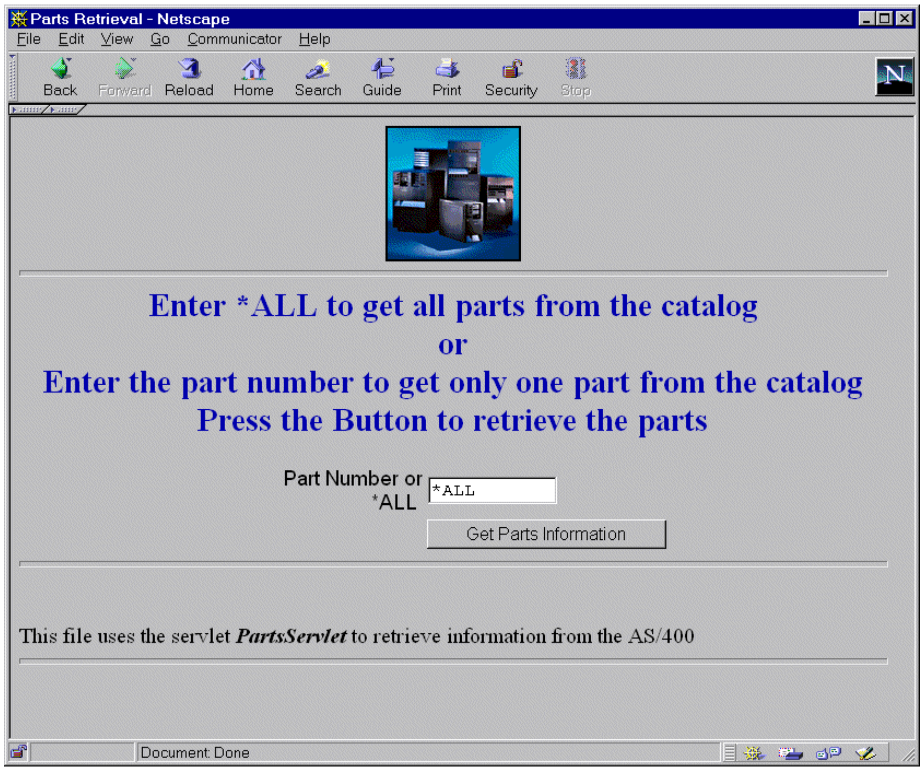
You can find the log file used by the Domino Go Webserver and ServletExpress in the directory \ServletExpress\logs\ncfservice under the name of event_log. When you open this file using NotePad, for example, you can see that there are some lines at the end that were produced by the PartsServlet class. This logging facility can be helpful when trying to find program problems.

4.6.1.3 Running the Enhanced Servlet

To run the servlet using the HTML file discussed in Section 4.5.1, “Enhancing the Servlet” on page 172, in the browser, enter:

```
http://server/apptest/Parts.html
```

The page shown in Figure 143 on page 177 appears. In this scenario, you can select either one specific part or all parts from the parts database.


Parts Retrieval - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Document: Done

Enter *ALL to get all parts from the catalog
or
Enter the part number to get only one part from the catalog
Press the Button to retrieve the parts

Part Number or *ALL: *ALL

Get Parts Information

This file uses the servlet *PartsServlet* to retrieve information from the AS/400

Figure 143. Enhanced Servlet

4.6.2 Running under the IBM HTTP Server for AS/400

We use the combination of the IBM HTTP Server for AS/400 and the WebSphere Application Server to provide a two-tier implementation for the servlet. This support is available with OS/400 V4R3 and later. In Figure 144, *client* provides the end user interface and *AS/400* provides the HTTP server and the database server.

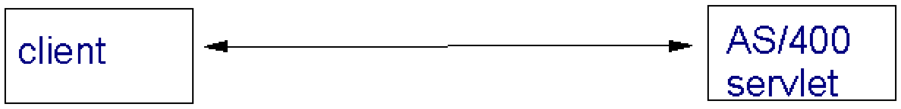


Figure 144. Two-Tier Servlet Architecture

For the IBM HTTP Server for AS/400 and the WebSphere Application Server, we export the classes to the **\QIBM\ProdData\IBMWebAS\servlets** directory in the AS/400 Integrated File System.

To view the configuration steps for the IBM HTTP Server for AS/400, see Section 9.3, “IBM HTTP Server for AS/400” on page 293.

4.6.3 Running Servlets on the AS/400 System

The IBM WebSphere Application Server is IBM's Java servlet-based Web application server that helps you deploy and manage Web applications. They range from simple Web sites to powerful e-business solutions. For detailed information on how to obtain and use the IBM WebSphere Application Server for AS/400, see the following URL: http://www.as400.ibm.com/HTTP

You can configure a server instance of the IBM HTTP Server for AS/400 to run WebSphere. It is enabled through the Server API of the HTTP Server. For information about how to configure the IBM HTTP Server for AS/400 and the IBM WebSphere Application Server for AS/400, see Chapter 9, “HTTP Server Configuration” on page 287. For an entire listing of the IBM HTTP Server for AS/400 configuration file used for this redbook, refer to Appendix B, “IBM HTTP Server for AS400 Configuration” on page 301.

4.6.4 Running the PartsServlet Servlet on the AS/400 System

Once you configure the IBM HTTP Server for AS/400 and the WebSphere application server to serve servlets, you can run the PartsServlet servlet on the AS/400 system. Enter:

```
http://yourAS400Server:xxxx/servlet/PartsServlet
```

To run the servlet using the HTML file, enter:

```
http://yourAS400Server:xxxx/apptest/Parts.html
```

The variable `yourAS400Server` is the name of your AS/400 system. The variable `xxxx` is equal to the TCP/IP port over which you are running the IBM HTTP Server for AS/400.

Chapter 5. Overview of the Order Entry Application

This chapter covers an example RPG order entry application. This application represents a commercial application, although it does not include all the necessary error handling a business application requires. In Chapter 6, “Developing AS/400 Java Applets” on page 193, we convert this RPG application to a Java Internet-based application.

5.1 Overview of the Order Entry Application

This section provides an overview of the application and a description of how the application database is used.

5.1.1 The ABC Company

The ABC Company is a wholesale supplier with one warehouse and 10 sales districts. Each district serves 3000 customers (30000 total customers for the company). The warehouse maintains stock for the 100000 items sold by the Company.

The following diagram illustrates the company structure (warehouse, district, and customer).

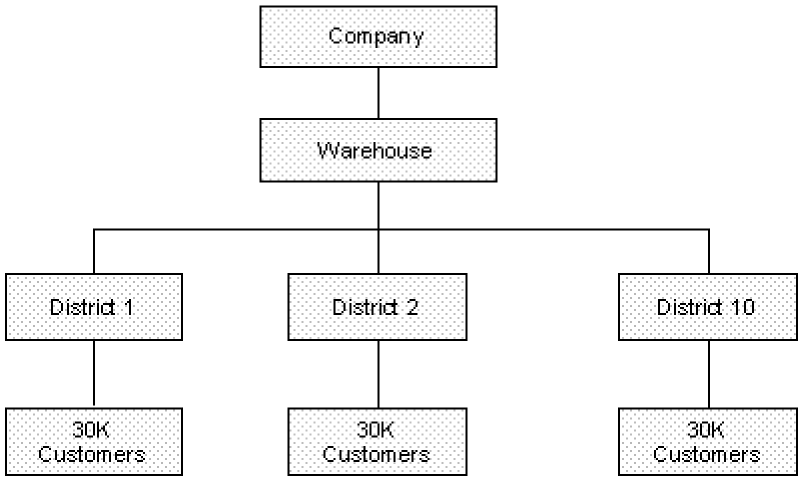


Figure 145. The Company Structure

5.1.2 The ABC Company Database

The company runs its business with a database. This database is used in a mission critical, OLTP (online transaction processing) environment. The database includes tables with the following data:

- District information (next available order number, tax rate, and so on)
- Customer information (name, address, telephone number, and so on)
- Order information (date, time, shipper, and so on)
- Order line information (quantity, delivery date, and so on)
- Item information (name, price, item ID, and so on)
- Stock information (quantity in stock, warehouse ID, and so on.)

5.1.3 A Customer Transaction

A customer transaction occurs based on the following series of events:

1. Customers telephone one of the 10 district centers to place an order.
2. The district customer service representative answers the telephone, obtains the following information, and enters it into the application:
   - Customer number
   - Item numbers of the items the customer wants to order
   - The quantity required for each item
3. The customer service representative may prompt for a list of customers or a list of parts.
4. The application then performs the following actions:
   a. Reads the customer last name, customer discount rate, and customer credit status from the Customer Table (CSTMR).
   b. Reads the District Table for the next available district order number. The next available district order number increases by one and is updated.
   c. Reads the item names, item prices, and item data for each item ordered by the customer from the Item Table (ITEM).
   d. Checks if the quantity of ordered items is in stock by reading the quantity in the Stock Table (STOCK).
5. When the order is accepted, the following occurs:
   a. Inserts a new row into the Order Table to reflect the creation of the new order (ORDERS).
   b. A new row is inserted into the Order Line Table to reflect each item in the order.
   c. The quantity is reduced by the quantity ordered.
   d. A message is written to a data queue to initiate order printing.

5.1.4 Application Flow

The RPG Order Entry Application consists of the following components:

Note

To download the sample code used in this redbook, please refer to Appendix A.1, “Downloading the Files from the Internet Web Site” on page 299, for more information.

- ORDENTD (Parts Order Entry)—Display File
- ORDENTR (Parts Order Entry)—Main RPG processing program
- PRTORDERP (Parts Order Entry)—Print File
- PRTORDERR (Print Orders)—RPG server job
- SLTCUSTD (Select Customer)—Display file
- SLTCUSTR (Select Customer)—RPG SQL stored procedure
- SLTPARTD (Select Part)—Display file
- SLTPARTR (Select Part)—RPG stored procedure

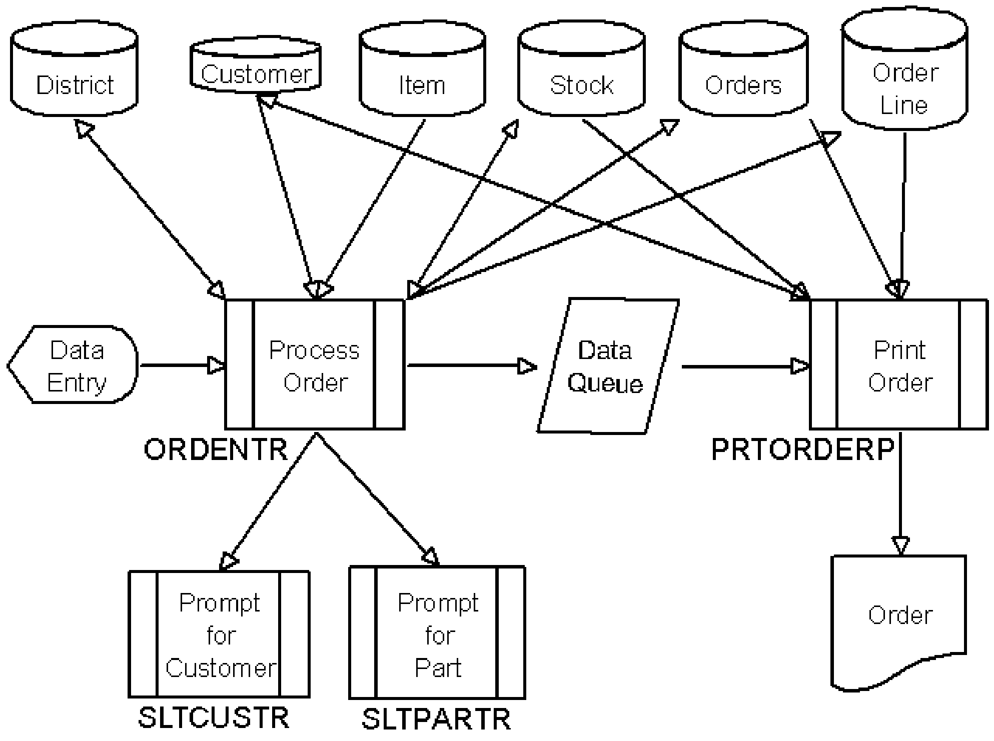


Figure 146. RPG Application Flow

ORDENTR is the main RPG program. It is responsible for the main line processing. It calls two supporting RPG programs that are used to prompt for and select end-user input. They are **SLTCUSTR** which handles selecting a customer, and **SLTPARTR** which handles selecting part numbers. **PRTODERR** is an RPG program that handles printing customer orders. It reads order records that were placed on a data queue and prints them in a background job.

5.1.5 Customer Transaction Flow

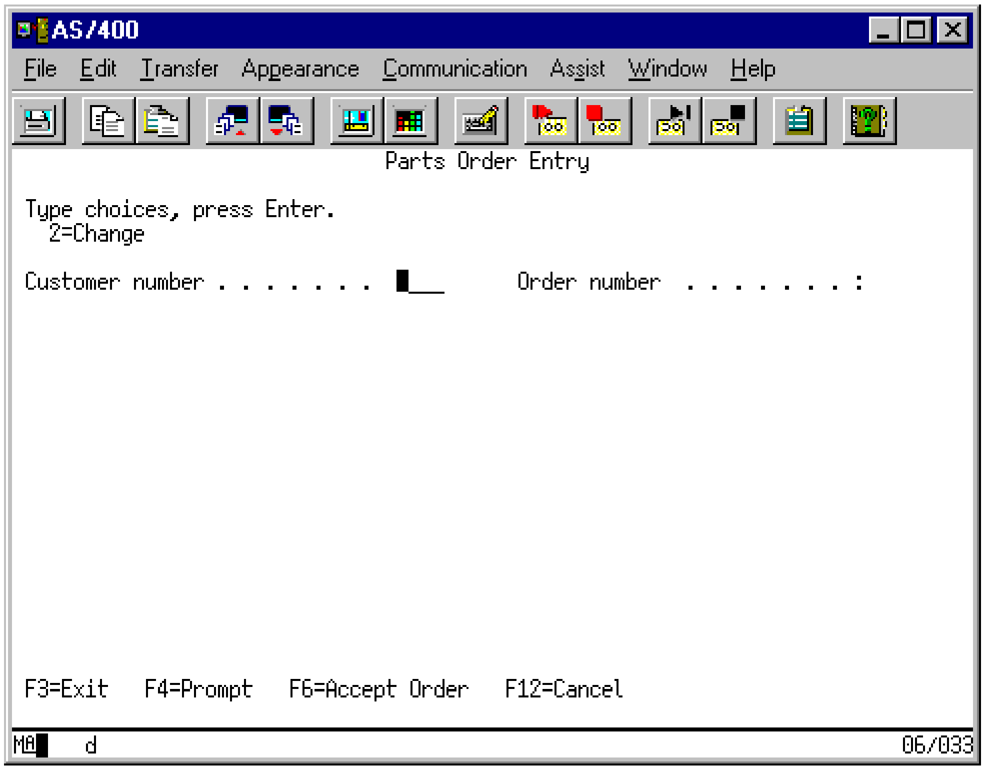
The following scenario walks through a customer transaction showing the application flow. By understanding the flow of the AS/400 application, you can understand the changes made to this application to support a graphical client.

5.1.5.1 Starting the Application

To start the application, the customer calls the main program from an AS/400 command line:

```
CALL ORDENTR
```

When the order entry application is started, the display in Figure 147 on page 182 appears.


AS/400
File Edit Transfer Appearance Communication Assist Window Help

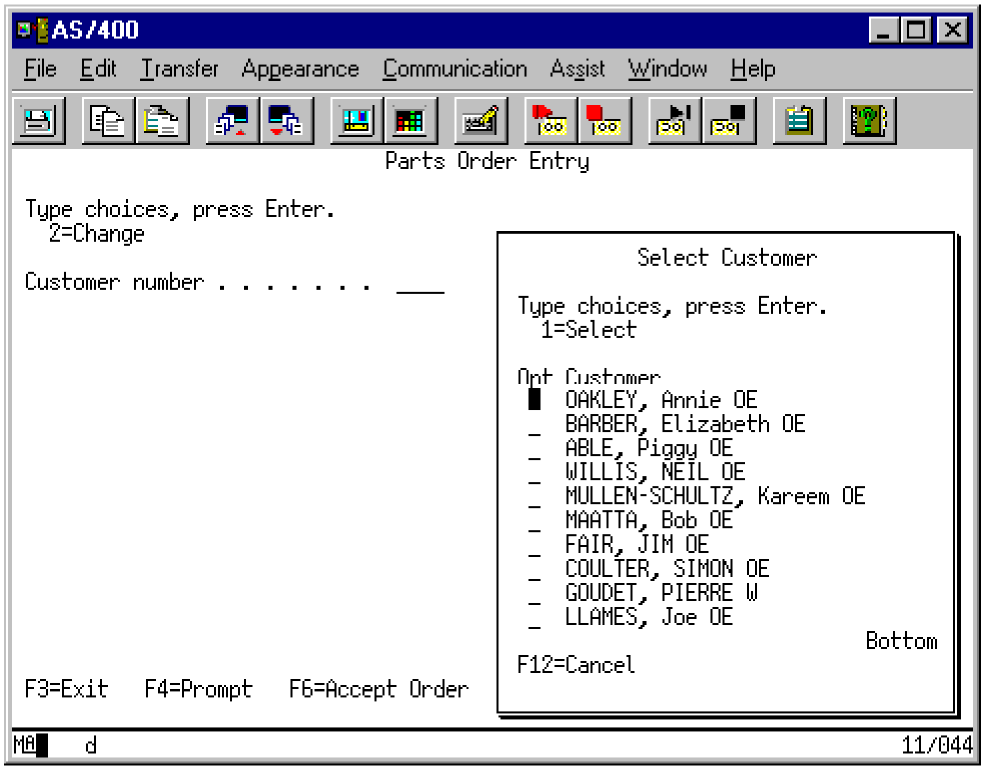
```
                                   Parts Order Entry

Type choices, press Enter.
  2=Change

Customer number . . . . . . . ____      Order number . . . . . . . :









F3=Exit   F4=Prompt   F6=Accept Order   F12=Cancel
MA     d                                                                   06/033
```

Figure 147. Parts Order Entry

When the Parts Order Entry display appears, the user has two options:

- Type in a customer number and press the Enter key
- End the program by pressing either F3 or F12.

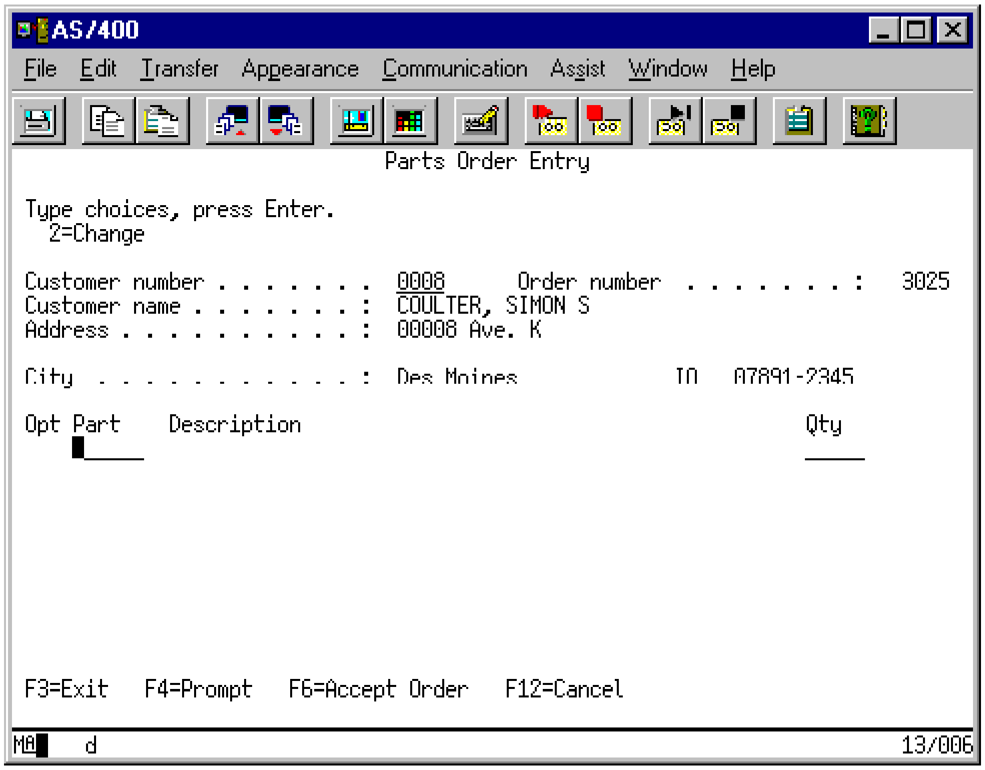
If they do not know the customer number, the user can press F4 to view a window containing a list of available customers.


AS/400
File Edit Transfer Appearance Communication Assist Window Help

```
                             Parts Order Entry

Type choices, press Enter.
  2=Change                              Select Customer

Customer number . . . . . . .  ____    Type choices, press Enter.
                                          1=Select

                                        Opt Customer
                                         █  OAKLEY, Annie OE
                                         _  BARBER, Elizabeth OE
                                         _  ABLE, Piggy OE
                                         _  WILLIS, NEIL OE
                                         _  MULLEN-SCHULTZ, Kareem OE
                                         _  MAATTA, Bob OE
                                         _  FAIR, JIM OE
                                         _  COULTER, SIMON OE
                                         _  GOUDET, PIERRE W
                                         _  LLAMES, Joe OE
                                                                Bottom
                                        F12=Cancel
F3=Exit   F4=Prompt   F6=Accept Order

MA█   d                                                        11/044
```

Figure 148. Select Customer

The user presses F12 to remove the window and return to the initial panel, or scrolls through the items in the list until they find the customer they want. By typing a 1 in the option field and pressing the Enter key, the user indicates their choice. The selected customer is then returned to the initial panel.

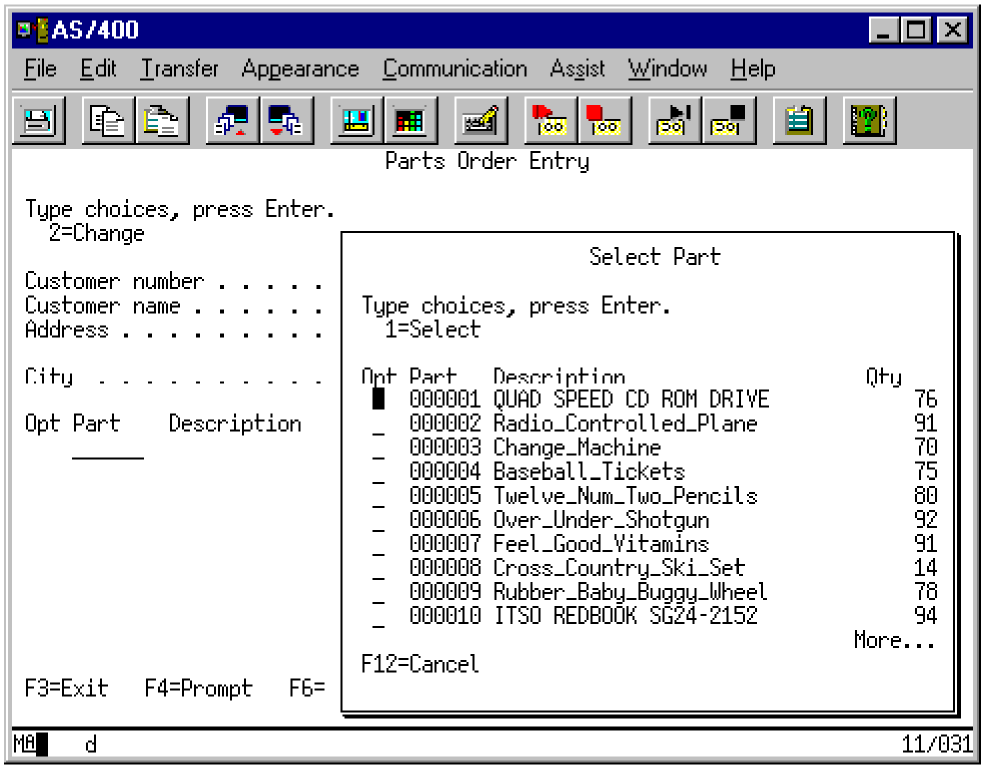

AS/400
File Edit Transfer Appearance Communication Assist Window Help

```
                               Parts Order Entry

Type choices, press Enter.
  2=Change

Customer number . . . . . . .  0008      Order number  . . . . . . . :   3025
Customer name . . . . . . . :  COULTER, SIMON S
Address . . . . . . . . . . :  00008 Ave. K

City  . . . . . . . . . . . :  Des Moines              IO   07891-2345

Opt Part     Description                                         Qty
    _____                                                        _____






F3=Exit    F4=Prompt    F6=Accept Order    F12=Cancel
MA     d                                                            13/006
```

Figure 149. Parts Order Entry

After selecting a customer from the list, or typing a valid customer number and pressing the Enter key, the customer details are shown and an order number is assigned. An additional prompt is displayed allowing the user to type a part number and quantity.

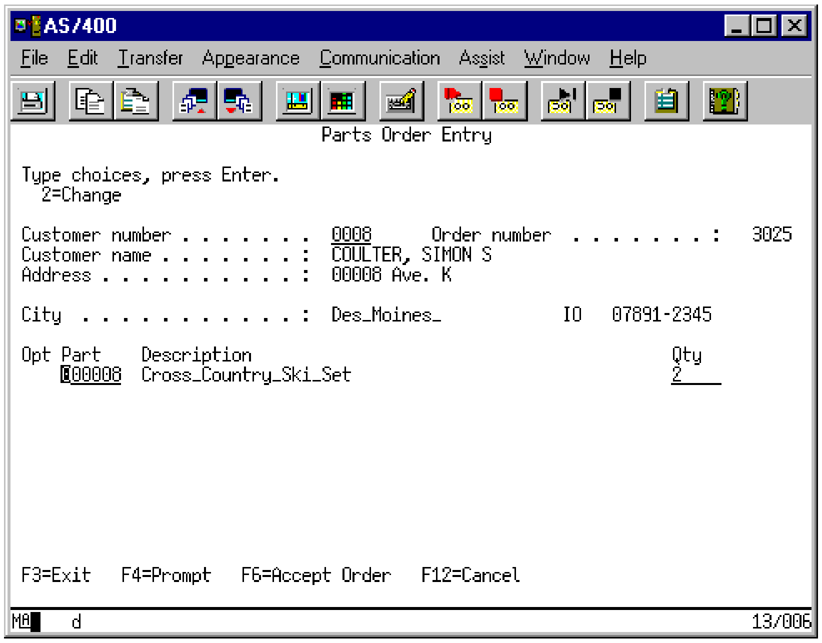
If the user does not know the part number, they can press F4 to view a window containing a list of available parts.


AS/400
File Edit Transfer Appearance Communication Assist Window Help

```
                        Parts Order Entry

Type choices, press Enter.
  2=Change                ┌─────────────────────────────────────────────────┐
                          │                  Select Part                    │
Customer number . . . . . │                                                 │
Customer name . . . . . . │ Type choices, press Enter.                      │
Address . . . . . . . . . │   1=Select                                      │
                          │                                                 │
City  . . . . . . . . . . │ Opt Part    Description                  Qty    │
                          │  █  000001 QUAD SPEED CD ROM DRIVE          76  │
Opt Part     Description  │  _  000002 Radio_Controlled_Plane           91  │
    ______                │  _  000003 Change_Machine                   70  │
                          │  _  000004 Baseball_Tickets                 75  │
                          │  _  000005 Twelve_Num_Two_Pencils           80  │
                          │  _  000006 Over_Under_Shotgun               92  │
                          │  _  000007 Feel_Good_Vitamins               91  │
                          │  _  000008 Cross_Country_Ski_Set            14  │
                          │  _  000009 Rubber_Baby_Buggy_Wheel          78  │
                          │  _  000010 ITSO REDBOOK SG24-2152           94  │
                          │                                        More...  │
                          │ F12=Cancel                                      │
F3=Exit    F4=Prompt   F6=│                                                 │
                          └─────────────────────────────────────────────────┘
MA█    d                                                              11/031
```

Figure 150. Select Part

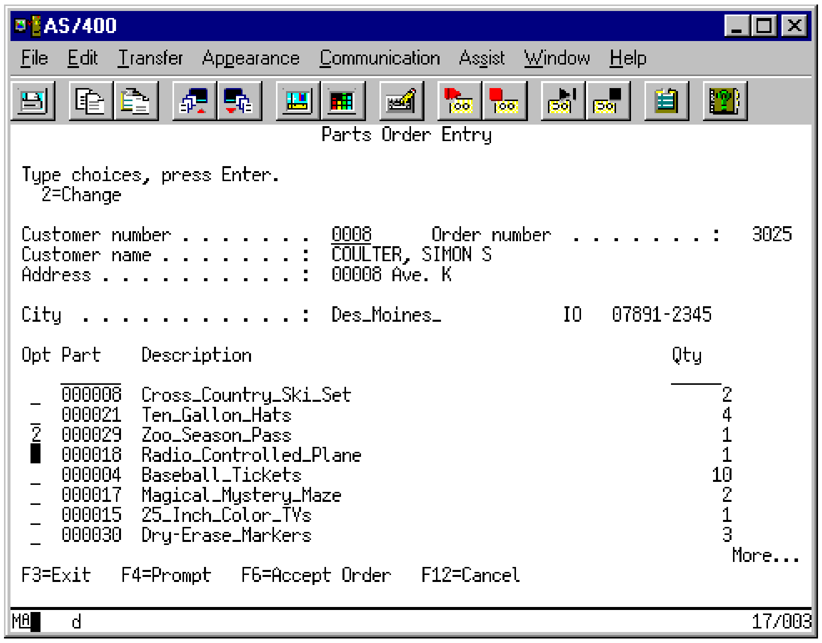
The user presses F12 to remove the window and return to the initial panel, or scrolls through the items in the list until they find the part they want. By typing a 1 in the option field and pressing the Enter key, they indicate their choice. The selected part is returned to the initial panel.


AS/400
File Edit Transfer Appearance Communication Assist Window Help

```
                        Parts Order Entry

Type choices, press Enter.
  2=Change

Customer number . . . . . . . :  0008      Order number  . . . . . . . :   3025
Customer name . . . . . . . . :  COULTER, SIMON S
Address . . . . . . . . . . . :  00008 Ave. K

City  . . . . . . . . . . . . :  Des_Moines_           IO   07891-2345

Opt Part     Description                                      Qty
    █00008   Cross_Country_Ski_Set                            2___





F3=Exit    F4=Prompt    F6=Accept Order    F12=Cancel
MA█    d                                                             13/006
```

Figure 151. Parts Order Entry

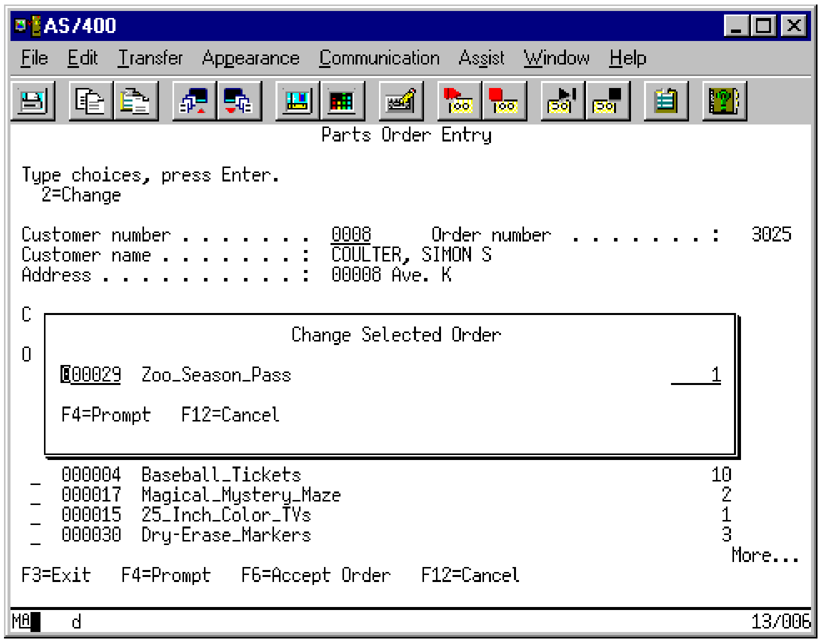
After selecting a customer from the list, or typing a valid customer number and pressing the Enter key, the part and quantity ordered are added to the list section below the part entry fields.


AS/400
File Edit Transfer Appearance Communication Assist Window Help

```
                              Parts Order Entry

Type choices, press Enter.
  2=Change

Customer number . . . . . . . :  0008      Order number  . . . . . . . :   3025
Customer name . . . . . . . . :  COULTER, SIMON S
Address . . . . . . . . . . . :  00008 Ave. K

City  . . . . . . . . . . . . :  Des_Moines_           IO   07891-2345

Opt Part    Description                                          Qty

 _  000008  Cross_Country_Ski_Set                                     2
 _  000021  Ten_Gallon_Hats                                           4
 2  000029  Zoo_Season_Pass                                           1
 _  000018  Radio_Controlled_Plane                                    1
 _  000004  Baseball_Tickets                                         10
 _  000017  Magical_Mystery_Maze                                      2
 _  000015  25_Inch_Color_TVs                                         1
 _  000030  Dry-Erase_Markers                                         3
                                                                   More...
F3=Exit   F4=Prompt   F6=Accept Order   F12=Cancel

MA     d                                                             17/003
```

Figure 152. Parts Order Entry

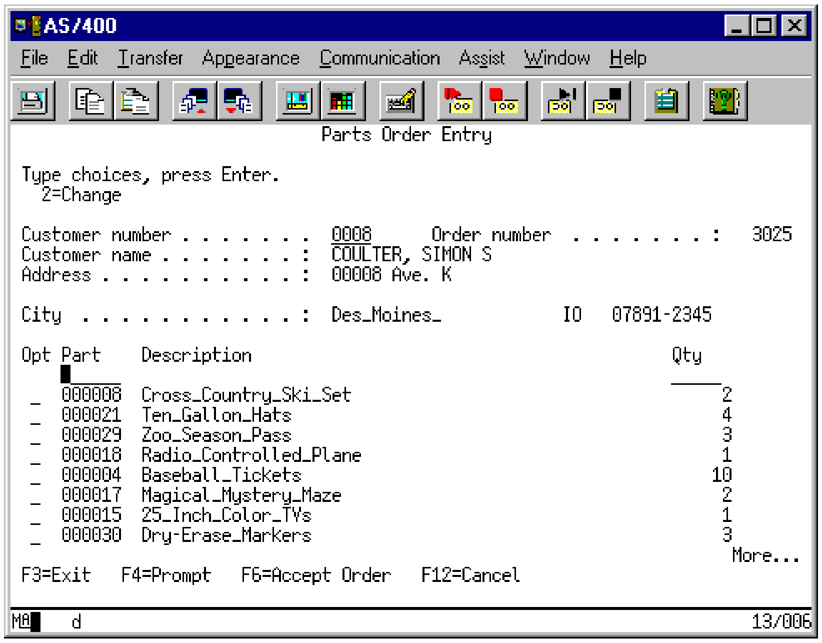
The user may type a 2 beside an entry in the list to change the order. When the Enter key is pressed, a window appears that allows the order line to be changed.


AS/400
File Edit Transfer Appearance Communication Assist Window Help

```
                              Parts Order Entry

Type choices, press Enter.
  2=Change

Customer number . . . . . . . :  0008      Order number  . . . . . . . :   3025
Customer name . . . . . . . . :  COULTER, SIMON S
Address . . . . . . . . . . . :  00008 Ave. K

C  ┌──────────────────────────────────────────────────────────────────┐
   │                     Change Selected Order                        │
O  │                                                                  │
   │ 000029  Zoo_Season_Pass                                      ___1 │
   │                                                                  │
   │ F4=Prompt   F12=Cancel                                           │
   │                                                                  │
   └──────────────────────────────────────────────────────────────────┘
 _  000004  Baseball_Tickets                                         10
 _  000017  Magical_Mystery_Maze                                      2
 _  000015  25_Inch_Color_TVs                                         1
 _  000030  Dry-Erase_Markers                                         3
                                                                   More...
F3=Exit   F4=Prompt   F6=Accept Order   F12=Cancel

MA     d                                                             13/006
```

Figure 153. Change Selected Order

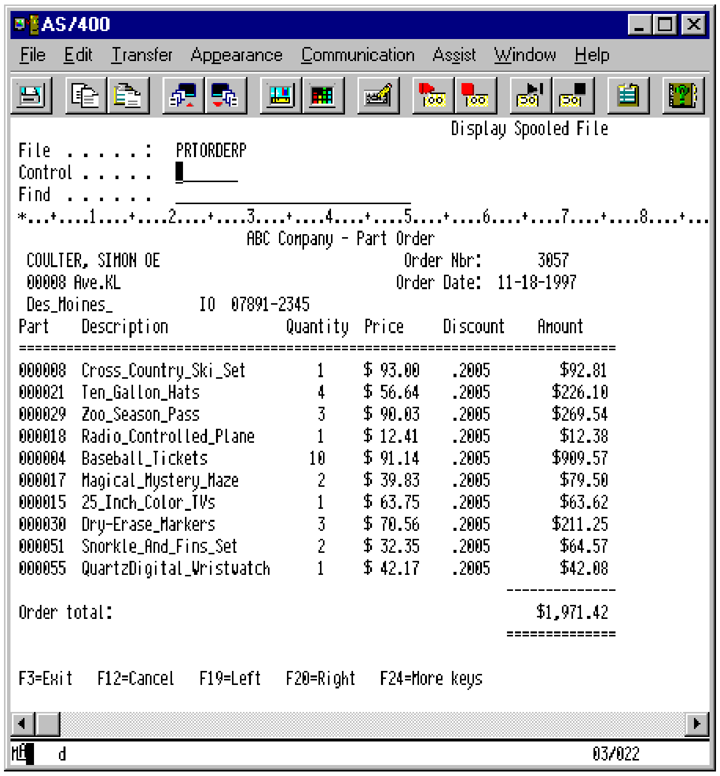
The user chooses to press F12 to cancel the change, press F4 to list the parts, or type a new part identifier or different quantity. Pressing the Enter key validates the part identifier and quantity. If valid, the order line is changed in the list and the window is closed.


AS/400
File Edit Transfer Appearance Communication Assist Window Help

```
                              Parts Order Entry

Type choices, press Enter.
  2=Change

Customer number . . . . . . . :  0008      Order number  . . . . . . . :   3025
Customer name . . . . . . . . :  COULTER, SIMON S
Address . . . . . . . . . . . :  00008 Ave. K

City  . . . . . . . . . . . . :  Des_Moines_           IO   07891-2345

Opt Part    Description                                          Qty
    █_____
                                                                 _____
 _  000008  Cross_Country_Ski_Set                                     2
 _  000021  Ten_Gallon_Hats                                           4
 _  000029  Zoo_Season_Pass                                           3
 _  000018  Radio_Controlled_Plane                                    1
 _  000004  Baseball_Tickets                                         10
 _  000017  Magical_Mystery_Maze                                      2
 _  000015  25_Inch_Color_TVs                                         1
 _  000030  Dry-Erase_Markers                                         3
                                                                       More...
F3=Exit   F4=Prompt   F6=Accept Order   F12=Cancel

MA█    d                                                                13/006
```

Figure 154. Completed Order

In Figure 154, you see the quantity for Zoo Season Pass is changed to 3. When the order is complete, the user presses F6 to update the database. Then, an order is placed on the data queue for printing.


AS/400
File Edit Transfer Appearance Communication Assist Window Help

```
                                             Display Spooled File
File  . . . . . :   PRTORDERP
Control . . . . .   █_______
Find  . . . . . .   ______________________________
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+...
                              ABC Company - Part Order
 COULTER, SIMON OE                                   Order Nbr:      3057
 00008 Ave.KL                                        Order Date:  11-18-1997
 Des_Moines_           IO  07891-2345
Part    Description               Quantity  Price     Discount     Amount
============================================================================
000008  Cross_Country_Ski_Set         1     $ 93.00    .2005        $92.81
000021  Ten_Gallon_Hats               4     $ 56.64    .2005       $226.10
000029  Zoo_Season_Pass               3     $ 90.03    .2005       $269.54
000018  Radio_Controlled_Plane        1     $ 12.41    .2005        $12.38
000004  Baseball_Tickets             10     $ 91.14    .2005       $909.57
000017  Magical_Mystery_Maze          2     $ 39.83    .2005        $79.50
000015  25_Inch_Color_TVs             1     $ 63.75    .2005        $63.62
000030  Dry-Erase_Markers             3     $ 70.56    .2005       $211.25
000051  Snorkle_And_Fins_Set          2     $ 32.35    .2005        $64.57
000055  QuartzDigital_Wristwatch      1     $ 42.17    .2005        $42.08
                                                              --------------
Order total:                                                     $1,971.42
                                                              ==============

F3=Exit   F12=Cancel   F19=Left   F20=Right   F24=More keys

MA█   d                                                            03/022
```

Figure 155. Printed Order

The printed order is created by a batch process. It shows the customer details and the items, quantities, and cost of the order.

5.1.6 Database Table Structure

The ABC Company database has eight tables:

- District
- Customer
- Order
- Order Line
- Item
- Stock
- Warehouse
- History

The relationship among these tables are shown in Figure 156.

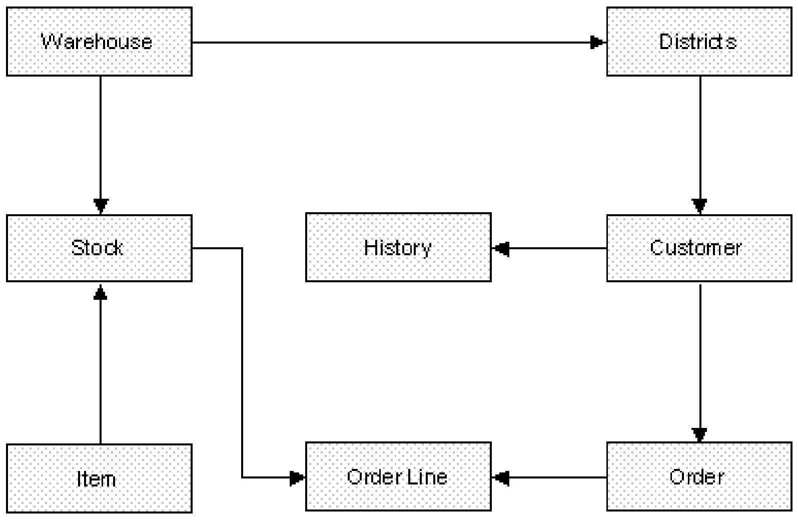


Figure 156. Table Relationships

5.1.7 Order Entry Application Database Layout

The sample application uses the following tables of the database:

- District
- Customer
- Order
- Order line
- Stock
- Item (catalog)

The following sections describe, in detail, the layout of the database.

5.1.7.1 Tables

Table 14. District Table Layout (Dstrct)

Field Name	Real Name	Type	Length
DID	District ID	Decimal	3
DWID	Warehouse ID	Character	4
DNAME	District Name	Character	10
DADDR1	Address Line 1	Character	20
DADDR2	Address Line 2	Character	20
DCITY	City	Character	20
DSTATE	State	Character	2
DZIP	Zip Code	Character	10
DTAX	Tax	Decimal	5
DYTD	Year to Date Balance	Decimal	13
DNXTOR	Next Order Number	Decimal	9
Primary Key: DID and DWID			

Table 15. Customer Table Layout (CSTMR)

Field Name	Real Name	Type	Length
CID	Customer ID	Character	4
CDID	District ID	Decimal	3
CWID	Warehouse ID	Character	4
CFIRST	First Name	Character	16
CINIT	Middle Initials	Character	2
CLAST	Last Name	Character	16
CADDR1	Address Line 1	Character	20
CCREDT	Credit Status	Character	2
CADDR2	Address Line 2	Character	20
CDCT	Discount	Decimal	5
CCITY	City	Character	20
CSTATE	State	Character	2
CZIP	Zip Code	Character	10
CPHONE	Phone Number	Character	16
CBAL	Balance	Decimal	7
CCRDLM	Credit Limit	Decimal	7
CYTD	Year to Date	Decimal	13

Field Name	Real Name	Type	Length
CPAYCNT	Payment	Decimal	5
CDELCNT	Delivery Qty	Decimal	5
CLTIME	Time of Last Order	Numeric	6
CDATA	Customer Information	Character	500
Primary Key: CID, CDID, and CWID			

Table 16. Order Table Layout (ORDERS)

Field Name	Real Name	Type	Length
OWID	Warehouse ID	Character	4
ODID	District ID	Decimal	3
OCID	Customer ID	Character	4
OID	Order ID	Decimal	9
OENTDT	Order Date	Numeric	8
OENTTM	Order Time	Numeric	6
OCARID	Carrier Number	Character	2
OLINES	Number of Order Lines	Decimal	3
OLOCAL	Local	Decimal	1
Primary Key: OWID, ODID, and OID			

Table 17. Order Line Table Layout (ORDLIN)r

Field Name	Real Name	Type	Length
OID	Order ID	Decimal	9
ODID	District ID	Decimal	3
OWID	Warehouse ID	Character	4
OLNBR	Order Line Number	Decimal	3
OLSPWH	Supply Warehouse	Character	4
OLIID	Item ID	Character	6
OLQTY	Quantity Ordered	Numeric	3
OLAMNT	Amount	Numeric	7
OLDLVD	Delivery Date	Numeric	6
OLDSTI	District Information	Character	24
Primary Key: OLWID, OLDID, OLOID, and OLNBR			

Table 18. Item Table Layout (ITEM)

Field Name	Real Name	Type	Length
IID	Item ID	Character	6
INAME	Item Name	Character	24
IPRICE	Price	Decimal	5
IDATA	Item Information	Character	50
Primary Key: IID			

Table 19. Stock Table Layout (Stock)

Field Name	Real Name	Type	Length
STWID	Warehouse ID	Character	4
STIID	Item ID	Character	6
STQTY	Quantity in Stock	Decimal	5
STDI01	District Information	Character	24
STDI02	District Information	Character	24
STDI03	District Information	Character	24
STDI04	District Information	Character	24
STDI05	District Information	Character	24
STDI06	District Information	Character	24
STDI07	District Information	Character	24
STDI08	District Information	Character	24
STDI09	District Information	Character	24
STDI10	District Information	Character	24
STYTD	Year to Date	Decimal	9
STORDERS	Quantity	Decimal	5
STREMORD	Quantity	Decimal	5
STDATA	Item Information	Character	50
Primary Key: STWID and STIID			

5.1.8 Database Terminology

This redbook concentrates on the use of the AS/400 system as a database server in a client/server environment. In some cases, we use SQL to access the AS/400 database. In other cases, we use native database access.

The terminology used for the database access is different in both cases. In Table 20, you find the correspondence between the different terms.

Table 20. Database Terminology

AS/400 Native	SQL
Library	Collection
Physical File	Table
Field	Column
Record	Row
Logical File	View or Index

Chapter 6. Developing AS/400 Java Applets

This chapter investigates a more complex applet development scenario. We build an AS/400 Internet-based shopping applet. It is actually a set of three applets. These applets use the CPW databases that are described in Chapter 5, “Overview of the Order Entry Application” on page 179. The databases are AS/400 databases and are accessed using JDBC. This suite of applets allow you to select items from the Item database, place and confirm orders, and check the status of orders. This application example is an Internet-based version of the RPG order entry application discussed in Section 5.1, “Overview of the Order Entry Application” on page 179.

The first two applets are the Toolbox applet and Cart applet. Use them to select items to order and to place an order:

- Toolbox applet
  - Used to query the Item database and select items to be ordered
- Cart applet
  - Used to check what items have been selected
  - Used to place and confirm an order for those items

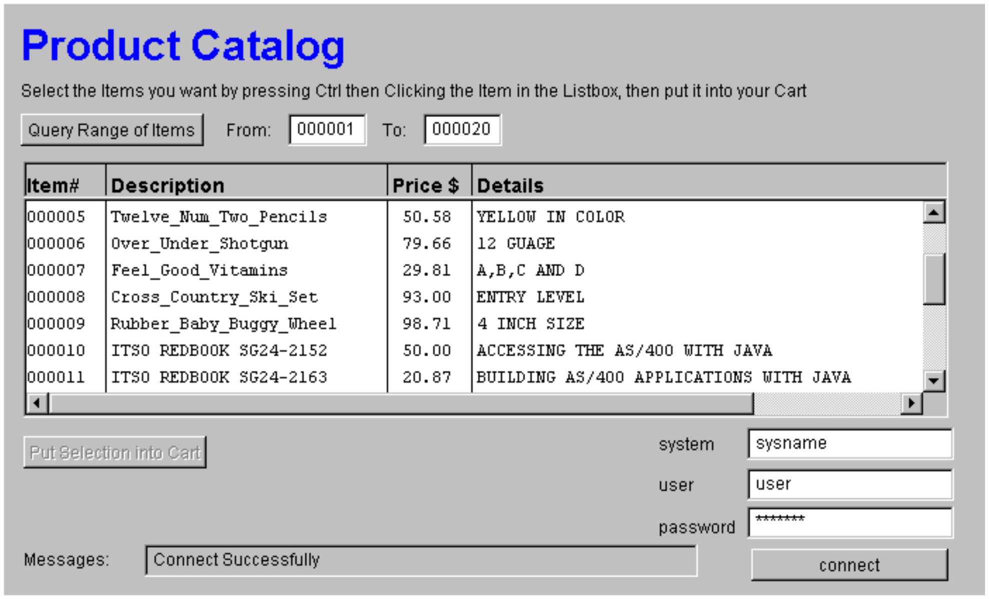
Normally the Toolbox applet works in conjunction with the Cart applet. The third applet is independent of the preceding two applets. It is the Status applet and is used to check the status of an order.

Note

The example programs discussed in this chapter are available for you to download from the redbook Web site. Refer to Appendix A.1, “Downloading the Files from the Internet Web Site” on page 299 for details.

6.1 Shopping Application User Interface

This section shows the "shopping" applet graphical user interfaces. The images shown were captured when running under Netscape Navigator.



Product Catalog

Select the Items you want by pressing Ctrl then Clicking the Item in the Listbox, then put it into your Cart

Query Range of Items From: 000001 To: 000020

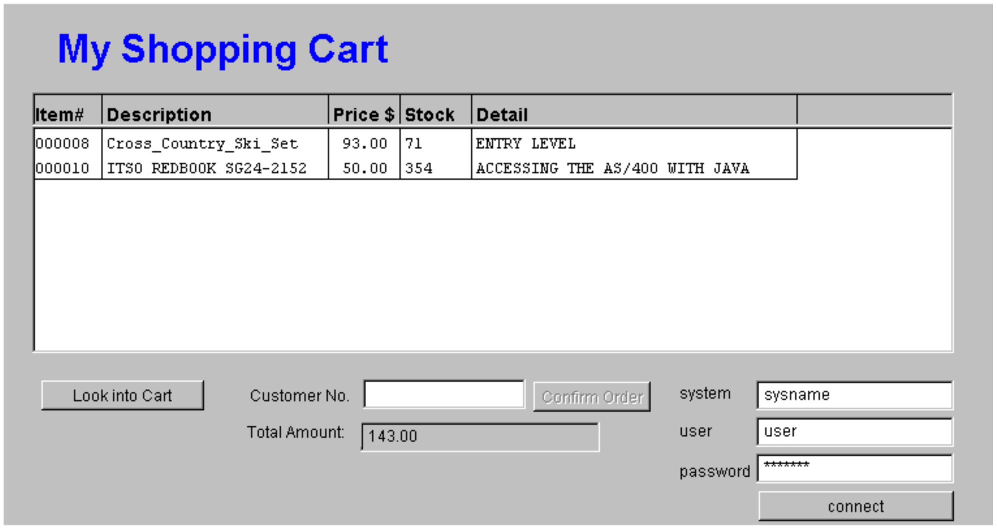
Item#	Description	Price $	Details
000005	Twelve_Num_Two_Pencils	50.58	YELLOW IN COLOR
000006	Over_Under_Shotgun	79.66	12 GUAGE
000007	Feel_Good_Vitamins	29.81	A,B,C AND D
000008	Cross_Country_Ski_Set	93.00	ENTRY LEVEL
000009	Rubber_Baby_Buggy_Wheel	98.71	4 INCH SIZE
000010	ITSO REDBOOK SG24-2152	50.00	ACCESSING THE AS/400 WITH JAVA
000011	ITSO REDBOOK SG24-2163	20.87	BUILDING AS/400 APPLICATIONS WITH JAVA

system sysname
user user
password *******

Messages: Connect Successfully

Figure 157. Toolbox Applet

Figure 157 shows the Toolbox applet, which is the first applet we investigate. It allows you to query the AS/400 Items database for information and display the query results in a listbox. You can select the items you want and put them into a “Shopping Cart.” Then, use the Cart applet to check your selected items and place an order. You can also use the Status applet to check the status of your orders.



My Shopping Cart

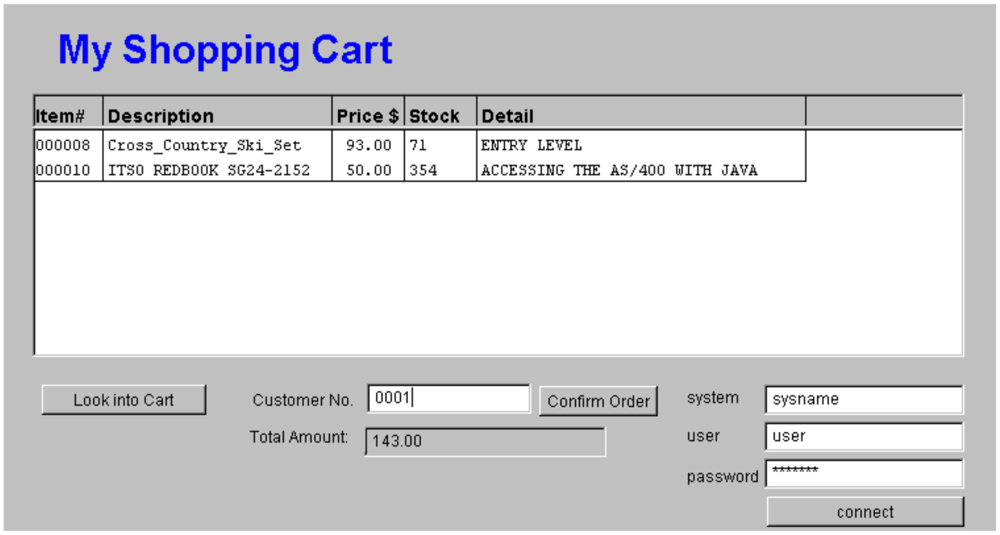
Item#	Description	Price $	Stock	Detail
000008	Cross_Country_Ski_Set	93.00	71	ENTRY LEVEL
000010	ITSO REDBOOK SG24-2152	50.00	354	ACCESSING THE AS/400 WITH JAVA

Customer No.
Total Amount: 143.00

system sysname
user user
password *******

Figure 158. Cart Applet

Figure 158 shows the Cart applet. It displays the items that you select using the Toolbox applet. Normally, the Toolbox applet and the Cart Applet are run together in a browser.

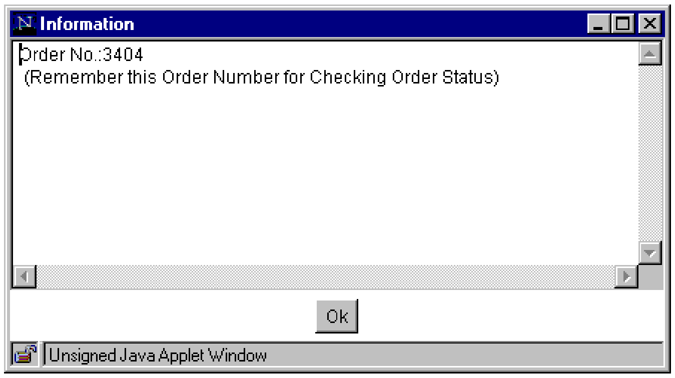

My Shopping Cart

Item#	Description	Price $	Stock	Detail
000008	Cross_Country_Ski_Set	93.00	71	ENTRY LEVEL
000010	ITSO REDBOOK SG24-2152	50.00	354	ACCESSING THE AS/400 WITH JAVA

Customer No. 0001
Total Amount: 143.00
system sysname
user user
password *******

Figure 159. Placing an Order

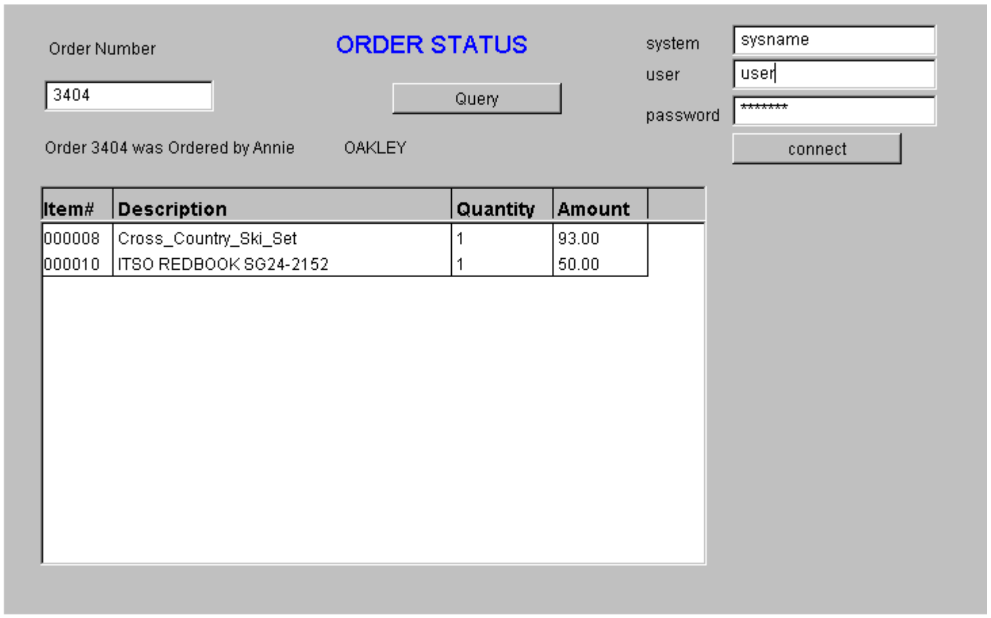
In the Cart applet, confirm your order by entering a valid customer number. The number entered is checked against the AS/400 Customer database. If the customer number is valid, the “Confirm Order” button is enabled as shown in Figure 159. You can place an order by clicking on the Confirm Order button.


Information

Order No.:3404
(Remember this Order Number for Checking Order Status)

Figure 160. Order Confirmation

Upon confirmation of the order, the Cart applet returns a message box, which displays an order number. Use the order number to track the order status.



Order Number	3404
system	sysname
user	user
password	*******

Order 3404 was Ordered by Annie OAKLEY

Item#	Description	Quantity	Amount
000008	Cross_Country_Ski_Set	1	93.00
000010	ITSO REDBOOK SG24-2152	1	50.00

Figure 161. Status Applet

The Order Status applet, shown in Figure 161, allows you to check the status of an order by entering the order number and clicking on the Query button.

The Shopping application has limitations that can easily be eliminated by adding additional function to the applets. These limitations include:

- The Toolbox applet only allows you to order a quantity of one.
- The Cart applet does not allow you to delete items from the cart.

6.2 Shopping Application Objects and Classes

Figure 162 on page 197 shows the design of the Shopping application. It consists of five classes. Three of the classes are applets that provide a graphical user interface, while the other two are supporting classes. All of the programs discussed in the chapter are available for you to ownload from the redbook Web site. The "Shopping" application was created using VisualAge for Java 2.0 Enterprise Edition.

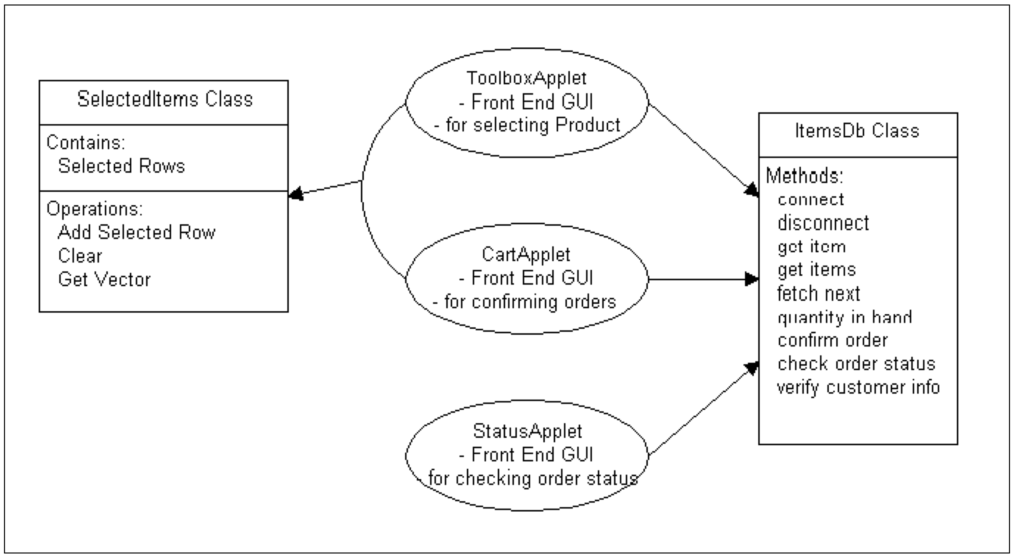


Figure 162. Shopping Application Design

A project named “AppletWorkshop” that holds a package named ToolboxApplet and the classes is in the VisualAge for Java Integrated Development Environment(IDE). The project includes:

- **ToolboxApplet**—A GUI applet that allows items to be selected.
- **CartApplet**—A GUI applet that allows orders to be placed.
- **StatusApplet**—A GUI applet that checks the order status.
- **ItemsDb**—A supporting class that provides access to the AS/400 database (it is used by the GUI applets).
- **SelectedItems**—A class for storing the items selected. A cart object is instantiated from this class. The GUI applets can put items into the cart or display what is in the cart.

The ItemsDb and SelectedItems classes are created to promote reusability. The other classes use them. This allows you to easily change and add functionality to the application.

Figure 163 on page 198 shows the ToolboxApplet package in the VisualAge for Java WorkBench. The *runner* superscript on the top right hand corner of a class means that it is a runnable Java applet.

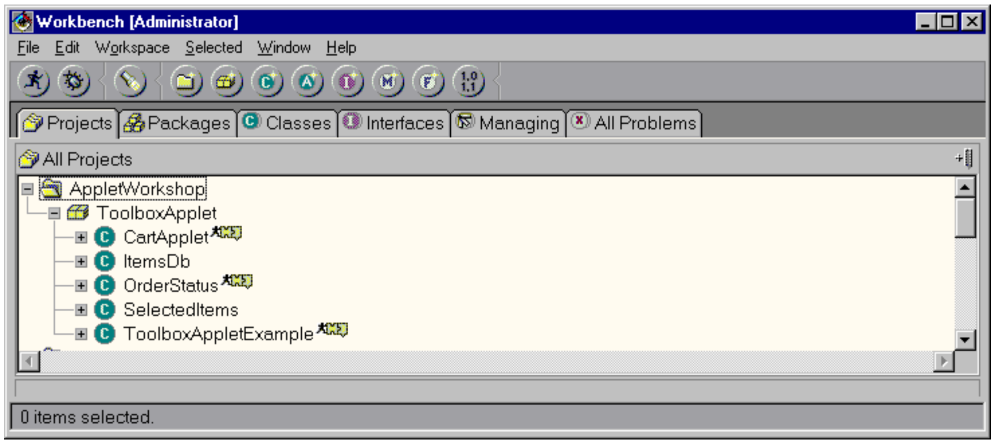

Workbench [Administrator]
File Edit Workspace Selected Window Help
Projects Packages Classes Interfaces Managing All Problems
All Projects
AppletWorkshop
ToolboxApplet
CartApplet
ItemsDb
OrderStatus
SelectedItems
ToolboxAppletExample
0 items selected.

Figure 163. ToolboxApplet Package

In addition to the classes shown in the ToolboxApplet package, we use several other classes:

- The MultiColumnListbox from the IBM Enterprise Access Libraries project to display the selected items
- The AS/400 Toolbox for Java classes to allow access to the AS/400 system
- The Java classes in the Java Classes Library project to provide Java support
- The Vector class from the java.util package to store the items in the "shopping" cart
  - A Vector is a collection of different objects.
  - The difference between a Vector and an Array is:
    - An array is a collection of **the same type** of objects.
    - A vector is a collection of different or the same type of objects.
  - VectorEnumeration is a support class:
    - It allows you to scroll through the entire list of objects inside a Vector.
    - It provides two main methods:
      - getNextElement()
      - hasmoreElements()
  - Elements can be of different types in a vector; you have to type cast the element to the proper type before using.

6.3 The SelectedItems Class

The SelectedItems class is a supporting class used with the Toolbox applet and the Cart applet. SelectedItems acts as a buffer that stores items that you select. As you view items that are available, you can select items and place them in your “Shopping Cart.” You can view the items you select, which are stored in a SelectedItems object.

The SelectedItems class contains a **static** vector named *wanted* that contains all of the items selected. This is the “shopping cart” where we keep the selected items. It also contains a **static** BigDecimal variable named totalAmount that stores the total value of the items selected. We use the **static** keyword because we want other classes to share these variables.

6.3.1 Writing the Class

The class definition for the SelectedItems class is shown in Figure 164.

```
import java.math.BigDecimal;
import java.util.Vector;
public class SelectedItems
{
   private static Vector wanted;
   public static BigDecimal totalAmount;
}
```

Figure 164. SelectedItems Class Definition

The two import statements tell the compiler where the supporting classes are located. The BigDecimal class is in the java.math package. The Vector class is in the java.util package.

In some cases, generic import statements are used. Instead of writing “import java.math.BigDecimal;” we can write “import java.math.*”, so that all classes in the java.math package are known to the compiler.

The advantage of using the specific import statement rather than the generic one is that specific import statements document exactly what additional classes are used in the application. This facilitates modifying code and packaging the application using JAR files.

6.3.2 Writing the Methods

First, we discuss the getVector method which allows other classes or methods to access the Vector named **wanted** (we declare this name as private). Declaring a variable as private gives the class owner more control over it because other users cannot access it directly. They can only access it through owner-supplied public methods. Here, we supply the public getVector method. Also, if the wanted Vector is not instantiated, we instantiate it here. This is called *lazy initialization* so if the Vector is disposed later, it is regenerated when needed.

```
public Vector getVector()
{
if (wanted == null)
 wanted = new Vector();
return wanted;
}
```

Figure 165. The getVector Method

The clear method allows other applets to clear the cart and remove selections from the cart.

```
public void clear()
{
  wanted = null;
}
```

Figure 166. The clear Method

Finally, the addSelectedRows method allows items to be added to the wanted Vector that was previously created. An object array is used as the input parameter.

We add the object array to the vector and add the price of the item to the totalAmount variable. Because we may not have initialized the totalAmount variable, we set it to zero if it is empty.

```
public void addSelectedRow(Object[] row)
{
getVector().addElement(row);
if (totalAmount == null)
{
  totalAmount = new BigDecimal("0");
}
String Column3 = new java.lang.String((String) row[2]);
totalAmount = totalAmount.add(new BigDecimal(Column3.trim()));
}
```

Figure 167. The addSelectedRow Method

6.4 The ItemsDb Class

The ItemsDb class is a supporting class that provides access to the AS/400 system. All the graphical user interface applets use this class to access the AS/400 databases. The class definition is shown in Figure 168 on page 201.

```
import java.math.*;
import java.util.*;
public class ItemsDb extends java.lang.Object
{
private java.sql.Connection dbConnect;
private java.sql.PreparedStatement psItem;
private java.sql.PreparedStatement psItemRange;
private java.sql.PreparedStatement psCustomerDb;
private java.sql.PreparedStatement psQuantityInHand;
private java.sql.Statement sGetInetOrderNo;
private String systemName = new String("");
private String userid = new String("");
private String password = new String("");
private java.sql.ResultSet rs = null;
public String itemId;
public String itemName;
public BigDecimal itemPriceBigDecimal;
public String itemPrice;
public String itemInfo;
public Object[] row;
public String validCustomerId = null;
protected java.util.Vector aConnectionListener = null;
}
```

Figure 168. The ItemsDb Class Definition

The class definition declares all the variables that we use in this class. An explanation of the variables is shown in Table 21 on page 202.

Table 21. ItemsDb Class Variables

Variables	Description
java.sql.Connection dbConnect	The connection to AS/400
java.sql.PreparedStatement psItem	Optimized Query - see *connect()* for def (Prepared Statements).
java.sql.PreparedStatement psItemRange	Optimized Query - see *connect()* for def.
java.sql.PreparedStatement psCustomerDb	Optimized Query - see *connect()* for def.
java.sql.PreparedStatement psQuantityInHand	Optimized Query - see *connect()* for def.
java.sql.Statement sGetInetOrderNo	Dynamic Query - used to get Internet Order Number. (slower than Prepared Statement, but has more flexibility where SQL can be changed on the fly.)
String systemName	Stores the AS/400 system name to connect to (Internet server name - for example, www.as400.com).
String user ID	Stores the Default User Id for logon from Internet shopping applications.
String password	Stores the password for the Default User Id.
java.sql.ResultSet rs	As a temporary variable for the Query Result Set returned.
String itemId	Store the item Id value of the current record. - **IID** of **ITEM** database in **CSDB** library
String itemName	Store the itemName **INAME** value of the current record.
BigDecimal itemPriceBigDecimal	Store the itemPrice **IPRICE** of the current record in BigDecimal format for caculation.
String itemPrice	itemPrice in String format
String itemInfo	itemInfo - **IDATA** field of **ITEM** database
String validCustomerId	The Valid Customer Id for ordering through Internet.

6.4.1 Common Methods All Applets Use

The ItemsDb class provides some methods that are used by all GUI applets.

6.4.1.1 The connect Method

The **connect** method is used to connect to the AS/400 system. It uses the system name, user ID, and password defined in the class variables. It also prepares the JDBC statements that are used to access the AS/400 databases.

```
public String connect()
{
try
{
netscape.security.PrivilegeManager.enablePrivilege("UniversalConnect");
java.sql.DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());
dbConnect = java.sql.DriverManager.getConnection("jdbc:as400://" + systemName +
"/apilib;naming=system;errors=full;date format=iso", userid, password);
psItem = dbConnect.prepareStatement("SELECT * FROM apilib/ITEM WHERE IID = ?");
psItemRange = dbConnect.prepareStatement("SELECT * FROM apilib/ITEM WHERE IID >= ? AND IID <= ?");
psCustomerDb = dbConnect.prepareStatement("SELECT CID FROM apilib/CSTMR WHERE CID = ? AND CDID=001 AND
CWID='0001'");
psQuantityInHand = dbConnect.prepareStatement("SELECT STQTY FROM apilib/STOCK WHERE STWID= '0001' AND
STIID=?");
}
catch (Exception e)
{
System.out.println("connect(): " + e);
e.printStackTrace();
return "Connect: " + e;
}
return "Connect Successfully";
}
```

Figure 169. The connect Method

If we load an applet from the local workstation disk, we must handle security. By default, applets do not have access to any remote system. In this case, the AS/400 system is a remote system. Here, we run under Netscape Navigator, so we use the security support provided by Netscape. We use the enablePriviledge method from the PrivilegeManger class. This causes a security dialog to appear when we attempt to connect to the AS/400 system. If we use the AS/400 HTTP server to serve the applets, we automatically have access to the AS/400 system, so we do not need to provide any special security support. For more information about security and deploying applets, refer to Chapter 3, “Introduction to AS/400 Applets” on page 51.

6.4.1.2 The disconnect Method

The **disconnect** method is used for closing all AS/400 connections.

```
public void disconnect() throws Exception
{
dbConnect.close();
psItem.close();
psItemRange.close();
psCustomerDb.close();
psQuantityInHand.close();
return;
}
```

Figure 170. The disconnect Method

6.4.1.3 The finalize Method

The **finalize** method automates using the disconnect method. Every time the ItemsDb class is disposed, it disconnects from the AS/400 system and releases all resources allocated.

```
protected void finalize()
{
try
{
  disconnect();
  super.finalize();
}
catch (Throwable t)
{
  System.out.println(t);
}
return;
}
```

Figure 171. The finalize Method

6.4.2 Methods Used by the Toolbox Applet

The ItemsDb class has some methods that are used only by the Toolbox applet class.

6.4.2.1 The fetchNextItem Method

The **fetchNextItem** method is used after executing a JDBC statement that returns a resultset. It fetches the next record from the current resultset and puts the corresponding field values in the public class variables. It returns itself (*this*) to allow for the cascading of methods.

```
public ItemsDb fetchNextItem()
{
try
{
if (rs.next())
{
  itemId = rs.getString("IID");
  itemName = rs.getString("INAME");
  itemPriceBigDecimal = rs.getBigDecimal("IPRICE", 2);
  itemPrice = itemPriceBigDecimal.toString();
  itemInfo = rs.getString("IDATA");
}
else
{
  itemId = null;
  itemName = null;
  itemPriceBigDecimal = null;
  itemPrice = null;
  itemInfo = null;
}
}
catch (Exception e)
{
  System.out.println("fetchnext fail: " + e);
}
return this;
}
```

Figure 172. The fetchNextItem Method

6.4.2.2 The getItem Method

The **getItem** method queries the ITEM table using the itemno parameter passed to it as input. It executes the psItem prepared statement object that was prepared in the connect method. It uses the fetchNextItem method to load the information from the resultset into the class variables.

```
public ItemsDb getItem(String itemno)
{
try
{
  psItem.setString(1, itemno);
  rs = psItem.executeQuery();
  fetchNextItem();
}
catch (Exception e)
{
  System.out.println("getItem fail: " + e);
}
return this;
}
```

Figure 173. The getItem Method

6.4.2.3 The getItems Method

The **getItems** method queries the ITEM database for items matching a range of item numbers. It executes the psItemRange prepared statement object created in the connect method. It uses the fetchNextItem method to load the data into the class variables. If there are no more records, the class variables are set to null.

```
public java.sql.ResultSet getItems(String itemnoMin, String itemnoMax)
{
if (itemnoMax.length() == 0)
{
  getItem(itemnoMin);
}
else
{
try
{
  psItemRange.setString(1, itemnoMin);
  psItemRange.setString(2, itemnoMax);
  rs = psItemRange.executeQuery();
}
catch (Exception e)
{
  System.out.println("getItemS fail: " + e);
}
}
return rs;
}
```

Figure 174. The getItems Method

6.4.3 Methods Used by CartApplet

The ItemsDb class provides some methods that are only used by the Cart applet.

6.4.3.1 The quantityInHand Method

The **quantityInHand** method returns the Quantity in Stock value for an item from the AS/400 Stock table. It executes the psQuantityInHand prepared statement object that was prepared in the connect method.

```
public BigDecimal quantityInHand(String itemNo)
{
try
{
// Get next Order No. and the Inet YTD Balance
 psQuantityInHand.setString(1, itemNo);
 rs = psQuantityInHand.executeQuery();
 rs.next();
 return rs.getBigDecimal("STQTY", 0);
}
catch (Exception e)
{
 System.out.println(e);
}
return null;
}
```

Figure 175. The quantityInHand Method

6.4.3.2 The verifyCustomer Method

The **verifyCustomer** method checks whether the customerId passed as an input parameter is valid. It executes the psCustomerDb prepared statement object created in the connect method. It returns as true or false depending on the result of the check. If the customerId is valid, it saves it in the class variable validCustomerId.

```
public boolean verifyCustomer(String customerId)
{
boolean isvalid = false;
try
{
 psCustomerDb.setString(1, customerId);
 rs = psCustomerDb.executeQuery();
if (rs.next())
{
 isvalid = true;
 validCustomerId = customerId;
}
}
catch (Exception e)
{
 validCustomerId = null;
}
return isvalid;
}
```

Figure 176. The verifyCustomer Method

6.4.3.3 The confirmOrder Method

The **confirmOrder** method creates an order from the items inside the SelectedItems object. The customerId is validated by the verifyCustomer method prior to confirming the order.

The **confirmOrder** method uses JDBC to access the AS/400 database. It contains the following logic:

- Verify if there are items in the cart.
- Retrieve the next order number and district YTD balance from the AS/400 District table.
- Update the District YTD balance with the new order total.
- Increment the District next order number by one.
- Insert an order record to the AS/400 Order table.
- Update the stock balance in the AS/400 Stock table.
- Insert an order line record in the AS/400 Order Line table for each item ordered.
- Return the order number used.
- Clear the items from the cart.

When confirming the order, the confirmOrder method performs the following actions:

- Retrieves the next order number **DNXTOR** from the District table(**DSTRCT)**
- Increments it by one
- Writes it back to the District table

6.4.4 Methods Used by the StatusApplet

The ItemsDb class provides some methods that are used only by the Status applet.

6.4.4.1 The checkOrderStatus Method

The **checkOrderStatus** method receives an order number as a parameter, which it uses to query the AS/400 database for order information. It returns a Vector named orderStatus which contains all the details about the order.

```
public Vector checkOrderStatus(String orderIdString)
{
// return Vector contains CustomerLastName, CustomerFirstName,Object[],Object[],...
//        where Object[] is OrderLineDetail => [ItemId,ItemName,QtyOrdered,Amount]
Vector orderStatus = new Vector();
if (orderIdString.length() > 9 || orderIdString.length() == 0)
 return null;
try
{
 sGetInetOrderNo = dbConnect.createStatement();
 rs = sGetInetOrderNo.executeQuery("SELECT OCID,OLINES FROM ORDERS WHERE OWID='0001' AND ODID=001 AND
OID=" + orderIdString);
 rs.next();
 String customerId = rs.getString("OCID");
 BigDecimal orderLines = rs.getBigDecimal("OLINES", 0);
 rs = sGetInetOrderNo.executeQuery("SELECT CFIRST,CLAST FROM CSTMR WHERE CWID='0001' AND CDID=001 AND
CID='" + customerId + "'");
 rs.next();
 String lastName = rs.getString("CLAST");
 String firstName = rs.getString("CFIRST");
 orderStatus.addElement(lastName);
 orderStatus.addElement(firstName);
.
.
.
return orderStatus;
}
```

Figure 177. The checkOrderStatus Method

6.5 The Toolbox Applet

This applet allows you to enter a system name, user ID and password, and connect to the AS/400 system. If the connection is successful, the QueryRange of Items button is enabled. You can click on the Query Range of Items button to get the items found in the ITEM table for the range specified by TextField1 and TextField2. The item information is displayed in the listbox. It uses the ItemsDb class for all the AS/400 Database accesses.

6.5.0.1 Basic Class Definition

Figure 178 shows the basic class structure, class variables, and the multicolumn listbox declaration.

```
import java.applet.*;
import java.awt.*;
import java.net.URL;
import java.util.*;
public class ToolboxAppletExample extends Applet implements
java.awt.event.ActionListener, java.awt.event.ItemListener {
  static SelectedItems selected = new SelectedItems();
  private com.ibm.ivj.eab.dab.IMulticolumnListbox ivjIMulticolumnListbox1 =
null;
  private java.sql.DriverManager ivjDriverManager1 = null;
}
```

Figure 178. ToolboxAppletExample Class Definition

We declare a variable named **selected**, which is based on the SelectedItems class. Because selected is declared as static, it is shared by all other classes that declare it. We declare a multicolumn listbox named **ivjIMulticolumnListbox1**. It is based on the IMulticolumnListbox class from the com.ibm.eab.dab package, which is included with VisualAge for Java Enterprise Edition. We declare a DriverManager, which is used for the JDBC connection.

6.5.1 The addAllRows Method

The **addAllRows** method is called after a JDBC statement that returns a resultset is executed. It uses the fetchNextItem method to loop through the resultset and retrieve all returned items. It formats the item information into an array, which is used to populate the listbox.

```
public void addAllRows(ToolboxApplet.ItemsDb anItemDb)
{
anItemDb.fetchNextItem();
while (anItemDb.itemId != null)
{
 String[] array = new String[4];
 array[0] = anItemDb.itemId;
 array[1] = anItemDb.itemName;
 array[2] = insertSpaces(anItemDb.itemPrice, 6);
 array[3] = anItemDb.itemInfo;
 getIMulticolumnListbox1().addRow(array, array[0]);
 anItemDb.fetchNextItem();
}
}
```

Figure 179. The AddAllRows Method

6.5.2 The getSelectedIndexes Method

The **getSelectedIndexes** method is called when the user clicks on the Put Selection into Cart button. It determines which listbox indexes are selected. It adds the selected rows of the listbox to the SelectedItems class object named **selected**. It places the selected items into the “cart.”

```
public void getSelectedIndexes()
{
 Object key = getIMulticolumnListbox1().getSelectedObject();
 selected.addSelectedRow(getIMulticolumnListbox1().getRowData(key));
}
```

Figure 180. The getSelectedIndexes Method

6.5.3 Checking the Connections

Figure 181 on page 211 shows the Toolbox applet in the VisualAge for Java Visual Composition Editor(VCE).

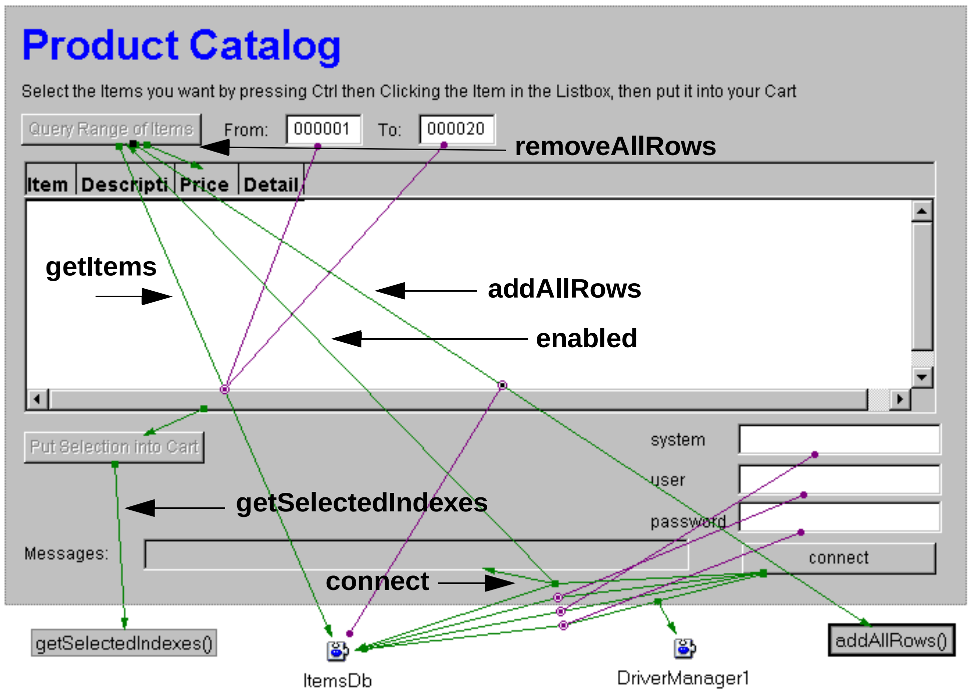


Figure 181. Toolbox Applet in the VCE

We use the following connections in this applet:

- connect button:
  - ItemsDb—setSystemName method
  - ItemsDb—setUserID method
  - ItemsDb—setPassword method
  - ItemsDb—connect method
    - Connect the normal result to the message label
    - Connect the normal result to enable the Query Range of Items button
- Query Range of Items button:
  - ToolboxAppletExample—removeAllRows method to clear the list box
  - ItemsDb—getItems method to retrieve the items from the AS/400 ITEM table
  - ToolboxAppletExample—addAllRows method to add the items to the listbox
- Put selections in cart button:
  - ToolboxAppletExample—getSelectedIndexes method to add the row selected in the listbox to the cart vector

6.6 The Cart Applet

This applet allows you to enter a system name, user ID and password, and then connect to the AS/400 system. If the connection is successful, the Look into Cart button is enabled. You can click on the Look into Cart button to display the items currently in the shopping cart, in the listbox. You can place an order for the items in the cart by entering a valid cutomer number in the customer number text field. The number entered is validated against the AS/400 Customer table. If the customer number is valid, the Confirm Order button is enabled. Clicking on the Confirm Order button generates an order. The created order number is returned in a message box. This class uses the ItemsDb class for all AS/400 Database accesses.

In the design of the *Cart* applet, we use a “Look into Cart” button to refresh the listbox. We can also implement an event for the automatic refresh of the cart whenever an item is put into the cart. For simplicity, we choose to implement the Look into Cart button.

6.6.1 Writing the Class

In the class definition, we instantiate a SelectedItems object and name it cart. This variable is shared with the Toolbox applet. The Toolbox applet puts items into the cart. The Cart applet allows us to view what is in the cart and create an order for the items in the cart.

```
import java.applet.*;
import java.awt.*;
import java.util.*;
public class CartApplet extends Applet implements
java.awt.event.ActionListener, java.awt.event.KeyListener {
 SelectedItems cart = new SelectedItems();
}
```

Figure 182. CartApplet Class Definition

6.6.2 Viewing the Methods

The *showCart* method is used to display what is in the cart Vector in the listbox. We use an object array of five elements (Object[5]) to store the elements from the cart Vector. The cart Vector has four objects (columns) in it:

- Item Identification
- Item Name
- Item Price
- Item Detail

We display the four values from the cart Vector, plus the Quantity in stock for the item in the listbox. We use the listbox addRow method to add rows to the listbox. It requires an array as an input parameter. We populate the array with the values from the cart Vector.

We insert the quantity in stock of the item, which we retrieve from the AS/400 system in the fourth element (Object[3]) of the array. We obtain the quantity in stock by calling the quantityInHand method in the ItemsDb class. We convert it to a String value for display.

Finally, we add the object array as a row to the listbox, display the total amount in the Total Amount TextField, and display the listbox.

```
public void showCart()
{
try
{
 if (cart.getVector() != null)
{
  Enumeration enum = cart.getVector().elements();
  while (enum.hasMoreElements())
  {
   String myObject[] = new String[5];
   Object[] element = ((Object[]) enum.nextElement());
   myObject[0] = (String) element[0]; //ItemId
   myObject[1] = (String) element[1]; //ItemName
   myObject[2] = (String) element[2]; //Price
    //[3] is qty in stock.
   myObject[3] = (String) (getItemsDb().quantityInHand(((String)
element[0]))).toString();
   myObject[4] = (String) element[3]; //Details
   getIMulticolumnListbox1().addRow(myObject, myObject[0]);
   }
  getTextField3().setText(cart.totalAmount.toString());
 return;
}
}
catch (Exception e)
{
 e.printStackTrace();
 System.out.println(e);
}
return;
}
```

Figure 183. The showCart Method

6.6.2.1 Viewing the Connections

Figure 184 shows the Cart applet in the VisualAge for Java Visual Composition Editor.

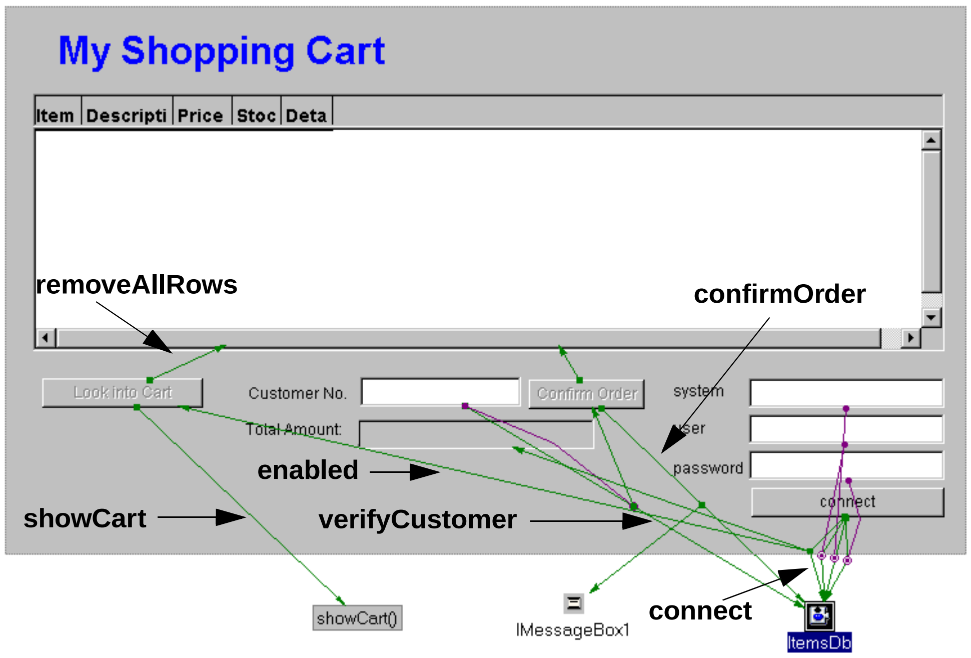


Figure 184. The Cart Applet in the VCE

We use the following connections:

- connect button:
  - – ItemsDb—setSystemName method
  - – ItemsDb—setUserID method
  - – ItemsDb—setPassword method
  - – ItemsDb—connect method
    - Connect the normal result to the message label
    - Connect the normal result to enable the Look into Cart button
- Look into Cart button:
  - – CartApplet—removeAllRows method to clear the listbox
  - – CartApplet—showCart method to display the items in the cart Vector
- Validate Customer Number:

  We validate the customer number field so that the Confirm Order button is only enabled if the customer number is valid. We use the verifyCustomer method in the ItemsDb class to validate it. We connect it with the KeyReleased event of the Customer Number Entry Field and the Confirm Order button. It enables the Confirm Order button if the customer number in Entry Field is found in the Database.
- Confrim Order button:
  - – ItemsDb—confirmOrder method, with the cart as the input parameter. The normal result is connected to the MessageBox's show method to display the order number.

– CartApplet—clear method. After confirming the order, items in the list box are cleared.

6.7 The Order Status Applet

The OrderStatus applet allows you to enter a system name, user ID, and password and connect to the AS/400 system. If a connection to the AS/400 system is successfully made, the Query button is enabled. You can then enter an order number and click on the Query button to check on the status of an order. This class uses the ItemsDb class for all of the AS/400 Database accesses.

```
public class OrderStatus extends Applet implements
java.awt.event.ActionListener {
 private com.ibm.ivj.eab.dab.IMulticolumnListbox ivjIMulticolumnListbox1 =
null;
 private com.ibm.ivj.eab.dab.IMessageBox ivjIMessageBox1 = null;
}
```

Figure 185. OrderStatus Class Definition

We declare a multicolumn listbox named ivjIMulticolumnListbox1 and a message box named ivjMessageBox1. They are based on classes from the com.ibm.eab.dab package, which is included with VisualAge for Java Enterprise Edition.

To implement the Check Order Status Applet, two user-written methods are used. The *fillListbox* method calls the ItemsDb class checkOrderStatus method with OrderId as a parameter. The *checkOrderStatus* method returns a vector that contains lastname, firstname, and an array of order detail. If the order is found, the items ordered are displayed in the listbox.

```
public void fillListbox(String OrderId)
{
Vector orderStatus = getItemsDb().checkOrderStatus(OrderId);
if (orderStatus == null)
{
 getLabel2().setText("Order No. " + OrderId + " Not Found !!!");
 return;
}
Enumeration detailLine = orderStatus.elements();
String lastName = ((String) detailLine.nextElement());
String firstName = ((String) detailLine.nextElement());
getLabel2().setText("Order " + OrderId + " was Ordered by " + firstName + "
" + lastName);
while (detailLine.hasMoreElements())
{
 String[] detail = ((String[]) detailLine.nextElement());

 getIMulticolumnListbox1().addRow(detail, detail[0]);
}

return;
}
```

Figure 186. The fillListbox Method

6.7.0.1 Viewing the Connections

In Figure 187 on page 216, we show the Order Status applet in the VisualAge for Java Visual Composition Editor.

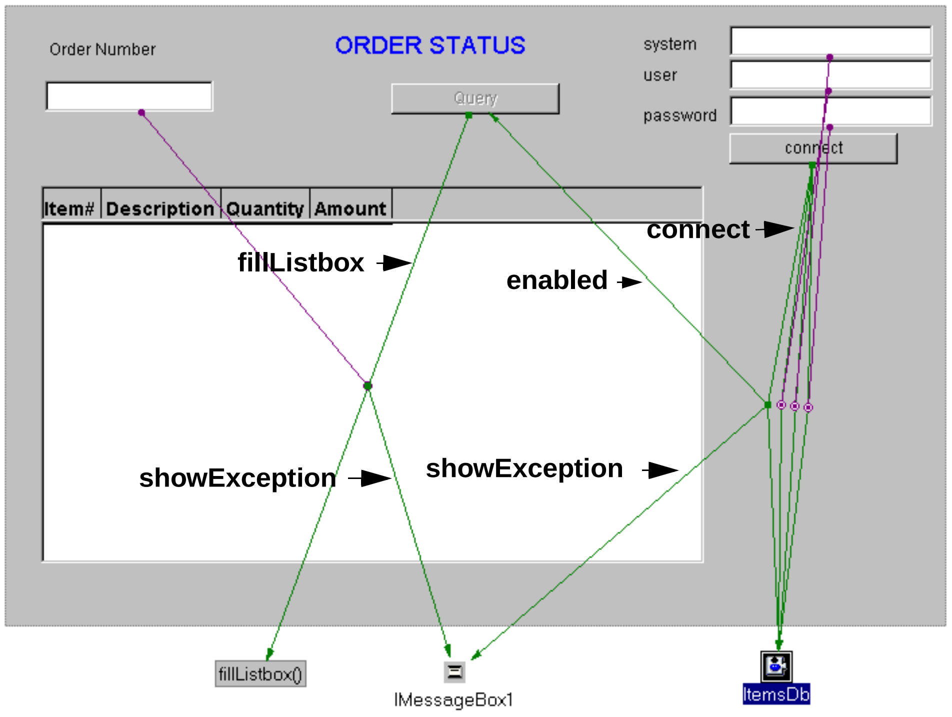


Figure 187. Order Status Applet

We use the following connections:

- connect button
  - – ItemsDb—setSystemName method
  - – ItemsDb—setUserID method
  - – ItemsDb—setPassword method
  - – ItemsDb—connect method
    - Connect the normal result to the message label
    - Connect the normal result to enable the Query button
- Query button:
  - – OrderStatus—fillListbox method to retrieve and display the order information for the order number in the TextField.

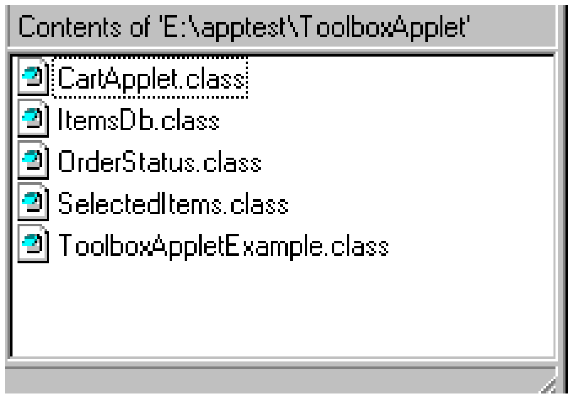
6.8 Testing the Applets

You can run the Toolbox applet and the Status applet inside the VisualAge for Java Integrated Development Environment. The Cart applet must be exported and run outside the IDE. We use Netscape Navigator to test outside the IDE.

To run the applets outside the VisualAge for Java IDE, we export the applet classes. We use **C:\apptest** as the directory in this example. We export the following classes to the apptest directory:

- ToolboxAppletExample
- CartApplet
- StatusApplet
- ItemsDb
- SelectedItems

The package in which the classes are stored becomes a subdirectory of the directory to which the export is done. In this case, ToolboxApplet becomes a subdirectory of apptest.



Contents of 'E:\apptest\ToolboxApplet'

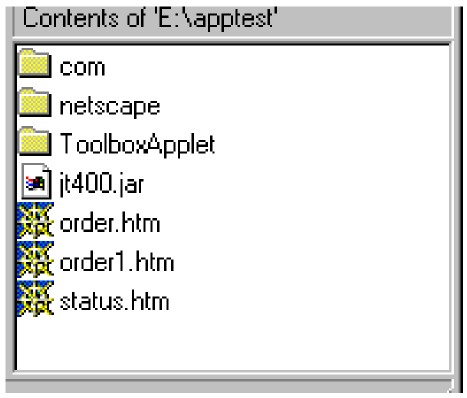
CartApplet.class
ItemsDb.class
OrderStatus.class
SelectedItems.class
ToolboxAppletExample.class

Figure 188. The apptest\ToolboxApplet Directory

We use html files to control running the applets. The html files refer to the applet class files using the APPLET tag. An applet viewer or browser tries to find the classes in the current directory from where the html file is loaded.

Place the AS/400 Toolbox classes and any third party classes under the current directory. A jar file can also be used to hold classes. We use a jar file, named jt400.jar, to hold the AS/400 Toolbox for Java classes.

In addition to the AS/400 Toolbox for Java classes, we also need several classes from the IBM Enterprise Access libraries. We export these classes to the com subdirectory of the apptest directory. We also export the classes from the netscape.security package to provide security support. Figure 189 shows the contents of the apptest directory.



Contents of 'E:\apptest'

com
netscape
ToolboxApplet
jt400.jar
order.htm
order1.htm
status.htm

Figure 189. The apptest Directory

Running Java applets is different than running Java applications. Java applications normally use the CLASSPATH environment variable to find classes that are located locally on the workstation. In many cases, applets are loaded from a remote resource. We assume that the browser or applet viewer does not use the local disk to find supporting classes. We must make them available so they can be found. Three html files (Table 22) are available to run the applets.

Table 22. Applet HTML Files

"AppletViewer Order.htm"	**-- which runs both "ToolboxApplet" and "CartApplet"**
"AppletViewer Order1.htm"	**-- which only runs "ToolboxApplet"**
"AppletViewer status.htm"	**-- which runs "StatusApplet"**

All of the html files point to the class files in the apptest directory. Figure 190 on page 219 shows the html file for Order.htm, which runs both the Toolbox applet and Cart applet under the same browser session.

```
HTML>
<applet archive="jt400.jar" code="ToolboxApplet.ToolboxAppletExample.class" width=800 height=420>
<hr>
This Applet would only be seen on JDK1.1 compatible Browser
<hr>
</applet><p>

<applet archive="jt400.jar" code="ToolboxApplet.CartApplet" width=800 height=400>
<hr>
This Applet would only be seen on JDK1.1 compatible Browser
<hr>
</applet><p>
</HTML>
```

Figure 190. Order.htm

Figure 191 shows the html file for Order1.htm which runs only the Toolbox applet.

```
<applet archive="jt400.jar" code="ToolboxApplet.ToolboxAppletExample.class" width=800 height=420>
<hr>
This Applet would only be seen on JDK1.1 compatible Browser
<hr>
</applet><p>
```

Figure 191. Order1.htm

Figure 192 shows the html file for the Status applet.

```
<HTML>
<applet archive="jt400.jar" code="ToolboxApplet.OrderStatus.class" width=800 height=450>
<hr>
This Applet would only be seen on JDK1.1 compatible Browser
<hr>
</applet><p>
</HTML>
```

Figure 192. Status.htm

6.9 Serving the Applets from the AS/400 System

We can also run the applets by using the AS/400 HTTP server. This involves three steps:

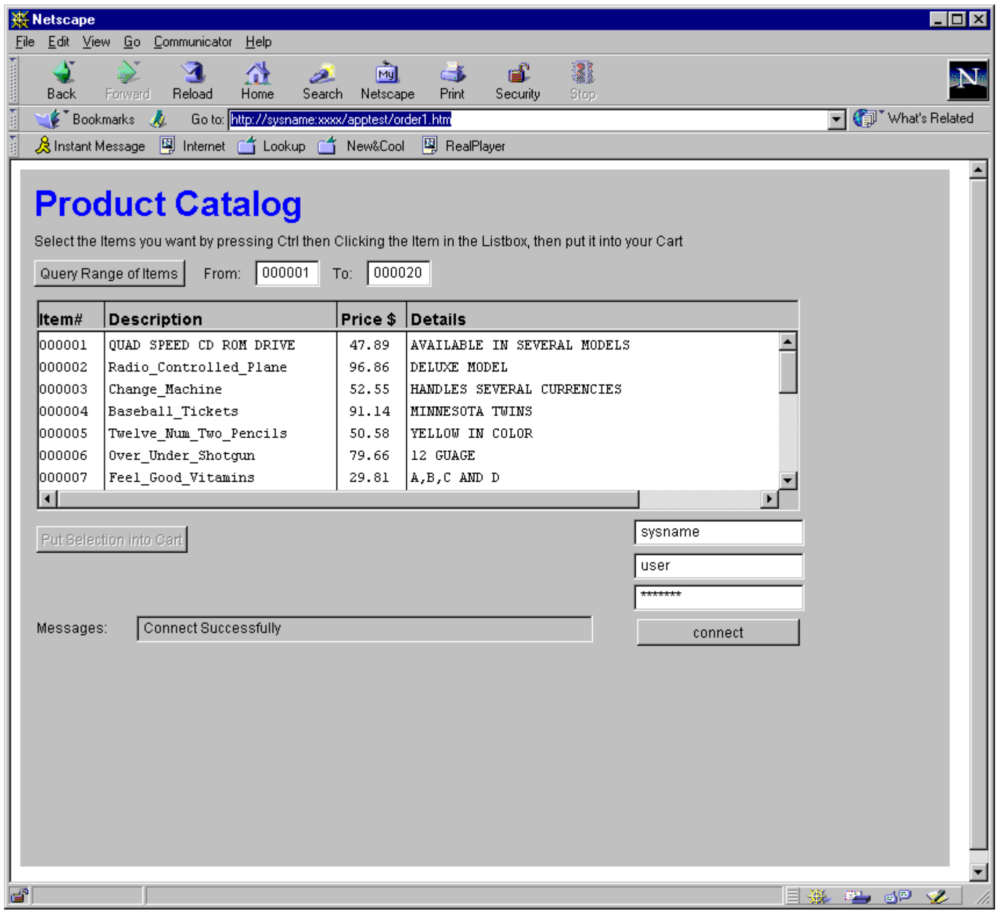
1. Copy the apptest directory to the AS/400 system Integrated File System.
2. Configure the AS/400 HTTP server to allow the html files in apptest to be executed.

   Add the following directive to the HTTP configuration:

   `Pass /apptest/* /apptest/*`
3. Point the browser to the html files stored on the AS/400 system.

Refer to Section 9.3, “IBM HTTP Server for AS/400” on page 293, for information about configuring the AS/400 HTTP server.

Figure 193 shows the shopping applet running under Netscape Navigator and serving it from the IBM HTTP Server for AS/400.


Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Go to: http://sysname:xxxx/apptest/order1.htm What's Related
Instant Message Internet Lookup New&Cool RealPlayer

Product Catalog

Select the Items you want by pressing Ctrl then Clicking the Item in the Listbox, then put it into your Cart

Query Range of Items From: 000001 To: 000020

Item#	Description	Price $	Details
000001	QUAD SPEED CD ROM DRIVE	47.89	AVAILABLE IN SEVERAL MODELS
000002	Radio_Controlled_Plane	96.86	DELUXE MODEL
000003	Change_Machine	52.55	HANDLES SEVERAL CURRENCIES
000004	Baseball_Tickets	91.14	MINNESOTA TWINS
000005	Twelve_Num_Two_Pencils	50.58	YELLOW IN COLOR
000006	Over_Under_Shotgun	79.66	12 GUAGE
000007	Feel_Good_Vitamins	29.81	A,B,C AND D

Put Selection into Cart

sysname
user
****
connect

Messages: Connect Successfully

Figure 193. Shopping Applet Running under Netscape Navigator

Chapter 7. Developing AS/400 Java Servlets

In this chapter, we build a more complex servlet example. This application demonstrates building Java programs that:

- Run as servlets
- Run as applets
- Provide security password validation and protection
- Use AS/400 Toolbox for Java classes to access AS/400 resources
- Run on a Windows/NT platform under Domino Go Webserver
- Run on the AS/400 platform under the IBM HTTP Server for AS/400

Note

The example programs discussed in this chapter are available for you to download from the redbook Web site. Refer to Appendix A.1, “Downloading the Files from the Internet Web Site” on page 299, for details.

This application runs under the control of a Web browser. The application displays shown in this chapter use the Microsoft Internet Explorer 4.0 browser. This application was also tested using Netscape Navigator 4.x.

To start the application, enter the URL of the application server. For example, to run it on an AS/400 system running the IBM HTTP Server for AS/400 and the WebSphere Application Server in the browser, enter:

```
http://AS400ABC:xxxx/servlet/Signon
```

Note

In this application, we pass information across the network that we may want to protect. For example, we pass in a password. In this case, we may want to use the IBM HTTP Server for AS/400 Secure Sockets Layer (SSL) support. For infomation about how to install and use SSL support on the AS/400 system, see Chapter 8, “Security Considerations” on page 261. When running the application under SSL, we enter:

```
https://AS400ABC:xxxx/servlet/Signon
```

AS400ABC is the name of the AS/400 system. xxxx is the TCP/IP port over which the IBM HTTP Server for AS/400 is running. Signon is a Java program, which handles security, and runs as a Java servlet.

You can also run this application under the control of other HTTP servers. For example, you can run under the Domino Go Webserver. In this case, we use a three-tier approach:

1. The client workstation running a browser
2. The HTTP server running a server, for example Windows/NT
3. The AS/400 system

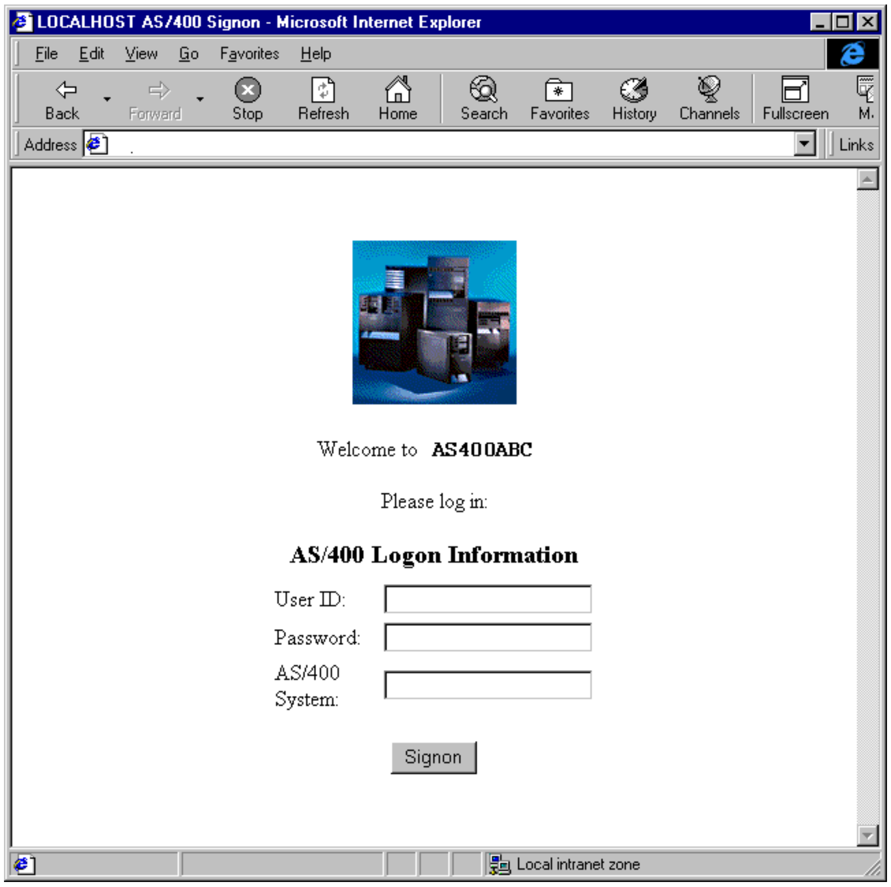
To run the application, enter:

```
http://server:xxxx/servlet/Signon
```

In this case, server is the name of an HTTP server, which supports running servlets. We tested using Domino Go Webserver and ServletExpress running on a Windows/NT platform.

7.1 Running the Application

When the application is started, the AS/400 Sign On window appears as shown in Figure 194.


LOCALHOST AS/400 Signon - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen M.
Address
Links
Local intranet zone

Welcome to AS400ABC

Please log in:

AS/400 Logon Information

User ID:

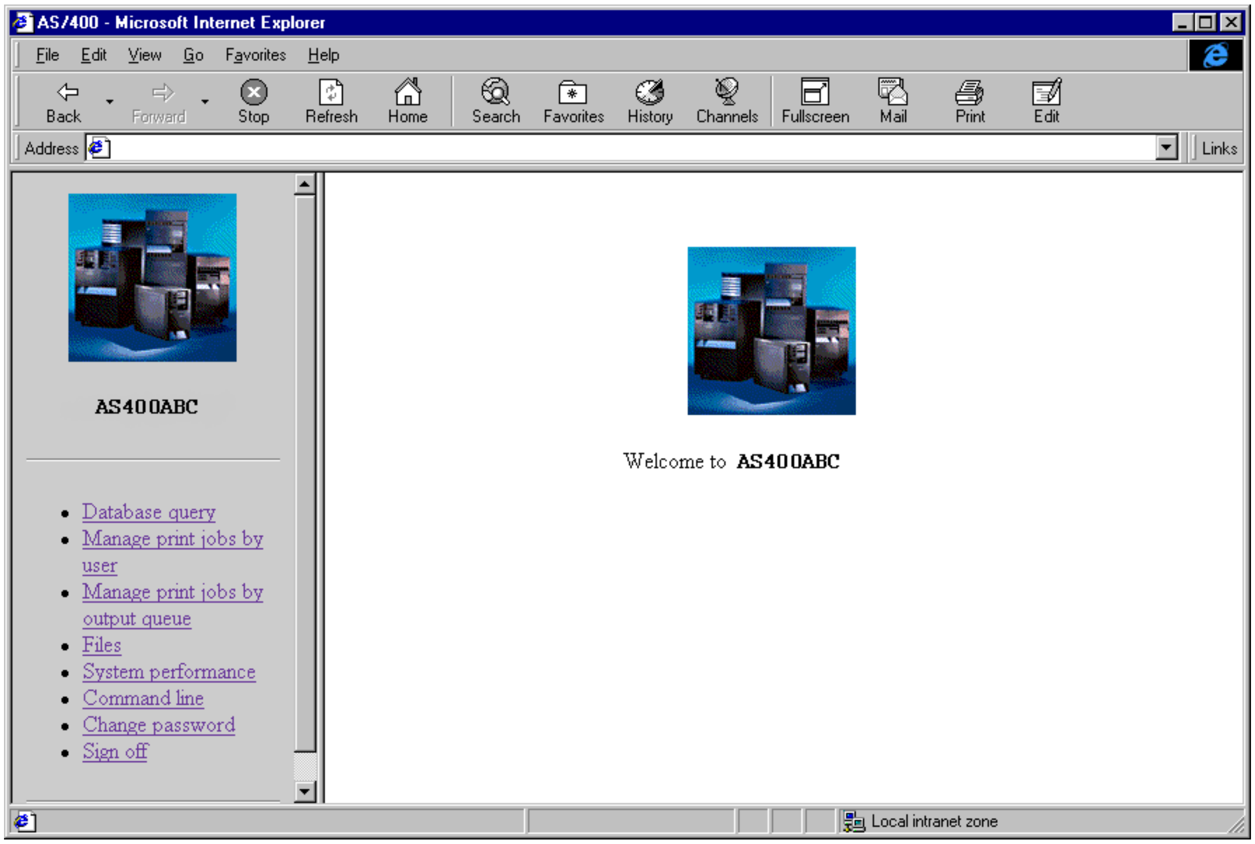
Password:

AS/400 System:

Signon

Figure 194. Servlet Sign On Window

The Sign On window allows you to enter security information and sign on to the AS/400 system. After you enter a user ID, password, and AS/400 system name, click on the Signon button. The information entered is validated. A Java program running as a servlet does the validation using AS/400 Toolbox for Java classes. If the information is valid and the AS/400 system is available, the application menu window shown in Figure 195 on page 223 appears.


AS/400 - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print Edit
Address
Links
Local intranet zone

AS400ABC

- Database query
- Manage print jobs by user
- Manage print jobs by output queue
- Files
- System performance
- Command line
- Change password
- Sign off

Welcome to **AS400ABC**

Figure 195. Servlet Application Menu Window

The application menu window allows you to select an application to run. Do this by clicking the mouse pointer on the option. If you select the Database query option, the Query Recall window appears as shown in Figure 196 on page 224.

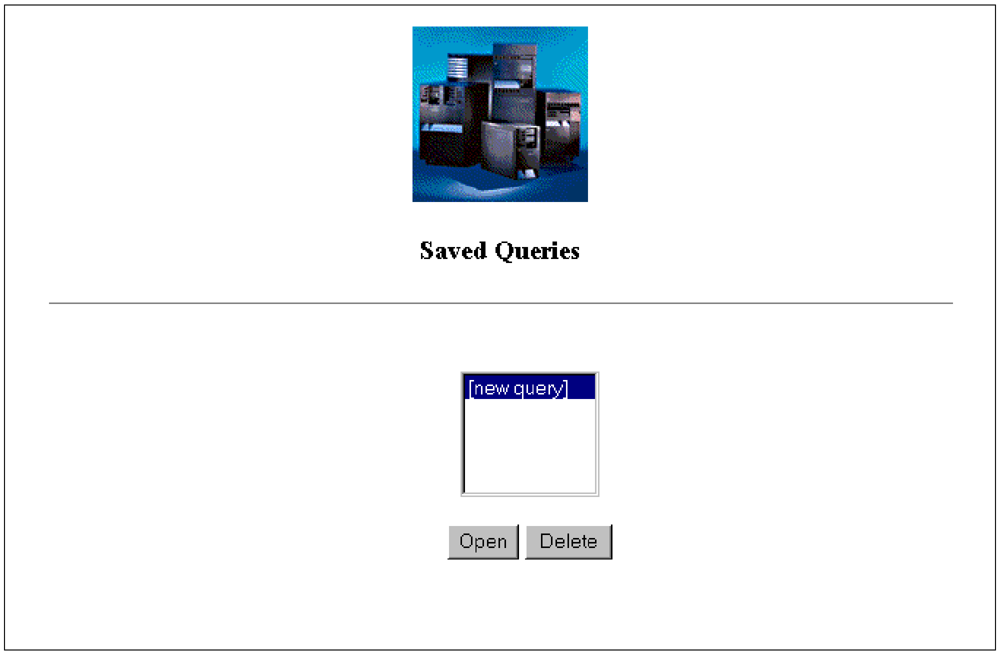
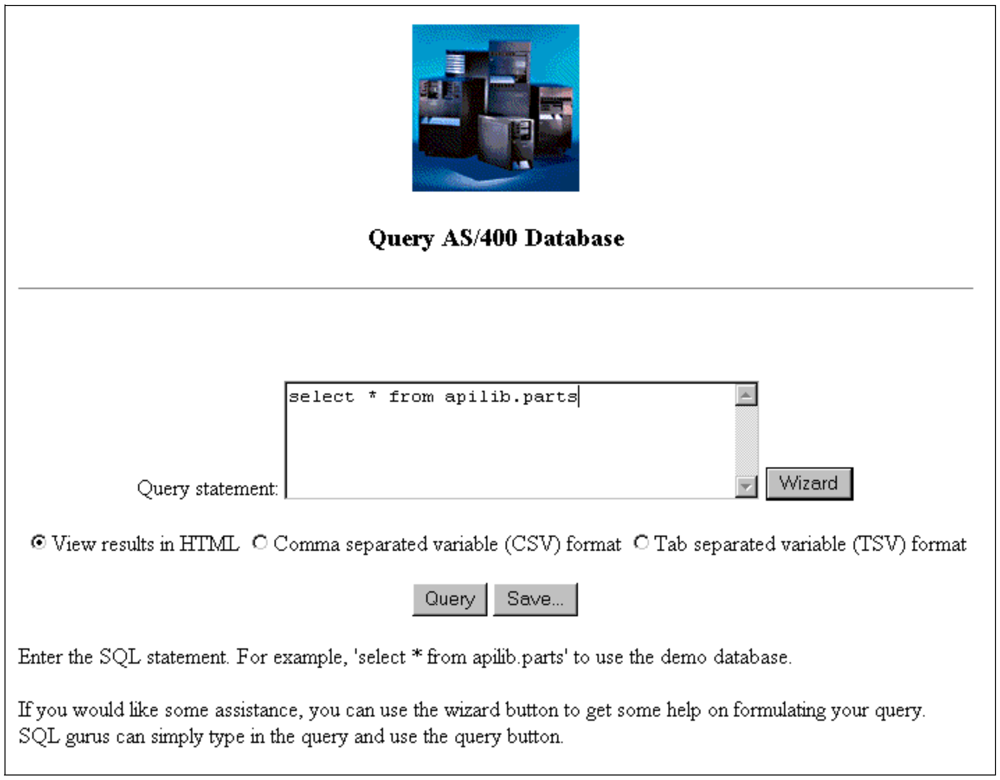


Figure 196. Query Recall Window

The Query Recall window allows you to recall a previous query or build a new query. To build a new query, highlight **new query**, and click on the **Open** button. This displays the Query Builder window, as shown in Figure 197.



Query statement:

```
select * from apilib.parts
```

[✓ View results in HTML] Comma separated variable (CSV) format   Tab separated variable (TSV) format

Enter the SQL statement. For example, 'select * from apilib.parts' to use the demo database.

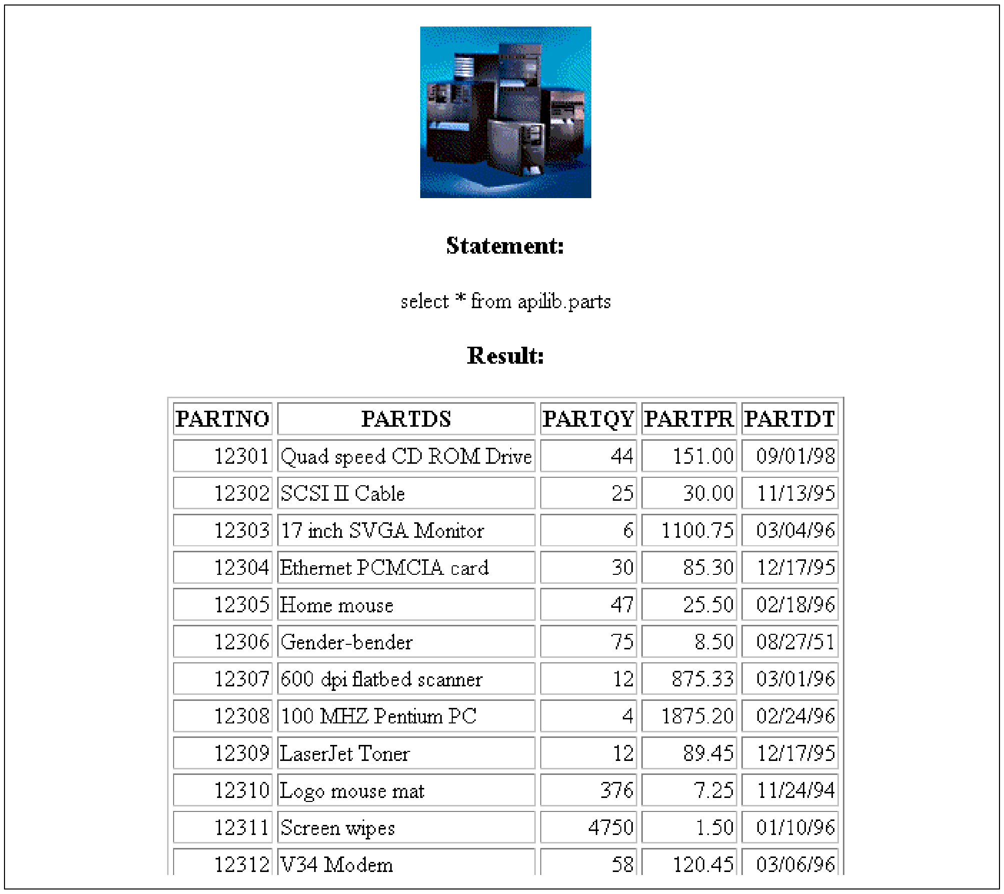
If you would like some assistance, you can use the wizard button to get some help on formulating your query. SQL gurus can simply type in the query and use the query button.

Figure 197. Query Builder Window

The Query Builder Screen allows you to enter an SQL statement and run it on the AS/400 system. For example, to retrieve all rows from the Parts table in the apilib library, enter:

```
select * from from apilib.parts
```

Clicking on the Query button causes the SQL statement to run on the AS/400 system. The results appear in a table as shown in Figure 198.



Statement:

select * from apilib.parts

Result:

PARTNO	PARTDS	PARTQY	PARTPR	PARTDT
12301	Quad speed CD ROM Drive	44	151.00	09/01/98
12302	SCSI II Cable	25	30.00	11/13/95
12303	17 inch SVGA Monitor	6	1100.75	03/04/96
12304	Ethernet PCMCIA card	30	85.30	12/17/95
12305	Home mouse	47	25.50	02/18/96
12306	Gender-bender	75	8.50	08/27/51
12307	600 dpi flatbed scanner	12	875.33	03/01/96
12308	100 MHZ Pentium PC	4	1875.20	02/24/96
12309	LaserJet Toner	12	89.45	12/17/95
12310	Logo mouse mat	376	7.25	11/24/94
12311	Screen wipes	4750	1.50	01/10/96
12312	V34 Modem	58	120.45	03/06/96

Figure 198. Query Results

Rather than writing an SQL statement, click on the Wizard button and allow the Query Wizard to help you build the SQL statement that you want to run. The Query Wizard prompts you for the name of the table that you want to use. Click on the **Browse** button to display the available tables or enter the name of the table, as shown in Figure 199 on page 226.

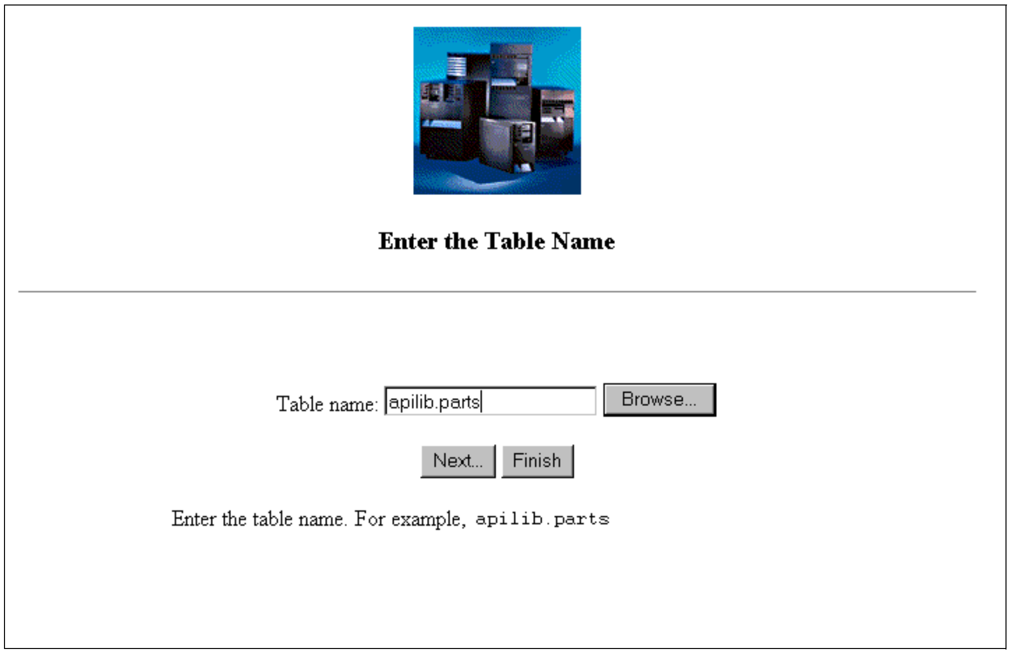


Table name: apilib.parts

Enter the table name. For example, apilib.parts

Figure 199. Query Wizard Table Prompt

After a table name is entered, click on the **Next** button. The Select Fields prompt window appears as shown in Figure 200.

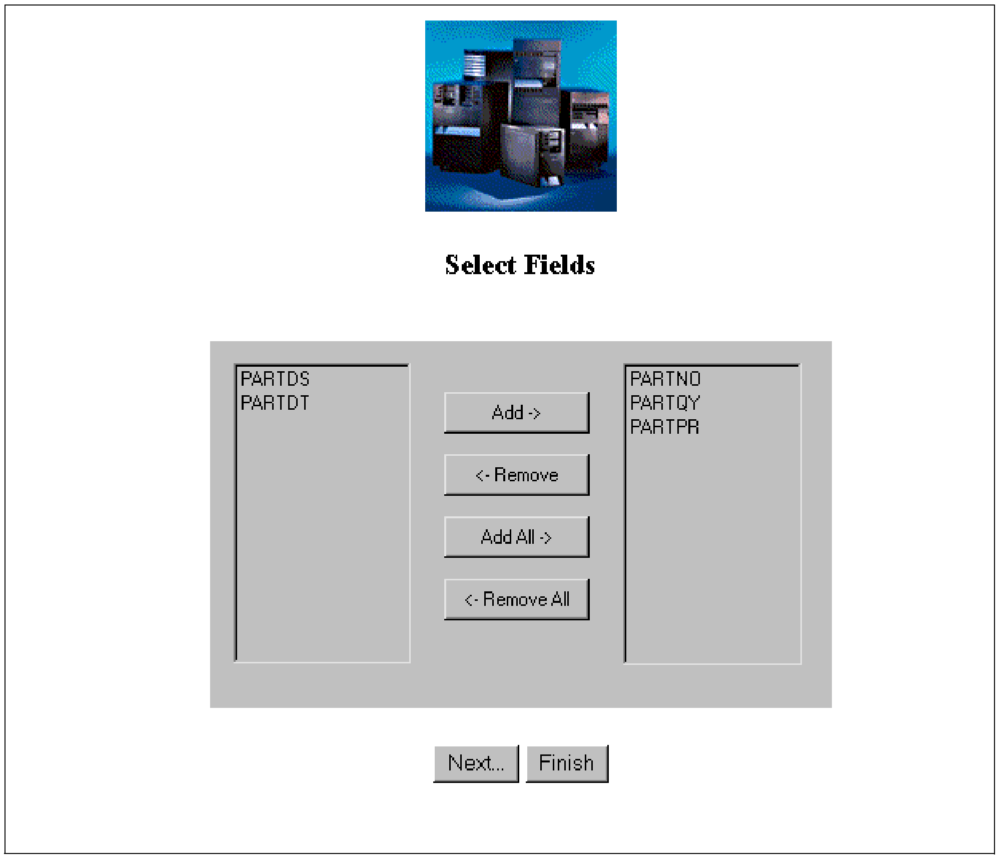


Figure 200. Select Fields Prompt Window

The Select Fields window allows you to select which fields you want to include in the query. To select a field, highlight it and click on the **Add** button. In this example, we select the PARTNO, PARTQY, and PARTPR fields. After selecting the fields you require, click on the **Next** button to display the Select Conditions window shown in Figure 201.

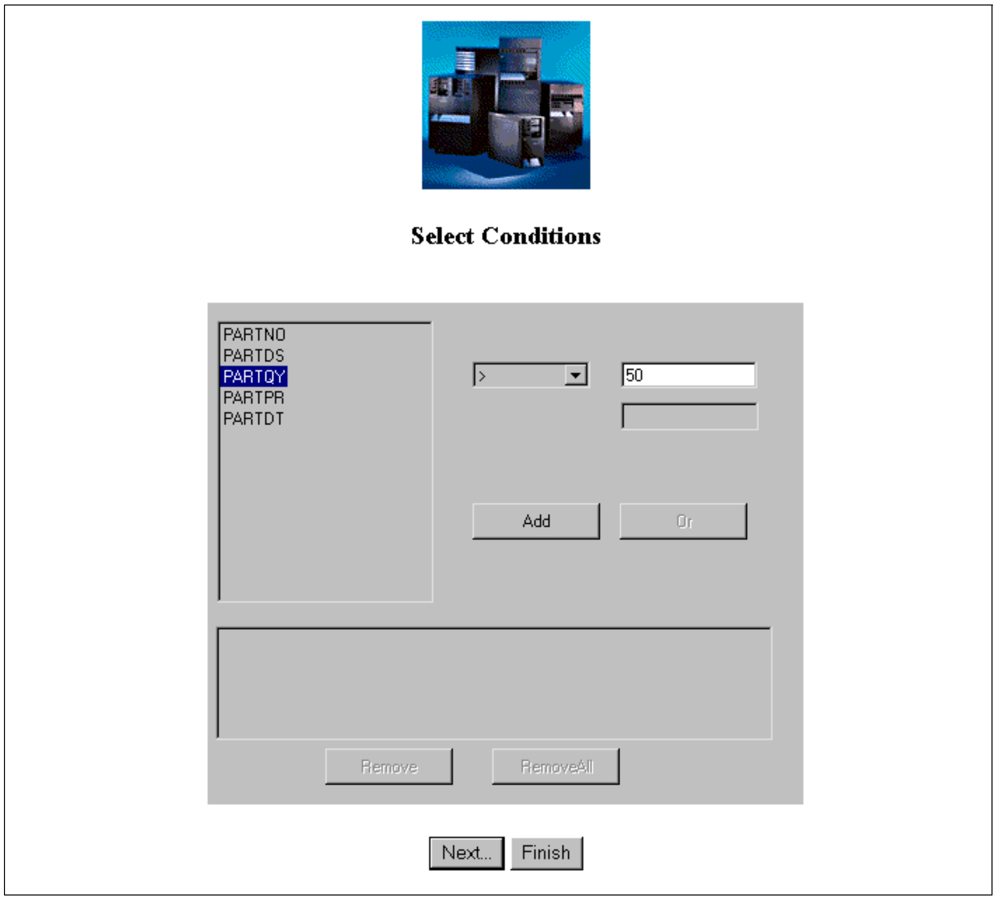


Figure 201. Select Conditions Prompt Window

The Select Conditions prompt allows you to add conditions to the SQL statement. For example, to select only those records that have a PARTQY value of greater than 50, perform the following steps:

1. Highlight the **PARTQY** field.
2. Select **">"** from the choice box.
3. Enter **50** in the TextField.

After setting the conditions, click on the **Next** button to display the Select Order prompt window shown in Figure 202 on page 228.

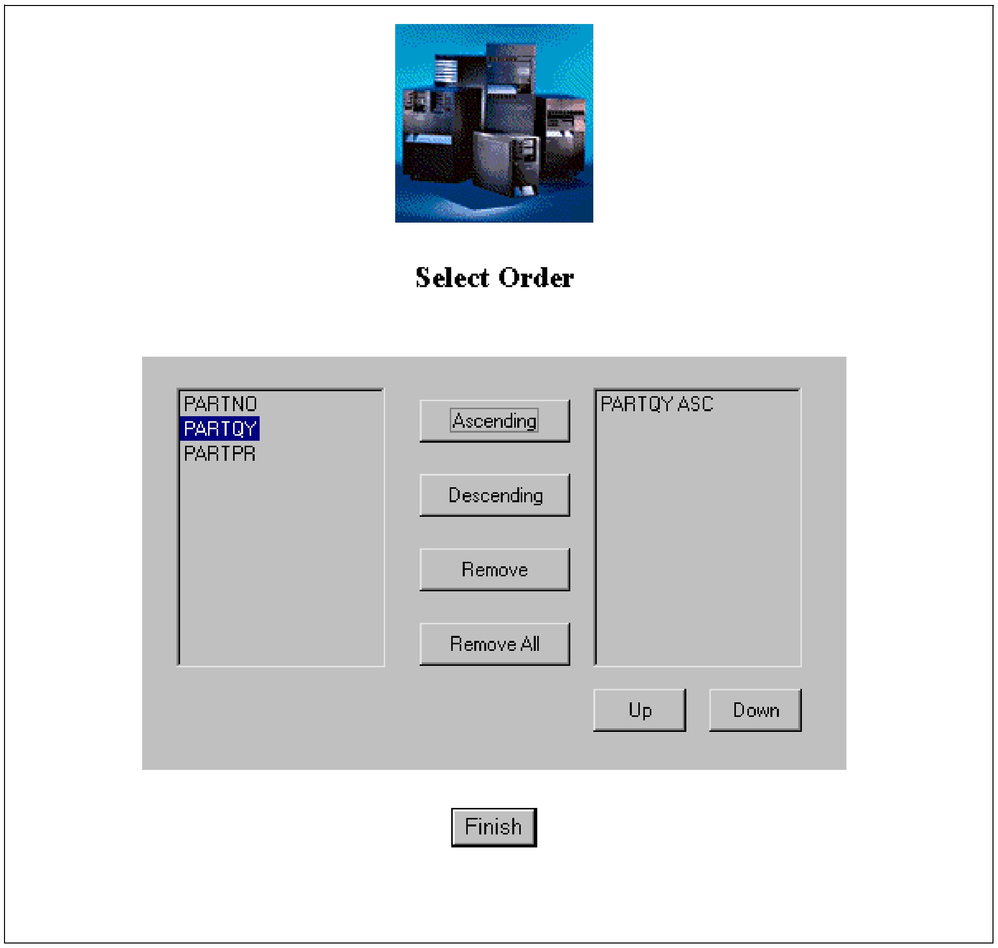
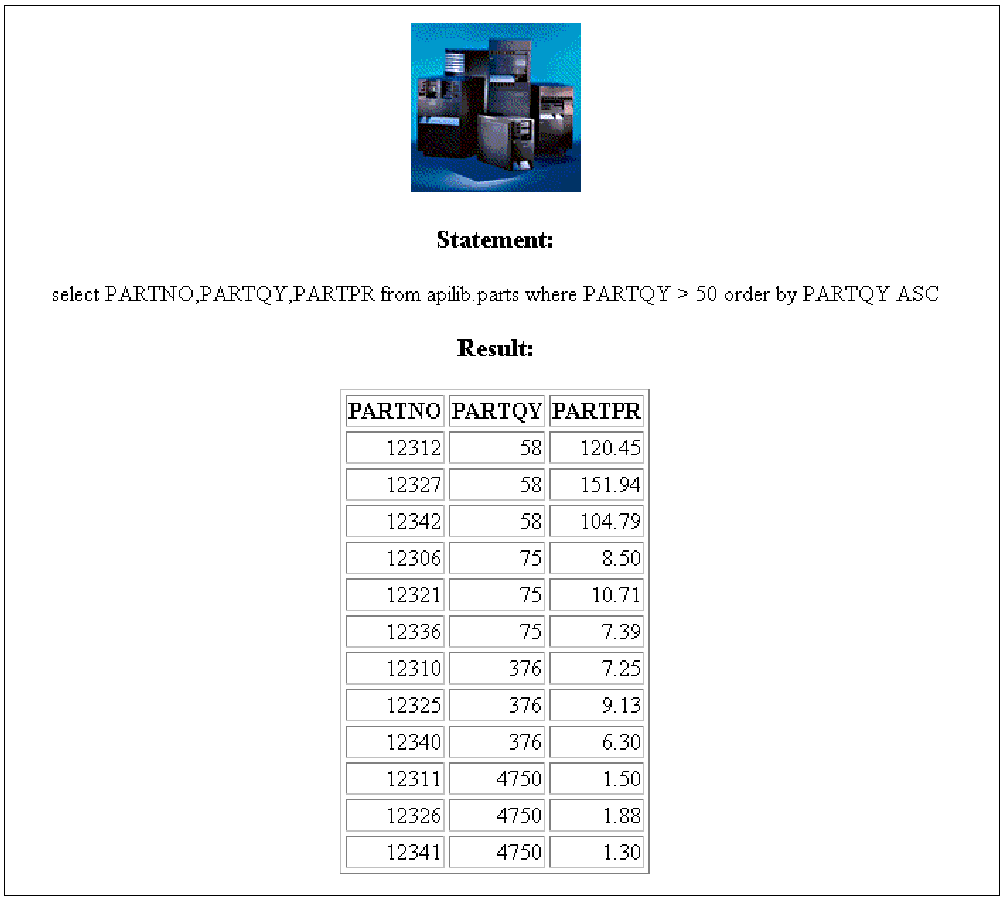


Figure 202. Select Order Prompt Window

The Select Order screen allows you to add ordering information to the SQL statement. For example, to order the records based on the PARTQY field in ascending order, perform these steps:

1. Highlight the **PARTQY** field.
2. Click on the **Ascending** button.

Click on the **Finish** button to see the SQL statement so you can run it. Figure 203 on page 229 shows the SQL statement and the results of running it.



Statement:

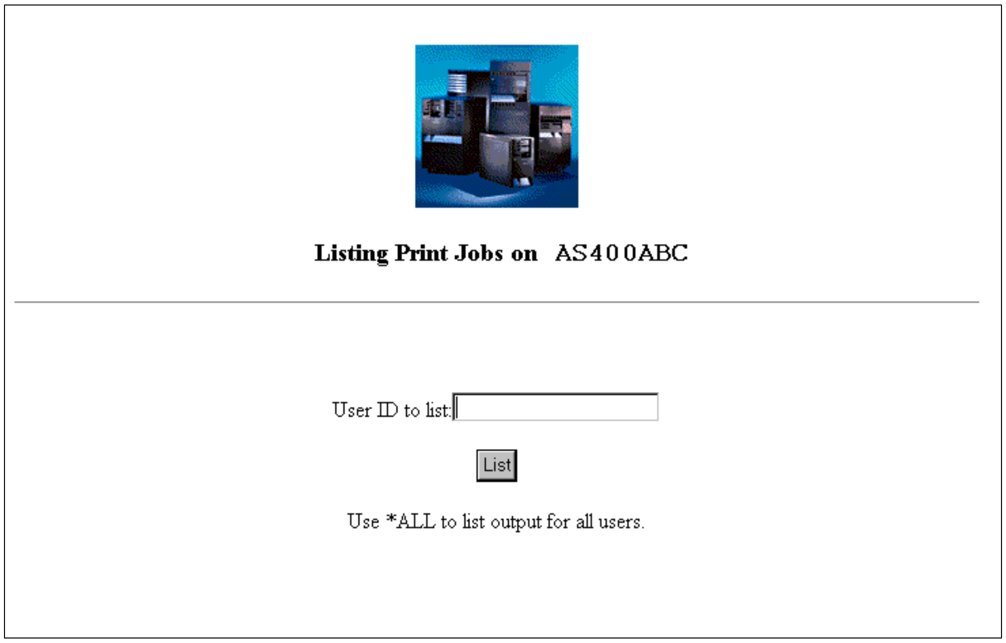
select PARTNO,PARTQY,PARTPR from apilib.parts where PARTQY > 50 order by PARTQY ASC

Result:

PARTNO	PARTQY	PARTPR
12312	58	120.45
12327	58	151.94
12342	58	104.79
12306	75	8.50
12321	75	10.71
12336	75	7.39
12310	376	7.25
12325	376	9.13
12340	376	6.30
12311	4750	1.50
12326	4750	1.88
12341	4750	1.30

Figure 203. Query Results

The second option on the application menu shown in Figure 195 on page 223, allows you to manage print jobs by user. Selecting this option displays the User ID Prompt shown in Figure 204 on page 230.



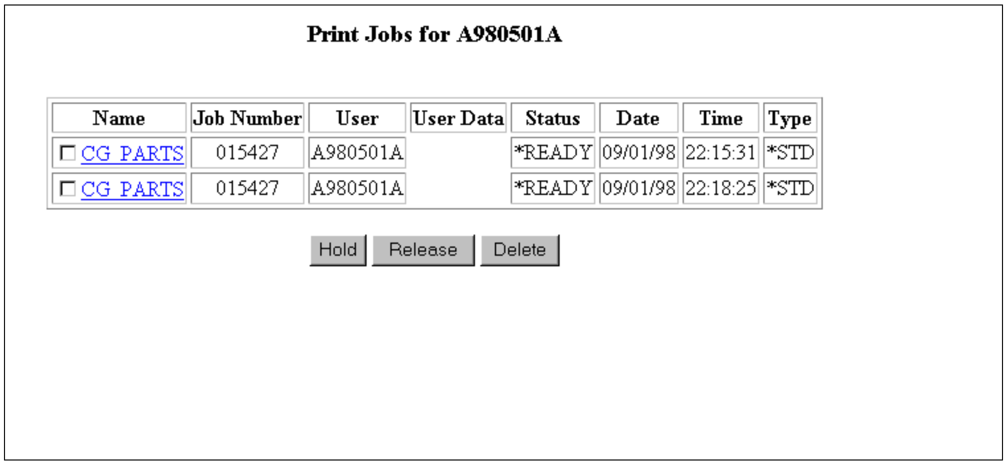
Listing Print Jobs on AS400ABC

User ID to list:

Use *ALL to list output for all users.

Figure 204. User ID Prompt

Enter a valid user ID to view all the print jobs available for the user as shown in Figure 205.



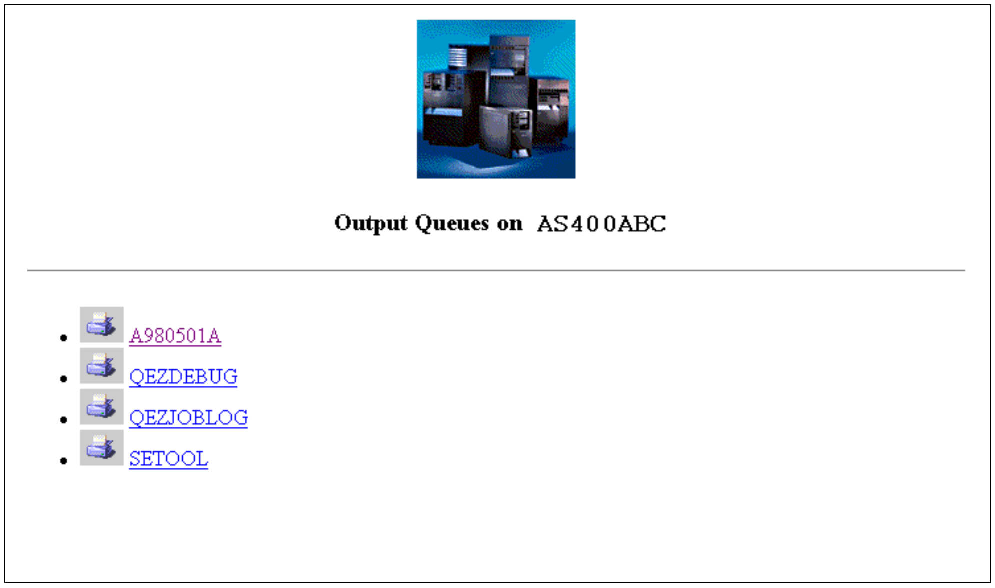
Print Jobs for A980501A

Name	Job Number	User	User Data	Status	Date	Time	Type
☐ CG PARTS	015427	A980501A		*READY	09/01/98	22:15:31	*STD
☐ CG PARTS	015427	A980501A		*READY	09/01/98	22:18:25	*STD

Figure 205. Print Jobs By User

The Print Jobs window allows you to hold the print job, release it, or delete it from the output queue.

You can also work with print jobs by output queue. Select **Manage print jobs by output queue** to view the AS/400 Output Queues as shown in Figure 206 on page 231.

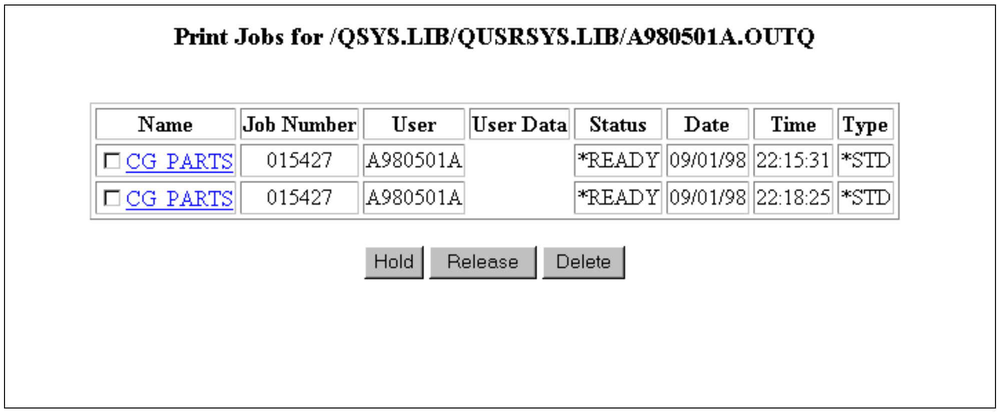


Output Queues on AS400ABC

- A980501A
- QEZDEBUG
- QEZJOBLOG
- SETOOL

Figure 206. Output Queues Display

Click on an output queue to view the print jobs in that particular output queue as shown in Figure 207.



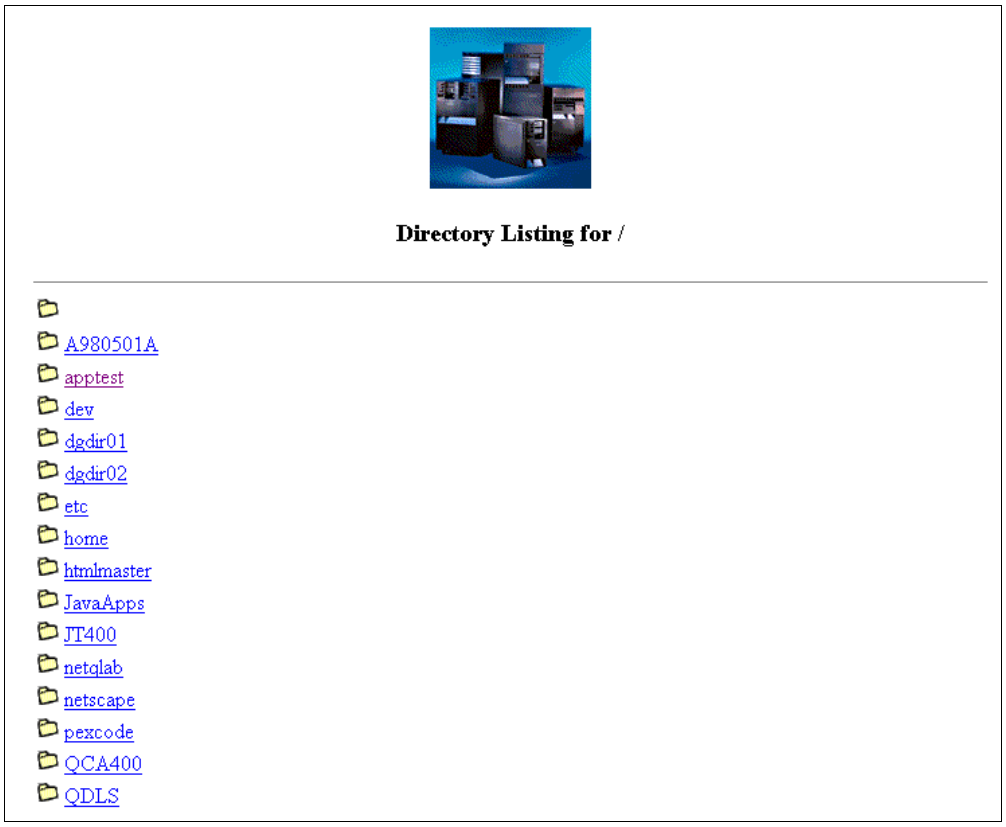
Print Jobs for /QSYS.LIB/QUSRSYS.LIB/A980501A.OUTQ

Name	Job Number	User	User Data	Status	Date	Time	Type
☐ CG PARTS	015427	A980501A		*READY	09/01/98	22:15:31	*STD
☐ CG PARTS	015427	A980501A		*READY	09/01/98	22:18:25	*STD

Figure 207. Print Jobs By Output Queue

The Print Jobs window allows you to select a job and hold it, release it, or delete it from the output queue.

The Files option on the Application menu allows you to list the directories in the AS/400 integrated file system. Figure 208 on page 232 shows a directory listing for the root of the AS/400 integrated file system.

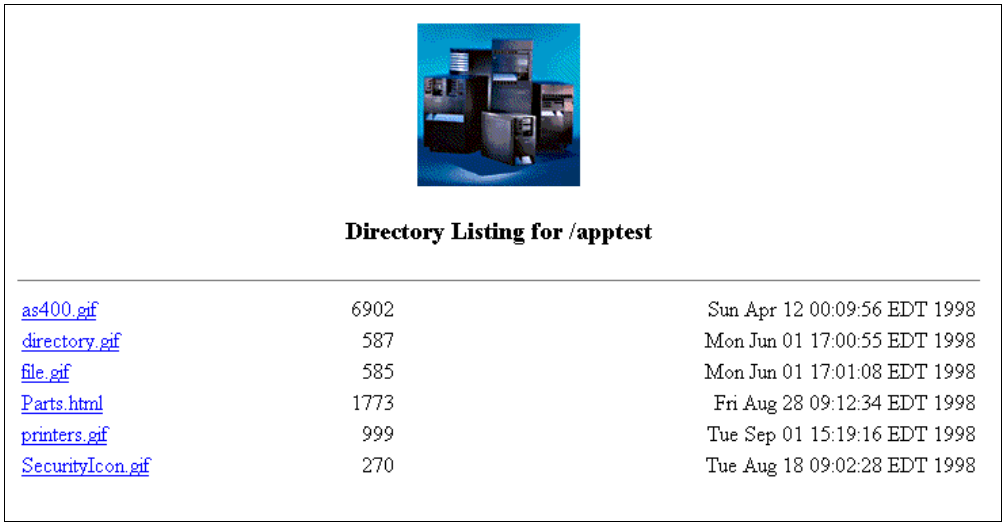


Directory Listing for /

-
- A980501A
- apptest
- dev
- dgdir01
- dgdir02
- etc
- home
- htmlmaster
- JavaApps
- JT400
- netqlab
- netscape
- pexcode
- QCA400
- QDLS

Figure 208. AS/400 Integrated File System Directories

Select and click on one of the directories listed to view the files stored in that directory. Click on the apptest directory to view the contents of that directory as shown in Figure 209.

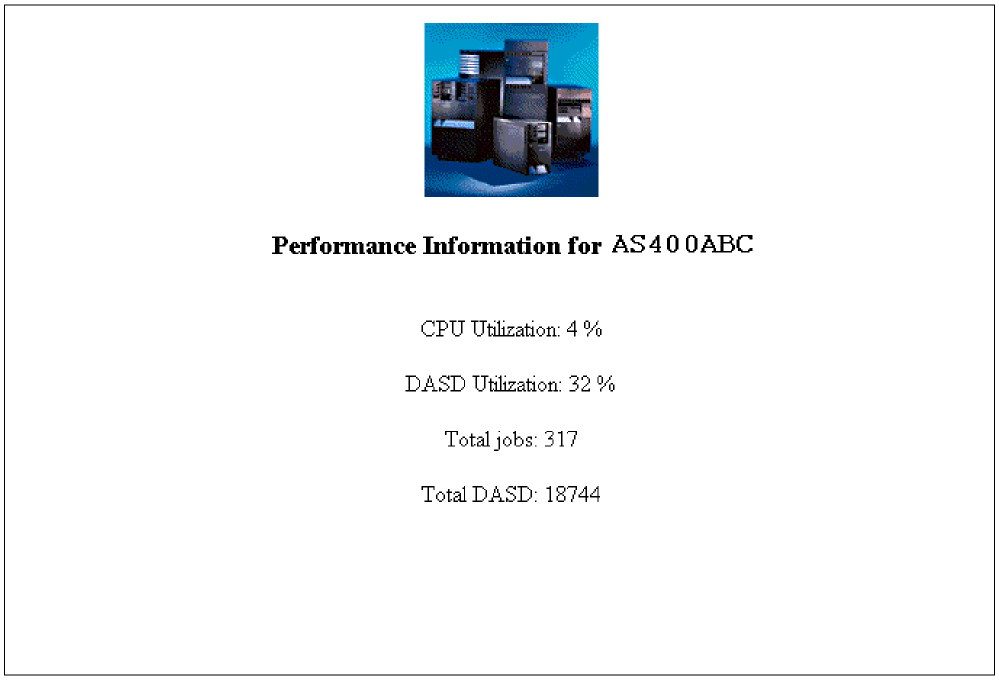


Directory Listing for /apptest

as400.gif	6902	Sun Apr 12 00:09:56 EDT 1998
directory.gif	587	Mon Jun 01 17:00:55 EDT 1998
file.gif	585	Mon Jun 01 17:01:08 EDT 1998
Parts.html	1773	Fri Aug 28 09:12:34 EDT 1998
printers.gif	999	Tue Sep 01 15:19:16 EDT 1998
SecurityIcon.gif	270	Tue Aug 18 09:02:28 EDT 1998

Figure 209. Directory Listing

Selecting System Performance from the application menu retrieves and displays AS/400 system performance information as shown in Figure 210 on page 233.



Performance Information for AS400ABC

CPU Utilization: 4 %

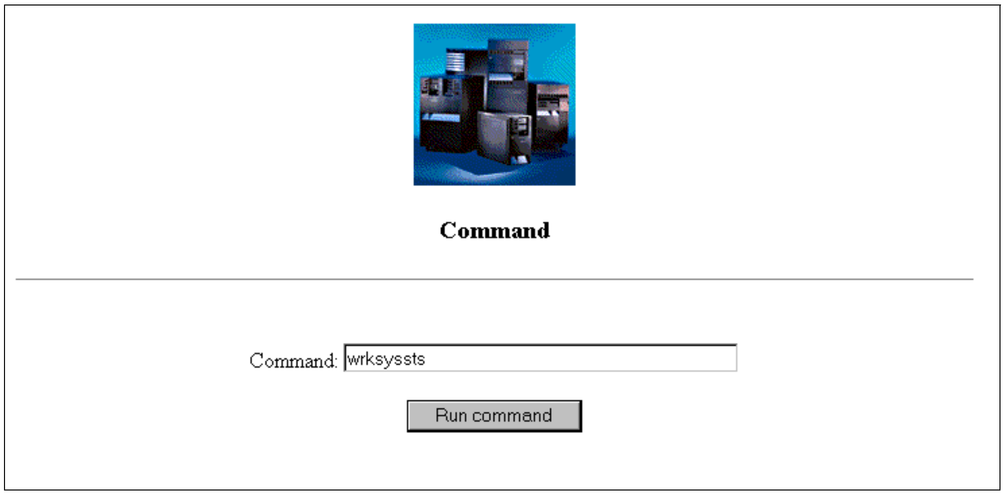
DASD Utilization: 32 %

Total jobs: 317

Total DASD: 18744

Figure 210. AS/400 Performance Information

The Command line option allows you to enter an AS/400 command as shown in Figure 211.



Command

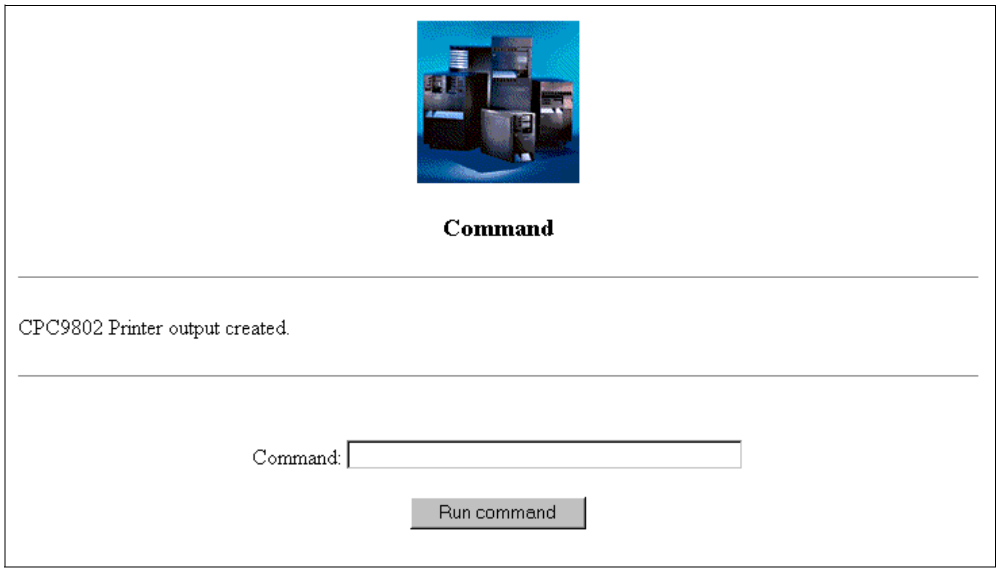
Command: wrksysst s

Figure 211. AS/400 Command

If you enter a valid AS/400 command, for example:

```
wrksyssts
```

The command runs on the AS/400 system, and the command output is written to an AS/400 print file. You are notified that a print file is created as shown in Figure 212 on page 234.



CPC9802 Printer output created.

Figure 212. AS/400 Command Output

The Change password option allows you to change your password on the AS/400 system. Selecting this options displays the screen shown in Figure 213.

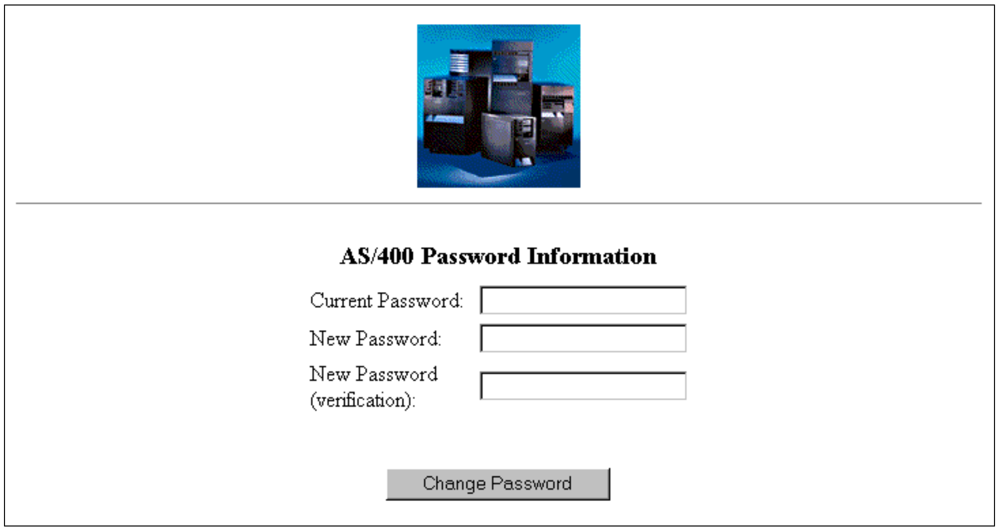


Figure 213. Change Password

The final option is the Sign off option. Selecting it signs you on the AS/400 system and displays the initial sign on menu.

7.2 Application Programs

This section covers the Java programs that support this application. This application consists of a number of Java classes. They are supporting class, servlets, and applets. We cover some of the key classes here to give you an understanding of how the application works. The entire application is available for you to download from our Web site.

Note

If you load the servlets that make up this application into the VisualAge for Java 2.0 Integrated Development environment, some of the classes are marked with a warning message. If you read the warning messages, they indicate that the `getParameter` method is deprecated. However, the Java Servlet API specification (version 2.1a, November 1998), shows this to be a supported method. Originally, Sun decided to deprecate this method, but has since reconsidered.

7.2.1 How the Application Works

This application works by using HTML to display windows in a browser and reading input requests from the HTML screens. The Java programs build the HTML files "on the fly." There are no HTML source files used. Access to the AS/400 system is done through classes provided by the AS/400 Toolbox for Java.

The **Signon** class is a servlet that provides support for signing on to the application. It provides the following support:

- Display initial sign on menu
- Validate sign on information
- Display application main menu

Two methods are provided to allow a servlet to interface with a client:

- doGet
- doPost

Each of these methods has two parameters that are passed in:

- ServletRequest that encapsulates the request to the servlet
- ServletResponse that encapsulates the response from the servlet

Using the ServletRequest interface or its subclass HttpServletRequest, servlets can access protocol-specific header information such as the scheme of the URL used in the request or the value of the specified parameters. After retrieving the data from the HttpServletRequest, the servlet performs the requested task and sends the information back to the client using the ServletResponse object. It allows the servlet to set the MIME content type and a Writer, through which the servlet can pass the information back to the client.

The Signon servlet is run from a browser by entering:

```
http://server:xxxx/servlet/Signon
```

Entering this command from the browser runs the doGet method of the Signon class.

```
 public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
 {
 log("Signon: doGet: Entered.");
String urlStr = new String(HttpUtils.getRequestURL(req));
ServletOutputStream out = res.getOutputStream();
imgBase = "http://" + req.getServerName() + ":" + Integer.toString(req.getServerPort()) + "/";
ServletCallLog.logCaller(req, res);
Enumeration e = req.getParameterNames();
if (!e.hasMoreElements()) {
// a GET request came in with no parameters -
// Generate a signon form
   genSignonForm(out, new String(HttpUtils.getRequestURL(req)));
} else {
    String sysName = req.getParameter("system");
    String userId = req.getParameter("user");
    String urlBase = urlStr.substring(0, urlStr.lastIndexOf("/") + 1);
    String cmd = req.getParameter("cmd");
    if (cmd.equalsIgnoreCase("menu")) {
      genMenu(out, urlBase, sysName, userId);
    } else
        if (cmd.equalsIgnoreCase("signoff")) {
            signOff(req, res);
        } else {
           genInfo(out, imgBase, sysName, userId);
        }
  }
  out.close();
 }
```

Figure 214. SignOn doGet Method

To help you understand the processing that takes place, we added logging to the SignOn class. The log for the sign on processing shows the following entries.

- Signon: Signon: doGet: Entered.

  The first time the doGet method is called, no parameters are passed in. It calls the genSignonForm method to generate and display the Sign on menu.

- Signon: Signon: genSignonForm: Entered.

  The genSignonForm method shown in Figure 215, builds the Sign on menu using HTML tags. It specifies that the Post method be used and the Signon program called.

```
private void genSignonForm(ServletOutputStream out, String urlBase)
throws IOException
{
log("Signon: genSignonForm: Entered.");
log("Signon: genSignonForm: imgBase = " + imgBase);
String host = getHostFromURL(urlBase);
out.println("<HTML><HEAD><TITLE> " + host + " AS/400 Signon </TITLE></HEAD>");
out.println("<BODY>");
out.println("<br><br>");
out.println("<center>");
out.println("<img src=" + imgBase + "apptest/as400.gif alt=\"" + host + "\">");

out.println("<br><br>");
out.println("<p>Welcome to " + host);
out.println("<p>Please log in:");
out.print("<FORM ACTION=\"");
out.print(urlBase);
out.println("\" METHOD=\"POST\">");
out.println("<TABLE ALIGN=CENTER WIDTH=40%>");
out.println("<CAPTION><BIG><STRONG>AS/400 Logon Information</STRONG></BIG></CAPTION>");
out.println("<TR>");
out.println("<TD></TD>");
out.println("<TD></TD>");
out.println("</TR>");
out.println("<TR>");
out.println("<TD>User ID:</TD>");
out.println("<TD><INPUT TYPE=\"text\" NAME=\"user\" MAXLENGTH=\"10\"></TD>");
out.println("</TR>");
out.println("<TR>");
out.println("<TD>Password:</TD>");
out.println("<TD><INPUT TYPE=\"password\" NAME=\"pw\" MAXLENGTH=\"10\"></TD>");
out.println("</TR>");
out.println("<TR>");
out.println("<TD>AS/400 System:</TD>");
out.println("<TD><INPUT TYPE=\"text\" NAME=\"system\"  MAXLENGTH=\"10\"></TD>");
out.println("</TR>");
out.println("</TABLE>");
// output submit button
out.println("<P ALIGN=Center>");

out.println("<INPUT TYPE=submit VALUE=\"Signon\">");
out.println("</FORM>");

out.println("</center>");

out.println("</BODY></HTML>");

}
```

Figure 215. SignOn genSignonForm Method

The urlBase variable, contains the name of the program to run. This causes the Sign on menu shown in Figure 194 on page 222 to be displayed. The HTML code that is generated by the genSignonForm method is shown in Figure 216 on page 238.

```
<HTML><HEAD><TITLE> LOCALHOST AS/400 Signon </TITLE></HEAD>
<BODY>
<br><br>
<center>
<img src=http://localhost:80/apptest/as400.gif alt="LOCALHOST">
<br><br>
<p>Welcome to LOCALHOST
<p>Please log in:
<FORM ACTION="http://sysname:xxxx/servlet/Signon" METHOD="POST">
<TABLE ALIGN=CENTER WIDTH=40%>
<CAPTION><BIG><STRONG>AS/400 Logon Information</STRONG></BIG></CAPTION>
<TR>
<TD></TD>
<TD></TD>
</TR>
<TR>
<TD>User ID:</TD>
<TD><INPUT TYPE="text" NAME="user" MAXLENGTH="10"></TD>
</TR>
<TR>
<TD>Password:</TD>
<TD><INPUT TYPE="password" NAME="pw" MAXLENGTH="10"></TD>
</TR>
<TR>
<TD>AS/400 System:</TD>
<TD><INPUT TYPE="text" NAME="system"  MAXLENGTH="10"></TD>
</TR>
</TABLE>
<P ALIGN=Center>
<INPUT TYPE=submit VALUE="Signon">
</FORM>
</center>
</BODY></HTML>
```

Figure 216. Generated Sign On HTML

Two key points to notice about the HTML file are:

- The FORM ACTION tag specifies that the **Signon** servlet will be run using the Post method.
- The INPUT TYPE tag specifies that clicking on a **Signon** button causes the servlet to run.

After the Signon button is clicked, the doPost method of the SignOn class is called:

- Signon: Signon: doPost: Entered.

  This method calls the addUserToCache method in the SuperServlet class, which tries to create an as400 object using the sign on information. If this is successful, the user is valid and the main application window appears.

```
public void doPost (HttpServletRequest req, HttpServletResponse res)throws ServletException, IOException
{
log("Signon: doPost: Entered.");
ServletCallLog.logCaller(req, res);

String sysName = req.getParameter("system");
String userId = req.getParameter("user");
String password = req.getParameter("pw");


String urlStr = new String(HttpUtils.getRequestURL(req));
String urlBase = urlStr.substring(0, urlStr.lastIndexOf("/") + 1);
log("urlBase = " + urlBase);

ServletOutputStream out = res.getOutputStream();

// set content type and other response header fields first
res.setContentType("text/html");

try
{
    addUsertoCache(req, res, sysName, userId, password);

    genMain(out, urlBase, sysName, userId);
}
catch (Exception e)
{
out.println("<HEAD><TITLE> AS/400 </TITLE></HEAD><BODY>");
out.println("<br>");

out.println("<p>Signon to AS/400 failed");
out.println("<p>" + e.getMessage());
out.println("</BODY>");

e.printStackTrace();
}

    // then write the data of the response
out.close();
}
```

Figure 217. SignOn doPost Method

- Signon: urlBase = http://localhost/servlet/
- Signon: Signon: genMain: Entered.

  After a successful sign on, the genMain method displays the main application menu and the information menu shown in Figure 195 on page 223.

```
private void genMain(ServletOutputStream out, String urlBase, String sysName, String userId)
throws IOException
{
log("Signon: genMain: Entered.");
out.println("<html>");
out.println("<head>");
out.println("<title>AS/400</title>");
out.println("</head>");

out.println("<frameset cols=25%,*>");
out.print("<frame src=");
out.println(urlBase + "Signon?cmd=menu&system=" + sysName
+ "&user=" + userId + " name=nav>");
out.print("<frame src=");
out.println(urlBase + "Signon?cmd=info&system=" + sysName
+ "&user=" + userId + " name=content>");
out.println("</frameset>");
out.println("</html>");
}
```

Figure 218. SignOn genMain Method

- Signon: Signon: doGet: Entered.
- Signon: Signon: genMenu: Entered.

  The SignOn doGet method is called twice. The first time it is called with a parameter named **cmd**, which is set equal to **menu**. This causes the genMenu method to be called to generate the application menu. The HTML tags generated by the genMenu method are shown in Figure 219 on page 240.

```
<HEAD><TITLE> AS/400 </TITLE></HEAD>
<BODY TEXT="#000000" BGCOLOR="#CCCCCC" LINK="#0000EE" VLINK="#551A8B" ALINK="#FF0000">
<center>
<img src=http://AS400ABC:1040/apptest/as400.gif alt="AS400ABC">
<h3>AS400ABC</h3>
</center>
<hr width="100%">
<ul>
<li><A HREF=<li><A
HREF=http://AS400ABC:1040/servlet/DbSelectServlet?system=AS400ABC&user=auser&cmd=qr
ylist target=content>Database query</A></li>
<li><A HREF=http://AS400ABC:1040/servlet/PrintJob?system=AS400ABC&user=auser&cmd=init&type=user
target=content>Manage print jobs by user</A></li>
<li><A HREF=http://AS400ABC:1040/servlet/PrintJob?system=AS400ABC&user=auser&cmd=init&type=outq
target=content>Manage print jobs by output queue</A></li>
<li><A HREF=http://AS400ABC:1040/servlet/IfsFileServlet?system=AS400ABC&user=auser&cmd=list&dir=/
target=content>Files</A></li>
<li><A HREF=http://AS400ABC:1040/servlet/PerfMon?system=AS400ABC&user=auser target=content>System
performance</A></li>
<li><A HREF=http://AS400ABC:1040/servlet/CmdCallServlet?system=AS400ABC&user=auser
target=content>Command line</A></li>
<li><A HREF=http://AS400ABC:1040/servlet/ChangePwdServlet?system=AS400ABC&user=auser
target=content>Change password</li>
<li><A HREF=http://AS400ABC:1040/servlet/Signon?system=AS400ABC&user=auser&cmd=signoff target=_top>Sign
off</A></li>
</ul>
<hr width="100%">
</BODY>
```

Figure 219. Application Menu HTML

The HTML tags control how the other Java servlets are run. To understand how it works, look at the tag for the Database query option:

```
<A
HREF=http://AS400ABC:1040/servlet/DbSelectServlet?system=AS400ABC&user=auser
&cmd=qrylist target=content>Database query
</A>
```

HTML stands for hypertext markup language. Links are the hyper part of hypertext. Links, which are also called anchors, mark text or images as elements that point to other documents, images, applets, or, in this case, servlets. Links are made up of three elements:

- An anchor tag <A>, which marks the text or image as a link
- An attribute, HREF=" ", which is located within the opening anchor tag
- An address (URL), which tells the browser what to link to, http://AS400ABC:1040/servlet/DbSelectServlet

In this case, clicking on the text **DataBase query** links us to the servlet named DbSelectServlet:

- Signon: Signon: doGet: Entered.
- Signon: Signon: genInfo: Entered.

Finally the doGet method is called again with a parameter of info. The genInfo method is called to generate the right portion of the application menu. The window shown in Figure 220 appears.

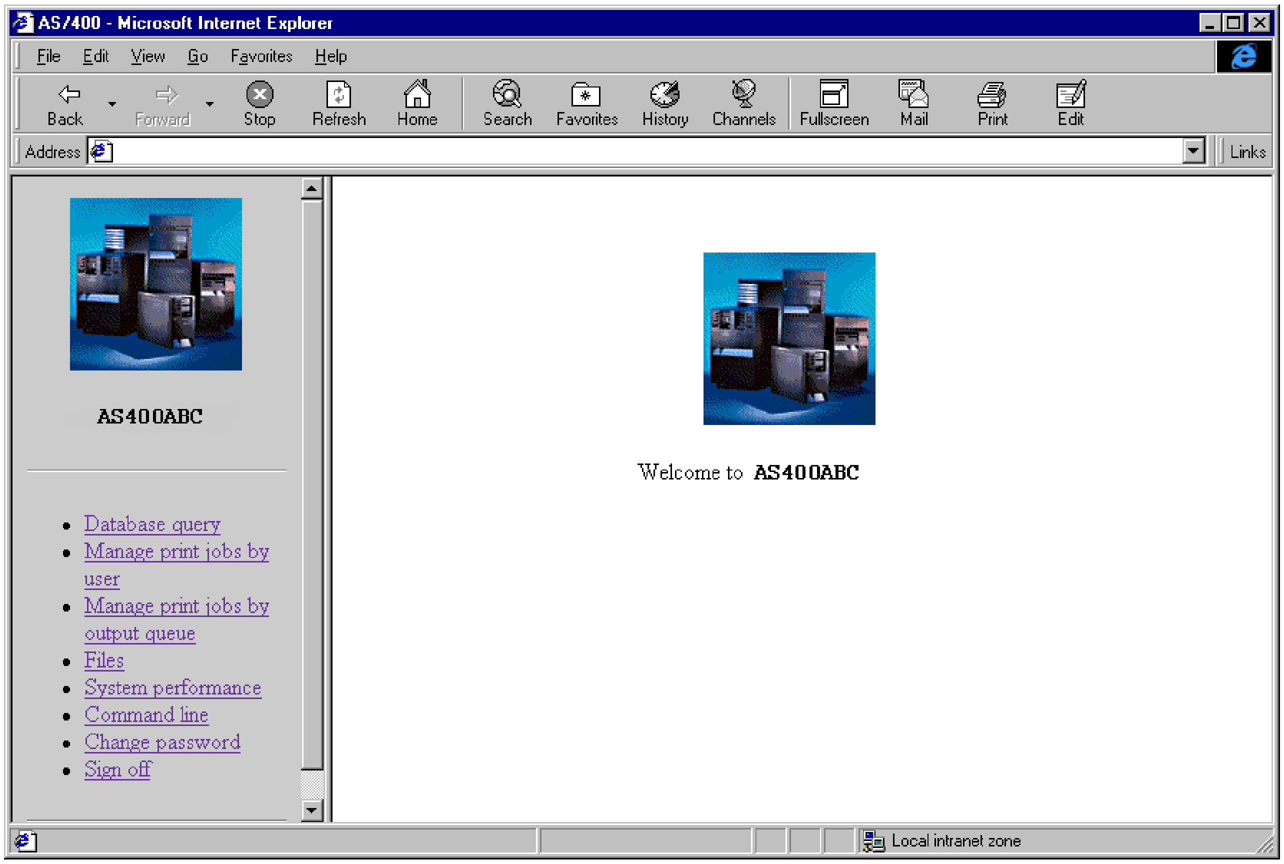

AS/400 - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print Edit
Address
Links
AS400ABC
Database query
Manage print jobs by user
Manage print jobs by output queue
Files
System performance
Command line
Change password
Sign off
Welcome to AS400ABC
Local intranet zone

Figure 220. Application Menu

7.3 The Java Application Programs

Now that you understand how the servlets are invoked, you can look at how they work. The application servlets are similar. This section looks at a fairly simple servlet, which calls a program on the AS/400 system and a more complex application which allows SQL query statements to be created and executed on the AS/400 system. By understanding how these applications work, you can look at the code and understand how the other applications work.

7.3.1 System Performance Servlet

This section looks at the System performance servlet. This servlet uses the AS/400 Toolbox for Java distributed program call (DPC) class to call a program on the AS/400 system, which returns system performance information. The information returned by the AS/400 program is displayed in the browser by imbedding it in an HTML file.

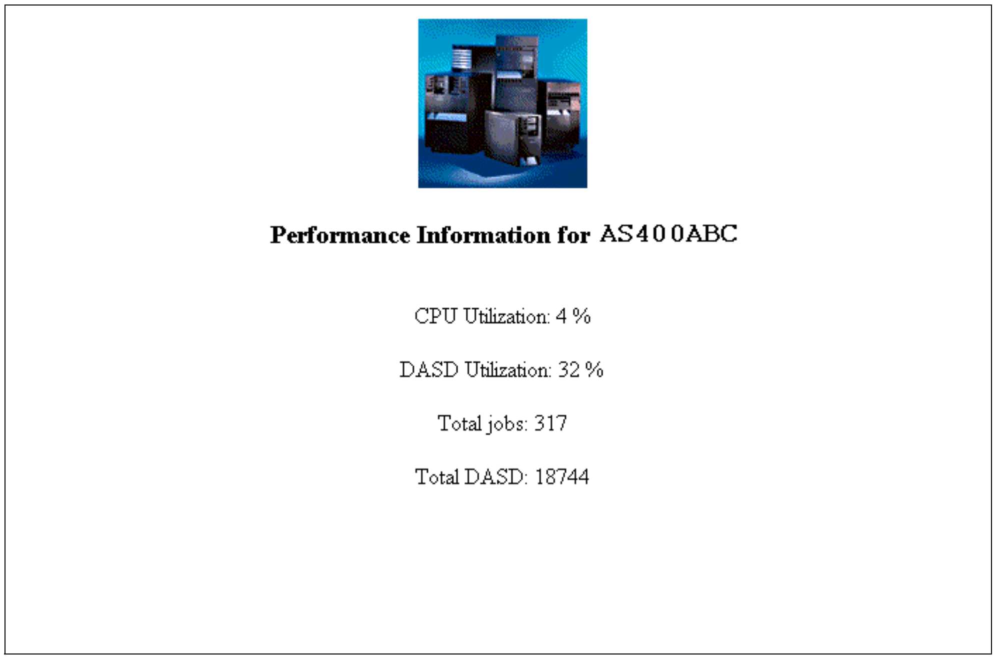
```
 public void doGet (HttpServletRequest req, HttpServletResponse res)throws ServletException, IOException
 {
AS400 sys = null;
imgBase = "http://" + req.getServerName() + ":"+ Integer.toString(req.getServerPort())
+ "/";
String sysName = req.getParameter("system");
String userId = req.getParameter("user");
ServletCallLog.logCaller(req, res);

ServletOutputStream out = res.getOutputStream();

// set content type and other response header fields first
res.setContentType("text/html");

String realSystem = sysName.toUpperCase();
if (sysName.equalsIgnoreCase("localhost"))
{
   String earl = HttpUtils.getRequestURL(req).toString();
   realSystem = getHostFromURL(earl);
}
    // then write the data of the response
System.out.println("PerfMon: doGet: Starting output");
out.println("<HEAD><TITLE> AS/400 Performance Monitor </TITLE></HEAD><BODY>");
out.println("<center>");
out.println("<img src=" + imgBase + "apptest/as400.gif alt=\""  + realSystem + "\">");
out.println("<h3> Performance Information for " + realSystem + " </h3>");
out.println("<br>");

try {
   sys = getSysFromUserCache(req, res, sysName, userId);
   getStatus(out, sys);
} catch (Exception e) {
   out.println(e.toString());
   e.printStackTrace();
}

out.println("</BODY>");
out.close();
sys.disconnectService(AS400.COMMAND);
 }
```

Figure 221. The PerfMon Class doGet Method

Clicking on the System Performance option invokes the doGet method of the PerfMon class, which is shown in Figure 221 on page 242, of the PerfMon servlet. This method builds the HTML tags for the display headings. It then calls the getStatus method.

The getStatus method performs the following actions:

1. Creates a ProgramCall object named pgm.
2. Creates the parameters for the pgm object.
3. Sets the name of the AS/400 program to call equal to **QSYS.LIB/QWCRSSTS.PGM.**
4. Calls the AS/400 program.
5. Formats the returned values into HTML tags for:
   – CPU utilization
   – DASD utilization
   – Total Jobs
   – Total DASD



Performance Information for AS400ABC

CPU Utilization: 4 %

DASD Utilization: 32 %

Total jobs: 317

Total DASD: 18744

Figure 222. System Performance Information

7.3.2 Database Query

The Database Query application is the most complex of the applications. It is actually made up of a combination of applets and servlets. The following classes make up the DataBase Query application:

- DbSelectServlet—Servlet
- SQLOrder—Applet
- SQLWhere—Applet
- SQLWizard—Applet

When you click on the Database query option from the application menu, the doGet method of the DbSelectServlet class is called with a parameter of qrylist. Refer to Figure 219 on page 240, to see the HTML file used. The doGet method checks the parameter to determine which method to call. In this case, the getSavedQueries method is called to allow you to recall previously created queries. This method generates the HTML file shown in Figure 223.

```
<HEAD><TITLE>AS/400 Database Query</TITLE></HEAD>
<BODY>
<center>
<img src=http://AS400ABC:1040/apptest/as400.gif alt="AS400ABC">
<h3>Saved Queries</h3>
</center>
<hr width="100%">
<ul>
<form action=http://AS400ABC:1040/servlet/DbSelectServlet method=post>
<input type=hidden name=cmd value=qryprompt>
<input type=hidden name=system value=AS400ABC>
<input type=hidden name=user value=auser>
<input type=hidden name=key value=as400abcauser908479492129>
<br>
<center>
<SELECT NAME="query" SIZE=5>
<OPTION VALUE=new selected>[new query]
</SELECT>
<br><br>
<input type=submit name=open value=Open>
<input type=submit name=delete value=Delete>
</center>
</BODY>
```

Figure 223. HTML Generated by the getSavedQueries Method

Displaying this HTML file in a browser shows the window that appears in Figure 224 on page 245. The action of this HTML file is to call the DbSelectServlet class using a post method. It sets the following parameters:

- name=cmd, value=qryprompt
- name=open, value=Open
- name=delete, value=Delete

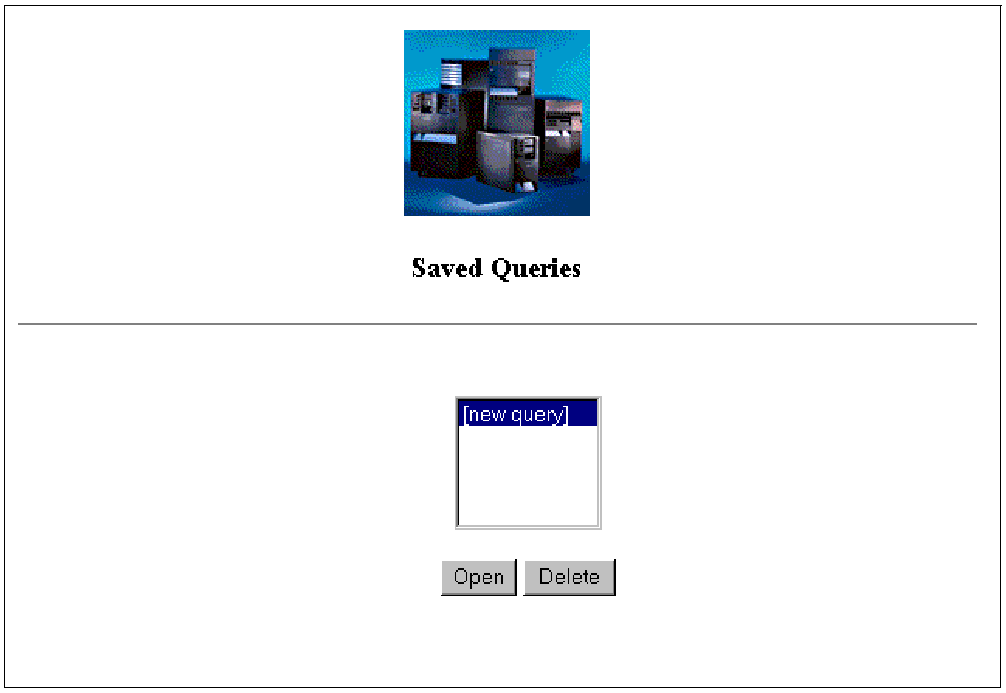


Figure 224. Saved Queries Window

Clicking on the Open button runs the doPost method, passing in a cmd parameter value of qryprompt. If a cmd value of qryprompt is received in the doPost method, the qryPrompt method is called.

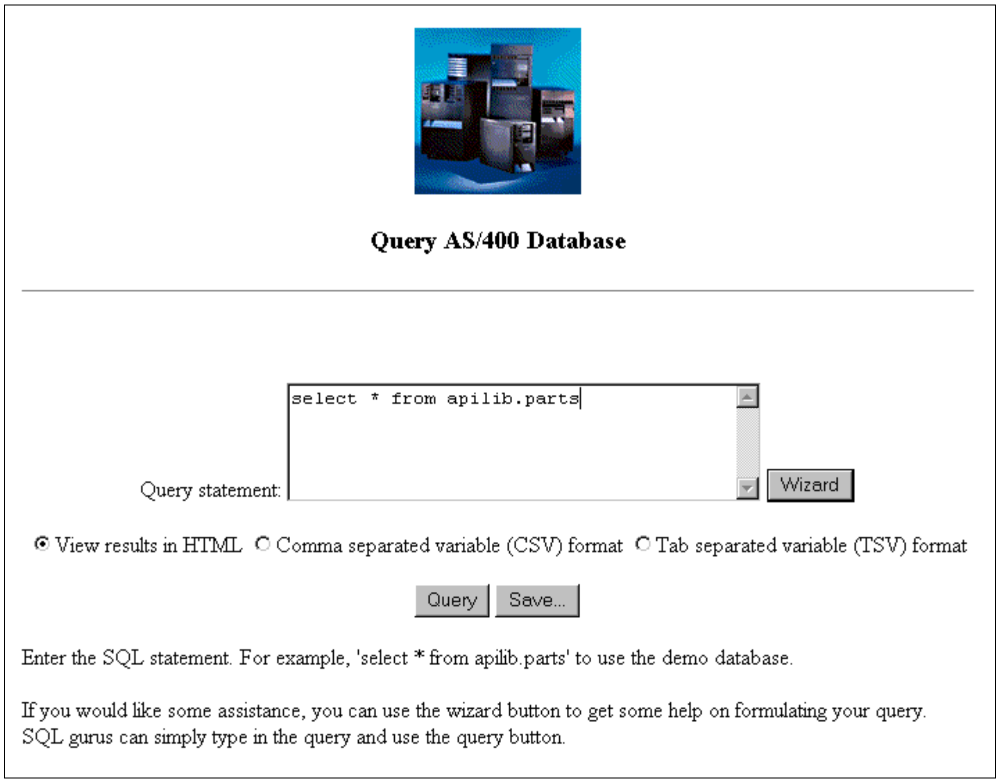
The qryPrompt method builds and displays the HTML file shown in Figure 225 on page 246.

```
o<HEAD><TITLE> Query Statement </TITLE></HEAD><BODY>
 <center>
 <img src=http://AS400ABC:1040/apptest/as400.gif alt="Database Query">
 <h3>Query AS/400 Database</h3>
 </center>
 <hr width="100%">
 <center>
 <form action=http://AS400ABC:1040/servlet/DbSelectServlet method=post>
 <input type=hidden name=cmd value=query>
 <input type=hidden name=system value=as400abc>
 <input type=hidden name=user value=auser>
 <input type=hidden name=key value=as400abcauser908479492129>
 <br><br>
 Query statement:
 <textarea name=statement rows=5 cols=40>
 </textarea>
 <input type=submit name=wizard value=Wizard>
 <br><br>
 <input type=radio name=qtype value="html" checked>View results in HTML
 <input type=radio name=qtype value="CSV">Comma separated variable (CSV) format
 <input type=radio name=qtype value="TSV">Tab separated variable (TSV) format
 <br><br>
 <input type=submit value=Query>
 <input type=submit name=save value=Save...>
 </center>
 </form>
 <p>Enter the SQL statement.
 For example, 'select * from apilib.parts'
 to use the demo database.
 <p>If you would like some assistance, you can use the
 wizard button to get some help on formulating your query.
 SQL gurus can simply type in the query and use the query button.
 </body>
```

Figure 225. HTML Generated by the queryPrompt Method

Displaying this HTML file in a browser shows the window that appears in Figure 226 on page 247. The action of this HTML file is to call the DbSelectServlet class using a post method. It sets the following parameters:

- name=cmd, value=query
- name=system, value=as400abc
- name=user, value=auser
- name=key, value=as400abcauser908479492129
- radio name=qtype, default=html
- name=wizard, value=Wizard
- name=save, value=Save...


Query AS/400 Database

Query statement:

```
select * from apilib.parts
```

[✓ View results in HTML] Comma separated variable (CSV) format Tab separated variable (TSV) format

Enter the SQL statement. For example, 'select * from apilib.parts' to use the demo database.

If you would like some assistance, you can use the wizard button to get some help on formulating your query. SQL gurus can simply type in the query and use the query button.

Figure 226. Query Statement Window

7.3.2.1 Writing your Own SQL Statements

At this point in the Database Query application, you can enter your own SQL statements or use the Wizard to help you build an SQL statement. To see how the Wizard works, refer to Section 7.3.2.2, “Using the Wizard” on page 248. In this section, write our own SQL statement:

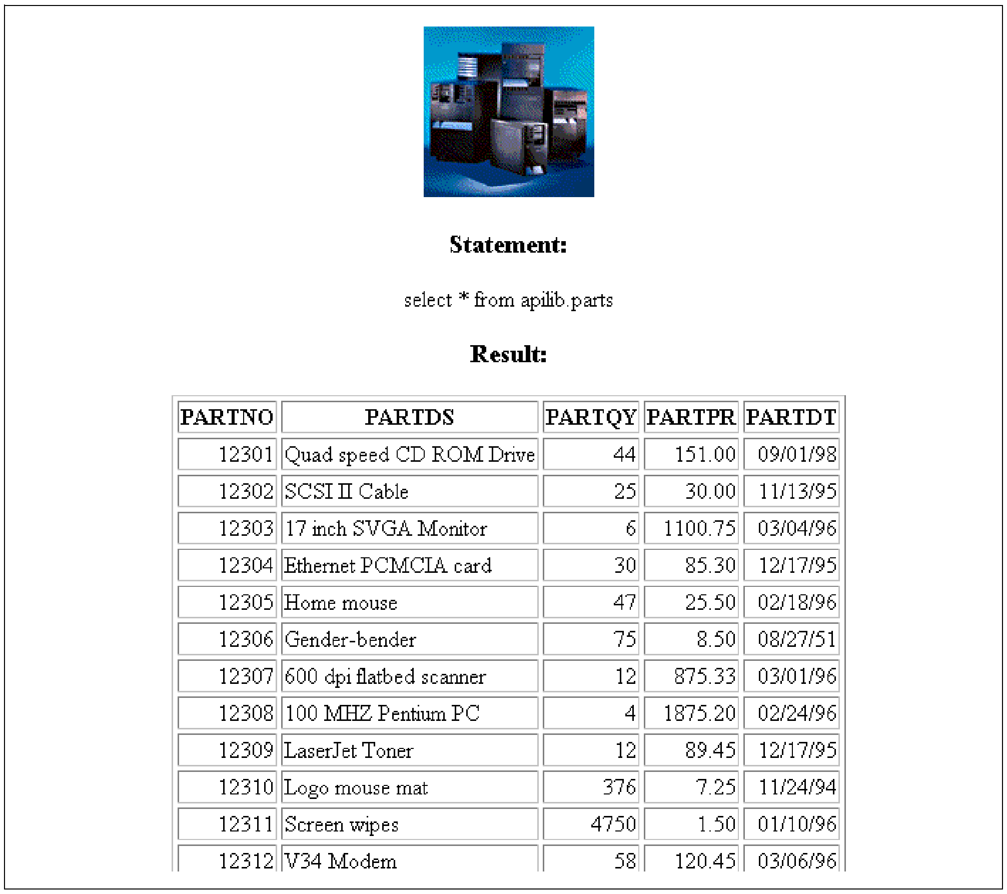
```
select * from apilib.parts
```

Clicking on the Query button runs the doPost method. If a cmd value of query is received in the doPost method, the performQuery method is called. The performQuery method sees how we want the results returned by checking which radio button is selected. In this case, the View results in HTML radio button is selected, so the fillTable method is called.

The fillTable method actually causes the sql statement to be executed on the AS/400 system. It does the following:

- Creates a connection object using the JDBC DriverManager getConnection method
- Creates a statement object using the createStatement method
- Execute the SQL statement we entered using the statement object
- Formats the rows into a table using HTML tags if a resultset is returned
- Causes the browser to execute the HTML file to show the results

The output from the HTML file displayed by the browser is shown in Figure 227 on page 248.



Statement:

select * from apilib.parts

Result:

PARTNO	PARTDS	PARTQY	PARTPR	PARTDT
12301	Quad speed CD ROM Drive	44	151.00	09/01/98
12302	SCSI II Cable	25	30.00	11/13/95
12303	17 inch SVGA Monitor	6	1100.75	03/04/96
12304	Ethernet PCMCIA card	30	85.30	12/17/95
12305	Home mouse	47	25.50	02/18/96
12306	Gender-bender	75	8.50	08/27/51
12307	600 dpi flatbed scanner	12	875.33	03/01/96
12308	100 MHZ Pentium PC	4	1875.20	02/24/96
12309	LaserJet Toner	12	89.45	12/17/95
12310	Logo mouse mat	376	7.25	11/24/94
12311	Screen wipes	4750	1.50	01/10/96
12312	V34 Modem	58	120.45	03/06/96

Figure 227. Query Results

7.3.2.2 Using the Wizard

If the Wizard button is clicked, the wizPrompt method is executed. It builds and shows the HTML file shown in Figure 228 on page 249.

```
<HEAD><TITLE> Query Wizard </TITLE></HEAD><BODY>
<center>
<img src=http://AS400ABC:1040/apptest/as400.gif alt="Database Query">
<h3>Enter the Table Name</h3>
<center>
<hr width="100%">
<center>
<form action=http://as400abc:1040/servlet/DbSelectServlet method=post>
<input type=hidden name=cmd value=qwcolinfo>
<input type=hidden name=system value=as400abc>
<input type=hidden name=user value=auser
<input type=hidden name=key value=as400abcauser908479492129>
<br><br>
Table name: <input type=text name=tblname>
<input type=submit name=wizact value=Browse...>
<br><br>
<input type=submit name=wizact value=Next...>
<input type=submit name=wizact value=Finish>
</center>
</form>
<p>Enter the table name.
For example, 'apilib.parts'
</body>
```

Figure 228. HTML File Generated by the wizPrompt Method

The output of this HTML file appears in a browser as shown in Figure 229.

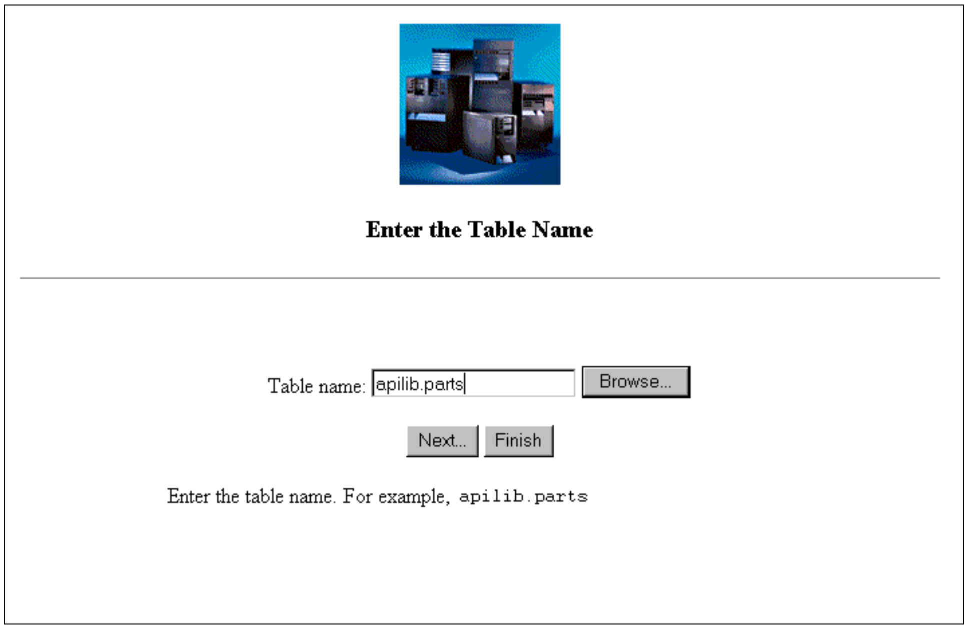

Enter the Table Name

Table name: apilib.parts

Enter the table name. For example, apilib.parts

Figure 229. Enter Table Name Prompt

The action of this HTML file is to call the DbSelectServlet class using a post method. It sets the following parameters:

- name=cmd, value=`qwcolinfo`
- name=system, value=`as400abc`
- name=user, value=`auser`

- name=key, value=as400abcauser908479492129
- radio name=qtype, default=html
- name=wizact, value=`Browse...`
- name=wizact, value=`Next...`
- name=wizact, value=`Finish`

Clicking on the Next button runs the doPost method with a cmd parameter equal to qwcolinfo. This action, in turn, runs the wizColInfo method. It generates and displays the HTML file shown in Figure 230.

```
<HEAD><TITLE> Query Wizard </TITLE></HEAD><BODY>
<center>
<img src=http://AS400ABC:1040/apptest/as400.gif alt="AS400ABC">
<center>
<h3>Select Fields</h3>
<form action=http://AS400ABC:1040/servlet/DbSelectServlet method=post>
<input type=hidden name=cmd value=qwwhere>
<input type=hidden name=system value=AS400ABC>
<input type=hidden name=user value=auser>
<input type=hidden name=key value=AS400ABCauser908479492129>
<br>
<applet code=SqlWizard.class codebase=http://AS400ABC:1040/applets/ height=230 width=390>
<param name=system value="AS400ABC">
<param name=user value="auser">
<param name=key value="AS400ABCauser908479492129">
<param name=url value=http://AS400ABC:1040/servlet/DbSelectServlet>
<param name=fields value="PARTNO,PARTDS,PARTQY,PARTPR,PARTDT">
</applet>
<br><br>
<input type=submit name=wizact value=Next...>
<input type=submit name=wizact value=Finish>
</center>
</form>
</body>
```

Figure 230. HTML Generated By the wizColInfo Method

This HTML file runs an applet named SqlWizard.

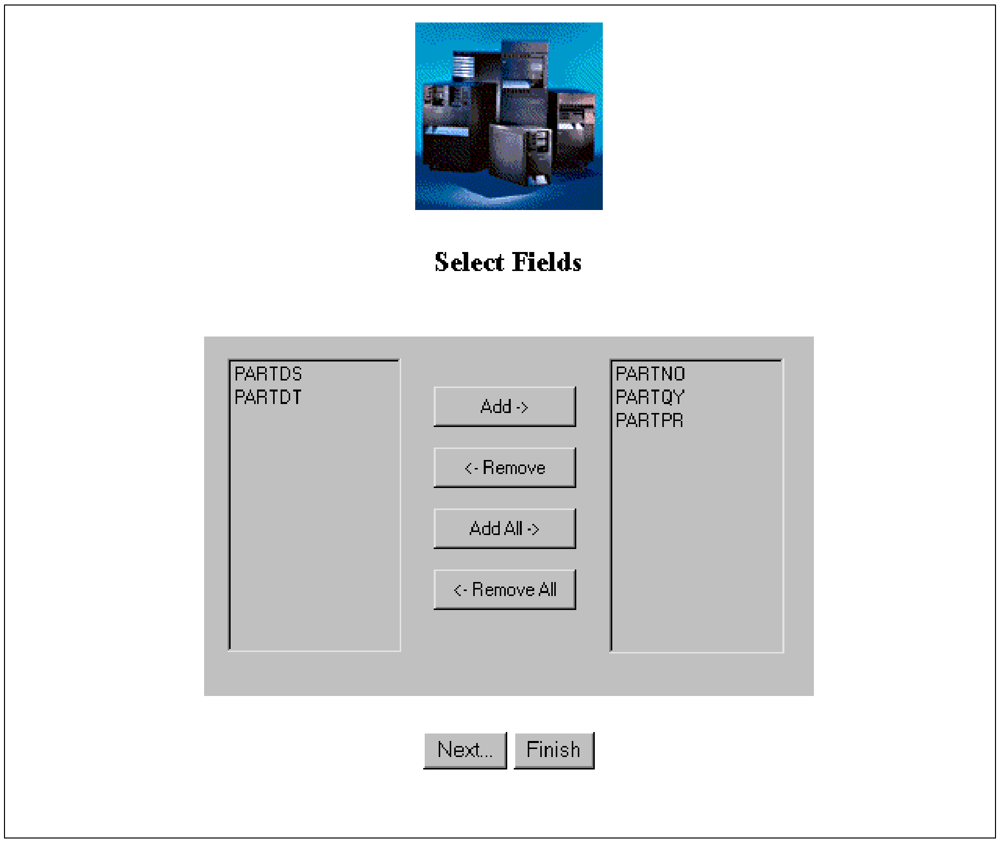


Figure 231. SQLWizard Applet

The SqlWizard applet allows you to select which table columns you want to add to the selection criteria. As shown in Figure 231, we select the PARTNO, PARTQY, and PARTPR columns.

The action of this HTML file is to call the DbSelectServlet class using a post method. It sets the following parameters:

- name=cmd, value=`qwwhere`
- name=system, value=`as400abc`
- name=user, value=`auser`
- name=key, value=`as400abcauser908479492129`
- name=wizact, value=`Next...`
- name=wizact, value=`Finish`

Click on the Next button to run the doPost method, with the cmd parameter set to qwwhere. Now, the wizWhere method is run. It generates and executes the HTML file shown in Figure 232 on page 252.

```
<HEAD><TITLE> Query Wizard </TITLE></HEAD><BODY>
<center>
<img src=http://AS400ABC:1040/apptest/as400.gif alt="AS400ABC">
<center>
<h3>Select Conditions</h3>
<form action=http://AS400ABC:1040/servlet/DbSelectServlet method=post>
<input type=hidden name=cmd value=qworder>
<input type=hidden name=system value=AS400ABC>
<input type=hidden name=user value=auser>
<input type=hidden name=key value=AS400ABCauser908479492129>
<br>
<applet code=SqlWhere.class codebase=http://AS400ABC:1040/applets/ height=358 width=425>
<param name=system value="AS400ABC">
<param name=user value="auser">
<param name=key value="AS400ABCauser908479492129">
<param name=url value=http://AS400ABC:1040/servlet/DbSelectServlet>
<param name=fields value="PARTNO,PARTDS,PARTQY,PARTPR,PARTDT">
</applet>
<br><br>
<input type=submit name=wizact value=Next...>
<input type=submit name=wizact value=Finish>
</center>
</form>
</body>
```

Figure 232. HTML File Generated by the wizWhere Method

The HTML file shown in Figure 232 runs an applet named SqlWhere.

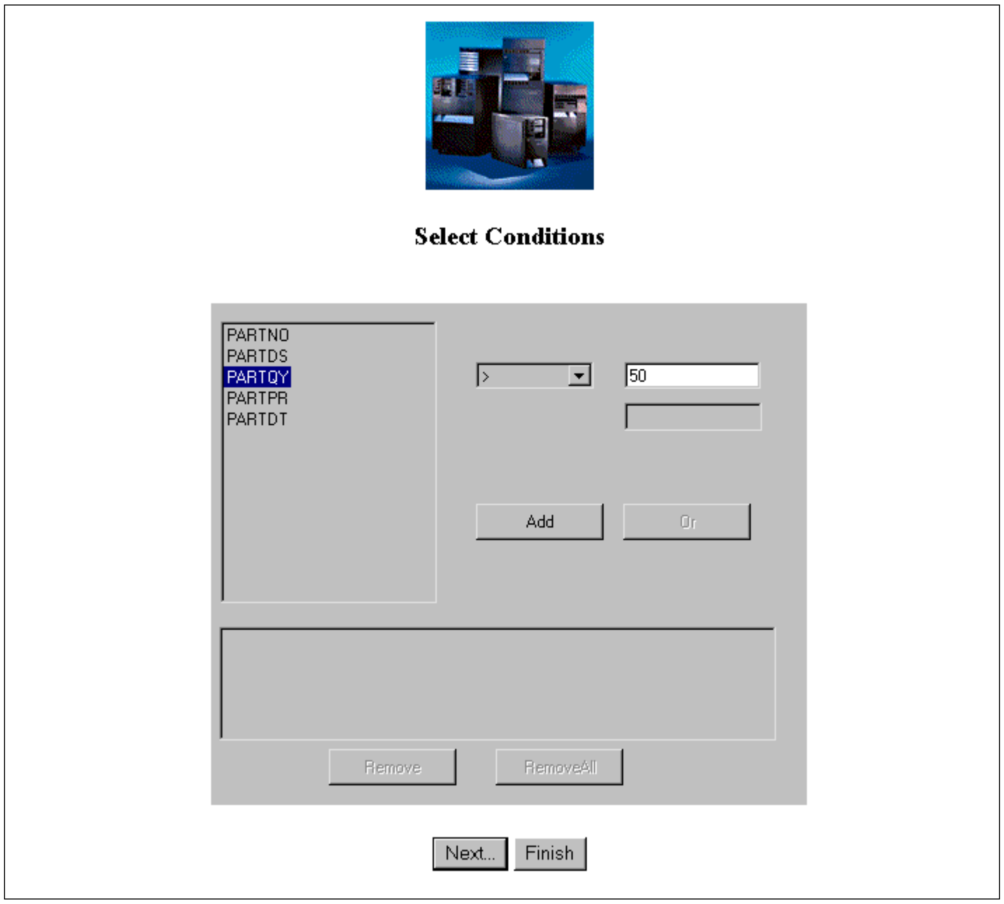


Figure 233. SQLWhere Applet

The SQLWhere applet allows you to add a where clause to the SQL statement. In this case, we want only those records that have a PARTQY field value of greater than 50.

The action of this HTML file is to call the DbSelectServlet class using a post method. It sets the following parameters:

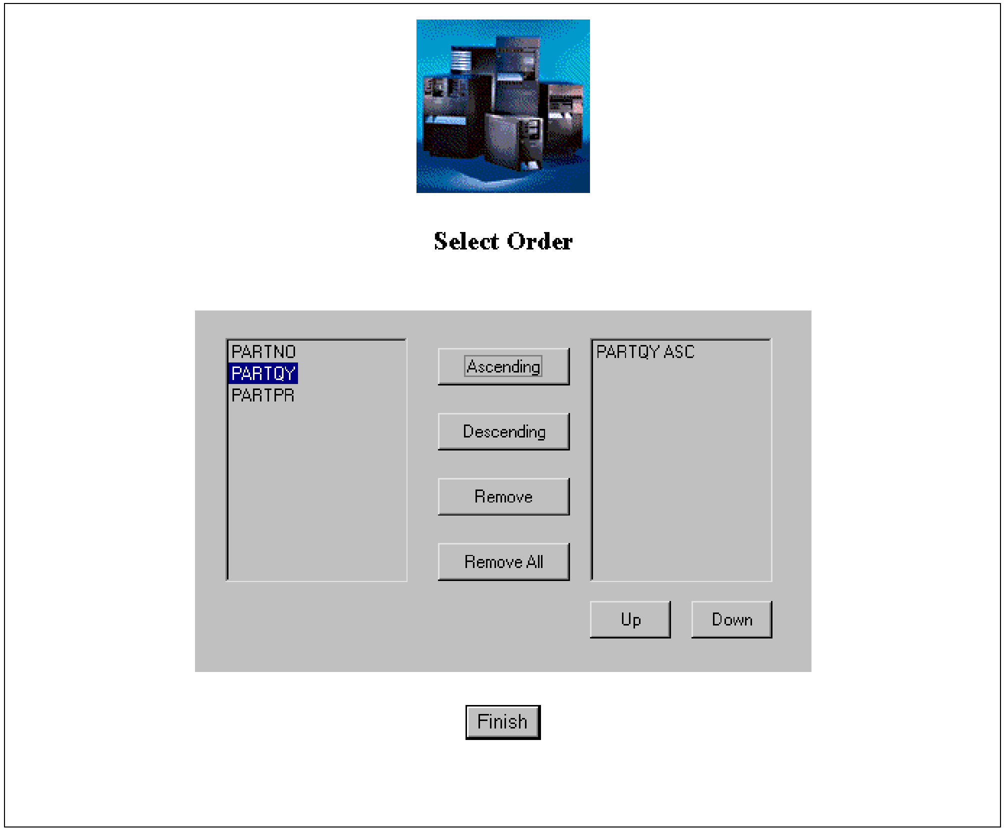
- name=cmd, value=`qworder`
- name=system, value=`as400abc`
- name=user, value=`auser`
- name=key, value=`as400abcauser908479492129`
- name=wizact, value=`Next...`
- name=wizact, value=`Finish`

Clicking on the Next button runs the doPost method, with the cmd parameter set to `qworder`. Now the wizOrder method runs. It generates and executes the HTML file shown in Figure 234 on page 254.

```
<HEAD><TITLE> Query Wizard </TITLE></HEAD><BODY>
<center>
<img src=http://AS400ABC:1040/apptest/as400.gif alt="AS400ABC">
<center>
<h3>Select Order</h3>
<form action=http://AS400ABC:1040/servlet/DbSelectServlet method=post>
<input type=hidden name=cmd value=qryprompt>
<input type=hidden name=system value=AS400ABC>
<input type=hidden name=user value=auser>
<input type=hidden name=key value=AS400ABCauser908479492129>
<br>
<applet code=SqlOrder.class codebase=http://AS400ABC:1040/applets/ height=250 width=426>
<param name=system value="AS400ABC">
<param name=user value="auser">
<param name=key value="AS400ABCauser908479492129">
<param name=url value=http://AS400ABC:1040/servlet/DbSelectServlet>
<param name=selected value="PARTNO,PARTQY,PARTPR">
</applet>
<br><br>
<input type=submit name=wizact value=Finish>
</center>
</form>
</body>
```

Figure 234. HTML File Generated by the wizOrder Method

This HTML file runs an applet named SqlOrder.



Select Order

Figure 235. SqlOrder Applet

The SqlOrder applet, allows us to add an *order by* clause to the sql statement. In this case, we want to display the results in ascending order, based on the PARTQY field.

The action of this HTML file is to call the DbSelectServlet class using a post method. It sets the following parameters:

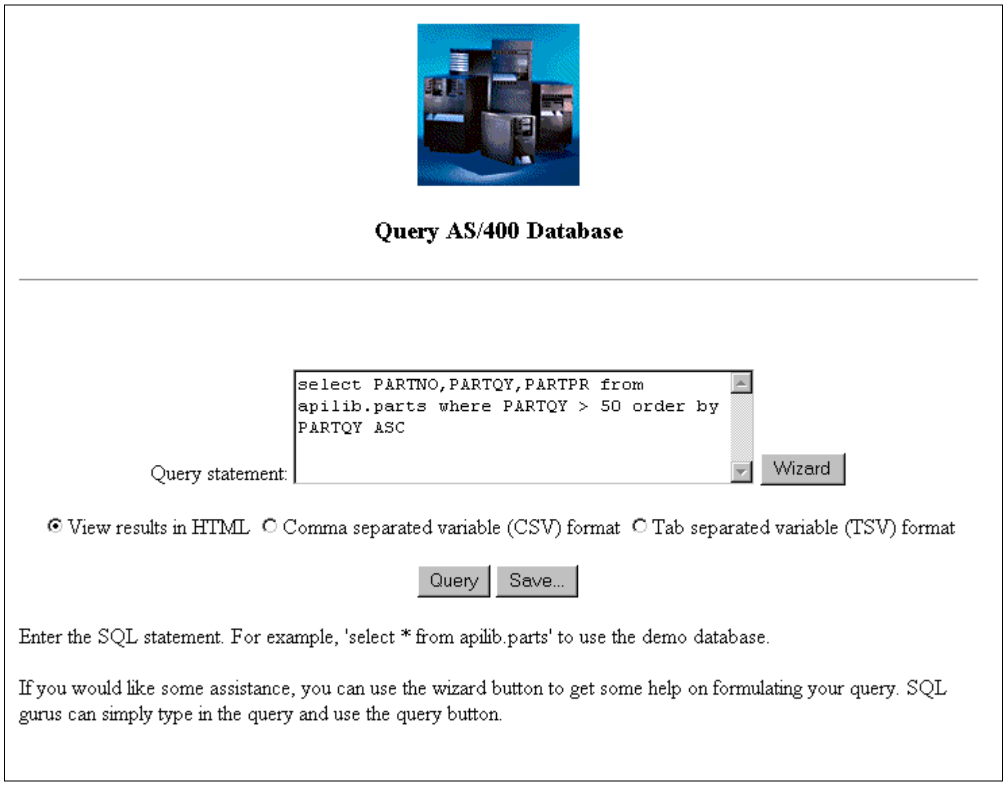
- name=cmd, value=`qryprompt`
- name=system, value=`as400abc`
- name=user, value=`auser`
- name=key, value=`as400abcauser908479492129`
- name=wizact, value=`Next...`
- name=wizact, value=`Finish`

Clicking on the Finish button starts the doPost method, with the cmd parameter set to `qryprompt`. If a cmd value of qryprompt is received in the doPost method, the qryPrompt method is called. The qryPrompt method builds and displays the HTML file shown in Figure 236.

```
<HEAD><TITLE> Query Statement </TITLE></HEAD><BODY>
<center>
<img src=http://AS400ABC:1040/apptest/as400.gif alt="Database Query">
<h3>Query AS/400 Database</h3>
</center>
<hr width="100%">
<center>
<form action=http://AS400ABC:1040/servlet/DbSelectServlet method=post>
<input type=hidden name=cmd value=query>
<input type=hidden name=system value=AS400ABC>
<input type=hidden name=user value=auser>
<input type=hidden name=key value=AS400ABCauser908479492129>
<br><br>
Query statement:
<textarea name=statement rows=5 cols=40>
select PARTNO,PARTQY,PARTPR from apilib.parts where PARTQY > 50 order by PARTQY ASC
</textarea>
<input type=submit name=wizard value=Wizard>
<br><br>
<input type=radio name=qtype value="html" checked>View results in HTML
<input type=radio name=qtype value="CSV">Comma separated variable (CSV) format
<input type=radio name=qtype value="TSV">Tab separated variable (TSV) format
<br><br>
<input type=submit value=Query>
<input type=submit name=save value=Save...>
</center>
</form>
<p>Enter the SQL statement.
For example, 'select * from apilib.parts'
to use the demo database.
<p>If you would like some assistance, you can use the
wizard button to get some help on formulating your query.
SQL gurus can simply type in the query and use the query button.
</body>
```

Figure 236. HTML File for the queryPrompt Method

Displaying this file in a browser returns you to the Query AS/400 Database window shown in Figure 237 on page 256.


Query AS/400 Database

Query statement:

```
select PARTNO,PARTQY,PARTPR from
apilib.parts where PARTQY > 50 order by
PARTQY ASC
```

[✓ View results in HTML] Comma separated variable (CSV) format Tab separated variable (TSV) format

Enter the SQL statement. For example, 'select * from apilib.parts' to use the demo database.

If you would like some assistance, you can use the wizard button to get some help on formulating your query. SQL gurus can simply type in the query and use the query button.

Figure 237. Query AS/400 Database

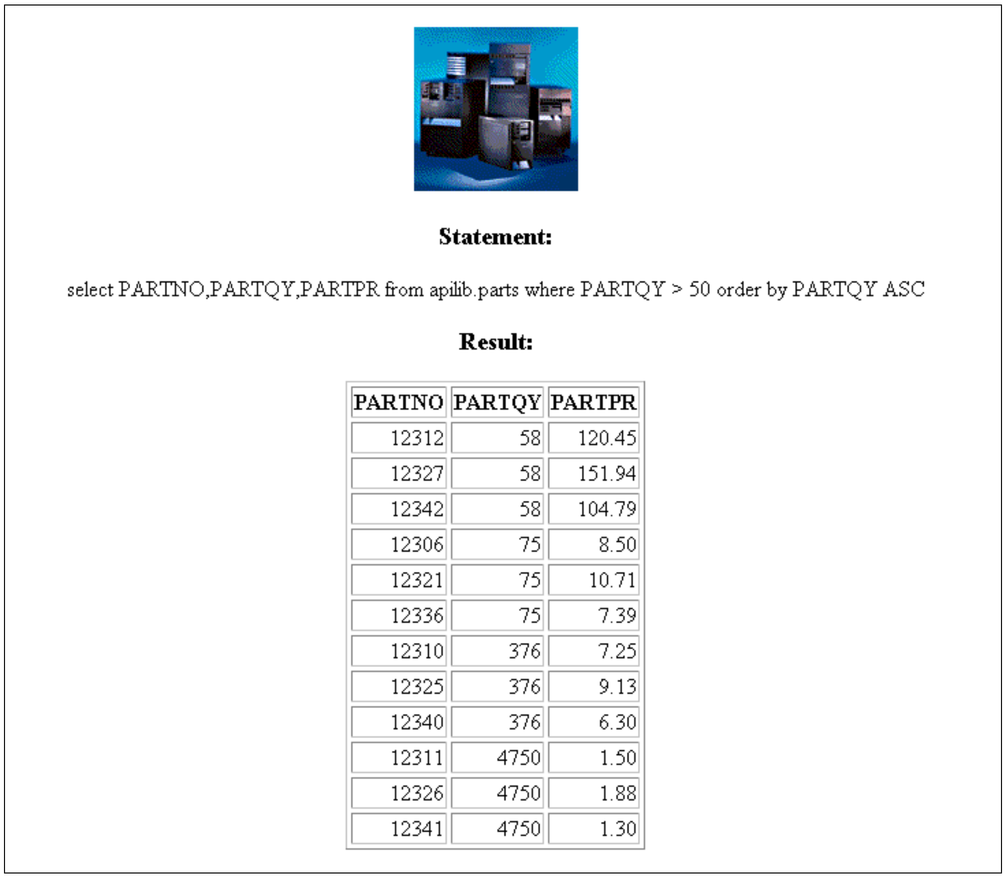
The text area of the Query AS/400 Database window displays the SQL statement that we created.

Clicking on the Query button runs the doPost method. If a cmd value of query is received in the doPost method, the performQuery method is called. The performQuery method verifies how we want the results returned by checking which radio button is selected. In this case, the **View results in HTML** radio button is selected, so the fillTable method is called.

The fillTable method actually runs the SQL statement on the AS/400 system. It does the following:

- Creates a connection object using the JDBC DriverManager getConnection method
- Creates a statement object using the createStatement method
- Executes the SQL statement we entered using the statement object
- Formats the rows into a table using HTML tags if a resultset is returned
- Causes the browser to execute the HTML file to display the results

Clicking on the Query button runs the statement. The results are shown in Figure 238 on page 257.



Statement:

select PARTNO,PARTQY,PARTPR from apilib.parts where PARTQY > 50 order by PARTQY ASC

Result:

PARTNO	PARTQY	PARTPR
12312	58	120.45
12327	58	151.94
12342	58	104.79
12306	75	8.50
12321	75	10.71
12336	75	7.39
12310	376	7.25
12325	376	9.13
12340	376	6.30
12311	4750	1.50
12326	4750	1.88
12341	4750	1.30

Figure 238. SQL Result

7.4 Running the Application

The application discussed in this chapter uses servlets to access AS/400 resources. To run it, we need a server that supports Java servlets. We tested the application using two servers:

- Domino Go Webserver and ServletExpress running on a Windows/NT platform
- The IBM HTTP Server for AS/400 and the WebSphere Application Server for AS/400 running on an AS/400 system

7.4.1 Domino Go Webserver

For details about configuring Domino Go Webserver and ServletExpress, see Section 9.1, “Domino Go Webserver” on page 287. For the application to be served by the Domino Go Webserver, you must:

1. Export the class files to a directory where the Web server can find them.
2. Add the appropriate directives to the Domino Go server request routing table.

By default the Domino Go Webserver serves servlets from following directory:

```
\ServletExpress\servlets
```

We export the servlet classes to this directory. Figure 239 on page 258 shows the servlet classes exported to the \ServletExpress\servlets directory.

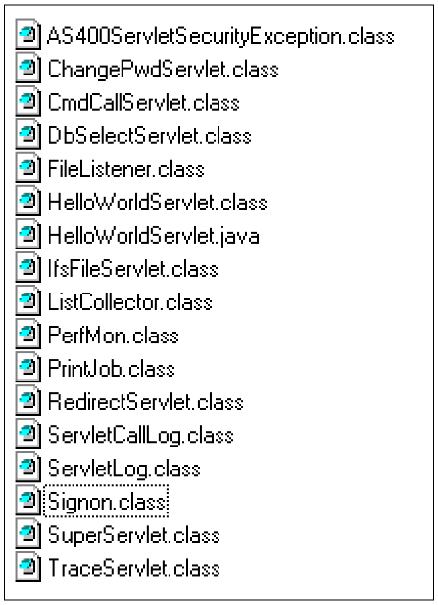

AS400ServletSecurityException.class
ChangePwdServlet.class
CmdCallServlet.class
DbSelectServlet.class
FileListener.class
HelloWorldServlet.class
HelloWorldServlet.java
IfsFileServlet.class
ListCollector.class
PerfMon.class
PrintJob.class
RedirectServlet.class
ServletCallLog.class
ServletLog.class
Signon.class
SuperServlet.class
TraceServlet.class

Figure 239. ServletExpress\servlets Directory

We also use three applets as part of this application. To keep access to the applets secure and separate from the servlets, we export them to the following directory:

```
\ServletExpress\servlets\applets
```

Figure 240 shows the content of the \Servlet\Express\applets directory.

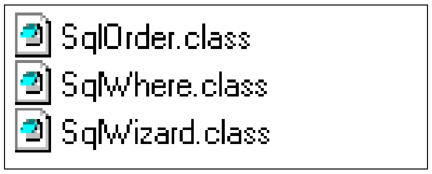

SqlOrder.class
SqlWhere.class
SqlWizard.class

Figure 240. Applet Directory

The application uses gif files to display images. The gif files are stored in the apptest directory. Figure 241 shows the content of the apptest directory.

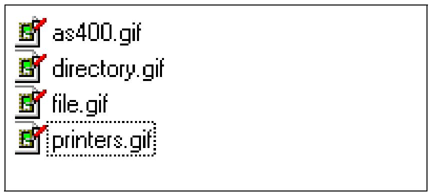

as400.gif
directory.gif
file.gif
printers.gif

Figure 241. apptest Directory

We add the following directives to the Domino Go server request routing table:

- Pass /apptest/* \apptest*

  This directive allows us serve the gif files from the apptest directory.

- Pass /applets/* c:\ServletExpress\servlets\applets\*

  This directive allows us to serve the applets from the applets directory.

7.4.2 IBM HTTP Server for AS/400

We can also serve this application using the IBM HTTP Server for AS/400 and the WebSphere Application Server for AS/400. For configuration details about running under the IBM HTTP Server for AS/400, see Section 9.3, "IBM HTTP Server for AS/400" on page 293.

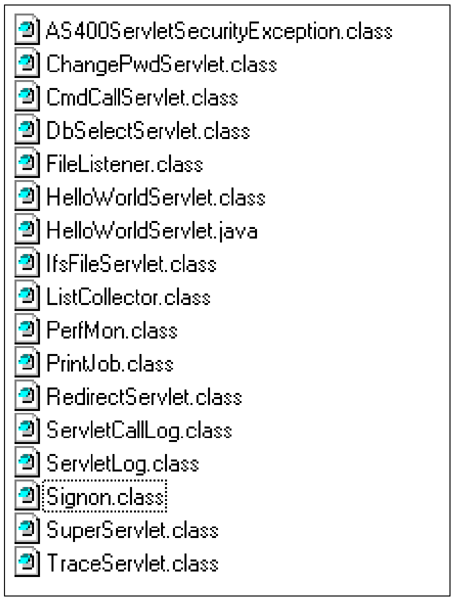
To allow the application to be served by the IBM HTTP Server for AS/400, you must:

1. Export the class files to a directory where the Web server can find them.
2. Add the appropriate directives to the Domino Go server request routing table.

By default IBM HTTP Server for AS/400 serves servlets from the following directory:

```
\QIBM\ProdData\IBMWebAS\servlets
```

We export the servlet classes to this directory. Figure 242 shows the servlet classes exported to the `\QIBM\ProdData\IBMWebAS\servlets`.



AS400ServletSecurityException.class
ChangePwdServlet.class
CmdCallServlet.class
DbSelectServlet.class
FileListener.class
HelloWorldServlet.class
HelloWorldServlet.java
IfsFileServlet.class
ListCollector.class
PerfMon.class
PrintJob.class
RedirectServlet.class
ServletCallLog.class
ServletLog.class
Signon.class
SuperServlet.class
TraceServlet.class

Figure 242. \QIBM\ProdData\IBMWebAS\servlets Directory

We also use three applets as part of this application. To keep access to the applets secure and separate form the servlets, we export them to the following directory:

```
\QIBM\ProdData\IBMWebAS\servlets\applets
```

Figure 243 on page 260 shows the content of the \Servlet\Express\applets directory.

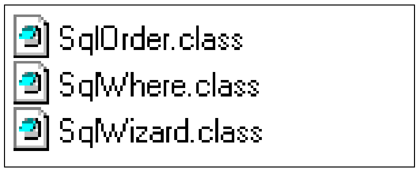

SqlOrder.class
SqlWhere.class
SqlWizard.class

Figure 243. Applet Directory

The application uses gif files to display images. The gif files are stored in the apptest directory. Figure 244 shows the content of the apptest directory.

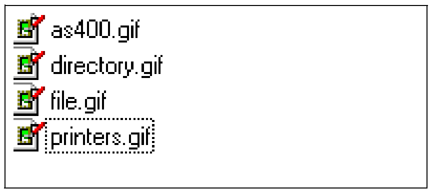

as400.gif
directory.gif
file.gif
printers.gif

Figure 244. The apptest Directory

We add the following directives to the IBM HTTP Server for AS/400 request routing table:

- Pass /apptest/* /apptest/*

  This directive allows us serve the gif files from the apptest directory.

- Pass /applets/* /`QIBM/ProdData/IBMWebAS/servlets/applets`

  This directive allows us to serve the applets from the applets directory.

Chapter 8. Security Considerations

This chapter explains how you can provide security for your Internet application. This chapter describes:

- An overview of the elements of transaction security available on the Internet
- A high level explanation of the Secure Sockets Layer (SSL) protocol
- How to use Digital Certificate Manager on AS/400 to create an intranet Certificate Authority (CA) and server certificates
- How to configure the IBM HTTP Server for AS/400 to use SSL
- Running a servlet under SSL

8.1 Internet Security Elements

There is no one single answer to Internet security. Some people think that by installing a firewall between their networks and the Internet the company's network will be safe.

Is it simply a firewall that shields your company from any inappropriate Internet access? No, security is not a single device or procedure. Security is a concept. It is a set of different security measures that are selected based on the needs of a specific installation. Therefore, it is essential to discuss first the type of Internet security you need to achieve. Security is not simply a firewall.

First, the policy established by high-level management indicates how your company wants to deal with the Internet and what level of security is to be achieved. Various Internet security features, such as cryptography or host system security functions, help you to implement what is designed.

Users must be educated to follow and maintain the implemented security procedures, as well as to observe specific rules when acting as Internet clients. These concepts are highlighted in Figure 245 on page 262.

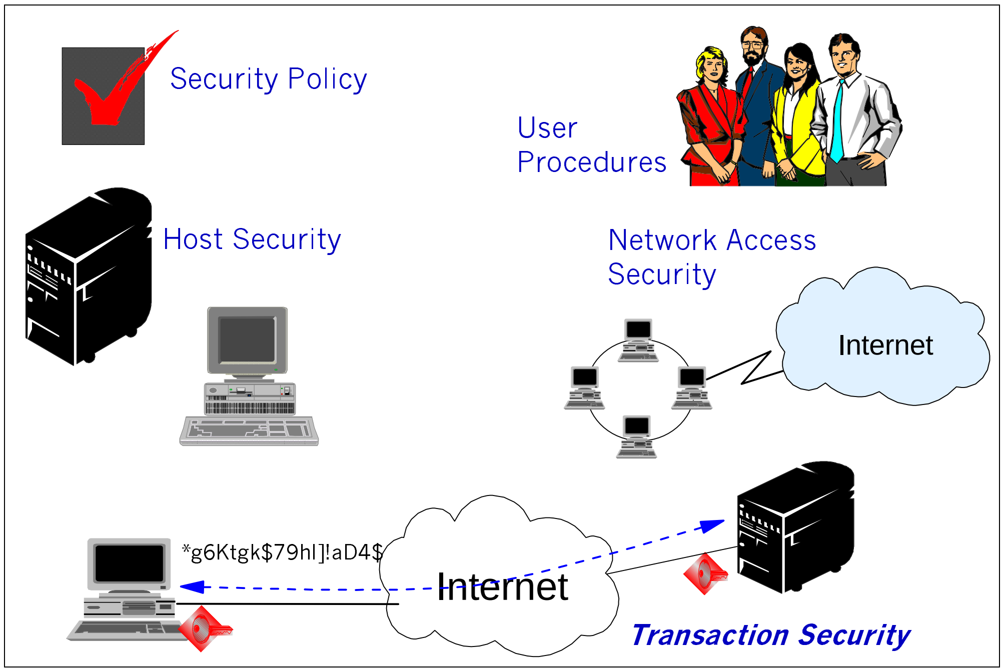


Figure 245. Internet Security Elements

8.1.1 Transaction Security and Secure Sockets Layer

Transaction security includes several basic elements, such as:

- Confidentiality and privacy
- Integrity
- Authentication
- Accountability

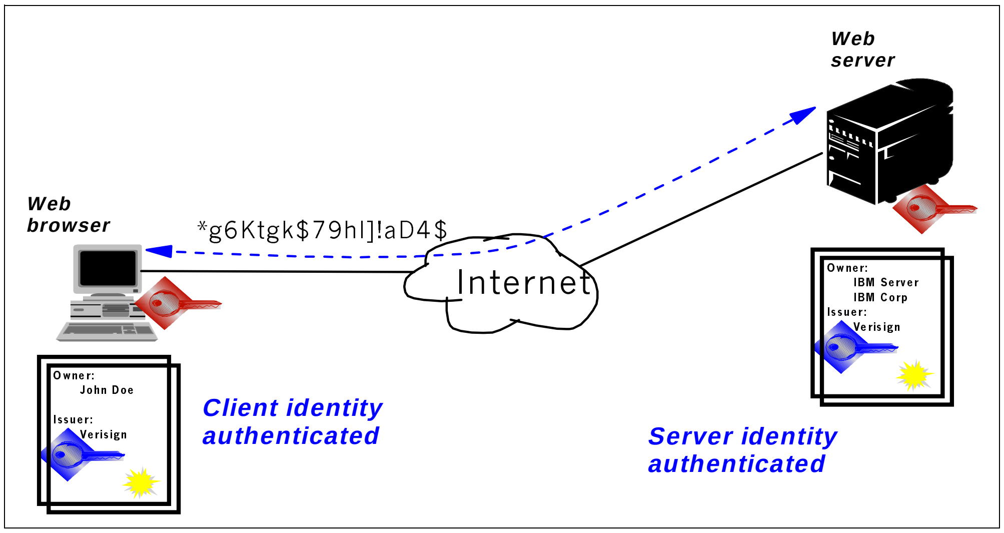


Figure 246. Transaction Security

SSL is the *Secure Sockets Layer* protocol defined by Netscape Communications Corporation. It provides a private channel between client and server that ensures privacy of data, authentication of session partners, and message integrity.

Digital certificates are used for session partner authentication. Server authentication is common. Client authentication is not yet common, but it is growing in popularity. Keys are the base for end-to-end information encryption. Figure 246 provides a high level view of SSL and transaction security.

TCP/IP applications must be rewritten to use SSL. Primarily SSL is used by HTTP (HTTPS) for Web browsing. In OS/400, the V4R3 Directory Services Server (LDAP) is SSL enabled. Other TCP/IP applications will follow.

8.1.1.1 Confidentiality

Consider this problem: Intruders can eavesdrop on private information as messages travel across the network. The solution lies in encryption. The sender scrambles the message, and the receiver unscrambles it using a secret key.

Confidentiality means that the contents of the messages remain private as they pass through the Internet. Without confidentiality, your computer broadcasts the message to the network, which is similar to shouting the information across a crowded room. *Encryption* ensures confidentiality.

8.1.1.2 Integrity

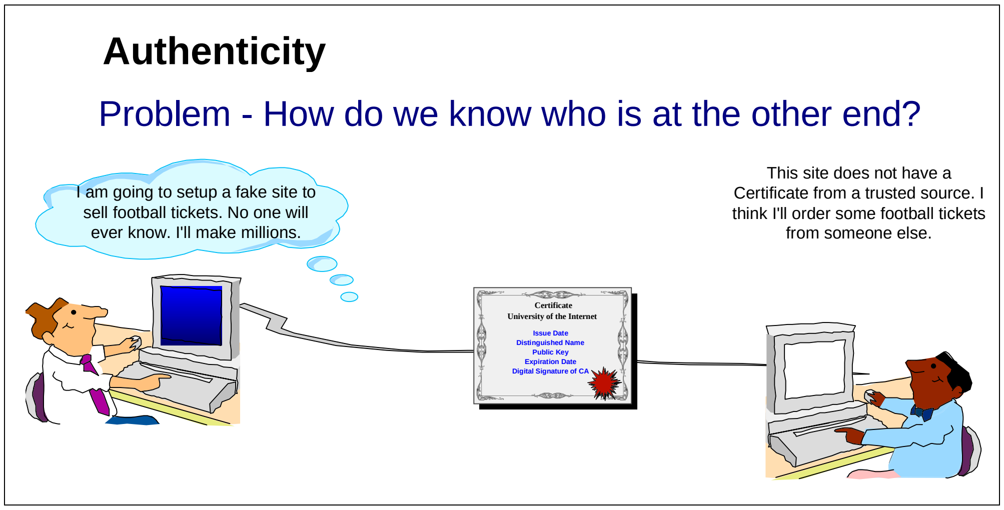
Consider, for example, that you want to know if the data received is the same as the data that was sent. You can determine this through two possible solutions: *digital signature (hashing)* and *encryption*.

The sending system calculates a value based on the data that is sent. The value is appended to the transmission. The receiving system uses the same calculation to generate a value. The receiving system compares the calculated value with the received value. If the values are different, it assumes that the data changed. Message hashing should be used with encryption for better protection.

Integrity means that the messages are not altered while being transmitted. Any router along the way can insert or delete text or garble the message as it passes by. Without integrity, you have no guarantee that the message you sent matches the message received. Encryption and ***digital signature*** ensure integrity.

8.1.1.3 Authenticity

Consider the scenario where you want to know who is at the other end of a Web site to test its authenticity. One way to find out is through the use of digital certificates and digital signatures (see Figure 247 on page 264).


Authenticity

Problem - How do we know who is at the other end?

I am going to setup a fake site to sell football tickets. No one will ever know. I'll make millions.
Certificate
University of the Internet
Issue Date
Distinguished Name
Public Key
Expiration Date
Digital Signature of CA
This site does not have a Certificate from a trusted source. I think I'll order some football tickets from someone else.

Figure 247. Verifying Identity—Digital Certificates and Digital Signatures

Authenticity means that you know who you are talking to and that you trust that person. Without authenticity, you have no way to be sure that anyone is who they say they are. ***Authentication*** through digital certificates and digital signatures ensure authenticity.

There are two ways in which the server uses authentication:

- Digital signature
- Digital certificates

A digital signature ensures accountability. But how do you know if the person sending you a message is who he says he is?

You look at the sender's *digital certificate*. A public key certificate is issued by a trusted third party known as the *certifying authority* (CA). The browser and server exchange information including their public key certificate. SSL uses the information to identify and authenticate the sender of the certificate.

A digital certificate is like a credit card with your picture on it and a picture of the bank president with his arm around you. A merchant trusts you more because you look like the picture on the credit card, and they know the bank president trusts you, too.

You base your trust for the authenticity of the sender on whether you trust the third party (a person or agency) that certified the sender. The third party or **certification authority (CA)** issues digital certificates.

How can you ensure that the person sending the message is really trustworthy? Consider the following an example, which illustrates this point.

If you wake up one day feeling ill, you may decide to visit a doctor. You can select a doctor from your phone book and go to his or her office for a visit. Once you arrive at the office, how can you be sure that the person about to examine you is really a doctor? After all, you have never met this person before. They may look like a doctor and act like a doctor, but how do you know that this person has successfully completed all the training necessary to become a doctor?

You need certification by a trusted third party to reassure you that this person really is a doctor. The doctor probably has a diploma on the wall stating that they have successfully completed their training. If the diploma is from a well-known school, you would probably be reassured that you are about to be examined by a real doctor. What if the diploma is from the medical school of a correspondence school whose name you don't recognize? You may not be reassured.

Authentication works the same way. Trusted third parties verify that the server really is who it claims to be. This verification is provided with a digital certificate (the digital equivalent of your doctor's diploma hanging on the wall). You base your trust for the authenticity of the server on whether you trust the third party that certified the server (the school that issued the diploma). That third party is called a Certifying Authority (CA).

The term *trusted root* is given to a trusted certifying authority (CA) on your server. A *trusted root key* is the key belonging to the CA.

Authentication can be used server to client (server authentication) or client to server (client authentication). Server authentication is described earlier. The clients authenticate the servers. With client authentication, the client is authenticated by the server. For example, if a server contains hospital patient information, we may use client authentication to verify that the client attempting to access the data is really who he said he is before allowing him access to patient records.

8.1.1.4 Accountability

Consider the situation where you want to prove that a transaction took place. We combine all the techniques we have seen. First the data is hashed using cryptography to assure its integrity. The data is encrypted using the keys derived from the public key exchange, which assures the identity of the session partners. This is used in combination with a time stamp in the data to provide a log of the transactions.

Accountability means that both sender and receiver agree that the exchange took place. Without accountability, the addressee can easily say that the message never arrived. Digital signatures ensure accountability. Accountability is *not* part of the SSL protocol.

8.1.2 HTTP Server Over SSL (HTTPS)

SSL ensures that data transferred between a client and a server remains private. It allows the client to authenticate the identity of the server. In addition, SSL V3 allows a server to authenticate a client.

Figure 248 on page 266 shows the high level view of the flow that takes place when a client (browser) sends an **https** request to an HTTP server.

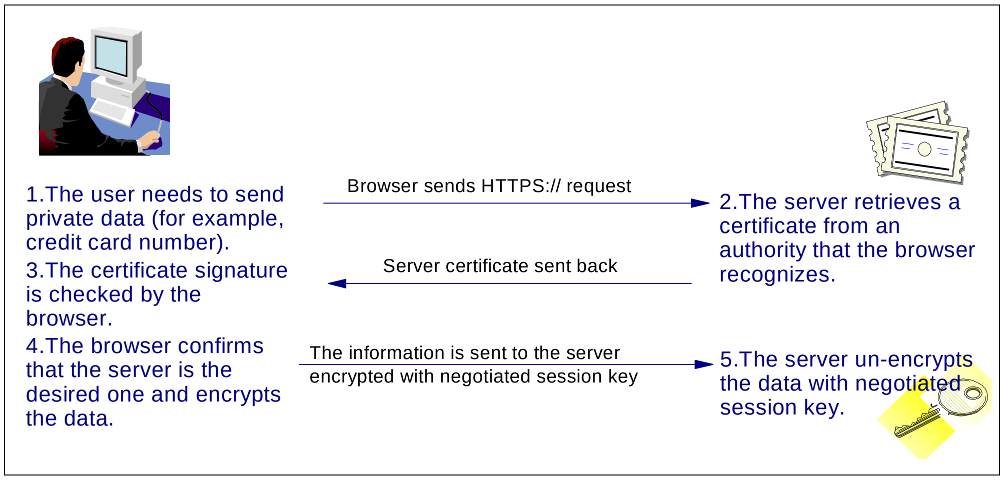

1.The user needs to send private data (for example, credit card number).
Browser sends HTTPS:// request
2.The server retrieves a certificate from an authority that the browser recognizes.
Server certificate sent back
3.The certificate signature is checked by the browser.
4.The browser confirms that the server is the desired one and encrypts the data.
The information is sent to the server encrypted with negotiated session key
5.The server un-encrypts the data with negotiated session key.

Figure 248. HTTP Server Using SSL

If SSL client authentication is configured, the server requests the client's certificate for any **https** request. The server establishes a secure session depending on whether the client has a valid certificate. This depends on the server configuration: no client authentication, optional client authentication, and mandatory client authentication.

Once your server has a digital certificate, SSL enabled browsers can communicate securely with your server using SSL. With SSL, you can easily establish a security-enabled Web site on the Internet or on your corporate network.

SSL uses a security handshake to initiate the secure TCP/IP connection between the client and the server. During the handshake, the client and server agree on the security keys that they will use for the session and the algorithms they will use for encryption and to compute message digest or hashes. The client authenticates the server. In addition, if the client requests a document protected by SSL client authentication, the server requests the client's certificate. After the handshake, SSL is used to encrypt and decrypt all information on both the **https** requests and the server response, including:

- The URL the client is requesting
- The contents of any form being submitted
- Access authorization information like user names and passwords
- All data sent between the client and the server

The benefits of HTTP using SSL include:

- Target server is verified for authenticity
- Information is encrypted for privacy
- Data is checked for transmission integrity

HTTPS is a unique protocol that combines SSL and HTTP. You need to specify https:// as an anchor in HTML documents that link to SSL, protected documents. A client user can open a URL by specifying https:// to request an SSL, protected documents.

Because HTTPS (HTTP + SSL) and HTTP are different protocols and usually use different ports (443 and 80, respectively), you can run both secure and non-secure servers at the same time. As a result, you can choose to provide information to all users using no security, and specific information only to browsers who make secure requests. This is how a retail company on the Internet can allow users to look through merchandise without security, complete order forms, and send their credit card numbers using SSL security. A browser that does not have support for HTTP over SSL naturally cannot request URLs using HTTPS. The non-SSL browsers do not allow users to send forms that need to be submitted securely.

Figure 249 shows how clients can access the same server instance in normal mode (port 80) or encrypted using SSL (port 443).

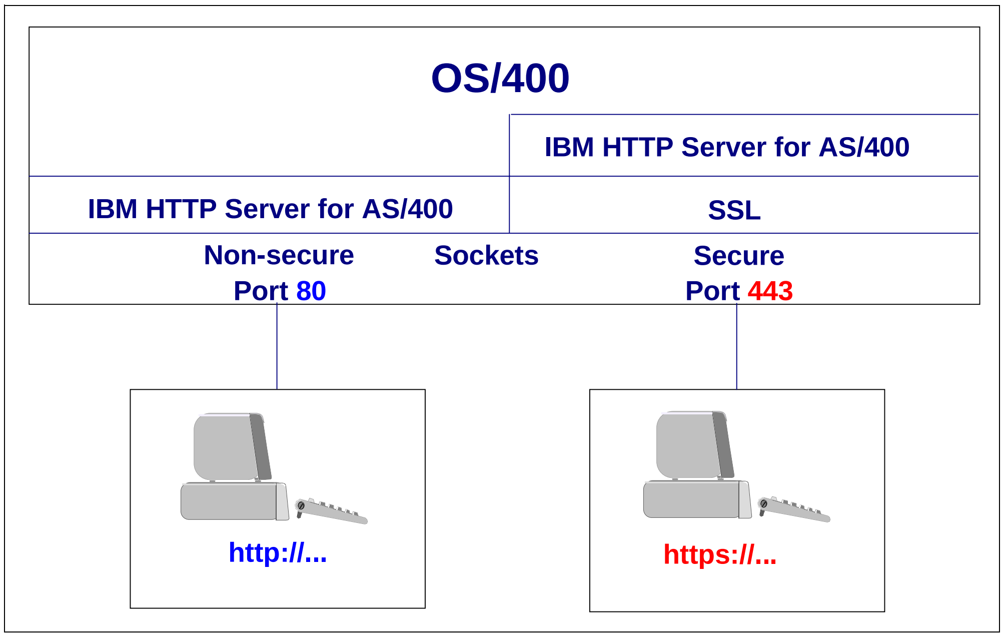


Figure 249. Accessing a Secure HTTP Session

8.2 Digital Certificates and Certificate Authority

A digital certificate identifies a user or a system and is required before SSL can be used. Once a server has a digital certificate, SSL-enabled browsers, such as the Netscape Navigator, can communicate securely with the server using SSL. A digital certificate consists of:

- Owner’s distinguished name
- Owner’s public key
- Digital signature of certificate authority (CA)
- Name of the CA
- Issue date of certificate
- Certificate expiration date
- Serial number

Plus, digital certificates have the following characteristics:

- Digital certificates are digital documents that validate the identity of a certificate's owner.
- There are three types of digital certificates: CA, server, and client certificates.

- Digital certificates contain public key—binds it to an identity.
- Digital certificates are created by trusted third parties called Certificate Authorities (CA).
- Digital certificates can be distributed freely.
- Digital signature in the digital certificate prevents tampering.

A digital certificate is issued by a certificate authority (CA). CAs are entities that are trusted to properly issue certificates and have controls in place to prevent fraudulent use. They are the equivalent to the Department of Motor Vehicles for a driver's license. An individual may have many certificates from different CAs just as we have many forms of personal identification (Social Security card, Blue Cross/Blue Shield card, gym membership card.) If we can trust a CA, we can be reasonably assured that any certificate they issue properly represents the individual that is holding it.

The Certificate Authority charges a fee for issuing a certificate.

- Certificate Authorities broadcast their public key and Distinguished Name.
- People add them as trusted root key to web servers and browsers.
- This means your server will trust anyone who has a certificate from that CA.
- There are several common CA's in the marketplace.
- Servers and browsers are shipped with several default trusted root keys and more can be added as needed.

Some examples of universally recognized Internet Certificate Authorities (CA) include:

- Thawte
- VeriSign
- US Postal Service
- AT&T
- MCI

For testing purposes or for applications that will be used exclusively in an intranet environment, you may issue digital certificates using an intranet Certificate Authority. The AS/400 system with Digital Certificate Manager (DCM) can act as an intranet Certificate Authority.

For secure communications, the receiver must trust the CA that issued the certificate, whether the receiver is a browser or a server. Any time a sender signs a message, the receiver must have the corresponding CA certificate and public key designated as trusted root key.

8.3 AS/400 Implementation of Digital Certificate Management

You can configure your AS/400 system as an intranet Certificate Authority. Digital Certificate Manager (DCM) is a Web-browser based administration facility that allows you to create, manage, and use certificates within an enterprise and with partners of an enterprise. You can use DCM to request digital certificates from Internet Certificate Authorities such as VeriSign and Thawte.

DCM allows you to create your own intranet Certificate Authority (CA). You can then use the CA to dynamically issue digital certificates to servers and users (client certificates) on your intranet. When you create a server certificate, DCM automatically generates the private key and public key for the certificate.

You can also use DCM to register and use digital certificates from Verisign or other commercial organizations on your intranet or the Internet.

Digital Certificate Manager is option 34 of OS/400 (5769-SS1 option 34). You must install this option to use DCM. DCM is a link in the AS/400 Tasks page, which runs in the *ADMIN HTTP server instance. Therefore, you must have installed IBM HTTP Server for AS/400 (5769-DG1) and use it to access DCM. In addition, you must install IBM Cryptographic Access Provider licensed program (5769-AC1,or AC2, or AC3) to create certificate keys. These cryptographic products determine the maximum key length permitted for cryptographic algorithms on your AS/400 system. Government export and import regulations determine which version is available in your country. To use all the options available in DCM, you must have *SECOFR and *SECADM authority.

To access the Digital Certificate Manager, click on the hyperlink for **Digital Certificate Manager** from the AS/400 Tasks Page. When using Digital Certificate Manager, you can click the **Help** button on any page at any time to access on-line help.

8.3.1 Configuring a Digital Certificate Environment

You can use your AS/400 system to configure a digital certificate environment. You can also configure the HTTP server to use digital certificates and run over SSL.

Perform the following series of steps to configure an intranet digital certificate environment using the AS/400 system as a Certificate Authority:

1. Use DCM to create an intranet CA in one or more AS/400 system.
2. Using DCM, the intranet CA issues server certificates that can be used in the local server (same AS/400 system where the CA is configured) or exported to a remote server.
3. For the clients to recognize and trust the server certificates issued by the intranet CA, the CA certificate must be installed in the browsers and designated as a trusted root.
4. If the server requests client certificates for client authentication, the users must request and install client certificates in their browsers.
5. The HTTP server must be configured to enable SSL (**SSL On**) and specify the key ring file where the server certificate is stored (**keyfile**). To optionally authenticate client certificates (**SSL_ClientAuth client**), add PROTECTION/PROTECT directives to protect resources.

8.4 Creating a Self-Signed Certificate

This section describes how to create a self-signed certificate using your AS/400 system as an intranet Certificate Authority.

Because self-signed certificates are not recognized by visitor’s browsers as coming from a trusted third party, they should not be used in customer transaction situations over the Internet. Use them only on your test and development systems, and for demonstration purposes. You can also use a self-signed certificate for intranet applications.

To obtain a self-signed certificate, perform the following tasks:

1. Create an intranet Certificate Authority.
2. Create a server certificate with your intranet CA.
3. Configure your HTTP server to use the server certificate.

8.4.1 Creating an Intranet Certificate Authority

Digital Certificate Manager (DCM) allows you to create your own intranet CA in your AS/400 system and use it to issue server and client certificates for testing purposes or applications within your organization.

This section outlines the steps you must perform to create a CA on your AS/400 system. You only need to perform this task if the system administrator has not previously created an intranet Certificate Authority and if you want to use your AS/400 system to issue intranet server certificates.

To create an intranet CA in your AS/400 system, follow these steps:

1. Start the HTTP *ADMIN server on your AS/400 system. From the command line, enter the command:

   ```
   STRTCPSVR SERVER(*HTTP) HTTPSVR(*ADMIN)
   ```

2. Access the AS/400 Tasks page from your browser by entering the URL:

   ```
   http:// System_name:2001
   ```

3. You are prompted to enter user name and password. Sign on with a user that has *SECOFR and *SECADM authority.

The AS/400 Tasks Page appears as shown in Figure 250.

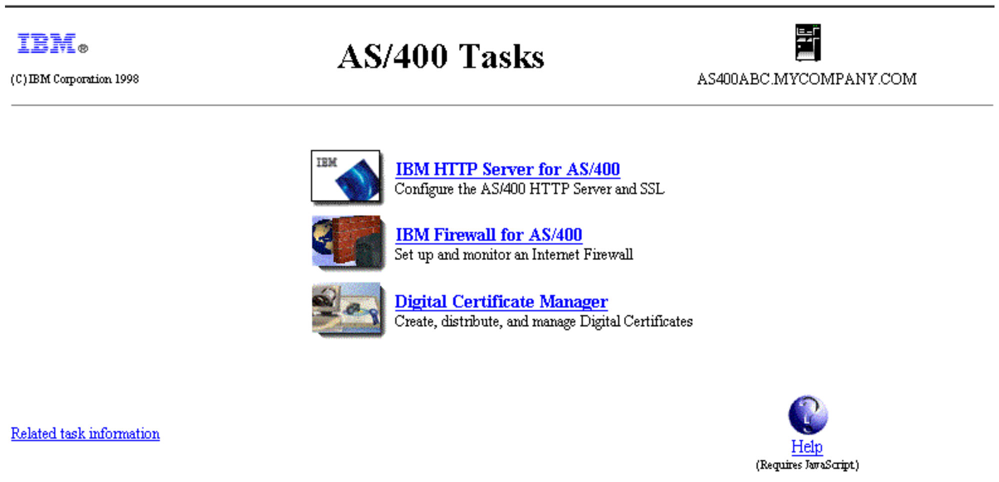

IBM
(C) IBM Corporation 1998
AS/400 Tasks
AS400ABC.MYCOMPANY.COM
IBM HTTP Server for AS/400
Configure the AS/400 HTTP Server and SSL
IBM Firewall for AS/400
Set up and monitor an Internet Firewall
Digital Certificate Manager
Create, distribute, and manage Digital Certificates
Related task information
Help
(Requires JavaScript)

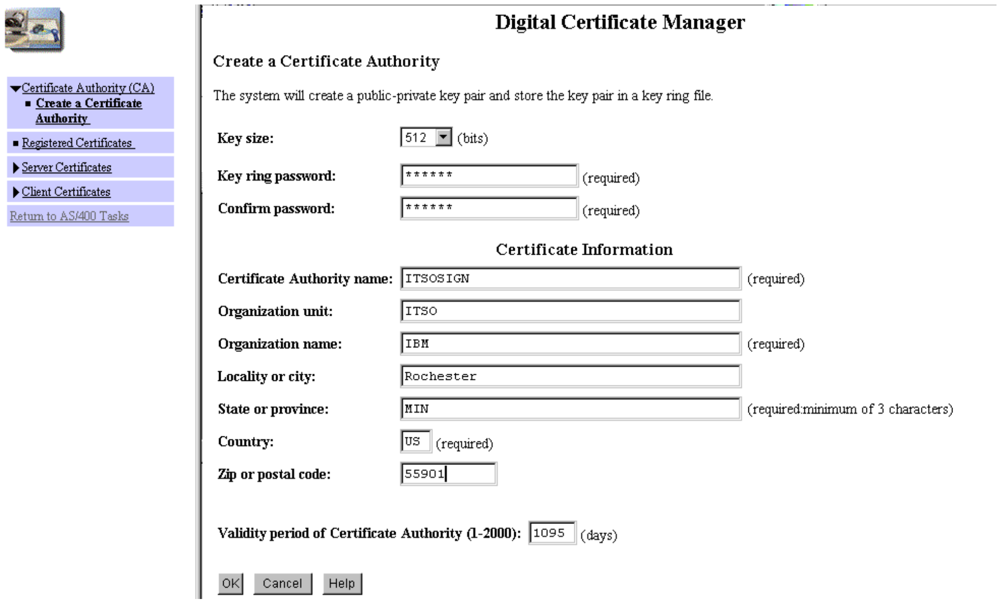
Figure 250. AS/400 Tasks Page

4. Click on **Digital Certificate Manager**.
5. Click on **Certificate Authority (CA)**.
6. Click on **Create a Certificate Authority**.

   Note: If a Certificate Authority (CA) was previously created on your system, the Create a Certificate Authority link does not appears.

7. Complete the Create a Certificate Authority form as shown in Figure 251.

   Replace the field values appropriately with your organization's information.



Digital Certificate Manager

Create a Certificate Authority

The system will create a public-private key pair and store the key pair in a key ring file.

Key size:	512	(bits)
Key ring password:	******	(required)
Confirm password:	******	(required)

Certificate Information

Certificate Authority name:	ITSOSIGN	(required)
Organization unit:	ITSO	
Organization name:	IBM	(required)
Locality or city:	Rochester	
State or province:	MIN	(required:minimum of 3 characters)
Country:	US	(required)
Zip or postal code:	55901	

Validity period of Certificate Authority (1-2000): 1095 (days)

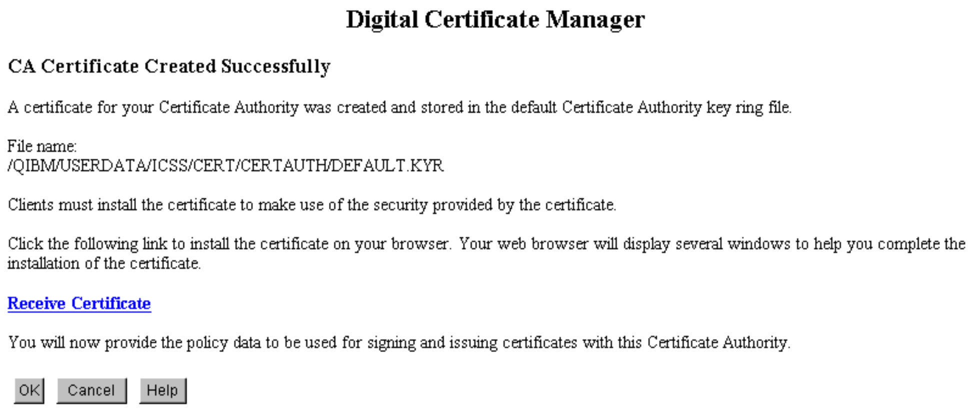
Figure 251. Create an Intranet Certificate Authority

   Click **OK**.

8. After DCM processes the form, it stores a copy of the CA certificate in the CA default key ring file:

   `/QIBM/USERDATA/ICSS/CERT/CERTAUTH/DEFAULT.KYR`

At this point, you can install the CA certificate in your browser so that it recognizes the certificates issued by the intranet CA. DCM displays the page shown in Figure 252.



Digital Certificate Manager

CA Certificate Created Successfully

A certificate for your Certificate Authority was created and stored in the default Certificate Authority key ring file.

File name:
/QIBM/USERDATA/ICSS/CERT/CERTAUTH/DEFAULT.KYR

Clients must install the certificate to make use of the security provided by the certificate.

Click the following link to install the certificate on your browser. Your web browser will display several windows to help you complete the installation of the certificate.

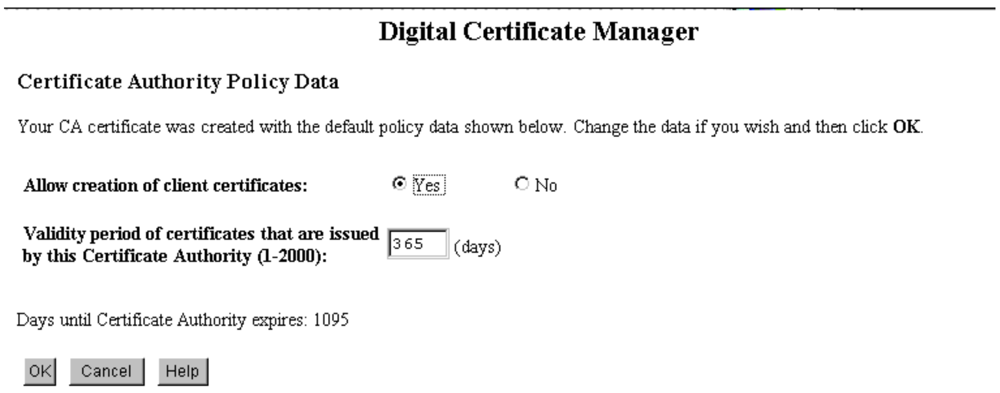
Receive Certificate

You will now provide the policy data to be used for signing and issuing certificates with this Certificate Authority.

Figure 252. CA Certificate Created Successfully

   Click **Receive Certificate** if you want to install the CA certificate in your browser now. Or, click **OK** to proceed to the next setup window, and install the CA certificate in your browser at later time. Notice the default path and file name where the intranet CA key ring file is stored.

9. Complete the CA Policy Data form to set the client certificate policy for your CA. See Figure 253.


Digital Certificate Manager

Certificate Authority Policy Data

Your CA certificate was created with the default policy data shown below. Change the data if you wish and then click **OK**.

Allow creation of client certificates: [✓ Yes] No

Validity period of certificates that are issued by this Certificate Authority (1-2000): 365 (days)

Days until Certificate Authority expires: 1095

Figure 253. Certificate Authority Policy

This is where you define whether your CA can issue and sign client certificates. If the CA can issue client certificates, indicate the length of time for which the certificates will be valid.

10. *The policy data for the Certificate Authority was successfully changed* message appears. At this point, you can continue to create a server certificate signed by your Certificate Authority. This allows server authentication by clients that use this system as a server.

8.4.2 Creating a Server Certificate with Your Intranet CA

Immediately after creating the intranet CA, DCM leads you to create a server certificate.

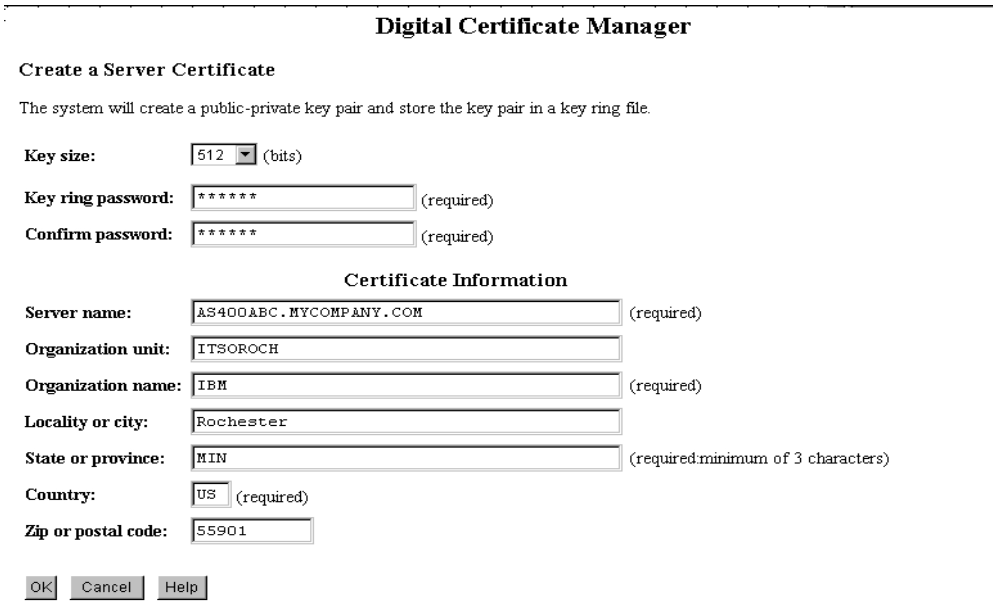
To use Secure Sockets Layer (SSL) for secure Web serving, your server must have a digital certificate. When you create a server certificate in DCM, the server certificate and keys are stored in the following default directory and file:

```
/QIBM/USERDATA/ICSS/CERT/SERVER/DEFAULT.KYR
```

Note: When you create a server certificate, Digital Certificate Manager (DCM) stores a copy of the CA certificate in the server's key ring and designates it as a trusted root.

1. Complete the Create a Server Certificate form as shown in Figure 254 replacing the field values with your organization information.

   The options for the key size are determined by the IBM Cryptographic Access Provider (5769-ACx) licensed program installed in your system. This is the key size that is used to generate your public and private keys.



Digital Certificate Manager

Create a Server Certificate

The system will create a public-private key pair and store the key pair in a key ring file.

Key size:	512	(bits)
Key ring password:	******	(required)
Confirm password:	******	(required)

Certificate Information

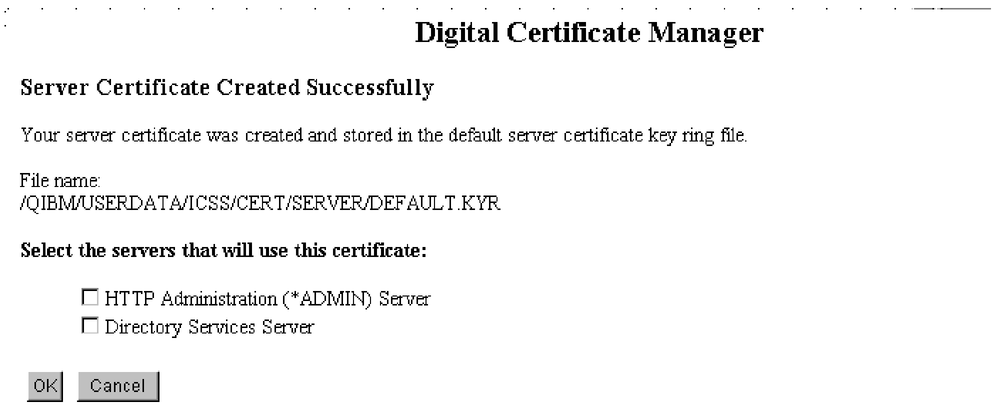
Server name:	AS400ABC.MYCOMPANY.COM	(required)
Organization unit:	ITSOROCH	
Organization name:	IBM	(required)
Locality or city:	Rochester	
State or province:	MIN	(required:minimum of 3 characters)
Country:	US	(required)
Zip or postal code:	55901	

Figure 254. Create a Server Certificate Page

By default, the system inserts the fully qualified name of the AS/400 system into the system name field. Do not change this name. This is the name used to describe your server. You can give the server any name. However, the fully qualified TCP/IP host name is usually used for the server name.

Click **OK**.

2. The Server Certificate Created Successfully page appears (see Figure 255).



Digital Certificate Manager

Server Certificate Created Successfully

Your server certificate was created and stored in the default server certificate key ring file.

File name:
/QIBM/USERDATA/ICSS/CERT/SERVER/DEFAULT.KYR

Select the servers that will use this certificate:

☐ HTTP Administration (*ADMIN) Server
☐ Directory Services Server

Figure 255. Server Certificate Created Successfully Page

From this page, you can select whether the HTTP ADMIN server or the Directory Services server (LDAP) uses this server certificate for SSL connections. Do *not* select any of these options.

3. Copy the file and path name where the server certificate is stored to the clipboard. It is:

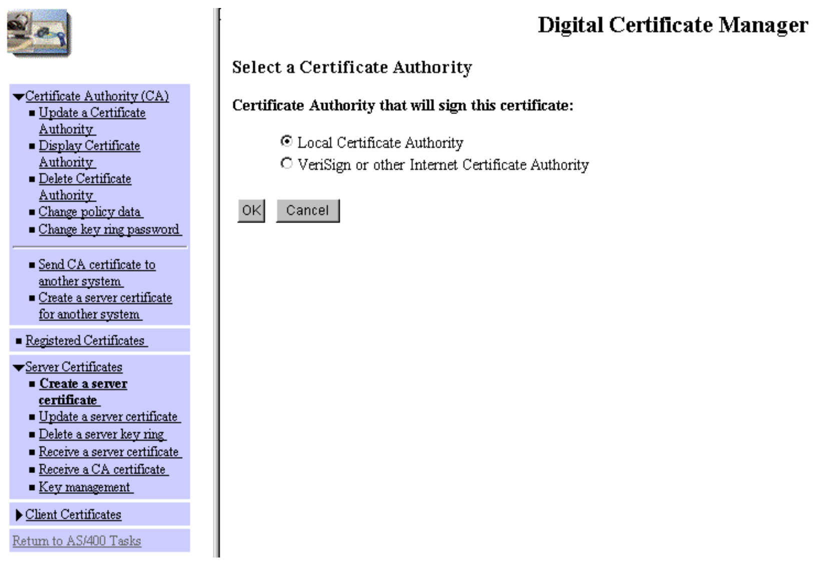
```
/QIBM/USERDATA/ICSS/CERT/SERVER/DEFAULT.KYR
```

Click **OK**. Click **Done**.

8.4.2.1 Creating a Server Certificate with an Existing Intranet CA

The steps to create a server certificate described in the previous section assume that you are creating the intranet CA for the first time. If your administrator has already created an intranet CA and server certificate, you can use the existing server certificate in your HTTP server configuration.

If you want to create a new server certificate using an existing intranet CA, start by clicking **Create a server certificate** under Server Certificates in DCM (see Figure 256).



Digital Certificate Manager

Select a Certificate Authority

Certificate Authority that will sign this certificate:

[✓ Local Certificate Authority] VeriSign or other Internet Certificate Authority

Certificate Authority (CA)
Update a Certificate Authority
Display Certificate Authority
Delete Certificate Authority
Change policy data
Change key ring password
Send CA certificate to another system
Create a server certificate for another system
Registered Certificates
Server Certificates
Create a server certificate
Update a server certificate
Delete a server key ring
Receive a server certificate
Receive a CA certificate
Key management
Client Certificates
Return to AS/400 Tasks
OK
Cancel

Figure 256. Create a Server Certificate with an Existing Intranet CA

Select **Local Certificate Authority** and Click **OK**.

The Create Server Certificate page appears next (see Figure 254 on page 273).

8.4.2.2 Authorizing QTMHHTTP to the Key Ring File

You may need to give QTMHHTTP (or the user profile under which your HTTP server runs) authority to the key ring and stash files. The key ring and stash files are created with *PUBLIC authority *EXCLUDE. QTMHHTTP (or the user profile under which the HTTP server runs) must have at least read rights to those files.

Perform the following steps:

1. To authorize QTMHHTTP to the key ring and stash file, enter the command:

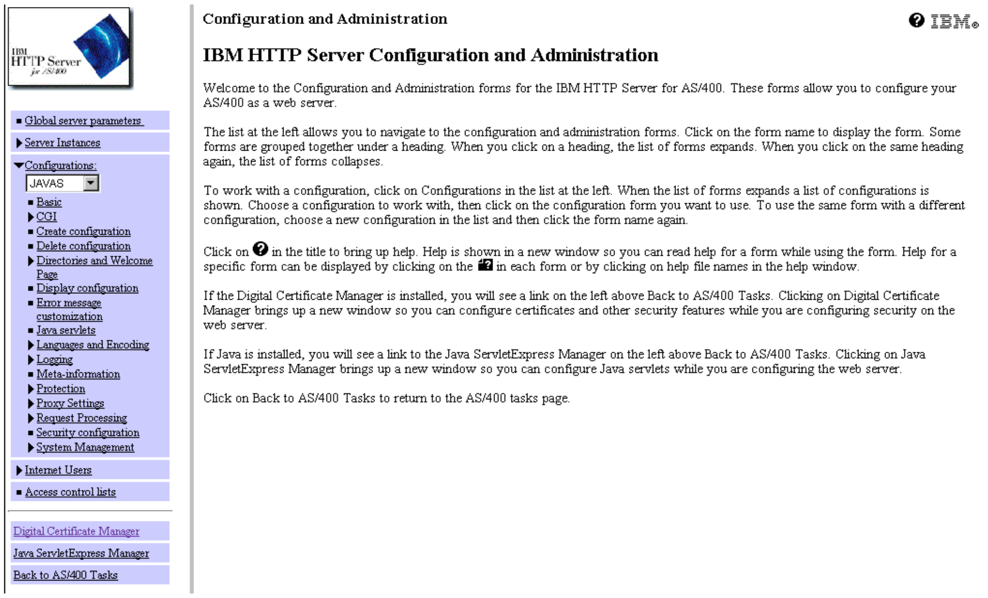
   ```
   WRKLNK '/QIBM/UserData/ICSS/Cert/Server'
   ```

2. Enter **5**, Next level, to display the files in the directory.
3. Enter **9**, Work with authority, by the key ring file (DEFAULT.KYR).
4. Enter **1**, Add user, User=QTMHHTTP, Data Authority=*R.
5. Repeat steps 1 through 3 to authorize QTMHHTTP to the stash file (DEFAULT.sth).

8.4.3 Configuring the Web Server to Use SSL with Server Authentication

The Web server must be configured to run over SSL and use the server certificate you created in Section 8.4.2, “Creating a Server Certificate with Your Intranet CA” on page 272. To configure your HTTP server to run over SSL and use a server certificate, you must perform the following tasks:

1. From Digital Certificate Manager, click on **Return to AS/400 Tasks**. The AS/400 Tasks page is displayed (see Figure 250 on page 270).
2. Click on **IBM HTTP Server for AS/400**.
3. Click on **Configuration and Administration**.
4. Click on **Configurations** in the left frame.
5. Select your **HTTP configuration file** in the drop-down box immediately beneath the Configurations link as shown in Figure 257.



Configuration and Administration

IBM HTTP Server Configuration and Administration

Welcome to the Configuration and Administration forms for the IBM HTTP Server for AS/400. These forms allow you to configure your AS/400 as a web server.

The list at the left allows you to navigate to the configuration and administration forms. Click on the form name to display the form. Some forms are grouped together under a heading. When you click on a heading, the list of forms expands. When you click on the same heading again, the list of forms collapses.

To work with a configuration, click on Configurations in the list at the left. When the list of forms expands a list of configurations is shown. Choose a configuration to work with, then click on the configuration form you want to use. To use the same form with a different configuration, choose a new configuration in the list and then click the form name again.

Click on ❓ in the title to bring up help. Help is shown in a new window so you can read help for a form while using the form. Help for a specific form can be displayed by clicking on the ❓ in each form or by clicking on help file names in the help window.

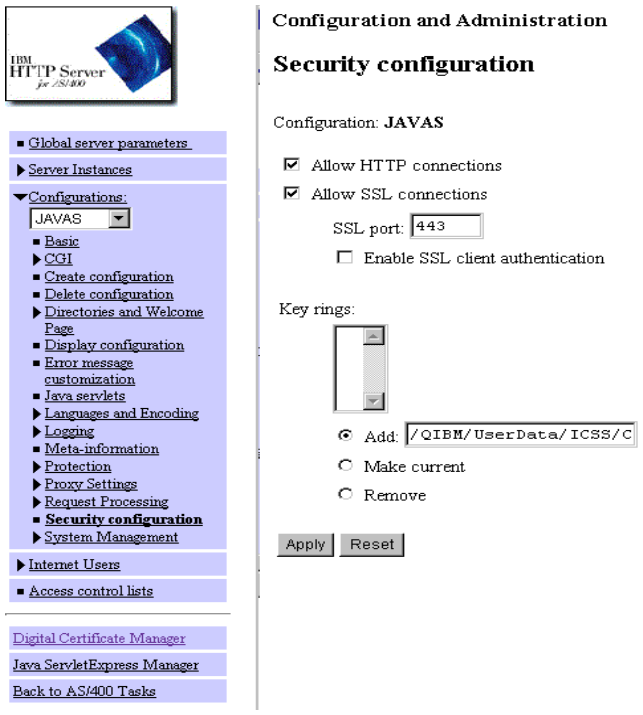
If the Digital Certificate Manager is installed, you will see a link on the left above Back to AS/400 Tasks. Clicking on Digital Certificate Manager brings up a new window so you can configure certificates and other security features while you are configuring security on the web server.

If Java is installed, you will see a link to the Java ServletExpress Manager on the left above Back to AS/400 Tasks. Clicking on Java ServletExpress Manager brings up a new window so you can configure Java servlets while you are configuring the web server.

Click on Back to AS/400 Tasks to return to the AS/400 tasks page.

Figure 257. HTTP Server Configuration

6. Click on **Security configuration**. Fill in the Security configuration page (see Figure 258 on page 276).
   a. Check **Allow SSL connections**.
   b. Accept the **default SSL port** (443) or specified the port you wish to use for SSL.
   c. Deselect **Enable SSL client authentication**.
   d. Add the **key ring path** and **file name**. If you copied it to the clipboard, you can paste it now:
   `/QIBM/USERDATA/ICSS/CERT/SERVER/DEFAULT.KYR`



Configuration and Administration

Security configuration

Configuration: JAVAS

☑ Allow HTTP connections

☑ Allow SSL connections

SSL port: 443

☐ Enable SSL client authentication

Key rings:

◉ Add: /QIBM/UserData/ICSS/C

○ Make current

○ Remove

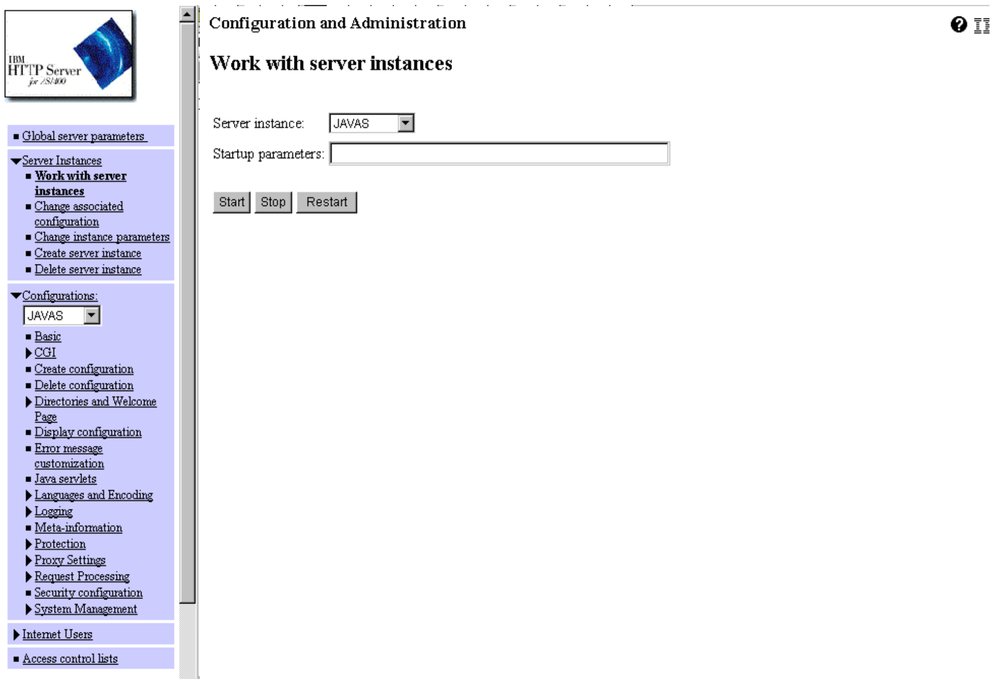
Figure 258. Security Configuration Page

Click **Apply**.

You should see this message at the top of the screen: *The configuration file was successfully updated. Server instances that are using this configuration must be stopped and started for the changes to take affect.*

You should also see your key ring file added in the Key rings box.

7. You should now stop the server instance and start it again. In the left pane window, click **Server Instances**.
8. Click on **Work with server instances**.
9. From the drop-down box, select your **server instance** (see Figure 259).


Configuration and Administration

Work with server instances

Server instance: JAVAS

Startup parameters:

Figure 259. Work with Server Instances

Click **Stop**. Wait until you see this message at the top of your window: *The server instance was successfully stopped.*

10. From the drop-down box, select your server instance (see Figure 259).

    Click **Start**.

    You should see this message: *The server instance was successfully started*.

You have now successfully configured your Web server to use SSL with server authentication.

8.5 Requesting a Server Certificate from an Internet CA

To conduct commercial business on the Internet, you should request your server certificate from an Internet Certificate Authority, such as VeriSign or Thawte, which are widely known by clients browsers and servers.

For your private Web network within your own company, university, or group, or for testing purposes you can, using Digital Certificate Manager (DCM), act as your own CA. Section 8.4, “Creating a Self-Signed Certificate” on page 269, explains this procedure.

This section describes how to obtain a server certificate from an Internet certificate authority. To use a server certificate issued by an Internet CA, perform these steps:

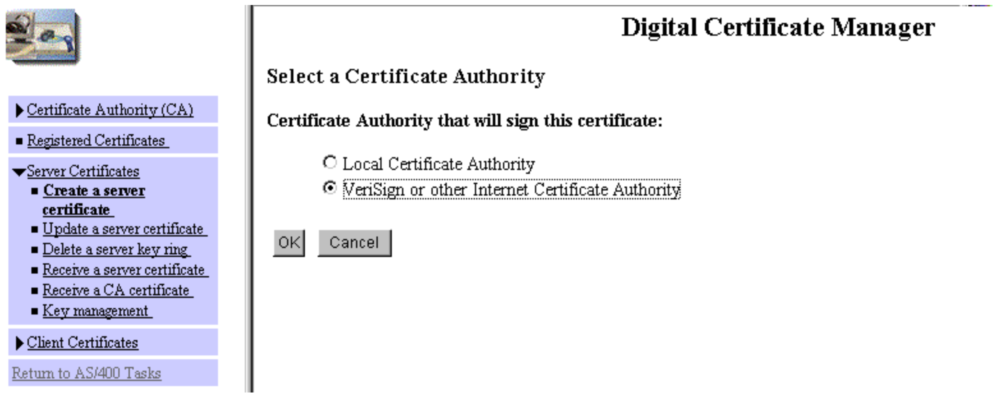
1. Request the server certificate from an Internet CA.
2. Receive a server certificate for this server.
3. Configure the HTTP server to use SSL and Server Authentication.

8.5.1 Requesting a Server Certificate from an Internet CA

To use SSL for secure Web serving, your server must have a digital certificate. You can use an intranet certification authority (CA) to issue a server certificate (see Section 8.4, “Creating a Self-Signed Certificate” on page 269), or you can use an Internet CA.

When you choose to use an Internet CA to issue a server certificate, you must first request the certificate. Follow these steps:

1. From the Digital Certificate Manager (DCM) page, click **Server Certificates** in the left-hand frame to display an extended list of server tasks.
2. Click on **Create a server certificate** from the list to display the Select a Certificate Authority page.
3. Select **VeriSign** or other **Internet Certificate Authority** as shown in Figure 260.



Select a Certificate Authority

Certificate Authority that will sign this certificate:

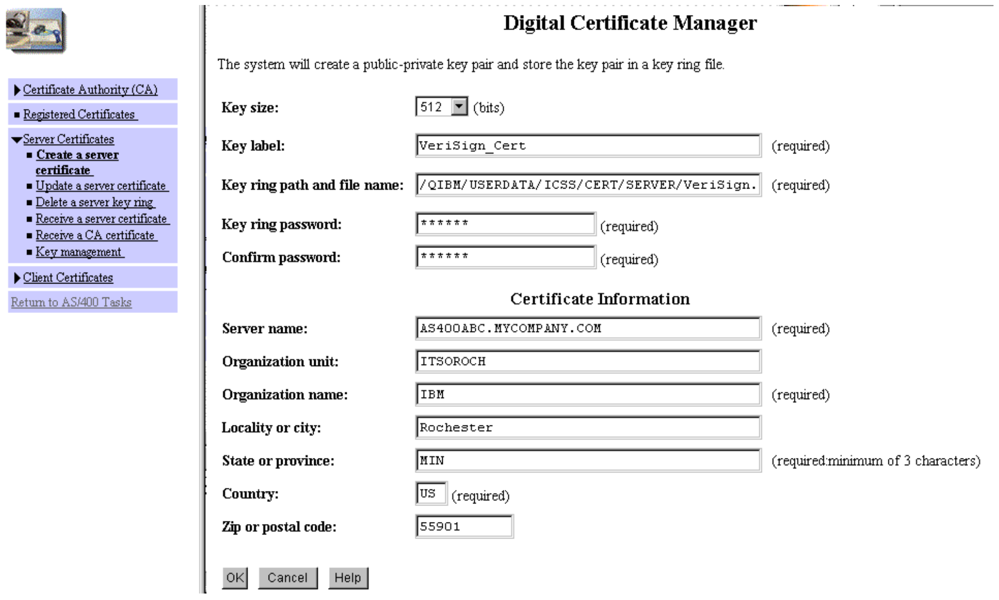
Local Certificate Authority [✓ VeriSign or other Internet Certificate Authority]

Figure 260. Requesting a Certificate from VeriSign or other Internet Certificate Authority

Click **OK** to display the Create a Server Certificate form.

4. Complete the Create a Server Certificate form as show in Figure 261 on page 279 replacing the field values with your organization information.

   The options for the key size are determined by the IBM Cryptographic Access Provider (5769-ACx) licensed program installed in your system. This is the key size that will be used to generate your public and private keys.



Digital Certificate Manager

The system will create a public-private key pair and store the key pair in a key ring file.

Key size:	512 (bits)	
Key label:	VeriSign_Cert	(required)
Key ring path and file name:	/QIBM/USERDATA/ICSS/CERT/SERVER/VeriSign.	(required)
Key ring password:	******	(required)
Confirm password:	******	(required)

Certificate Information

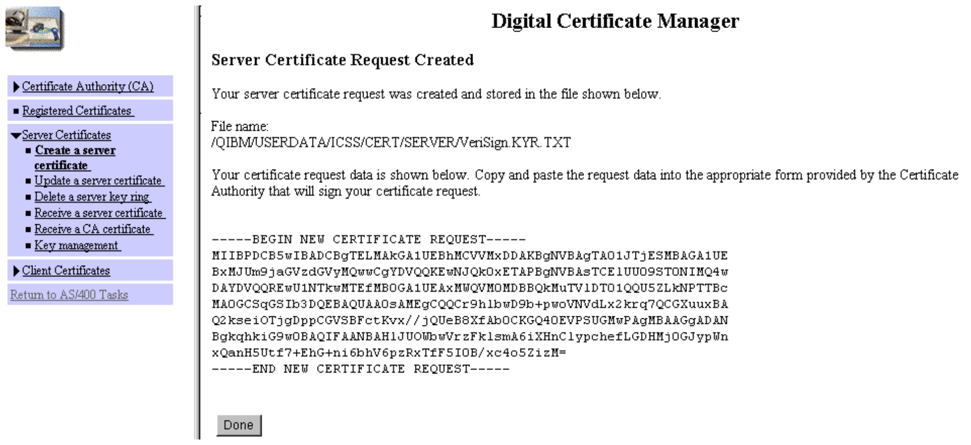
Server name:	AS400ABC.MYCOMPANY.COM	(required)
Organization unit:	ITSOROCH	
Organization name:	IBM	(required)
Locality or city:	Rochester	
State or province:	MIN	(required:minimum of 3 characters)
Country:	US	(required)
Zip or postal code:	55901	

Figure 261. Request a Server Certificate from an Internet CA

By default, the system inserts the fully qualified name of the AS/400 system into the system name field. Do not change this name. This is the name used to describe your server. You can give the server any name, although the fully qualified TCP/IP host name is usually used for the server name.

Click **OK** to process the Create a Certificate Request form.

You receive the Server Certificate Request Created page as shown in Figure 262.



Digital Certificate Manager

Server Certificate Request Created

Your server certificate request was created and stored in the file shown below.

File name:
/QIBM/USERDATA/ICSS/CERT/SERVER/VeriSign.KYR.TXT

Your certificate request data is shown below. Copy and paste the request data into the appropriate form provided by the Certificate Authority that will sign your certificate request.

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBPDCB5wIBADCBgTELMAkGA1UEBhMCVVMxDDAKBgNVBAgTA01JTjESMBAGA1UE
BxMJUm9jaGVzdGVyMQwwCgYDVQQKEwNJQk0xETAPBgNVBAsTCE1UU09STONIMQ4w
DAYDVQQREwU1NTkwMTEfMB0GA1UEAxMWQVM0MDBBQkMuTV1DT01QQU5ZLkNPTTBc
MA0GCSqGSIb3DQEBAQUAA0sAMEgCQQCr9h1bwD9b+pwoVNVdLx2krq7QCGXuuxBA
Q2kseiOTjgDppCGVSBFctKvx//jQUeB8XfAbOCKGQ40EVPSUGMwPAgMBAAGgADAN
BgkqhkiG9w0BAQIFAANBAH1JUOWbwVrzFklsmA6iXHnClypchefLGDHMjOGJypWn
xQanH5Utf7+EhG+ni6bhV6pzRxTfF5IOB/xc4o5ZizM=
-----END NEW CERTIFICATE REQUEST-----
```

Figure 262. Server Certificate Request Generated by DCM

Note: Do not click done or close the browser yet. You need to cut and paste the certificate request when you submit the Certificate Signing Request to the Internet CA.

5. Copy the Server Certificate Request to your clipboard. Start at -----BEGIN NEW CERTIFICATE REQUEST----- and end at -----END NEW CERTIFICATE REQUEST-----. Click **Done** to close the page.

6. Follow your Internet CA procedures to paste the certificate request. For example, to request a certificate from VeriSign, follow the instructions that are described on the following URL:

   `http://www.verisign.com`

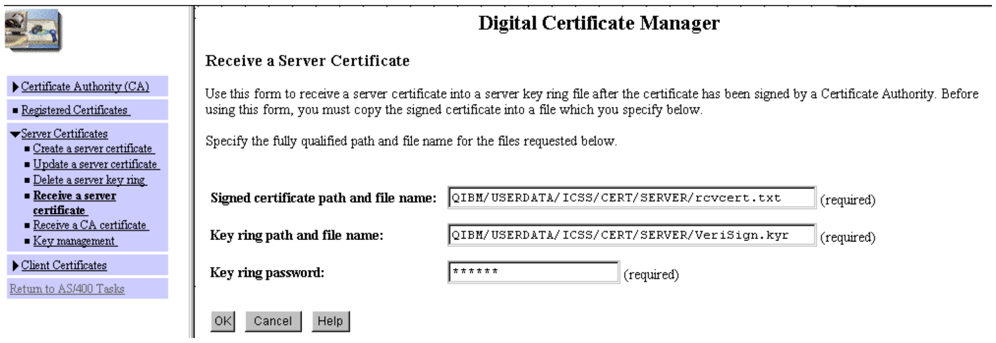
   When VeriSign is satisfied that you meet all of its requirements, it e-mails the secure server certificate to you. You should receive it in three to five business days. Other certificates authorities have their own procedures.

8.5.2 Receiving a Server Certificate for this Server

After you receive the certificate from the Internet CA, you need to copy the signed server certificate to a text file that DCM can access when you perform the Receive server certificate task. Perform the following steps:

1. Copy the signed server certificate presented to you by the Internet CA to your clipboard. Start at -----BEGIN CERTIFICATE REQUEST-----, and end at -----END CERTIFICATE REQUEST-----.
2. Paste the signed server certificate in your clipboard into a .txt file. Use a text editor of your choice, for example Notepad, to create a .txt file and paste the server certificate issued by the Internet CA.
3. Save the file in your AS/400 system IFS. Use a mapped network drive and save the .txt file that contains the server certificate issued by the Internet CA in the following path (enter a file name of your choice):

   `/QIBM/USERDATA/ICSS/CERT/SERVER/rcvcert.txt`
4. In DCM, click **Receive a server certificate** and complete the Receive a Server Certificate page (Figure 263).



Digital Certificate Manager

Receive a Server Certificate

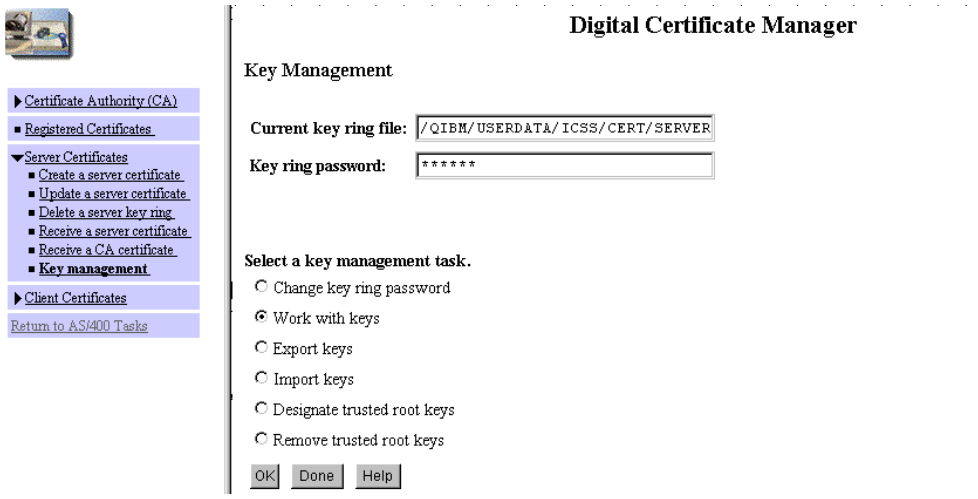
Use this form to receive a server certificate into a server key ring file after the certificate has been signed by a Certificate Authority. Before using this form, you must copy the signed certificate into a file which you specify below.

Specify the fully qualified path and file name for the files requested below.

Signed certificate path and file name:	QIBM/USERDATA/ICSS/CERT/SERVER/rcvcert.txt	(required)
Key ring path and file name:	QIBM/USERDATA/ICSS/CERT/SERVER/VeriSign.kyr	(required)
Key ring password:	******	(required)

Figure 263. Receiving a Server Certificate Issued by an Internet CA

5. The Certificate Received page is displayed. You have the option to use the received certificate with the ADMIN or LDAP server. Do not select these options. Click **OK**.
6. You should receive a Server Configuration Status message indicating the server certificate operations are complete. Click **Done**.
7. You must now set the key as the default key. In DCM, click K**ey management**. Complete the Key Management page and select **Work with keys** (see Figure 264 on page 281).



Digital Certificate Manager

Key Management

Current key ring file: /QIBM/USERDATA/ICSS/CERT/SERVER

Key ring password: ******

Select a key management task.

- ○ Change key ring password
- ● Work with keys
- ○ Export keys
- ○ Import keys
- ○ Designate trusted root keys
- ○ Remove trusted root keys

Figure 264. Key Management Page

8. Select the key with the label corresponding to the certificate you received from the Internet CA (VeriSign_Cert in our example). Select **Set key to be the default** and click **OK**.

8.5.3 Configuring the HTTP Server to Use SSL

This task is described in Section 8.4.3, “Configuring the Web Server to Use SSL with Server Authentication” on page 275.

8.6 Applying Security to the Applications

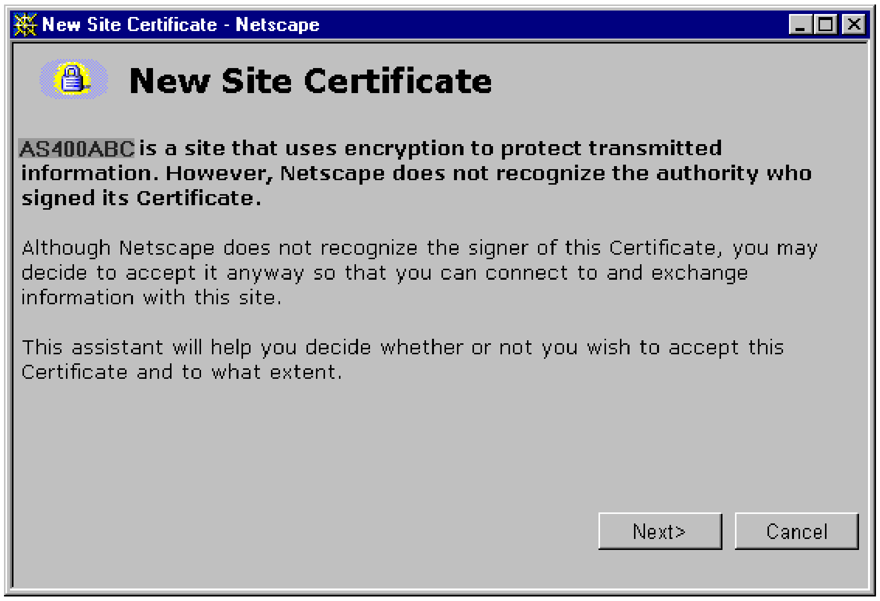
In this section, we run an application using digital certificates and SSL. Since we are using the security support provided by the IBM HTTP Server for AS/400, the benefits apply directly to servlets. For applets, we can use the HTTP server to initiate the applet, but once the applet starts and is communicating with the AS/400 system, it uses its own sockets connection to interface with the AS/400 system. In this case, you have to provide your own encryption to protect information going across an Internet connection. As discussed in Chapter 3, “Introduction to AS/400 Applets” on page 51, you can use the browser’s security classes to provide digital certificate support.

8.6.1 Servlets

In Section 8.3, “AS/400 Implementation of Digital Certificate Management” on page 268, we show configuring the test AS/400 system. We use this configuration to run the servlet discussed in Chapter 4, “Introduction to AS/400 Servlets” on page 159, using Netscape Communicator. If you run the application under another browser, you use similar dialogs to help you control the security of the application. To start the servlet, we enter:

```
https://AS400ABC/apptest/Parts.html
```

Before the servlet starts, we are presented with security dialogs. Since we did not obtain a digital certificate from a universally recognized Internet Certificate Authority, the browser displays the warning dialog shown in Figure 265 on page 282.



New Site Certificate

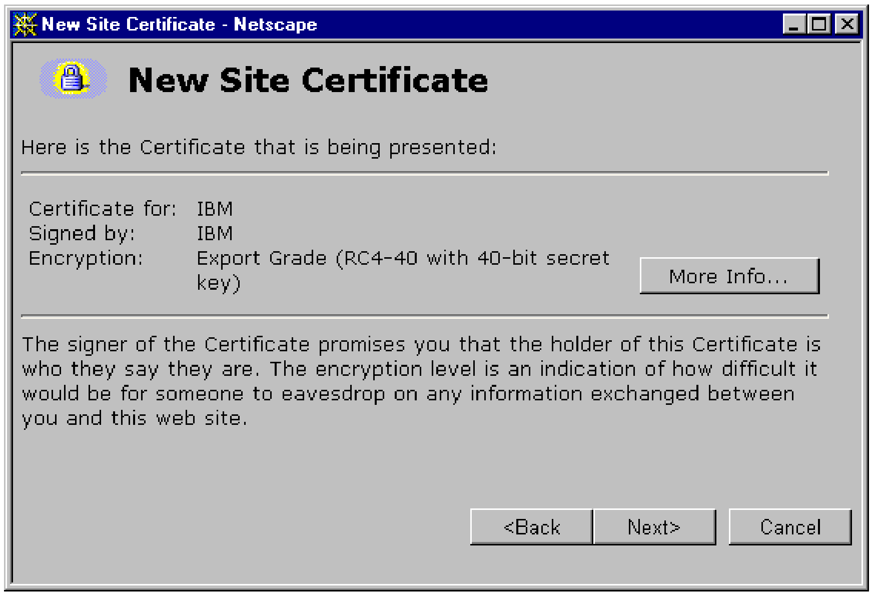
AS400ABC is a site that uses encryption to protect transmitted information. However, Netscape does not recognize the authority who signed its Certificate.

Although Netscape does not recognize the signer of this Certificate, you may decide to accept it anyway so that you can connect to and exchange information with this site.

This assistant will help you decide whether or not you wish to accept this Certificate and to what extent.

Figure 265. New Site Certificate

Clicking on the Next button causes the dialog shown in Figure 266 to appear. This dialog allows us to display more information about the certificate that is being presented.



New Site Certificate

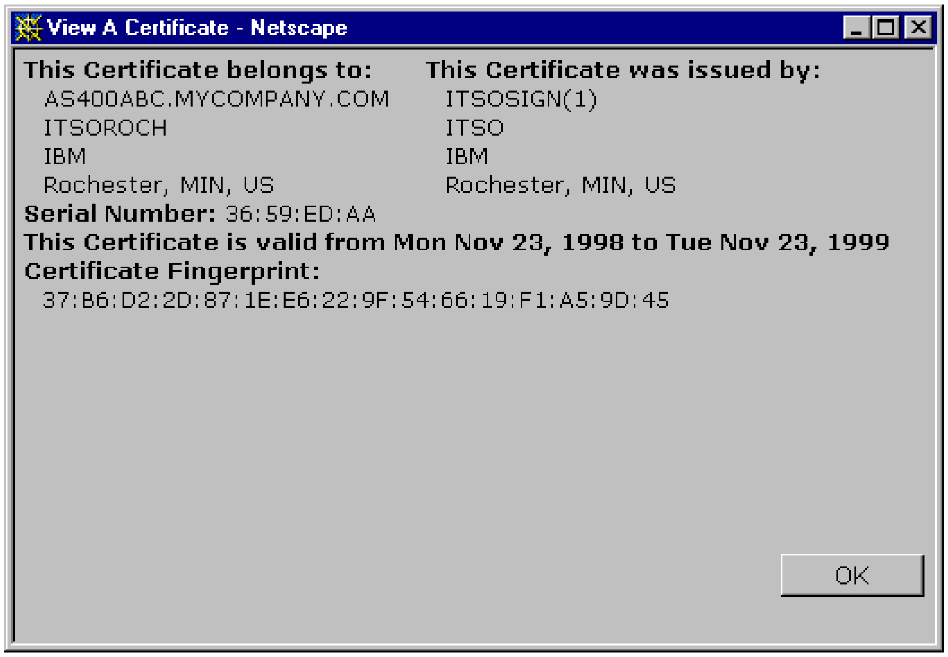
Here is the Certificate that is being presented:

Certificate for:	IBM
Signed by:	IBM
Encryption:	Export Grade (RC4-40 with 40-bit secret key)

The signer of the Certificate promises you that the holder of this Certificate is who they say they are. The encryption level is an indication of how difficult it would be for someone to eavesdrop on any information exchanged between you and this web site.

Figure 266. New Site Certificate Information

Clicking on the More Info... button shows a dialog which contains information about the issuer of the certificate.



View A Certificate - Netscape

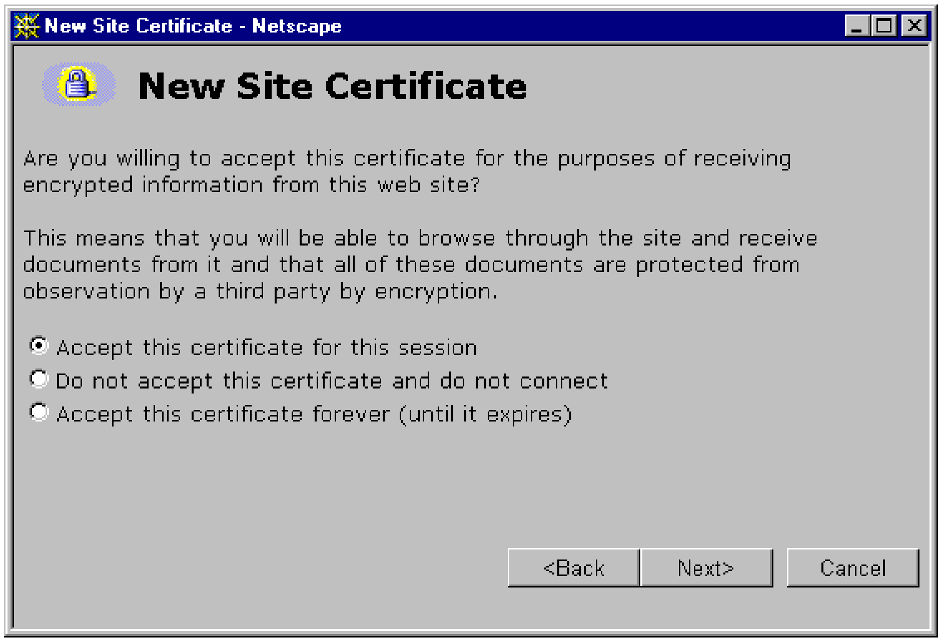
This Certificate belongs to:
AS400ABC.MYCOMPANY.COM
ITSOROCH
IBM
Rochester, MIN, US

This Certificate was issued by:
ITSOSIGN(1)
ITSO
IBM
Rochester, MIN, US

Serial Number: 36:59:ED:AA
This Certificate is valid from Mon Nov 23, 1998 to Tue Nov 23, 1999
Certificate Fingerprint:
37:B6:D2:2D:87:1E:E6:22:9F:54:66:19:F1:A5:9D:45

Figure 267. View a Certificate

Clicking on the Next button in the dialog displayed in Figure 266 on page 282, causes the dialog shown in Figure 268 to appear.



New Site Certificate - Netscape

New Site Certificate

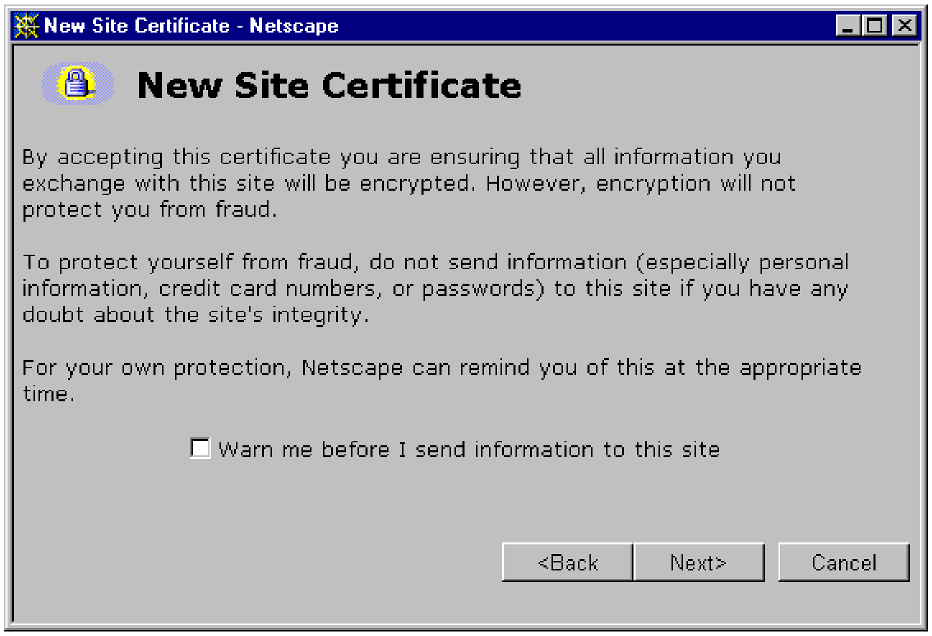
Are you willing to accept this certificate for the purposes of receiving encrypted information from this web site?

This means that you will be able to browse through the site and receive documents from it and that all of these documents are protected from observation by a third party by encryption.

● Accept this certificate for this session
○ Do not accept this certificate and do not connect
○ Accept this certificate forever (until it expires)

Figure 268. New Site Certificate Acceptance Dialog

The dialog shown in Figure 268 allows us to choose how we want to deal with the certificate received from the remote site. In this case, we recognize that it is a site that we can trust, so we set the **Accept this certificate for this session** radio button on and click on the Next button. This causes the dialog shown in Figure 269 on page 284 to appear.



By accepting this certificate you are ensuring that all information you exchange with this site will be encrypted. However, encryption will not protect you from fraud.

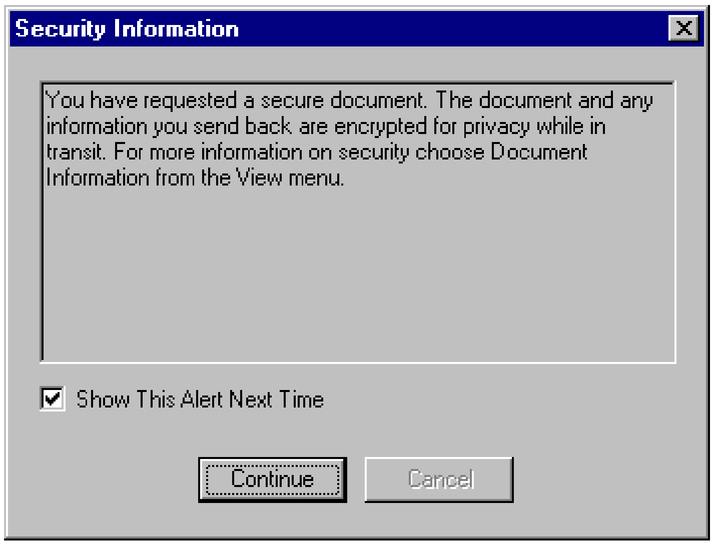
To protect yourself from fraud, do not send information (especially personal information, credit card numbers, or passwords) to this site if you have any doubt about the site's integrity.

For your own protection, Netscape can remind you of this at the appropriate time.

☐ Warn me before I send information to this site

Figure 269. Netscape Certificate Warning Dialog

After we accept the certificate, the browser displays a final warning that allows us to choose to be reminded with further warning messages. Clicking on the Next button starts the SSL session with the remote system. If this is the first time that we have requested a secure document, we are presented with the dialog shown in Figure 270.

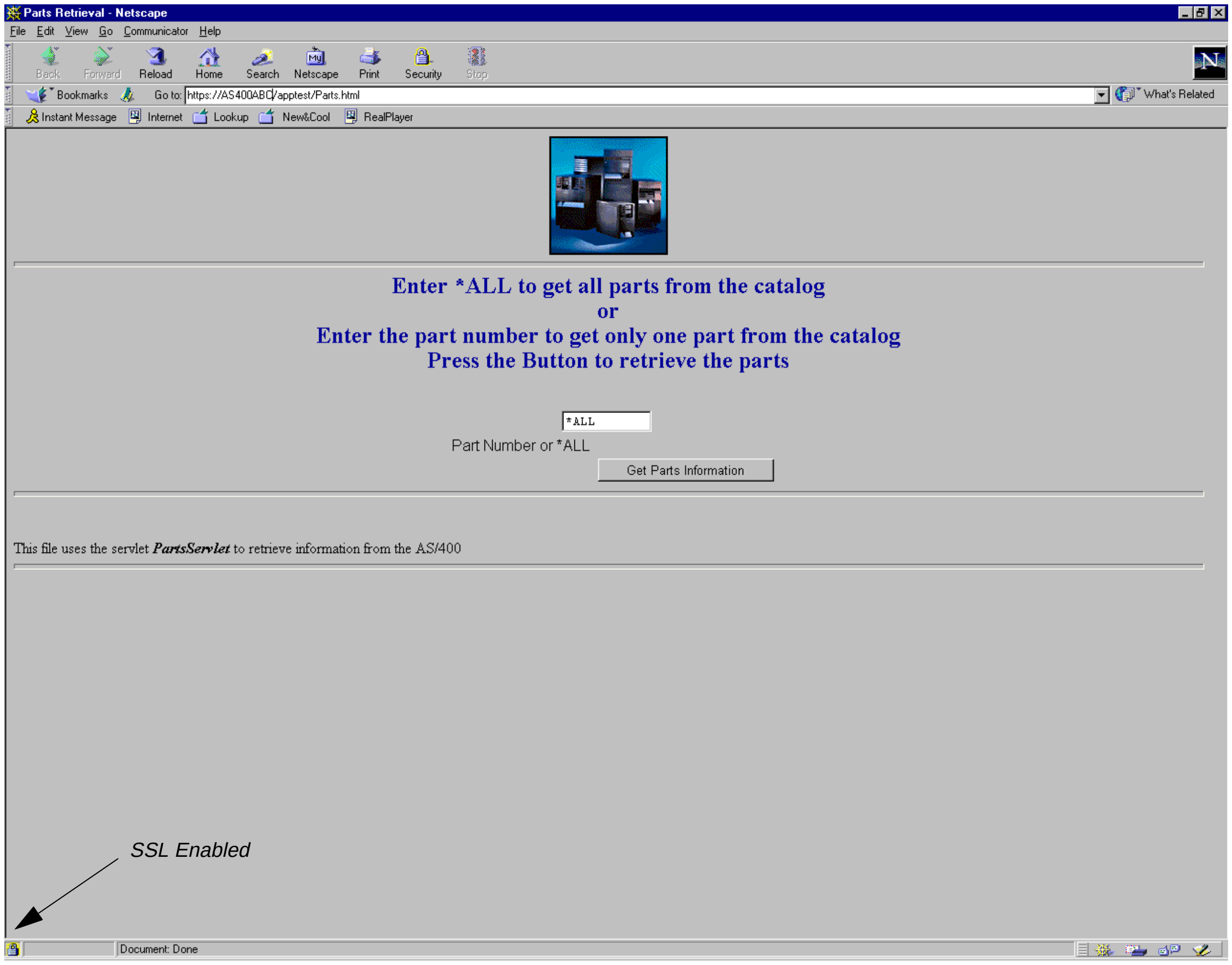


You have requested a secure document. The document and any information you send back are encrypted for privacy while in transit. For more information on security choose Document Information from the View menu.

☑ Show This Alert Next Time

Figure 270. Netscape Security Information Dialog

We can use the check box shown in Figure 270 to control whether we want to see this warning dialog in the future. If we click on the Continue button, the servlet application starts as shown in Figure 271 on page 285. The lock Icon shown in the lower left corner of the browser indicates that we are running under a SSL session.


Parts Retrieval - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Go to: https://AS400ABC/apptest/Parts.html
What's Related
Instant Message Internet Lookup New&Cool RealPlayer
Document: Done
SSL Enabled

Enter *ALL to get all parts from the catalog
or
Enter the part number to get only one part from the catalog
Press the Button to retrieve the parts

*ALL

Part Number or *ALL

Get Parts Information

This file uses the servlet *PartsServlet* to retrieve information from the AS/400

Figure 271. PartsServlet Running under SSL

8.6.2 Additional Resources

For additional information, consult the following resources:

- *HTTP Server for AS/400 Webmaster's Guide*
- *Securing Your AS/400 from Harm on the Internet,* SG24-4929
- http://publib.boulder.ibm.com/pubs/html/as400/ic2924/info/index.htm

  Click Internet**—>**Digital certificate management

- http://www.software.ibm.com/webservers/
- http://www.ibm.com/security
- http://www.ics.raleigh.com
- http://www.internet.ibm.com/commercepoint/registry/
- http://www.verisign.com/products/doc.html
- http://home.netscape.com/assist/security/ssl/index.html
- http://www.rsa.com

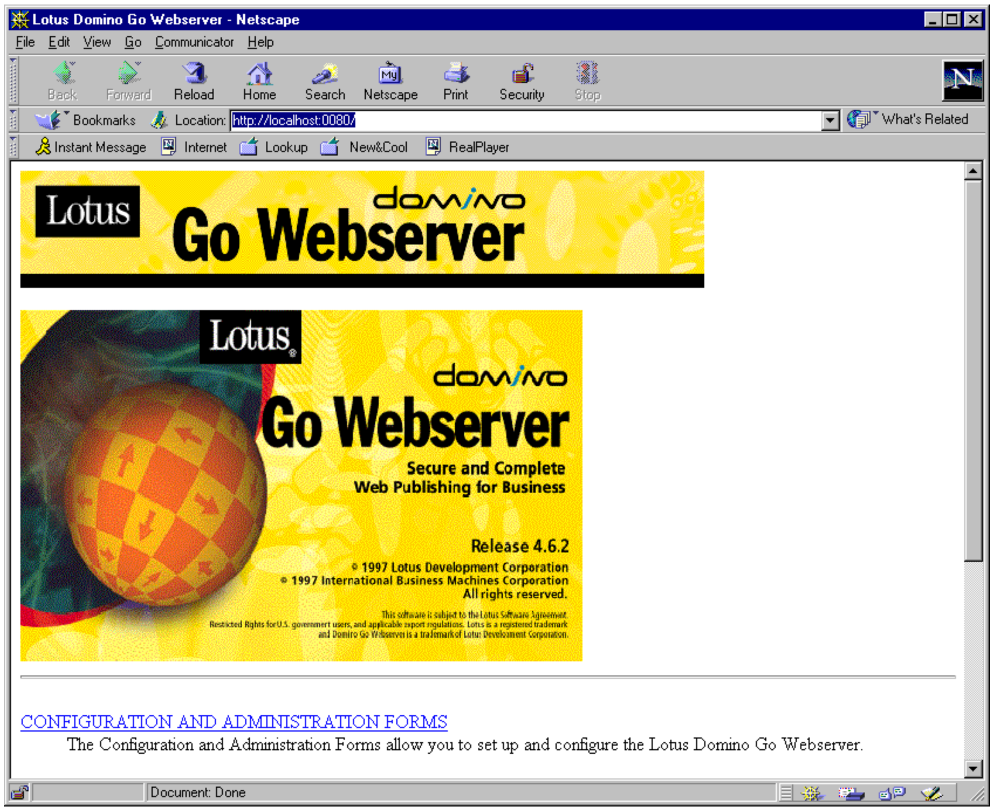
Chapter 9. HTTP Server Configuration

This chapter shows the HTTP server configurations that we use for this redbook. The chapter explains:

- Domino Go Webserver
- ServletExpress
- IBM HTTP Server for AS/400
- IBM WebSphere Application Server for AS/400

9.1 Domino Go Webserver

For more information about Domino Go Webserver, see the Web site: http://www.software.ibm.com/webservers/dgw.


Lotus Domino Go Webserver - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://localhost:0080/
What's Related
Instant Message Internet Lookup New&Cool RealPlayer
Lotus
domino
Go Webserver
Lotus
domino
Go Webserver
Secure and Complete
Web Publishing for Business
Release 4.6.2
© 1997 Lotus Development Corporation
© 1997 International Business Machines Corporation
All rights reserved.
Document: Done

This software is subject to the Lotus Software Agreement. Restricted Rights for U.S. government users, and applicable export regulations. Lotus is a registered trademark and Domino Go Webserver is a trademark of Lotus Development Corporation.

CONFIGURATION AND ADMINISTRATION FORMS

The Configuration and Administration Forms allow you to set up and configure the Lotus Domino Go Webserver.

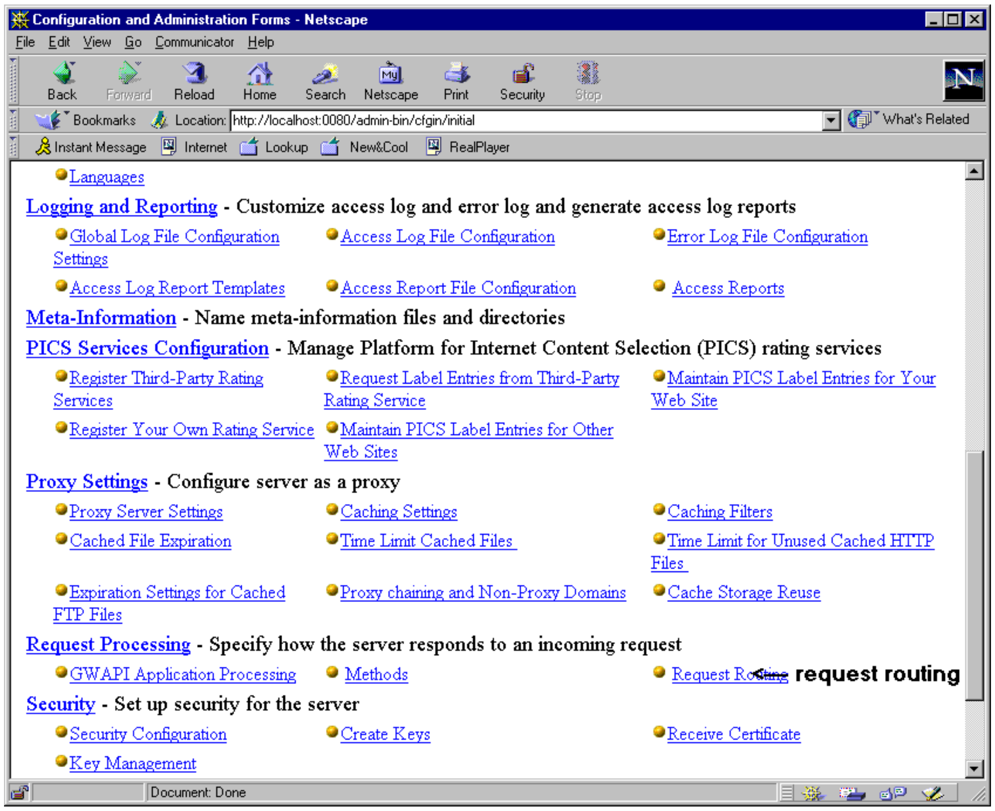
Figure 272. Domino Go Webserver Initial Screen

To see the initial Domino Go Webserver screen, enter the following URL in the browser Location field:

```
http://server:0080
```

If you are physically on the server system, you can replace the server name with localhost. Perform the following steps:

1. Select **CONFIGURATION AND ADMINISTRATION FORMS**.
2. You are prompted for security information. The default user ID and password are both **admin**.


Configuration and Administration Forms - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://localhost:0080/admin-bin/cfgin/initial What's Related
Instant Message Internet Lookup New&Cool RealPlayer
Document: Done

Languages

Logging and Reporting - Customize access log and error log and generate access log reports

Global Log File Configuration Settings · Access Log File Configuration · Error Log File Configuration

Access Log Report Templates · Access Report File Configuration · Access Reports

Meta-Information - Name meta-information files and directories

PICS Services Configuration - Manage Platform for Internet Content Selection (PICS) rating services

Register Third-Party Rating Services · Request Label Entries from Third-Party Rating Service · Maintain PICS Label Entries for Your Web Site

Register Your Own Rating Service · Maintain PICS Label Entries for Other Web Sites

Proxy Settings - Configure server as a proxy

Proxy Server Settings · Caching Settings · Caching Filters

Cached File Expiration · Time Limit Cached Files · Time Limit for Unused Cached HTTP Files

Expiration Settings for Cached FTP Files · Proxy chaining and Non-Proxy Domains · Cache Storage Reuse

Request Processing - Specify how the server responds to an incoming request

GWAPI Application Processing · Methods · Request Routing **request routing**

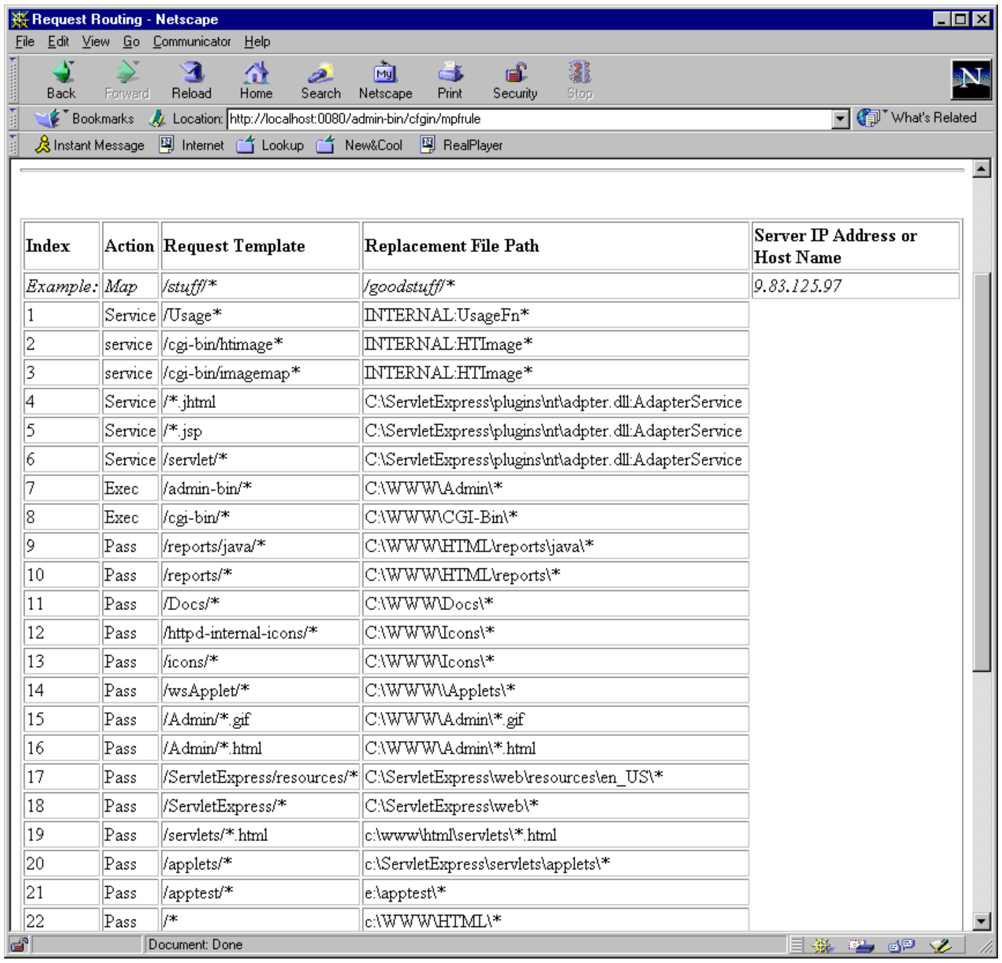
Security - Set up security for the server

Security Configuration · Create Keys · Receive Certificate

Key Management

Figure 273. Configuration and Administration Window

3. On the Configuration and Administration Form Screen, select **Request Routing**.


Request Routing - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://localhost:0080/admin-bin/cfgin/mpfrule
What's Related
Instant Message Internet Lookup New&Cool RealPlayer
Document: Done

Index	Action	Request Template	Replacement File Path	Server IP Address or Host Name
Example:	*Map*	*/stuff/**	*/goodstuff/**	*9.83.125.97*
1	Service	/Usage*	INTERNAL:UsageFn*	
2	service	/cgi-bin/htimage*	INTERNAL:HTImage*	
3	service	/cgi-bin/imagemap*	INTERNAL:HTImage*	
4	Service	/*.jhtml	C:\ServletExpress\plugins\nt\adpter.dll:AdapterService	
5	Service	/*.jsp	C:\ServletExpress\plugins\nt\adpter.dll:AdapterService	
6	Service	/servlet/*	C:\ServletExpress\plugins\nt\adpter.dll:AdapterService	
7	Exec	/admin-bin/*	C:\WWW\Admin\*	
8	Exec	/cgi-bin/*	C:\WWW\CGI-Bin\*	
9	Pass	/reports/java/*	C:\WWW\HTML\reports\java\*	
10	Pass	/reports/*	C:\WWW\HTML\reports\*	
11	Pass	/Docs/*	C:\WWW\Docs\*	
12	Pass	/httpd-internal-icons/*	C:\WWW\Icons\*	
13	Pass	/icons/*	C:\WWW\Icons\*	
14	Pass	/wsApplet/*	C:\WWW\\Applets*	
15	Pass	/Admin/*.gif	C:\WWW\Admin\*.gif	
16	Pass	/Admin/*.html	C:\WWW\Admin\*.html	
17	Pass	/ServletExpress/resources/*	C:\ServletExpress\web\resources\en_US\*	
18	Pass	/ServletExpress/*	C:\ServletExpress\web\*	
19	Pass	/servlets/*.html	c:\www\html\servlets\*.html	
20	Pass	/applets/*	c:\ServletExpress\servlets\applets\*	
21	Pass	/apptest/*	e:\apptest*	
22	Pass	/*	c:\WWW\HTML\*	

Figure 274. Domino Go Routing Table

Figure 274 shows the Domino Go Webserver routing table used for this redbook. The entries added were:

- Pass /apptest/* \apptest*

  This directive allows us serve the gif files from the apptest directory

- Pass /applets/* c:\ServletExpress\servlets\applets\*

  This directive allows us to serve applets from the applets subdirectory

9.2 ServletExpress

ServletExpress allows you manage the servlets running under the control of the Web server. You do not have to use ServletExpress to run servlets under the Domino Go HTTP server if they are stored in the default directory. For the PartsServlet class, it is stored in a package named servlets. When you export it, it is exported to a subdirectory named servlets in the default directory. You need to configure ServletExpress so it can find the PartsServlet class. ServletExpress also makes it easy for you to load and unload servlets without stopping the Web server.

We also use ServletExpress to configure the classpath environment variable that we use. In this case, we use ServletExpress to add the AS/400 Toolbox for Java to the classpath variable.

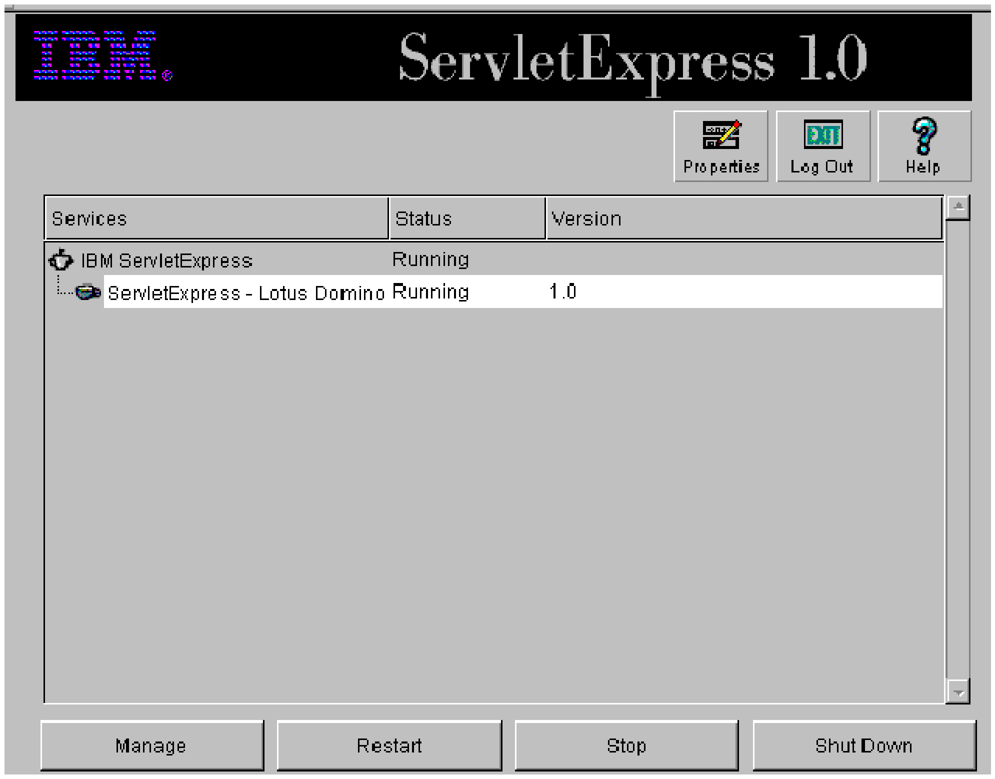
Start your Web browser, perform the following steps:

1. Enter **http://server:9090/** for the URL.
2. Make sure you add **:9090** after the server name since the administration tool server listens to the port 9090.

Note

If you are physically on the server, you can replace the server name with **localhost**.

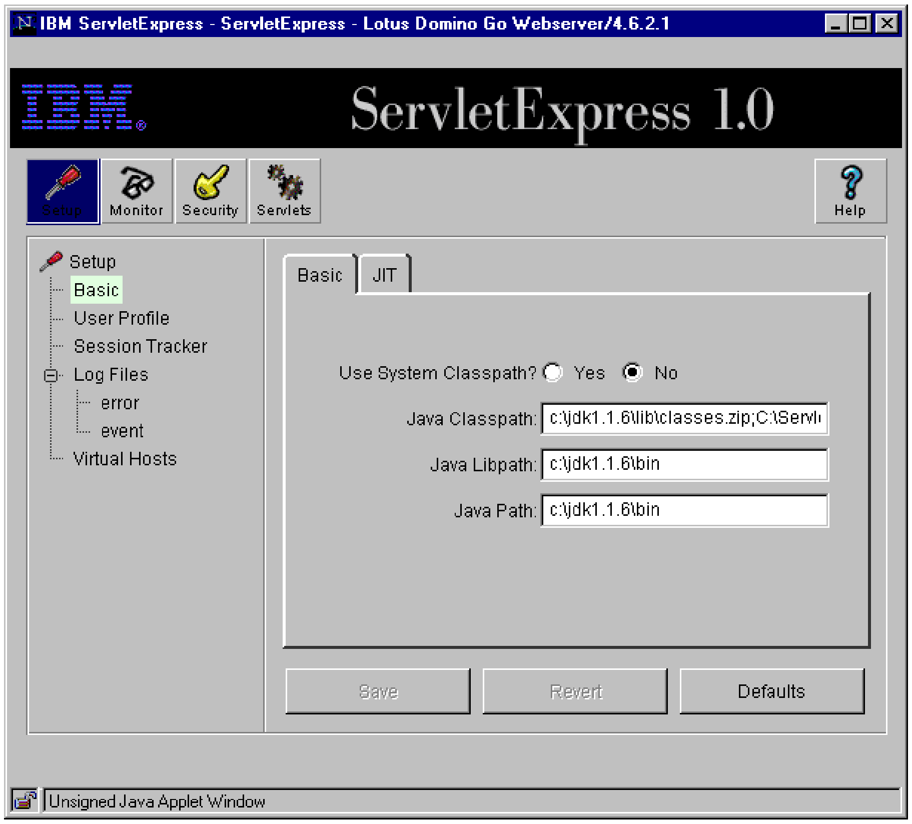
3. Login with the administrator user ID and password (the default is admin/admin). You should see a window similar to the one shown in Figure 275.



Services	Status	Version
IBM ServletExpress	Running	
ServletExpress - Lotus Domino	Running	1.0

Figure 275. ServletExpress Services Dialog

4. Click on the **Manage** button.



Use System Classpath? Yes [✓ No]

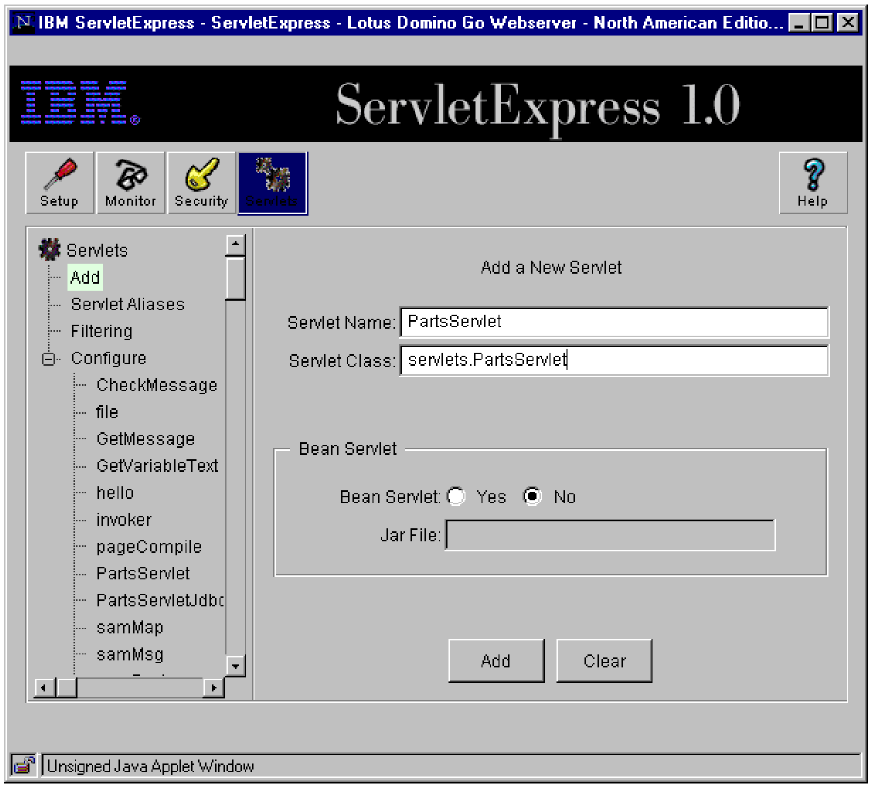
Java Classpath: c:\jdk1.1.6\lib\classes.zip;C:\Servl

Java Libpath: c:\jdk1.1.6\bin

Java Path: c:\jdk1.1.6\bin

Figure 276. ServletExpress Setup Window

5. In the Setup Window, use the Basic menu option to set the Classpath environment variable. Use the system Classpath or set your own. Be sure that the directory where the AS/400 Toolbox for Java zip file is stored is included in the Classpath.
6. After setting how the Classpath is resolved, select the **Servlets** button from the toolbar. A window appears as shown in Figure 277 on page 292.



Add a New Servlet

Servlet Name: PartsServlet

Servlet Class: servlets.PartsServlet

Bean Servlet

Bean Servlet: Yes [✓ No]

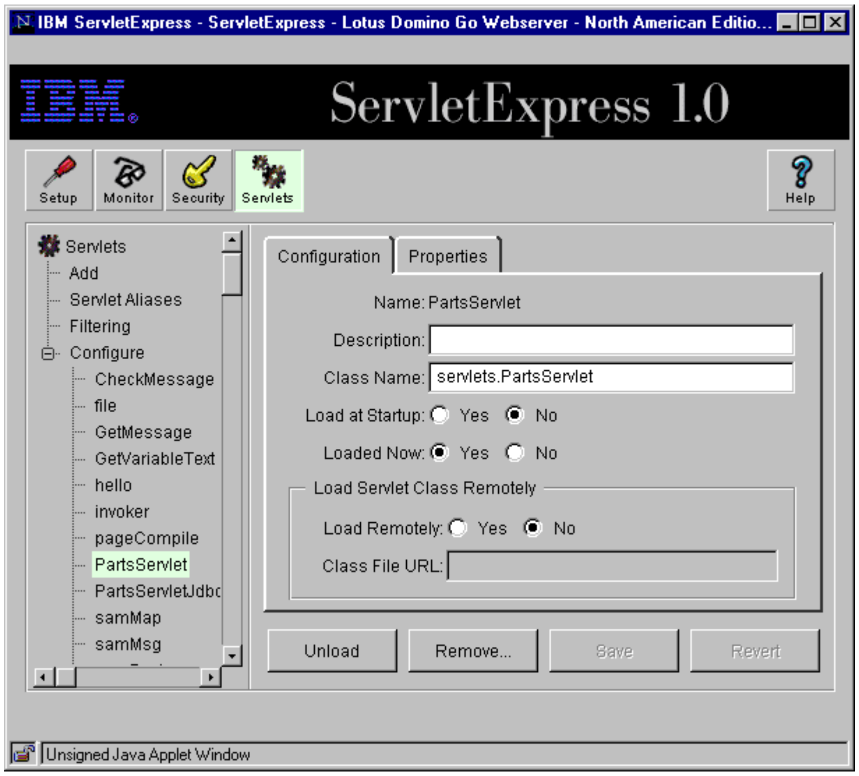
Jar File:

Figure 277. ServletExpress Add Servlet Dialog

7. You can use this window to add any new servlets. For example, to add the PartsServlet, complete these tasks:
   a. Select the **Add Servlet** options from the Servlets tree and add the information: **Specify PartsServlet** in the *Servlet Name:* text field
   b. Specify servlets.PartsServlet in the Servlet Class: text field.

      Note: Do not add the .class extension

   c. Press the **Add** button
8. In the next window that appears, you can see that there are options or buttons to load or unload servlets without the need for shutting down the whole HTTP server.



Name: PartsServlet

Description:

Class Name: servlets.PartsServlet

Load at Startup: Yes [✓ No]

Loaded Now: [✓ Yes] No

Load Servlet Class Remotely

Load Remotely: Yes [✓ No]

Class File URL:

Figure 278. ServletExpress Servlet Configuration Dialog

9.3 IBM HTTP Server for AS/400

For more information about the IBM HTTP Server for AS/400, see the Web site: http://www.as400.ibm.com/http

We store the class files, jar files, gif files, and HTML files used to run the applets and servlets discussed in this redbook in an Integrated File System (IFS) directory named `apptest`.

By default, IBM HTTP Server for AS/400 serves servlets from the following directory:

```
/QIBM/ProdData/IBMWebAS/servlets
```

We also use three applets as part of the servlet discussed in Chapter 7, “Developing AS/400 Java Servlets” on page 221. To keep access to the applets secure and separate from the servlets, we export them to the following directory:

```
/QIBM/ProdData/IBMWebAS/servlets/applets
```

To run the applets and servlets discussed in this redbook, you need to configure the HTTP Server for AS/400 so it can serve the servlets and applets from these directories.

Note

It is beyond the scope of this redbook to describe in detail how to use the configuration programs for the HTTP Server for AS/400. You can find additional information about how to configure and work the HTTP Server for AS/400 in the following manuals:

- *HTTP Server for AS/400 Quick Beginnings*, GC41-5433
- *HTTP Server Webmaster's Guide*, GC41-5434

The configuration task requires that you add directives to the HTTP Server configuration file for the server configuration that will be used for applet and servlet serving. The directives to be added are `PASS` directives:

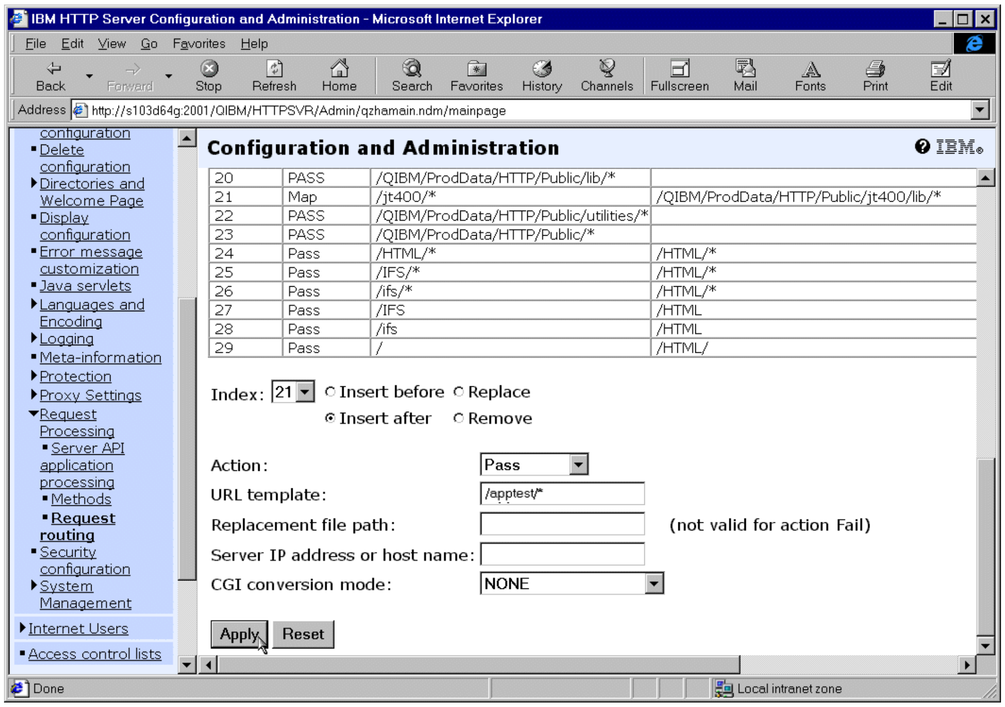
```
PASS    /apptest/*
PASS /applets/* /QIBM/ProdData/IBMWebAS/servlets/applets
```

If you want to run servlets on the AS/400 system, you must also add special directives to the HTTP configuration, see Section 9.4, “IBM WebSphere Application Server for AS/400” on page 296, for details.

There are two techniques that you can use to add the directives to the server configuration file:

- Use the browser-based configuration program. You start this program by entering one of the following URLs in your browser (substitute your server name or IP address in the URL):
  - `http://server_name:2001`
  - `http://ip_address:2001`

  Figure 279 on page 295 shows the V4R3 version of the browser-based configuration program. You need to navigate to the `Configurations-->Request Processing-->Request Routing` section of the configuration program to arrive at the page where you can enter the `PASS` directive.
- Use the OS/400 command Work with HTTP Configuration (`WRKHTTPCFG`). This command is used in a 5250 session to invoke a line editor that can be used to add the `PASS` directive to a selected configuration file member.


IBM HTTP Server Configuration and Administration - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Fonts Print Edit
Address http://s103d64g:2001/QIBM/HTTPSVR/Admin/qzhamain.ndm/mainpage
configuration
Delete configuration
Directories and Welcome Page
Display configuration
Error message customization
Java servlets
Languages and Encoding
Logging
Meta-information
Protection
Proxy Settings
Request Processing
Server API application processing
Methods
Request routing
Security configuration
System Management
Internet Users
Access control lists
Done
Local intranet zone

Configuration and Administration

20	PASS	/QIBM/ProdData/HTTP/Public/lib/*	
21	Map	/jt400/*	/QIBM/ProdData/HTTP/Public/jt400/lib/*
22	PASS	/QIBM/ProdData/HTTP/Public/utilities/*	
23	PASS	/QIBM/ProdData/HTTP/Public/*	
24	Pass	/HTML/*	/HTML/*
25	Pass	/IFS/*	/HTML/*
26	Pass	/ifs/*	/HTML/*
27	Pass	/IFS	/HTML
28	Pass	/ifs	/HTML
29	Pass	/	/HTML/

Index: 21 Insert before [✓ Insert after] Replace Remove

Action: Pass
URL template: /apptest/*
Replacement file path: (not valid for action Fail)
Server IP address or host name:
CGI conversion mode: NONE

Apply Reset

Figure 279. Browser-based HTTP Server Configuration Program

Note

Regardless of the technique you use to add the PASS directive to the HTTP Server configuration file, be sure that you place the PASS directive for the apptest directory before the PASS / directive.

The PASS / directive is considered to be a "catch-all" directive. If that directive is encountered before your PASS directive, the HTTP Server attempts to serve the requested HTML file from the directory associated with the PASS / directive.

Start or Restart the HTTP Server Configuration Instance

After adding your PASS directive to the HTTP Server configuration file, you need to start or restart the HTTP Server configuration instance.

If the HTTP Server instance is not active, use the following OS/400 command to start the server instance:

```
STRTCPSVR SERVER(*HTTP) HTTPSVR(DEFAULT)
```

If the HTTP Server instance is already active, use the following OS/400 command to restart the server instance:

```
STRTCPSVR SERVER(*HTTP) RESTART(*HTTP) HTTPSVR(DEFAULT)
```

For either of those commands, substitute your server instance name if you need to start or restart a server instance other than DEFAULT. For example, we use the name **JAVAS** for our configuration file.

9.4 IBM WebSphere Application Server for AS/400

The IBM WebSphere Application Server for AS/400 provides the same functionality of what the product provides on the other shipped platforms. On the AS/400 system, it enables servlet support. Please refer to the official WebSphere Web site for complete product information. It can be found at: http://www.software.ibm.com/webservers/appserv/index.html

Attention

A number of PTFs are required to enable the WebSphere support on the AS/400 system. The PTFs are not part of a cumulative tape at this point, so you need to order them. To see the current list of PTFs, go the Web page **http://www.as400.ibm.com/http** and click on the **WebSphere Application Server** link.

The AS/400 version has the same installation structure as the other platforms:

```
<INSTALL_ROOT>...
```

In this case, for the AS/400 system, <INSTALL_ROOT> is **/QIBM/ProdData/IBMWebAS/**.

You can configure a server instance of the IBM HTTP Server for AS/400 to run WebSphere. It is enabled through the Server API of the HTTP Server. Figure 280 shows the configuration directives you need to add to the HTTP server configuration file to enable WebSphere. For the entire listing of the IBM HTTP Server for AS/400 configuration file used for this redbook, refer to Appendix B, “IBM HTTP Server for AS400 Configuration” on page 301.

```
# Enable WebSphere support
Service    /servlet/* /QSYS.LIB/QHTTPSVR.LIB/QZHJSVLT.SRVPGM:AdapterService*
Service    /*.jsp     /QSYS.LIB/QHTTPSVR.LIB/QZHJSVLT.SRVPGM:AdapterService
ServerInit /QSYS.LIB/QHTTPSVR.LIB/QZHJSVLT.SRVPGM:AdapterInit
/QIBM/ProdData/IBMWebAS/properties/server/servlet/servletservice/jvm.properties
ServerTerm /QSYS.LIB/QHTTPSVR.LIB/QZHJSVLT.SRVPGM:AdapterExit
Pass       /IBMWebAS/samples/*       /QIBM/ProdData/IBMwebAS/samples/*
Pass       /IBMWebAS/doc/*           /QIBM/ProdData/IBMwebAS/doc/*
Pass       /IBMWebAS/system/admin/*  /QIBM/ProdData/IBMwebAS/system/admin/*
Pass       /IBMWebAS/*               /QIBM/ProdData/IBMwebAS/web/*
# end - WebSphere support
```

Figure 280. HTTP Server Directives to Enable WebSphere

In the near future, you can use the Administration and Configuration form of the HTTP Server to add those directives automatically when you select to enable the Java Servlet support.

Note: Today, the form is there, but the directives added are not correct for the WebSphere support.

Once you select the configuration file and add the directives shown in Figure 280 to enable WebSphere, you need to change the jvm.properties files under the **/QIBM/ProdData/IBMWebAS/properties/server/servlet/servletservice/** directory. Change the following line in the file to point to your configuration file, the shipped default is **CONFIG**. In this case, we use a configuration file named **JAVAS**.

```
# Properties for Domino Go
#ncf.native.httpd.cnf.path=c:\winnt\httpd.cnf
ncf.native.httpd.cnf.path=/QSYS.LIB/QUSRSYS.LIB/QATMHTTPC.FILE/JAVAS.MBR
```

Figure 281. JVM Properties File

Once you change this line, you are ready to start the IBM WebSphere Application Server on the AS/400 system. Start the HTTP server instance that points to the configuration file that you have configured for WebSphere.

Note

It may take a little longer than you expect for your HTTP server instance to come up due to the fact that it needs to create a Java Virtual Machine for WebSphere to run under. Watch the server instance on WRKACTJOB until you see it finish bring coming up and going into TIMW status.

9.4.0.1 Verifying the Installation

To verify that you have WebSphere set up correctly, try to run the HelloWorldServlet as your first servlet on the AS/400 system. Set the URL on your browser to:

`http://yourAS400Server:xxxx/servlet/HelloWorldServlet`

yourServer:xxxx is your server name and the corresponding port number you used for this server.

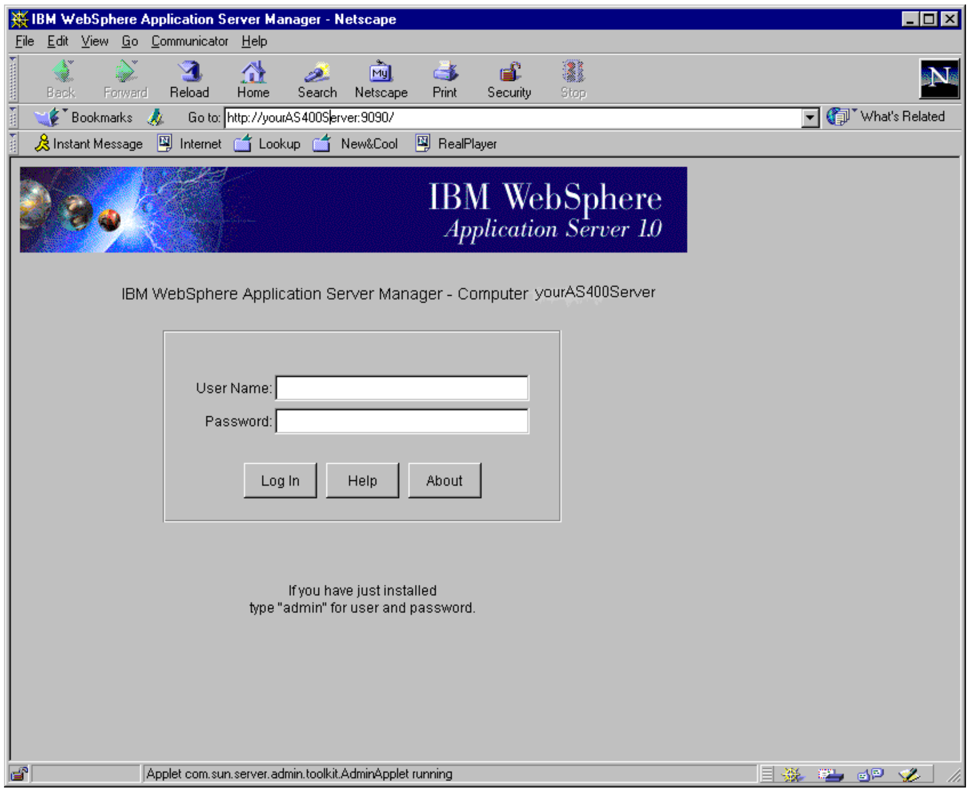
If you "Hello World" appears as the result, your WebSphere Application Server is set up and ready. Again, you need to be patient here to allow the servlet to be loaded into the Java Virtual Machine.

9.4.0.2 WebSphere Administration Graphical User Interface

The IBM WebSphere Application Server also provides its own Web-based GUI for configuration, called the Application Server Manager. To get to the GUI, point your browser to the following URL:

`http://yourAS400Server:9090`

The WebSphere Signon window appears as shown in Figure 282 on page 298.


IBM WebSphere Application Server Manager - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Go to: http://yourAS400Server:9090/
What's Related
Instant Message Internet Lookup New&Cool RealPlayer
IBM WebSphere Application Server 1.0
User Name:
Password:
Log In Help About
Applet com.sun.server.admin.toolkit.AdminApplet running

IBM WebSphere Application Server Manager - Computer yourAS400Server

If you have just installed
type "admin" for user and password.

Figure 282. WebSphere Sign On Screen

The graphical user interface provides configuration screens that allow you to configure servlets. The screens and the configuration scenario is the same as for ServletExpress used with the Domino Go Webserver. Please see Section 9.2, “ServletExpress” on page 289 for details about configuring servlets. You do not need to use the Application Server Manager to run servlets on the AS/400 system from the default directory. For the PartsServlet class, it is stored in a package named servlets. When you export it, it is exported to a subdirectory named servlets in the default directory. You need to configure WebSphere so it can find the PartsServlet class. The WebSphere graphical user interface also makes it easy for you to load and unload servlets without stopping the Web server.

We also use the Application Server Manager to configure the classpath environment variable that we use. In this case, we use the Application Server Manager to add the AS/400 Toolbox for Java to the classpath variable.

The Application Server Manager also makes it easy for you to load and unload servlets without starting and stopping the HTTP server.

Appendix A. Example Programs

The Java programs and the AS/400 programs and libraries used in this redbook are available for you to download from the Internet. These examples were developed using VisualAge for Java Version 2.0 Enterprise edition. OS/400 V3R2 or later is required. To run the servlet examples on the AS/400 system, OS/400 V4R3 or later is required. The following VisualAge for Java projects are available:

- **AdvancedServlet**—Advanced servlet example. See Chapter 7, “Developing AS/400 Java Servlets” on page 221.
- **AppletWorkshop**—Advanced applet example. See Chapter 6, “Developing AS/400 Java Applets” on page 193.
- **ServletExamples**—Beginning applets and servlet examples. See Chapter 3, “Introduction to AS/400 Applets” on page 51 and Chapter 4, “Introduction to AS/400 Servlets” on page 159.

Important Information

These example programs have not been subjected to any formal testing. They are provided "as is". Use them for *reference only*. Please refer to Appendix C, “Special Notices” on page 303, for more information.

A.1 Downloading the Files from the Internet Web Site

To use these files, you must download them to your personal computer from the Internet Web site. A file named **README.TXT** is included. It contains instructions for restoring the AS/400 libraries, the VisualAge for Java examples and runtime notes. The URL to access is: www.redbooks.ibm.com

Click on **Additional Materials** and select the directory **SG245337**. In the SG245337 directory, click on **readme.txt**.

A.2 Setting up VisualAge for Java

VisualAge for Java, Enterprise Edition, Version 2.0 requires the following software and hardware for development with the IDE:

- Windows 95 or Windows NT 4.0 with service pack 3
- TCP/IP communication protocol
- Pentium processor or higher recommended
- SVGA 800x600 display or higher (1024x768 recommended)
- 64 MB RAM minimum (80 MB recommended)
- Frames-capable Web browser
  - Netscape Navigator 4.04 or higher, or
  - Microsoft Internet Explorer 4.01 or higher
- Java Development Kit (JDK) 1.1 for deploying applications or
- JDK 1.1.2 for deploying applications using Swing components

The Java support classes described in the following sections are required:

A.2.1 The AS/400 Toolbox for Java Classes

The example programs require that the AS/400 Toolbox for Java classes be inside the VisualAge for Java Integrated Development Environment. You must import these classes inside the IDE. Enterprise Edition simplifies this process. After you install VisualAge for Java 2.0 Enterprise edition, the Toolbox classes are available in the repository as part of the IBM Enterprise Toolkit for AS/400 project. If you want to use the Toolbox classes, perform the following steps:

1. From the workbench, click on **File—>Quick Start**.
2. Click on **Features—>Add Feature—>OK**.
3. Select **IBM Enterprise Toolkit for AS/400—>OK**.

This adds the toolbox classes to your workspace. The IBM Enterprise Toolkit for AS400 is listed under All projects.

The alternative is to perform the following steps:

1. Install LPP 5763-JC1 on an AS/400 system.
2. Download the classes to your workstation.
3. Import the classes into the VisualAge for Java IDE.

A.2.2 IBM Enterprise Data Access Libraries

The example programs use these supporting classes. To add them, perform these tasks:

1. From the workbench, click on **File—>Quick Start**.
2. Click on **Features—>Add Feature—>OK**.
3. Select **IBM Enterprise Data Access Libraries—>OK**.

A.2.3 Sun JSDK Class Libraries

The example programs use these supporting classes. To add them, perform these steps:

1. Select **Selected** from the Workbench menu.
2. Select **Add—>Project** from the pulldown menu.
3. Click on the **Add projects** from the repository radio button.
4. Select the **Sun JSDK class libraries project**.
5. Click on the **Finish** button.

A.2.4 Netscape Security

The example programs use these supporting classes. To add them:

1. Select **Selected** from the Workbench menu.
2. Select **Add—>Project** from the pulldown menu.
3. Click on the **Add projects** from the repository radio button.
4. Select the **Netscape Security project**.
5. Click on the **Finish** button.

Appendix B. IBM HTTP Server for AS400 Configuration

This appendix contains the configuration file used for the IBM HTTP Server for AS/400. The name of the configuration file is JAVAS. It uses port 1040.

```
# * * * * * * * * * * * * * * * * * * * * * * * * * * *
#               IBM HTTP Server for AS/400
# * * * * * * * * * * * * * * * * * * * * * * * * * * *
#----------------------------------------------------
#             *** PORT DIRECTIVES ***
#
# The default port for HTTP is 80.  If you change this
# use a port number greater than 1024.
#
#
# Syntax:
#   Port                     <port number>
#----------------------------------------------------
Port                                **1040**
Pass /  /QIBM/ProdData/HTTP/Public/HTTPSVR/HTML/Welcome.html
Pass /sample/*  /QIBM/ProdData/HTTP/Public/HTTPSVR/HTML/*
Service       /servlet/*  /QSYS.LIB/QHTTPSVR.LIB/QZHJSVLT.SRVPGM:AdapterService*
Service       /*.jhtml  /QSYS.LIB/QHTTPSVR.LIB/QZHJSVLT.SRVPGM:AdapterService
Service       /*.jsp  /QSYS.LIB/QHTTPSVR.LIB/QZHJSVLT.SRVPGM:AdapterService
Map       /netdata/*  /QSYS.LIB/A980501A.LIB/DB2WWW.PGM/*
IconPath /QIBM/HTTPSVR/Icons/
AddIcon text.gif     text  text/*
AddIcon html.gif     html  text/html
AddIcon binary.gif   bin   application/*
AddIcon compress.gif Z     application/x-compress
AddIcon compress.gif gzip  application/x-gzip
AddIcon image.gif    img   image/*
AddIcon movie.gif    vid   video/*
AddIcon sound.gif    au    audio/*
#----------------------------------------------------
#                  *** AddType ***
#
# To bind files with a particular suffix to a MIME
# type/subtype, use AddType.  Multiple occurrences
# are allowed.
AddType .java text/plain binary 1.0
AddType .html text/html  8bit   1.0
AddType .htm  text/html  8bit   1.0
AddType .gif  image/gif  binary
Map       /IBMWebAS/samples/*  /QIBM/ProdData/IBMWebAS/samples/*
Map       /IBMWebAS/doc/*  /QIBM/ProdData/IBMWebAS/doc/*
Map       /IBMWebAS/system/admin/*  /QIBM/ProdData/IBMWebAS/system/admin/*
Map       /IBMWebAS/*  /QIBM/ProdData/IBMWebAS/web/*
Pass      /apptest/*
Pass      /applets/*  /Qibm/proddata/ibmwebas/servlets/applets/*
Exec       /QSYS.LIB/A980501A.LIB/*
ServerInit /QSYS.LIB/QHTTPSVR.LIB/QZHJSVLT.SRVPGM:AdapterInit
         /QIBM/ProdData/IBMWebAS/properties/server/servlet/servletservice/jvm.properties
ServerTerm /QSYS.LIB/QHTTPSVR.LIB/QZHJSVLT.SRVPGM:AdapterExit
Enable       GET
Enable       HEAD
Enable       POST
Enable       OPTIONS
Enable       TRACE
Disable       CONNECT
normalmode       On
sslmode       On
sslport       443
SSLClientAuth       Off
keyfile /QIBM/USERDATA/ICSS/CERT/SERVER/DEFAULT.KYR
```

Appendix C. Special Notices

This publication is intended to help anyone who want to build AS/400 Internet based applications with Java. The information in this publication is not intended as the specification of any programming interfaces that are provided by VisualAge for Java or the AS/400 Toolbox for Java. See the PUBLICATIONS section of the IBM Programming Announcement for VisualAge for Java and the AS/400 Toolbox for Java for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AFP	IBM ®
AIX	OS/390
AS/400	OS/2
Client Access	OS/400
DB2	VisualAge
	400

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

Java and HotJava are trademarks of Sun Microsystems, Incorporated.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

Pentium, MMX, ProShare, LANDesk, and ActionMedia are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

Appendix D. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

D.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How IBM Employees Can Get ITSO Redbooks” on page 307.

- *Building AS/400 Applications with Java*, SG24-2163
- *Application Development with VisualAge for Java Enterprise*, SG24-5081
- *Securing Your AS/400 from Harm on the Internet,* SG24-4929

D.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
Lotus Redbooks Collection	SBOF-6899	SK2T-8039
Tivoli Redbooks Collection	SBOF-6898	SK2T-8044
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
RS/6000 Redbooks Collection (PDF Format)	SBOF-8700	SK2T-8043
Application Development Redbooks Collection	SBOF-7290	SK2T-8037

D.3 Other Publications

These publications are also relevant as further information sources:

- *HTTP Server for AS/400 Quick Beginnings*, GC41-5433
- *HTTP Server Webmaster’s Guide*, GC41-5434
- *Java in a Nutshell*, ISBN 1-56592-183-6
- *Java Developer's Reference*, ISBN 1-57521-129-7
- *Object Oriented Technology: A Manager's Guide*, ISBN 0-201-56358-4

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at `http://www.redbooks.ibm.com/`.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Redbooks Web Site on the World Wide Web**

  `http://w3.itso.ibm.com/`

- **PUBORDER** – to order hardcopies in the United States
- **Tools Disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLCAT REDPRINT
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get BookManager BOOKs of redbooks, type the following command:

  ```
  TOOLCAT REDBOOKS
  ```

  To get lists of redbooks, type the following command:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  ```

  To register for information on workshops, residencies, and redbooks, type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1998
  ```

- **REDBOOKS Category on INEWS**
- **Online** – send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (`http://www.redbooks.ibm.com/redpieces.html`). 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.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** – send orders to:

	IBMMAIL	**Internet**
In United States	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** – send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29570	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **Fax** – send orders to:

United States (toll free)	1-800-445-9269
Canada	1-800-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1) 408 256 5422 (Outside USA)** – ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **On the World Wide Web**

Redbooks Web Site	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (`http://www.redbooks.ibm.com/redpieces.html`). 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.

IBM Redbook Order Form

Please send me the following:

Title	**Order Number**	**Quantity**

First name	Last name	
Company		
Address		
City	Postal code	Country
Telephone number	Telefax number	VAT number

☐ Invoice to customer number

☐ Credit card number

Credit card expiration date	Card issued to	Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

List of Abbreviations

AFP	advanced function printing
APA	all points addressable
AWT	Abstract Windowing Toolkit
CPW	Commercial Processing Workload
EAB	Enterprise Access Builder
DAX	Data Access Builder
DDM	Distributed Data Management
DPC	Distributed Program Call
FFST	First Failure Support Technology
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IBM	International Business Machines Corporation
IDE	Integrated Development Environment
ITSO	International Technical Support Organization
JAR	Java archive
JDBC	Java Database Connectivity
JDK	Java Development Toolkit
JFC	Java Foundation Classes
JIT	Just in Time Compiler
JVM	Java Virtual Machine
MI	Machine Interface
OOA	Object Oriented Analysis
OOD	Object Oriented Design
OOP	Object Oriented Programming
PTF	Program Temporary Fix
RAD	Rapid Application Development
RMI	Remote Method Invocation
SCS	SNA Character Set
SLIC	System Licensed Internal Code
SSL	secure sockets layer
TIMI	Technology Independent Machine Interface
UML	Unified Methodology Language
URL	Universal Resource Locator
VCE	Visual Composition Editor
WWW	World Wide Web

Index

ITSO Redbook Evaluation

Building AS/400 Internet-Based Applications with Java
SG24-5337-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.ibm.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?
_ Customer _ Business Partner _ Solution Developer _ IBM employee
_ None of the above

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction __________

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

What other redbooks would you like to see published?

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

SG24-5337-00

Printed in the U.S.A.

Building AS/400 Internet-Based Applications with Java

SG24-5337-00