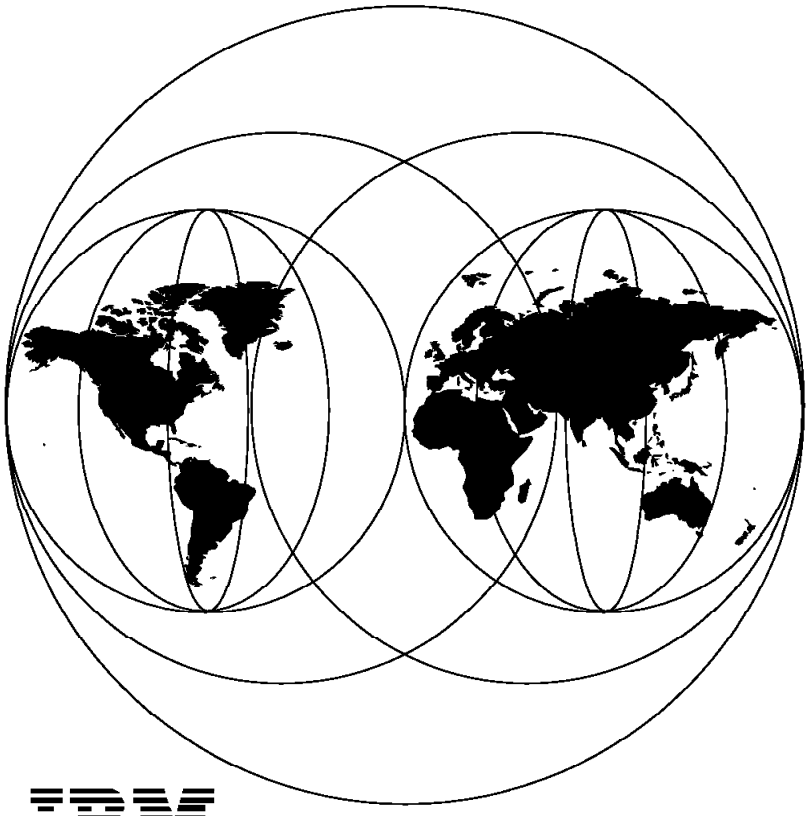
SG24-4935-00

Unleashing AS/400 Applications on the Internet

June 1997



IBM

International Technical Support Organization
Rochester Center

International Technical Support Organization

SG24-4935-00

Unleashing AS/400 Applications on the Internet

June 1997

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 275.

First Edition (June 1997)

This edition applies to Version 3 Release 2 and Version 3 Release 7 of the Operating System/400 (5763-SS1) and (5716-SS1).

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

Contents

Figures

Tables

Preface

This redbook is intended to give directions to enable an AS/400 application to the Internet. It gives you the necessary information to implement and run the Internet-related AS/400 applications using one or a combination of the following available techniques:

- Common Gateway Interface (CGI-Bin)
- Net.Data
- Workstation Gateway
- Java Applets and Java Scripts

It is also the intent of this redbook to provide you with a road map of the AS/400 application development scenario for the Internet and performance considerations.

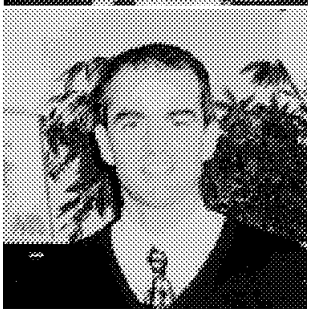
This redbook was written for technical services and consultants who are charged with the task of recommending and implementing an AS/400 network computing environment, and for the IBM, Business Partner and customer technical professional community.

Some knowledge of the AS/400 system, application development, and Internet programming is assumed.

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.

Ana Vallano Romero is a Systems Programmer working for Ecinsa, an IBM Business Partner in Madrid, Spain. She holds a degree and diploma in Computer Science. Her areas of expertise include Internet, CGI-Bin, and C language programming. She has more than 3 years of C programming and technical support experience with the IBM RS/6000 systems.



Jan P. Collins is a Systems Consultant working for Datalink International, an IBM Business Partner in Guernsey, England. During the past year, he has been actively involved in integrating his company's existing AS/400 applications with the Internet. He holds a degree and diploma of engineering in Naval Architecture from England, but his most recent areas of expertise include Internet, TCP/IP, Windows 95, HTML, and Java. He has more than 15 years of scientific programming and technical report writing.

Paul E. Hampton is an AS/400 Evangelist in the USA. He has 20 years of experience working with IBM customers and clients with S/36, S/38, and AS/400 Communications and Performance. During the last two years, he has helped several clients plan and execute successful AS/400 intranets and Internets.

Gottfried Schimunek is an IBM Chief Systems Engineer for the AS/400 system in Germany. He has more than 25 years of experience in the area of DP and IT systems, especially on IBM System/3x, System/38, and the AS/400 system. His areas of interest and expertise include AS/400 Architecture, System and Application Performance, and Internet Connection. He coauthored books about AS/400 Performance Management, Capacity Planning, Client/Server Performance, and SAP R/3.

Fernando R. Zuliani is an AS/400 Certified I/T Specialist in the ITSO Rochester. He has 10 years of experience in the I/T field. He has worked at IBM for 9 years. His areas of expertise include OS/400, Client/Server, VisualAge Generator, Application Development, and Performance. He holds a master's degree in Mathematics from the University of Sao Paulo, Brazil.

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

Allan E. Johnson
Systems Performance - AS/400 Development, IBM Rochester

Bill C. Rapp
Internet Technologies - AS/400 Development, IBM Rochester

Brian R. Smith
AS/400 Communications, ITSO Rochester Center

Daryl E. Woker
Internet Security and Firewalls - AS/400 Development, IBM Rochester

David D. Money
Network Computing - AS/400 Partners in Development, IBM Rochester

Jason Hunt
Network Computing - AS/400 Partners in Development, IBM Rochester

Jim J. Herring
Internet Technologies - AS/400 Development, IBM Rochester

Jose Carlos Duarte Goncalves
Network Computing, IBM Sao Paulo, Brazil

Marcela Adan
Internet Security and Firewalls, ITSO Rochester Center

Mark A. McKelvey
Internet Security and Firewalls - AS/400 Development, IBM Rochester

Mel L. Rothman
CGI-Bin Programming - AS/400 Customer Technology Center, IBM Rochester

Mike W. Schroeder
Net.Data/Internet - AS/400 Development, IBM Rochester

Nadir K. Amra
Net.Data/Internet - AS/400 Development, IBM Rochester

Paul D. Brown
Network Computing - AS/400 Partners in Development, IBM Rochester

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 285 to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Home Pages at the following URLs:

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

- Send us a note at the following address:

  `redbook@vnet.ibm.com`

Chapter 1. AS/400® System in Network Computing Environment

This redbook is intended to give directions to enable an AS/400 application to the Internet. It gives you the necessary information to implement and run the Internet-related AS/400 applications using one or a combination of the following available techniques:

- Common Gateway Interface (CGI-BIN)
- Net.Data
- Workstation Gateway
- JAVA Applets and JAVA Scripts

It is also the intent of this redbook to provide you with a road map of the AS/400 application developing scenario for the Internet and performance considerations.

This book, the same as other IBM redbooks, mentions related areas such as IBM offerings and AS/400 solutions.

As you may have seen, most product or solutions providers are offering many kinds of Internet services. As well as an Internet connection for the AS/400 system, IBM offers:

- Client and server software
- Firewalls
- IBM networking services
- IBM information super highway solutions
- IBM global network

In this chapter, we talk about:

Topic	Please see
What is the Internet	See Section 1.1, “What is the Internet” for more information.
History of the Internet	See Section 1.2, “History of Internet” on page 2 for more information.
Network Computing	See Section 1.3, “Network Computing” on page 3 for more information.
Why AS/400?	See Section 1.4, “Why Attach an AS/400 System to the Internet?” on page 4 for more information.

1.1 What is the Internet

The Internet is *the* network of networks. It is an interconnection of many smaller networks forming a larger network we call the Internet. As the number of hosts and other networks connecting to the Internet grow, so does the Internet.

Since the Internet′s conception, applications and protocols have been developed to utilize it. The popularity of applications such as FTP and e-mail caused a meteoric rise in the number of users connecting to the Internet. More recently, however, the World Wide Web is the area of usage showing the most rapid growth.

The original backbone of the Internet was provided by the U.S. Government, science institutes, and universities as a means of sharing research data. As usage has outgrown the original infrastructure, it has been continually upgraded by communications companies funded by the subscriptions of the end users.

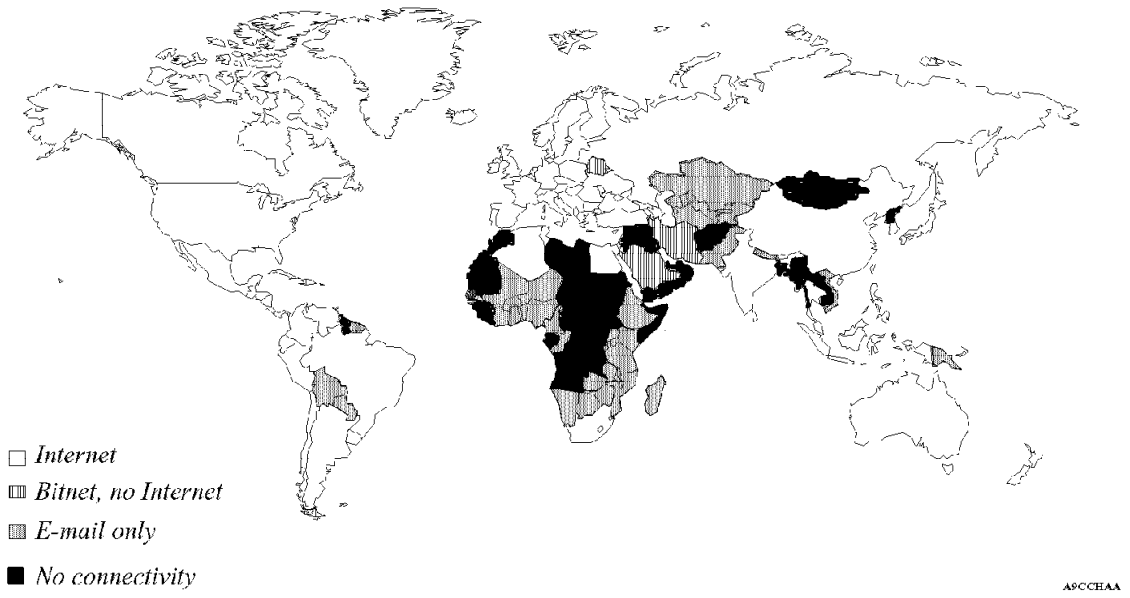


Figure 1. Internet World Map

This picture shows all of the current countries in the world and indicates by different shades of black through white the level of Internet access. White indicates full Internet access, which makes up most of this world map.

1.2 History of Internet

The Internet was born in the late 1960s as an effort to create a United States Department of Defense network called ARPANET. The main purpose for it was to connect defense contractors and universities that were working on projects for the Department of Defense.

In 1973, the ARPANET established connections to England and Norway, and in 1982, the TCP/IP became the protocol suite for ARPANET. This leads to one of the first references to an ″Internet″ of connected networks.

In 1983, the University of Wisconsin developed the name server that was introduced in 1984.

In 1986, NFSNET was created with a backbone speed of 56 Kbps and upgraded to T1 (1.544 Mbps) in 1989.

In 1989, the AS/400 system started supporting TCP/IP on Release 2.0 of OS/400® . This was even before we knew that this was version 1!

Gopher was introduced by the University of Minnesota in 1991. WWW (World Wide Web) was released by CERN in 1992 adding to established and popular applications such as Telnet, e-mail, and FTP.

Today, we have an estimated 4 000 000 hosts connected to the Internet and many more connected to internal ″intranets″. Network providers have upgraded the speeds of their backbones to T1, T2, T3, and so on to cope with the rising demand for bandwidth.

Now, with the Internet on the verge of becoming a huge commercial marketplace, the challenge is to create a suitable environment for secure electronic commerce.

Here are some interesting Internet facts:

- U.S.A. corporations establish a new site on the Internet every 12 minutes - Internet Society.
- By early 1996, there were 170 000 commercial sites on the Internet worldwide - InterNIC.
- U.S.A. shoppers purchased 436 million U.S. dollars worth of goods and services over the World Wide Web in 1995 - ActivMedia.
- In 1996, 81% of the top 200 U.K. companies viewed the Internet as a business opportunity with more than one-third already having an Internet site - Barclays Bank.
- In Asia, Europe, and the U.S., about 17.6 million people are expected to be connected to the Internet by the end of 1996, up from 10.6 million in 1995 - Forrester Research.
- Fifty years after the invention of the telephone, world residents had made a total of 38 million phone calls. Today on the Internet, around 38 million page visits occur every 24 hours - SRI International.

1.3 Network Computing

The latest stage in the computing industry′s evolution is the client/server model in which end users (clients) are able to use the services of multiple host systems (servers). These customers require highly effective products, services, and solutions to help integrate all of the legacy data from distributed systems. This takes us to the Network Computing (NC) model.

An example of the Network Computing model can be:

A single AS/400 system accessing vast amounts of data, stored on many different systems across a large network as if it were stored on a single entity. End users perceive information, applications, and services as being provided by the network rather than by a computer system. IBM is committed to Network Computing on the AS/400 system, a system that has the advantages of an integrated database, integrated security, and an object-based structure.

The AS/400 users can utilize the latest technology, reduce or eliminate many geographic barriers, exploit the Internet to their companies, and enable new ways of doing business.

Internet Connection for AS/400, which is an integrated package included with OS/400, helps customers conduct business on the Internet. It includes:

- World Wide Web Hypertext Transfer Protocol (HTTP) server
- Workstation Gateway (WSG) server
- Net.Data
- Logging of World Wide Web server access

TCP/IP has also been enhanced to provide:

- FTP support across all of the Integrated File System (IFS).
- FTP authentication exits points (Anonymous FTP).
- Asynchronous communications support in the form of SLIP.

Reference: URL http://www.as400.ibm.com

1.4 Why Attach an AS/400 System to the Internet?

Why not? If you are seriously considering putting a server onto the Internet, there are many distinct advantages in choosing an AS/400 system. If you have concerns about using an AS/400 system in this role, ask yourself whether the reason for your concern (such as security) is associated with the Internet itself rather than a specific system.

Consider the advantages of using an AS/400 system:

Communications An AS/400 system has exceptional communication capabilities. LAN, WAN, and dial-up capability are all part and parcel of OS/400 providing a choice of connection methods using TCP/IP.

Scaleability The common features present in OS/400 across the entire model range make it easy to accommodate future growth.

Functionality The AS/400 system provides the popular TCP/IP applications such as Telnet, FTP, e-mail, WWW server, and so on.

Software cost Almost all the AS/400 TCP/IP suite of applications comes free of charge when you order OS/400.

Training If you are already running an AS/400 operation, the training required to attach your AS/400 system successfully to the Internet is minimal.

Security The AS/400 system has extremely good, native, security features. These features alone can prevent many Internet-related security problems that afflict other systems. With user-exit programs, you can further enhance the security for FTP and Workstation Gateway servers.

Legacy Data Perhaps the main reason to consider an AS/400 system as an Internet server is when the information you want to serve is already resident on an AS/400 system. The new Internet Connection for AS/400 applications provide several methods of accessing your legacy data through the Internet. By using the Workstation Gateway server, most of your existing ″green-screen″ applications are Internet enabled now. With Net.Data, you can access your legacy data from WWW pages. With CGI-BIN programs, you can build new applications to be used from WWW pages.

Chapter 2. Internet Application Design Considerations

Millions of people are now connected to each other through the Internet, creating a magnificent spectrum of new application possibilities. This topic discusses the AS/400 specific Internet application capabilities.

2.1.1 Network-Centric Computing

The Internet is becoming more and more popular. Every day thousands of new users are entering cyberspace. They all are looking for, using, and exchanging information on the many servers that form the backbone of the Internet. The World-Wide Web (WWW) gives these users a single interface to these servers and so offers easy access to an overwhelming amount of information about almost everything on this planet.

However, the information presented on the pages of the WWW used to be essentially static. Using a Web browser, a user could access the information on the Web but could not change or manipulate it in real time. This changes with the availability of AS/400 Internet Connection (CGI-Programming, Net.Data, and Workstation Gateway). (We cover these functions in great detail in the following chapters.)

This will again change in the future when small programs, called applets, are linked from Web pages, downloaded to the Web browser and executed on the client machine. The technology that makes this possible and secure is the Java™ run-time environment designed for this purpose by Sun Microsystems. IBM has licensed the technology from Sun Microsystems and will use it on its platforms; beta versions of the Java toolkit for OS/2®, AIX®, and others are already available on the Internet (see:

`http://ncc.hursley.ibm.com/javainfo`

Java applets can be written in any language that can be compiled to the Java instruction codes. One such language is called Java, a simplified derivative of C++. IBM is actively investigating alternatives to the Java language for programming the Java environment. For more detailed information, see Chapter 6, “Further Enhancing Your AS/400 HTML Pages” on page 221.

2.2 AS/400 Internet Application Scenarios

When discussing AS/400 systems and the Internet, one needs to consider the various roles the system may play since they affect the design and technology requirements for an application.

- **Public Internet Presence**

  The AS/400 system can be a public server (for example, Web server) that is accessed directly by users from the Internet. The users that access the system are often not known in advance. Some services are generally open to the general public while others are reserved for authorized users.

- **Internet Accessible**

  The AS/400 system is a private system that is accessible by authorized users from the Internet; the Internet merely provides a convenient means for access.

- **Internet Access**

  A private AS/400 system accesses servers attached to the Internet (for example, AS/400 users FTP to Internet servers to obtain software fixes).

- **Intranet**

  A private AS/400 system is part of an internal corporate network that uses Internet-based protocols.

- **Extranet**

  Networks that extend beyond a single company to multiple organizations that must collaborate, communicate, and exchange documents to achieve joint goals. These networks take a company's existing intranet and extend it beyond the enterprise to enable collaborative business applications such as electronic commerce and supply-chain management or between one company and its customers and suppliers.

A given AS/400 system may take on multiple roles at the same time.

2.3 Different Web Models

Within each Net previously described, there are four distinct and different ways to present or to interact with the Web user:

2.3.1 Simple Web Model

The simple Web model consists of publishing static Web pages using the Hypertext Markup Language (HTML). The Web browser opens up a connection to the Web server and the server returns a page and closes the connection. This model is suitable for providing access to business information that is relatively stable. It originally provided the foundation for the Web. Although later enriched by extensions to HTML (such as tables and frames), the simple Web model remains a static model. The user's only interactive options are limited to clicking on a link to visit another page or requesting a page reload.

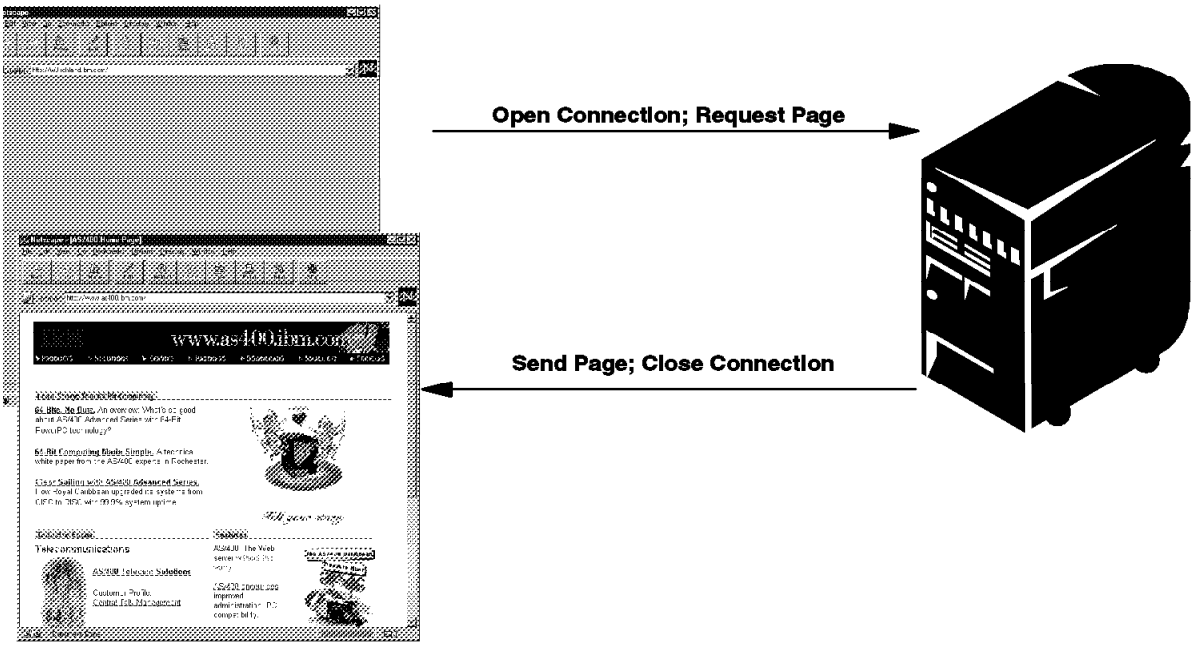


Figure 2. The Simple Web Model

2.3.2 Interactive Web Model

In the interactive Web model, pages can contain forms, fields, and buttons that allow users to enter data and choices. When they complete the forms and make their choices, the browser opens a connection with the server to allow the data and choices to be transmitted. The Web server passes the information to a custom server program or script that makes an inquiry or does calculations and passes back a new page for the browser to display. The connection is closed.

The interactive Web model supports a simple form of client/server computing using Hypertext Transfer Protocol (HTTP) as the middleware and the Web server's Common Gateway Interface (CGI) that calls a custom server program or script. This model works well for simple interactions that do not require high inter-activity between the client and server. However, it is quite expensive in terms of server resources and time when establishing a connection between the browser and the server for each interaction.

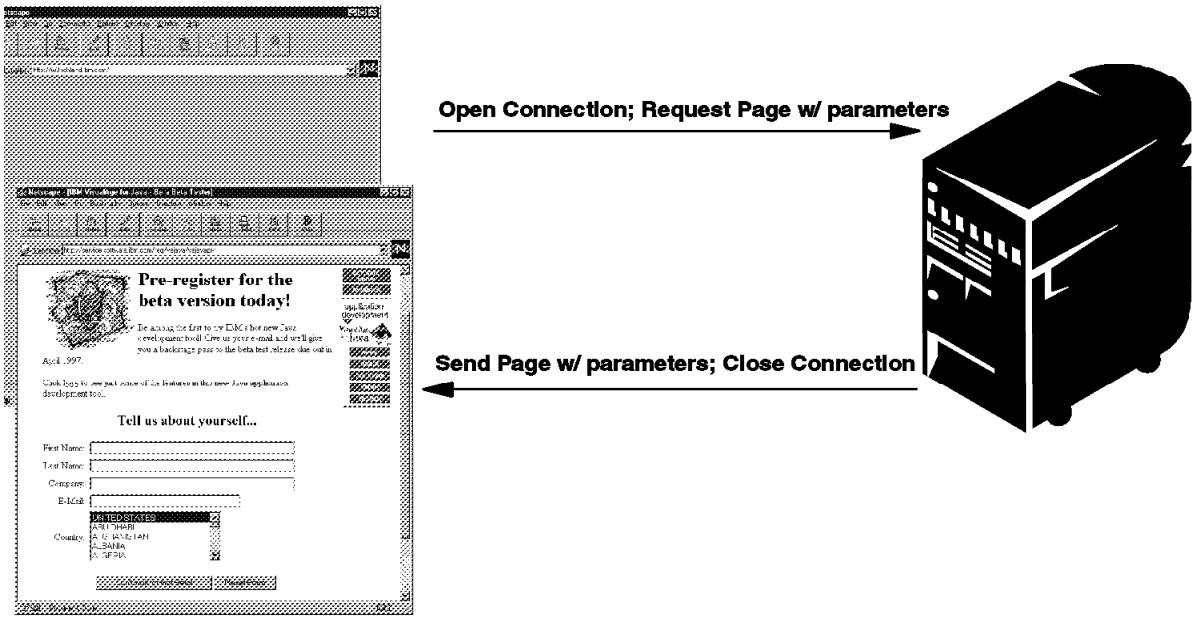


Figure 3. The Interactive Web Model

2.3.3 Distributed Web Model

The distributed Web, also known as Internet PC, introduces Java. With Java, a small application called an applet can be transmitted to the browser along with the Web page. The Java applet runs within the browser as users view the page. It can provide lively animation or sound as well as rich graphical user interface components such as the ones usually found on windowed PC applications. The applet can display a window or be displayed in a Web browser; it can also process user input. If server interaction is needed, the applet can open a connection to the Web server to access a server or script through the Web server's CGI. The distributed Web model enriches the user interface and off-loads tasks from the Web server (such as parameter checking in the browser), but it is still tied to HTTP for communications. The next section describes another model that supports additional communication protocols.

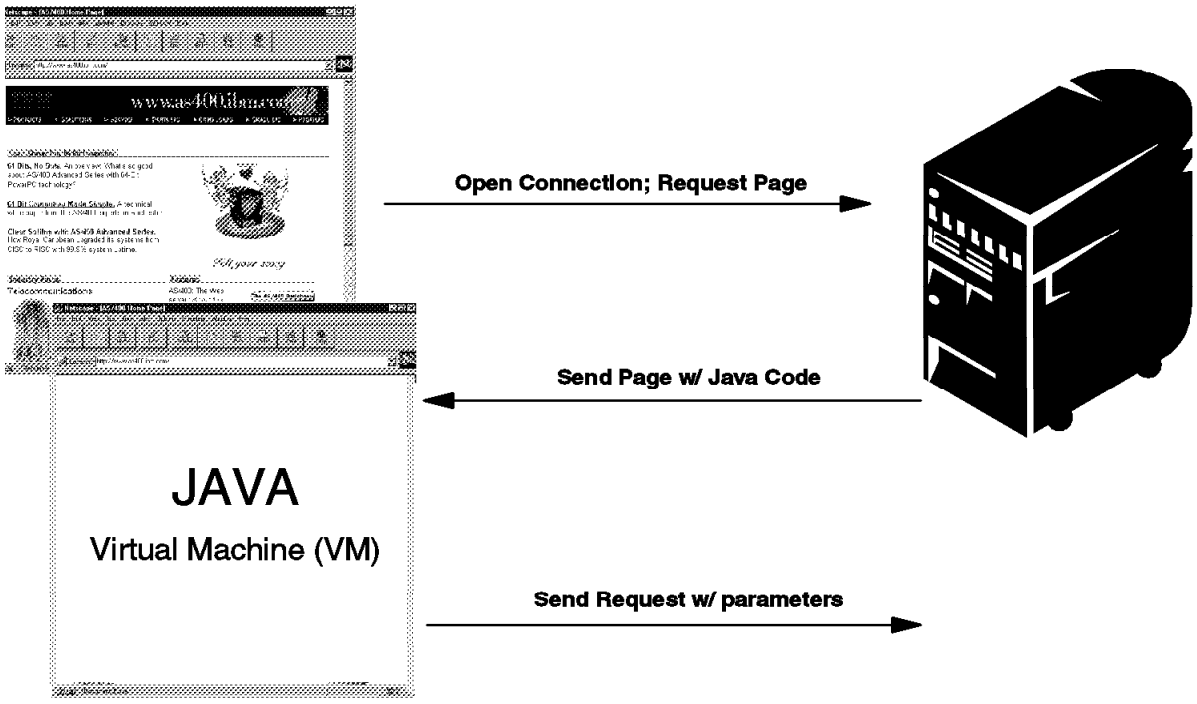


Figure 4. The Distributed Web Model

2.3.4 Enterprise Distributed Web Model

An additional application model that IBM intends to support is called the Enterprise Distributed Web model. Enterprise Distributed Web applications are true open client/server applications. The client is coded in downloadable Java, which runs in any Java-capable browser. A page containing the Java applet is downloaded by a Web server to a browser. While the applet runs in the browser, it opens its own communications session to the server. The applet also communicates with a server to provide access to the database, C++, Java application servers, or SOM™ objects, in addition to supporting a variety of middleware, include HTTP, TCP/IP, Secure Sockets Layer (SSL), DSOM, and MQ.

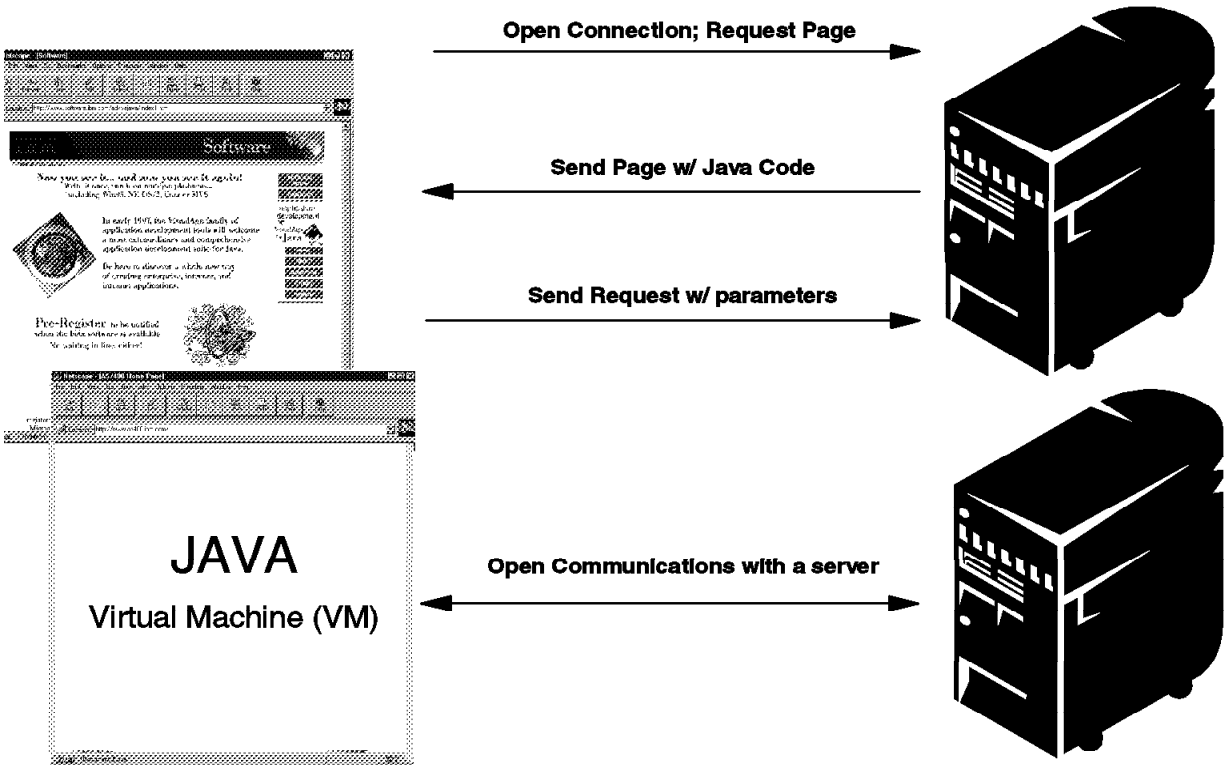


Figure 5. The Enterprise Distributed Web Model

2.4 AS/400 System Facilities for the Internet

This section gives an overview of the Internet-related services on the AS/400 system.

2.4.1 Database Services

The AS/400 database is accessible to Internet users through CGI programs and the Net.Data product. Net.Data uses a modified type of HTML file called a macro that includes HTML script as well as directives that allow dynamic documents to be built. These directives can be SQL statements, program invocations, include statements, and variable substitutions. Local and remote AS/400 databases can be accessed from a Net.Data macro. For more information, refer to the Net.Data description in Chapter 4, “Net.Data Implementation” on page 111.

2.4.2 Application Access

Existing (legacy) applications can be accessed using the 5250 Workstation Gateway function.

New internet applications can be written using HTTP server APIs, the CGI interface, or using the Net.Data product. HTTP server API programs can be written in the C programming language. CGI programs can be written in C, RPG,

and COBOL programming languages. Net.Data applications are written in a macro language that can reference AS/400 resources including the database and programs written in a variety of AS/400 languages (C, C++, COBOL, RPG, COBOL, REXX, and PERL).

All of these functions are available as part of OS/400.

Lotus Domino is another way of serving up Internet applications. Domino allows applications to be built using the rich application development capabilities of Lotus Notes. Domino allows existing Notes applications to be served up over the Web.

PERL has been ported to the AS/400 system by Mortice Kern Systems Inc. (MKS). More information about MKS can be found on the Web under the URL:

`http://www.mks.com`

This product was put out as a product preview option in December 1996.

2.4.3 Content Authoring

Internet content authoring for the AS/400 system is typically done using a PC editor or graphical tool. The content (HTML text, Net.Data macros, Java applets, images, audio, video, and so on) is created and tested on the PC and uploaded to the AS/400 IFS where it can be served to Internet clients.

2.4.4 Content Management

Internet content management deals with the difficult job that a Web administrator has of managing Web objects including HTML scripts, macros, CGI programs, Java applets, audio/video clips, bitmaps, and the links that tie them all together. It also deals with being able to determine how a Web site is navigated through navigation analysis tools.

A technology demo for a management tool (Visual Web Manager) is available at IBM on the Internet. Visual Web Manager allows you to view the structure of your Web site and HTML presentations and make modifications on the fly. Visual Web Manager gives you a simple visual environment in which you can develop and maintain your Web site.

The demo version of the software is suitable for small-to-medium sized Web sites that contain a maximum of 500 HTML pages. The software runs locally on your machine (which can be your server or any other machine where you develop a Web site or presentation) and operates on Web sites where the HTML pages are contained within the directory containing the home page or subdirectories under this directory.

The best way to become familiar with the software is to run it on the demo presentation that is supplied with the software. See the instructions in the README for running the demo presentation. For more information and to download a Beta version of an Internet Visual Web Management Tool, see:

`http://www.ics.raleigh.ibm.com/ics/icfbetas.htm`

2.4.5 Java Development Environment

VisualAge® Java for the AS/400 system includes a suite of visual builders for interactively building Java applications on client machines including OS/2 and Win95/NT clients. It includes an integrated development environment with a project manager, class browser, editor, and debugger. It also includes a set of remote debug classes that allow a Java application written on the AS/400 system to be debugged from a client machine. Other OEM tools can also be used to build AS/400 Java applications.

2.5 General Design Considerations

There are some general design considerations to be taken into account when writing or modifying applications for the Internet. The following sections cover some of the major design considerations and differences to native AS/400 application design.

2.5.1 Transactions over the World Wide Web using the Internet

Running an AS/400 application over the Internet can be quite different to what applications do in a ″normal″ environment. Since there is no constant connection between the server and the client, there is no transaction flow that is controlled from the server. The user typically gets a window (HTML), looks at it, changes some fields, and sends it back.

The user might also decide after receiving the data to do something else and not do anything with the data. So the server program cannot rely on the fact that since it sent some data, it is suppose to get some data back.

There is also no control over the arrival of the data; the data might get stuck on the wire and never see the end user. The server does not recognize it.

2.5.2 Synchronization between Web Browser and Server

This section discus how the requests are synchronized between the Web browsers and the Web server.

2.5.2.1 Sending Multiple Requests

Most of the applications on the AS/400 system do not allow you to submit a further request while another request is currently processing. This is due to the fact that during this time, the application is not sensitive for user input. With a Web browser, this is not the case. A user is allowed to request a new document or post data while another request is currently processed by a server. For example, this may happen when a user unintentionally double-clicks a button. Usually the second request causes the browser to discard the answer to the first one.

2.5.2.2 History Mechanism and Caching

Also with a native application, it is not possible to switch to another window without server communication. Browsers often have history mechanisms such as ″Back″ buttons and a history list that can be used to re-display a page from the cache retrieval earlier in the session. A history mechanism is meant to show exactly what the user saw at the time when the resource was retrieved on condition that the resource is still in the cache.

2.5.2.3 Transaction Context

In the beginning, browsers were used to retrieve inter-related documents. No context was kept by the HTTP server related to a user session. Here it made sense to switch back to another page without communication to a server even if the page was outdated. But to re-display a page from a cache does not make sense for transactions. Here a page should reflect the current transaction state. With transactions, a page also contains forms for posting data to a server in contrast to links that are (usually) used to request other documents. These forms make sense only in a certain context.

2.5.2.4 GET and POST

The GET method is always used in the HTTP request for links. For forms, the method is determined by the form attribute ″method″, which should be set to the POST method.

2.5.2.5 Caching Strategies for GET and POST

When using history mechanisms, the Web browser always tries to retrieve the page from cache. But when submitting a form or selecting a link, we must differentiate between the POST and GET method. When resubmitting a form with a POST method, the Web browser never tries to retrieve the page from cache even if the posted data is the same. Here the request is always submitted to the server in contrast to pages retrieved with a GET method.

2.5.2.6 Caching Control

It is possible for the server to request no caching for a page. Here the field ″expires″ (which is set to 0 in the HTTP header) and prevents caching. It is also possible to request no caching for an HTML document by using:

```
<meta http-equiv=″expires″ content=″0″>
```

in the header part of the HTML document.

2.5.3 End User Driven

Also different is the way the application is being driven. On the Internet, the end user determines the flow of the application. This is similar to a client/server application with a GUI front end where the user picks and chooses icons and menu items not controlled by the application. The user might decide to request some data from one server, request different data from the same server, data from different servers, and finally after hours, fills the first window and returns it to the first server.

To always have complete transactions, the server application has to send a simple, complete transaction on one request, or has to keep track of each user′s status. Most AS/400 applications today presume that the calling user stays with the application until it finishes it. Most of the transactions span several windows before a customer transaction is complete.

2.5.4 End User Anonymous

On the Internet (or intranet, for that matter) end users are anonymous, or unknown to the application when they request information from the server for the first time unless one of the first requests contain sign-on information such as USERID and password. The nature of how a user communicates to the server is again different than what we currently have in a typical AS/400 environment.

2.5.5 Status of User Requests

In a native AS/400 environment, there is a one-to-one relationship between one user and an invocation of the application program (one job per user). Also, it is quite common for an application that needs more than one transaction to the user to complete one business transaction. Because it is always the same user talking to the same program invocation, it is relatively easy to keep track of what the user did and what the next steps should be.

In the Internet server environment, however, each dialog is separate from each other and it is probably going to use another server job. So, there is no information being kept on the server regarding the client.

To keep track of a particular user and what it has processed so far, you can use what is called a **cookie** or hidden fields, a feature inside the browser. The following sections describe how cookies and hidden fields can be used.

2.5.5.1 How Cookies Work (Client Side Persistent Information)

A ″cookie″ is a small piece of information that a Web server can ask the browser to store temporarily.

Note

Not all browsers support ″cookies″ so it violates the rule of being a browser independent application.

A program can test for HTTP_USER_AGENT and use ″cookies″ if supported, or else use hidden fields or other available mechanisms.

This is useful by having the browser remember some specific information that the Web server can later retrieve. For example, when you browse through a ″virtual shopping mall″ and add items to your ″shopping cart″ as you browse, a list of the items you pick up is stored by your browser so that you can pay for all of the items at once when you are finished shopping. It is much more efficient for each browser to keep track of information instead of expecting the Web server to remember which browser (of the thousands of browsers that might be using it at a time) bought what.

As you browse the Web, cookies sent to your browser are stored in memory. When you quit the browser, any cookies that have not expired are written to a cookie file so they can be reloaded the next time you run the browser. (On a Mac, this file is named ″MagicCookie″; on UNIX®, it is ″cookies″, and on Windows™, it is ″cookies.txt″. Contrary to popular belief, a cookie file is **not** a secret way for a Web server to find out everything about you and what you have on your hard drive. The only way that any private information is in your cookie file is if you personally gave that information to a Web server in the first place and it decides to put that information into your cookie file for some reason. There is absolutely no way for a Web server to get access to any private information about your system through cookies.

To create a cookie, a Web server sends a ″Set-Cookie″ HTTP header line such as this one in response to a URL access from a browser:

```
Set-Cookie: NAME=VALUE; expires=DATE; path=PATH; domain=DOMAIN_NAME;
```

NAME is the name of the value you are storing on the browser, and VALUE is the actual data being stored in the cookie. DATE is the date and time at which the cookie information expires and is ″forgotten″ by the browser (and removed from the cookies file if it was written there). DOMAIN is a host or domain name for which the cookie is valid (servers outside of that domain can never see this particular cookie for security reasons). PATH specifies a subset of the URLs at that server for which the cookie is valid (other URLs do not see the cookie).

The only required field is NAME=VALUE.

Whenever the browser sends an HTTP request for a URL on a server where it has stored cookies (as long as DOMAIN and PATH match the URL), it includes a line of the form:

```
Cookie: NAME=VALUE; NAME=VALUE; ...
```

A complete technical explanation of how cookies work is available on

```
http://home.netscape.com/newsref/std/cookie_spec.html
```

2.5.5.2 How Hidden Fields are Used

Hidden variables are used to conceal the actual name of a variable from people who choose to view your HTML source with their Web browser.

Type=″hidden″ defines an invisible input field whose value is sent along with the other form values when the form is submitted. This is used to pass state information from one script or form to another:

```
Hidden Field:  INPUT TYPE=HIDDEN
```

An INPUT element with ″TYPE=HIDDEN″ represents a hidden field. The user does not interact with this field; instead, the VALUE attribute specifies the value of the field. The NAME and VALUE attributes are required.

Example:

```
<input type=hidden name=context value=″l2k3j4l2k3j4l2k3j4lk23″>
```

IMPORTANT!

Hidden fields are not totally safe; you can save and modify the HTML on the browser and resend the modified HTML back to the server, which, in fact, prevails the hidden fields.

Reference on the Internet

http://members.aol.com/htmlguru/tips/index.html

2.5.6 National Characters, Code Pages, Date Format

An Internet application that is supposed to give world wide support has to have support for national characters as well as the corresponding date formats, date separators, decimal position, and decimal characters. While in a native AS/400 application, this support is given by OS/400; in the Internet environment, it is up to the application and the browser.

It is a good practice to store language dependent code/HTML in separate directories and allow the user to choose their language at the first window (Welcome window).

2.5.6.1 CCSIDs and ISO Character Sets

While OS/400 uses CCSIDs to identify the way text data is encoded, the World Wide Web uses ISO character sets to identify the way text data is encoded. The following example shows some of the useful ISO character sets and associated CCSIDs:

ISO character set	ASCII CCSID
US-ASCII	367
ISO-8859-1	819
ISO-8859-2	912
ISO-8859-5	915
ISO-8859-7	813
ISO-8859-8	916
ISO-8859-9	920
ISO-2022-JP	5052

Note

Note that ISO-8859-1 (CCSID 819) is the default character set for HTTP.

Related Documentation

The following IBM manuals contain information that may be useful to you:

Character Data Representation Architecture Level 1	SC09-1390-00
Character Data Representation Architecture Level 2	SC09-1390-01
AS/400 International Application Development	SC41-3603-00
AS/400 National Language Support Planning Guide Version 2	GC41-9877-02

2.5.7 Internet - Intranet - Extranet Considerations

One of the major differences between running applications in the IntERnet versus the intRAnet and the EXTRAnet is security (or the lack of it) and what kind of information is presented to the user.

While using the intRAnet, all information is available to the users with the right authority (for example; a sales person, to all sales and customer related data; a book keeping person, to all personnel and human resource data, and so on).

On the IntERnet, however, everything should be disallowed except information to be used by the public. Under more restricted access, users are allowed to see and modify specific data, but always with the exposure of accidentally disclosing this information also to the public (hackers).

The EXTRAnet can be viewed as a private net within the IntERnet or a private network using either dialup or leased lines with it's own security and encryption algorithms. So data exchanged between businesses is pretty much safe. However, here it is much more important to have a mechanism in place to make sure that when you send something, the addressee knows exactly who sent it and also keep a receipt for delivery to have proof of sending it.

2.5.8 Internet Application Error Recovery

Internet application error recovery is up to the application developer and the program itself. Each programming model has different techniques to monitor and handle error messages and is described in more detail either in Chapter 5, “HTML Gateway Implementation” on page 169 for Workstation Gateway, Chapter 4, “Net.Data Implementation” on page 111 for Net.Data, or Chapter 3, “Common Gateway Interface (CGI-BIN) Implementation” on page 57 for CGI-Bin programming.

2.6 Programming Alternatives

On the AS/400 system, there are several options available to support users on the Internet.

The AS/400 system offers some alternatives to enable applications to the Internet. One does not require any change to the application itself; it only has to be enabled at the server side. Users can use AS/400 applications on the Internet (intranet) as they are used to using natively on a terminal. The function on the AS/400 system is called HTML Workstation Gateway and is part of OS/400.

Another method is to provide an easy access to the data in the database and allow queries to be executed, or call programs that access the database and return the requested data.

The following diagram shows the different ways to support users on the Internet. One might to decide to take one function only, some others might want to combine two or even all available approaches.

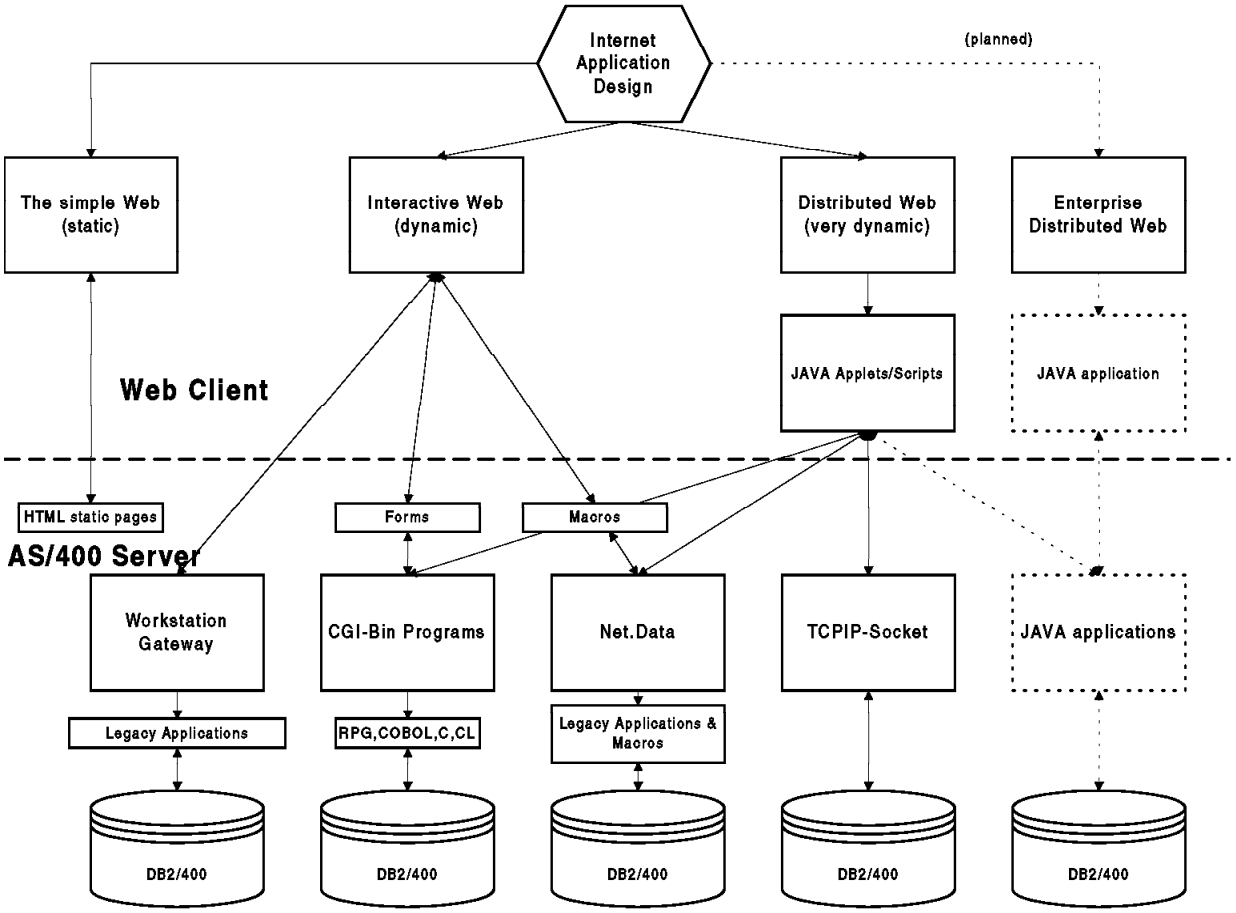


Figure 6. Internet Application Design Overview

Depending of what you want to get accomplished, you can choose either one or a combination of the following options:

- CGI - Common Gateway Interface
- Net.Data - Interactive Data Publishing with Universal Access to Dynamic Data
- HTML Gateway - Workstation Gateway
- Java applets accessing the AS/400 system directly

The following sections explain each option (programming model) in more detail to allow you to determine which one to use for your specific requirements.

2.7 Internet Programming Models

There are two dominant programming models currently in use today over the Internet: the HTTP CGI programming model and the Internet client/server programming model. The AS/400 system supports both of these models to varying degrees.

2.7.1 HTTP CGI Programming Model

The HTTP Common Gateway Interface (CGI) programming model provides a simple interface for running programs external to the HTTP server in a platform-independent manner. This interface has been in use by the World Wide Web since 1993. A good document that describes this interface can be found at URL:

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

The purpose of CGI is to extend the capability of an HTTP server by providing framework in which the HTTP server can interface with a program that is specified on a URL. The format of the URL allows parameters to be passed to the CGI program. On the server side, the interface describes how the program is started by the HTTP server and how parameters for the program are passed using a combination of standard-input and environment variables. It also describes how output information (usually HTML elements) are passed back to the HTTP server using standard output. Thus, in its simplest form, a CGI program can be defined as a program that:

1. Can be called as an executable and run as a child process of the HTTP server (this does not preclude the use of interpretive languages such as REXX and PERL. In this case, their respective interpreters run as child processes of the HTTP server).

   Note: The CGI program on the AS/400 runs within the same job as the HTTP server.

2. Is able to read from the standard input.
3. Is able to access environment variables.
4. Is able to write to the standard output.
5. Is able to access ″command line″ arguments passed to main()

CGI programs can be as simple or as complex as the programmer wants to make them, but clearly, the intent of the interface is to allow simple programs or scripts to be developed quickly.

Given this definition of the CGI interface, the following characteristics are for CGI programs:

- They can be written in compiled programming languages or interpreted scripts.
- They can be simple five or six-line programs that serve a specialized function or generalized business applications that do complex calculations and database transactions (for example, order processing applications).
- They are portable. As with any programming model, portability is a matter of degree. To the extent that the program uses any operating system unique interfaces or naming semantics, portability is reduced.
- Since they run as a child process of the HTTP server, they can benefit from the services provided by the server:
  - Communication over the HTTP protocol
  - Security
  - Firewall transparency. HTTP is usually allowed to pass through firewalls (through socks or application gateways).

  - Resource mapping and name indirection provided by HTTP server directives (program can be moved without affecting client URLs.)
  - Error logging and audit logging provided by HTTP server
  - Course grained navigation provided by HTML documents
- They provide process isolation. Since each CGI program runs in its own child process, it has the security and integrity advantages that separate processes offer.
- They run in a connectionless environment. The HTTP server starts the CGI program, it does its thing, and terminates. The output is sent to the browser and the connection is broken. Subsequent requests for that same program result in a new connection and a new instance. No state information is maintained unless the CGI program had stored previous state information in fields in the requesting HTML form or parameters in the requesting URL. This characteristic comes about as a consequence of the Web's hypertext model that allows users to hop around from machine-to-machine across the world at will. Forcing the browser to maintain persistent connections to all of these machines is ridiculous.
- Process start time often gates performance. The price to pay for process isolation is to suffer the costs of process start time. In a connectionless environment, this becomes an even bigger problem as the process must get started for each request. The AS/400 HTTP model minimizes this penalty by providing a pool of pre-started server jobs that minimize a majority of the start-up costs. Add to that the ability to exploit named activation groups that also eliminate program initialization costs, and the AS/400 CGI model starts to look better than what is provided on other platforms.
- They use short-running transactions. This is a consequence of the connectionless model. The length of a transaction is limited to a single URL request since that is the length of a process. This problem brings about the requirement for the persistent CGI model described later.
- The logic is almost entirely on the server. The CGI model is basically a distributed presentation model where the logic is on the server and the client is responsible for window presentation. The advantage of having the logic on the server is apparent when the number of window interactions is far smaller than the number of data accesses or the number of calculations performed by the server.

Although the CGI model is simple and universal, it has some definite shortcomings. This is primarily due to its connectionless nature.

- CGI program parameter data (passed as part of the URL) has portions encoded with escape sequences. It is cumbersome for CGI programs to decode these sequences after they have been converted from ASCII to EBCDIC.
- It is difficult for the CGI programs to determine and control what CCSIDs to use when handling ASCII and EBCDIC data.
- Multiple conversions from ASCII to EBCDIC of input and output data presents some performance problems.

2.7.2 Java Applet Communicating to CGI

Another dimension that can be added to the HTTP CGI model is to make use of Java applets running in the browser that communicate with CGI programs. It turns out that the Java virtual machines built into browsers have mechanisms that allow its applets to access HTML documents and CGI programs directly. With this model, a CGI program may send to a browser HTML that includes a Java applet. The Java applet may use HTTP to retrieve additional information from the server such as data stored in a file or output from another CGI program. This access is done independently of the browser session but using the HTTP services built into the browser. Since the HTTP protocol is being used, all of the advantages of HTTP are maintained including its authentication and encryption features. No changes are required to standard CGI or persistent CGI programs to support communications with a Java applet.

With this approach, logic can be split between the client and server with the client Java applet doing more than just providing a pretty window presentation. It can dynamically receive information from server files and programs at the user′s request.

2.7.3 Net.Data Macros

Net.Data provides a convenient mechanism for creating dynamic Web pages without complex programming. A user creates a dynamic HTML script called a macro that is interpreted by the Net.Data macro processor. The macro processor is implemented as a CGI program or a persistent CGI program (on other platforms, it is also implemented as a server API program). The macro contains HTML statements as well as macro directives that represent substitution variables, SQL calls, or program calls. Macros can be used to build complex forms and process information from those forms. They can also be used to generate Java applet calls, Java script, and Visual basic script statements.

2.7.4 The Internet Client/Server Programming Model

The second prominent Internet programming model is the Internet client/server model. This model is not that different from the traditional client/server model that allows program logic to be distributed between a client and a server communicating over some agreed-to protocol. The Internet client server model is different only in that Internet/intranet protocols are used (that is, something that runs over TCP/IP) and the programming languages and associated tools are platform independent (for example, Java).

What is attractive about the Internet client/server model (compared to the HTTP CGI) model, is that the program logic can be distributed as appropriate between the client and server, and a persistent connection is maintained allowing for long running transactions. However, since the HTTP protocol and the HTTP server are not being used for communications, it is the responsibility of client and server pair to agree on what application protocols are used (for example, roll-your-own over a sockets connection), and how things such as security, encryption, integrity, translation, and such are handled. This becomes especially difficult when dealing with transient Java applets.

For the purposes of this document, the Internet client/server programming model is defined as Java applications or applets communicating to server applications over Internet/intranet protocols. The server applications can be either new or existing procedural servers or new Java servers.

Why Java? Java is the common man's OO language. It is much simpler than C++ and it was designed for programming on the Internet. It comes with its own distributable object model and it runs everywhere. Technically, Java is the only language that deals with threads as part of its basic construction. This has the potential of giving it scalability characteristics that are orders of magnitude greater than the traditional process based model. And finally, Java is a good fit for the AS/400 system. Its byte code architecture maps well to our MI architecture and the fact that is uses no pointers makes it a natural for our secure AS/400 execution environment.

2.7.5 Lotus Domino

Domino allows application developers to use the powerful application development facilities of Notes to create Web applications. Notes includes its own graphical forms designer, directory services, replication services, messaging services, security, and workflow. Domino allows applications developed with Notes to be accessible through the Web.

2.8 Supporting the AS/400 Legacy Programming Models

The previous discussion described the "new" internet programming models available on the AS/400 system. It does not describe what we are doing to support the thousands of programs and millions of lines of code implemented with the AS/400 legacy programming models.

We use the legacy programming model here to describe those applications that were probably written in COBOL or RPG and communicate to a 5250 workstation using DDS. There is a well-defined set of tools and procedures available to create, debug, and deploy these applications and our ISVs have invested a great deal of time and money in training their developers to become proficient in this area.

Eventually, we want those developers to become comfortable with the new programming models and begin to re-tool their development processes. We must realize, however, that this re-tooling is gradual at best. We, therefore, have a responsibility to support and extend the existing legacy applications by providing mechanisms to make them accessible over the Internet. Some of these mechanisms allow this to happen transparently to the application. Others require various degrees of modification to portions of the application.

We provide some of the following mechanisms to support our legacy applications.

2.8.1 The 5250 Workstation Gateway

The 5250 workstation gateway is a specialized HTTP server that transforms the 5250 data stream into HTML on the fly. With the gateway, any browser can run any AS/400 application with no modification to that application.

Functionally, this is the best Internet support we have for legacy applications. It is completely transparent to the applications, works with system programs as well as user programs, and can be accessed from any browser on the Internet.

There is a price to pay for this functionality and transparency (namely usability and performance). Usability problems stem mainly from the fact that an Internet browser does not have the same functional capabilities as a 5250 terminal and

controller. It does not support function keys, it does not support special field editing capabilities (such as mandatory fill, mandatory entry, numeric-only fields, range checking, and check digit), and it does not maintain a persistent connection with the host system. The 5250 gateway compensates for these deficiencies the best as it can but usability suffers.

As far as performance goes, there is, of course, a penalty to pay for the translation of 5250 screen buffers into HTML. As you can imagine, there are very few clues in the screen buffer to indicate how fields should be translated into HTML. A brute force method is used to figure this out and it takes some time. The result is that response time for a browser using the gateway is several times greater than an equivalent 5250 terminal connection.

The 5250 workstation gateway does provide some capabilities not available to 5250 terminals. One is the capability to define HTML keywords in the display file DDS specification. A new DDS keyword (″HTML″) can be added to a DDS specification and is recognized by the gateway. These can be used to define, for example, HTML image tags to display graphics or a special page title, headers, footers, and so on.

Another unique capability of the 5250 gateway is the ability for the browser to specify the program it wants to run as part of the URL. An exit program (written by the customer) receives this request, verifies the client′s authority to the program, and allows the operation to take place. This eliminates the need to present the sign-on display to the browser user and force the user to navigate to the program the user wants to run. It makes for much better integration with other HTML documents.

The 5250 workstation gateway is a novel approach to providing legacy application access from the Web. It provides the AS/400 system with a differentiator that most systems today cannot offer.

2.8.2 ″On-the-Fly″ Translation versus the Toolkit Approach

The on-the-fly approach is what WSG currently uses. It involves intercepting the 5250 data stream and trying to figure out dynamically:

1. What the green screen layout was intended to be
2. How to best represent that with HTML controls (or, in the future, Java controls)

This is a compute intensive process and leads to a lowest common denominator approach whereby the resulting HTML window does not look too much different from the intended ″green screen″.

An alternative approach to this problem is being done by at least two of our business partners, Seagull and CST. They use a toolkit approach that translates all the windows for a particular application ahead of time rather than on-the-fly. The toolkit allows a customer or application developer to run through all of the windows of an application one-by-one. The toolkit does its best to translate the 5250 screen into an HTML or Java equivalent and shows the user the result. At that time, the user can change the translated look of that window by moving, modifying, or adding controls. The changes are recorded and the GUI representation of that 5250 screen is stored in a separate data store and indexed through a hash of the 5250 screen. This sequence is done for every window that can be produced by that particular application (which could be hundreds).

At run time, gateway code intercepts the 5250 data stream, builds a hash from it, and uses the hash to look up the associated GUI representation of that window. This pre-built GUI is sent to the client (browser or Java applet). If no index exists for this hash, an on-the-fly representation of the 5250 screen is built.

This approach has the advantage of being faster at run time and provides a more complete and tailorable GUI representation than what can be done simply on-the-fly. However, it does require someone (typically the application provider) to go through each window ahead of time and build the associated data store. If changes are made to the application or the DDS, portions of the data store must be rebuilt. Most likely, a separate data store has to be built for each translatable version of the application.

There are definite advantages and disadvantages to each approach and you have to decide which you want to invest in.

2.8.3 Telnet and Client Access™ - Personal Communication

Using a Telnet session is not really a programming model by it's own, but might be acceptable for many customers to allow direct access to their applications from the Web.

However, there is no security available today to preclude others from seeing entered USERIDs and passwords and all data flows clearly over the net (no encryption for Telnet available today).

The same is true for Personal Communication of Client Access/400, although it has some nice features that distinguish it from using plain Telnet (tn5250). Personal Communication allows you to copy and paste between an AS/400 session and any other PC-session, allows you to record and playback keystrokes, transfers files between the host and the PC, and some more functions to assist a workstation session configuration and customization.

2.9 HTML Programming Considerations

The following sections do ***not*** try to teach you how to program HTML, but rather give some basic information and some hints on writing HTML.

We strongly recommend that you buy one of the many available books on the market about HTML and Web Publishing.

Note

A good reference book and book to learn about HTML is:

Teach yourself Web Publishing with HTML 3.x from Laura Lemay published by sams.net.

There are a lot of nice HTML tools available that makes HTML coding easy, in fact, so easy that you do not even have to do know about any HTML statement. All of it is done through templates and prepared forms and wizards.

Because of this, there is no need to learn all about HTML, but the is no harm in knowing about the basics. The following section gives you an introduction to HTML and some references to more detailed information.

2.9.1 Hypertext Markup Language (HTML)

The Hypertext Markup Language (HTML) is a simple markup language used to create hypertext documents that are platform independent. HTML documents are SGML (Standard Generalized Markup Language) documents with generic semantics that are appropriate for representing information from a wide range of domains. SGML is an international standard for document markup conforming to ISO 8879.

The latest defined version of HTML is HTML3.2.

The major enhancements of HTML3.2 over HTML2.0 are:

- Split large documents across multiple servers.
- Support for tables
- Support for mathematical formulae
- Frame Support

HTML is similar to a computer programming language without being a programming language. There are commands called tags and syntax rules to be observed when writing in HTML.

HTML documents can be written using any word processor or text editor. The way they look when seen with a Web browser, however, is quite different from what the writer sees when editing them. It is not a *What You See Is What You Get* (WYSIWYG) approach. There are HTML editors currently available on the market that can be helpful and productive to use when writing your HTML documents.

HTML language provides support for the following features:

- Hypertext links to resources (documents, multimedia, or data files)
- Menu and forms
- Inline graphics
- Text formatting

2.9.2 The HTML Document Structure

The HTML documents are composed of two main parts.

A head Every document should start with a head. The head is the top part of the document. In general, it includes the document title. Why is it, then, important to define the head part in the document?

The different browsers on the market have different ways to display the document's title. The title is also the way by which documents are referenced when saved in the Hotlist or Quicklist of the browser. The title, therefore, must be descriptive but short so it fits into one line of the Quicklist window.

The head of a document cannot contain anchors, any kind of highlighting, or paragraphs. The tags to enclose the head are <HEAD> and </HEAD> (simply meaning start and end of head).

A body The body is the core part of the HTML document. It contains all of the information that is part of the document and controls the way this is presented to browser users.

The body can contain images, lists, menus, entry fields, plain text, or link to other resources. The tags to enclose the body are <BODY> and </BODY> (simply meaning start and end of body).

Figure 7 shows an example of an HTML document.

```
<HTML>
<HEAD>
<TITLE>Ordering an AS/400</TITLE>
</HEAD>
<BODY>
<IMG ALIGN=middle SRC="file:///htmldoc/as400.gif">
<B>PLEASE FILL IN THE FOLLOWING :</B>
<P>
<FORM METHOD="GET" ACTION="/QSYS.LIB/HTML.LIB/ORDAS400.PGM">
Name:
<INPUT TYPE="TEXT" NAME="NAME" SIZE="30" MAXLENGTH="40">
<P>
Address:
<TEXTAREA NAME="ADDRESS" ROWS=2 COLS=30>
</TEXTAREA>
<P>
<B>Which AS/400 would you like to order ?</B>
<BR>
<INPUT TYPE="RADIO" NAME="P1" VALUE="P1">Portable
<INPUT TYPE="RADIO" NAME="P1" VALUE="P2">Server
<INPUT TYPE="RADIO" NAME="P1" VALUE="P3">System
<P>
<B>Do you want the Support Line Service ?</B>
<BR>
<INPUT TYPE="CHECKBOX" NAME="w1" VALUE="w">Yes
<INPUT TYPE="CHECKBOX" NAME="w2" VALUE="f">No
<P>
Thanks for ordering
<INPUT TYPE="SUBMIT" VALUE="Order">
<INPUT TYPE="SUBMIT" VALUE="More Information">
<INPUT TYPE="SUBMIT" VALUE="Cancel">
</FORM>
</BODY>
</HTML>
```

Figure 7. HTML Document

If the HTML document is displayed with a browser, for example, Web browser for OS/2, you see that the title in the head part is displayed in the title bar of Figure 8 on page 27.

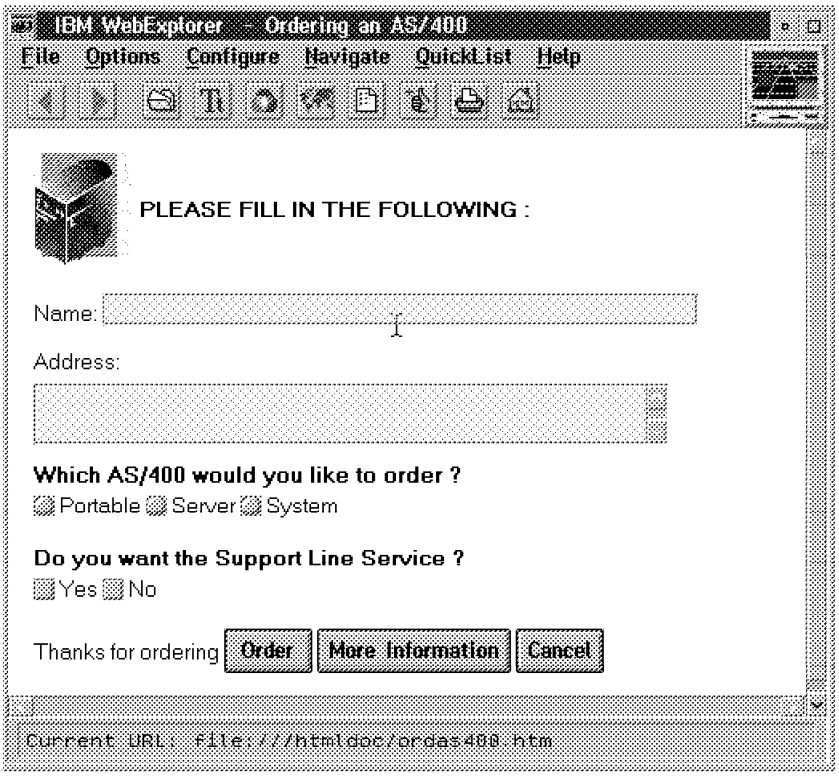


Figure 8. HTML Document Displayed Through an OS/2 Web Browser

2.9.3 HTML Syntax

Let's give you an introduction to the HTML syntax in short terms. The HTML language consists of markup tags to identify the elements of the document. All tags begin with a left angle bracket (<) and end with a right angle bracket (>). Almost every tag is a container. This means that there is always an opening tag and a closing tag the same as in the head and body. Table 1 lists the main HTML elements:

Table 1 (Page 1 of 3). HTML Main Elements

Name	Opening tag	Closing tag	Description
Anchor	< A >	< / A >	HyperLink to a resource.
Address	<ADDRESS>	</ADDRESS>	Format an address.
Bold	< B >	< / B >	Display test in bold.
Base	<BASE>	No closing tag	Record URL of document.
Body	<BODY>	</BODY>	Contain the document's body.
Blockquote	<BLOCKQUOTE>	</BLOCKQUOTE>	Include text in quotes.
Line Break	< B R >	No closing tag	Break current line.
Citation	<CITE>	</CITE>	Specify a citation.
Code	<CODE>	</CODE>	Enclose an example of code.

Table 1 (Page 2 of 3). HTML Main Elements

Name	Opening tag	Closing tag	Description
Definition list description	<DD>	No closing tag	Description of Definition list item
Directory List	<DIR>	</DIR>	Enclose a directory list.
Definition List	<DL>	</DL>	Enclose a list of terms and definitions.
Definition list item	<DT>	No closing tag	Item of definition list
Emphasis	<EM>	</EM>	Emphasize enclosed text.
Form	<FORM>	</FORM>	Define form of enclosed text.
Level 1 heading	<H1>	</H1>	Enclose level 1 heading.
Level 2 heading	<H2>	</H2>	Enclose level 2 heading.
Level 3 heading	<H3>	</H3>	Enclose level 3 heading.
Level 4 heading	<H4>	</H4>	Enclose level 4 heading.
Level 5 heading	<H5>	</H5>	Enclose level 5 heading.
Level 6 heading	<H6>	</H6>	Enclose level 6 heading.
Head	<HEAD>	</HEAD>	Define the head of the document.
Horizontal Rule	<HR>	No closing tag	Insert horizontal line.
HTML	<HTML>	</HTML>	Define HTML document.
Italics	<I>	</I>	Italicize enclosed text.
Image	<IMG>	No closing tag	Embed an image.
Input	<INPUT>	</INPUT>	Display entry field.
Index	<ISINDEX>	No closing tag	Define searchable URL.
Keyboard	<KBD>	</KBD>	Indicate user typed text.
List item	<LI>	No closing tag	Item of Directory list, Menu list, Ordered list, Unordered list
Link	<LINK>	No closing tag	Describe relationship between documents.
Menu	<MENU>	</MENU>	Enclose a menu list.
Ordered List	<OL>	</OL>	Enclose an ordered list.
Option	<OPTION>	No closing tag	Indicate one choice in a Select menu.
Paragraph	<P>	</P> (optional)	Define a paragraph.

Table 1 (Page 3 of 3). HTML Main Elements			
Name	**Opening tag**	**Closing tag**	**Description**
Pre-formatted text	<PRE>	</PRE>	Enclose pre-formatted text.
Sample	<SAMP>	</SAMP>	Indicate sample text.
Select	<SELECT>	</SELECT>	Define a set of selectable options.
Strong Emphasis	<STRONG>	</STRONG>	Strongly emphasize text.
Title	<TITLE>	</TITLE>	Define document's title.
Typetype	<TT>	</TT>	Display enclosed text in monospaced font.
Textarea	<TEXTAREA>	</TEXTAREA>	Enclose a text area.
Underlined	<U>	</U>	Underline text (unsupported by Mosaic).
Unordered List	<UL>	</UL>	Enclose an unordered list.
Variable	<VAR>	</VAR>	Indicate a variable.

HTML tags are case insensitive. Every command is interpreted by the browsers independently of the capitalization. The tag: <FORM> , for example, can either be written: <Form> or <form> or <fORm> without making any difference.

Note This!

If you want to know more about the Hypertext Makeup Language (HTML), there is a lot of information available on the Internet. There are also a lot of publications available. One of them is the redbook, *Using the Information Super Highway*, GG24-2499.

The most commonly used HTML tags are the headings, list, anchors or links, images, and form tags.

The form tag is used when requesting CGI programs as we shall see in Chapter 3, "Common Gateway Interface (CGI-BIN) Implementation" on page 57, and the image tag is, as it says, used when referring to images or clickable images. See Section 2.10, "Server Side Imagemap Support" on page 30 for more information.

2.9.4 Creating Static HTML Pages

The best approach is to create all the static HTML pages with a PC tool (there are many available on the market - check on the Internet) and store them when you are finished on the AS/400 system in a folder, or even better yet, in the Integrated File System (IFS).

For more information about IFS, refer to the 2.13, "IFS - Integrated File System Considerations" on page 43.

Note

For more detailed information about HTML coding and use, see:

http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html

2.9.5 Feedback

It is recommended that on your Web site, you include an e-mail address or point of contact for your company. This allows you to gain valuable feedback, or provide a means for users to report error conditions. For example, the following note might appear at the foot of your home page:

Contact webmaster@yoursite.com with comments.

The HTML used in this example is:

```
Contact <A SUBJECT="Comment on site home page"
HREF=mailto:webmaster@yoursite.com>webmaster@yoursite.com</A>
with comments.
```

The mailto action in the HREF tag causes the client's e-mail package to be opened with the subject field already filled with the text "Comment on site home page".

2.10 Server Side Imagemap Support

Suppose you want people to be able to choose a car, check if a certain model is available, and make reservations from your Web server. What would be better than to show them a map of the country and allow them to click on any location and make the desired reservation? That is what a clickable image map can do. It can have many other uses also. For example, it helps users better understand a picture or diagram by allowing them to click on a component represented in the graphic to see an explanation. It also lets users move down to finer levels of detail on a map of a campus, building, or whatever you want.

Clickable image maps obviously work only with graphically-oriented Web browsers. You should always try to provide alternate text-based methods of accessing the same information.

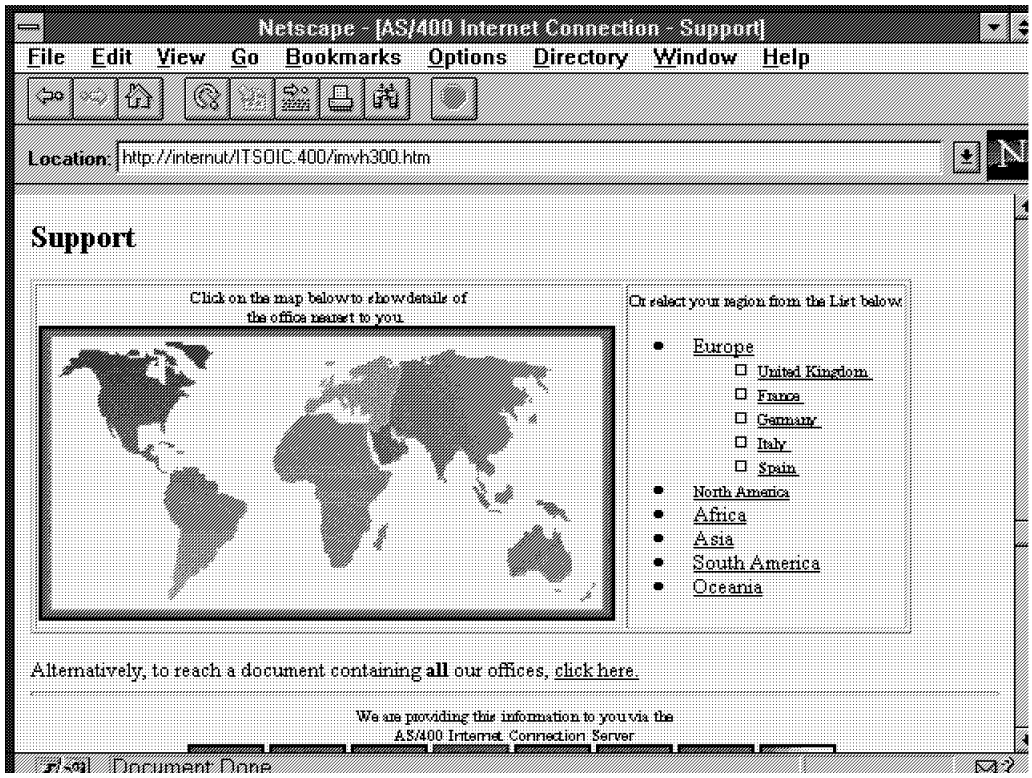


Figure 9. Clickable World Map. This figure shows a clickable map and an alternate text-based method of accessing the same information.

Image map requests are another form of program call or CGI. You can request image maps from either a server-provided image mapping program or a user-provided image mapping program. The way to provide this level of function for your Web application is to take advantage of a server-provided mapping program such as the one found on the AS/400 system. This IBM written CGI-BIN application is called QTMHIMAG in the QTCP library.

If an image map request is made from a user-provided image program, it is handled as a standard program call request that must meet the CGI interface.

There are several steps you need to take when creating a server-provided image map:

- Plan a clickable image map.
- Map the hotspots in a map file.
- Reference your clickable image map in HTML.
- Configure the HTTP server.

Let′s take a look at each of these steps in detail.

2.10.1.1 Planning the Clickable Image Map

First, determine the image you are going to use. As for any inline images, the graphics need to be in GIF or JPEG format for widest portability.

Define where you are going to map the clickable areas or hotspots.

Depending on the size and make up of the image you are mapping, you can use polygons, rectangles, or circles to identify hotspots.

2.10.1.2 Mapping the Hotspots in a Map File

The next step is to develop an image map file such as the one seen previously. You need to create a separate image map file for each clickable image map on your Web server. Each line in an image map file represents a hotspot by defining an area within the graphic and the corresponding URL to be returned if someone clicks on that area with their Web browser. By using a program on your PC such as MAPEDIT, hotspots on the image are formatted by:

Shape, Coordinate-1, Coordinate-2 ... Coordinate-n, URL.

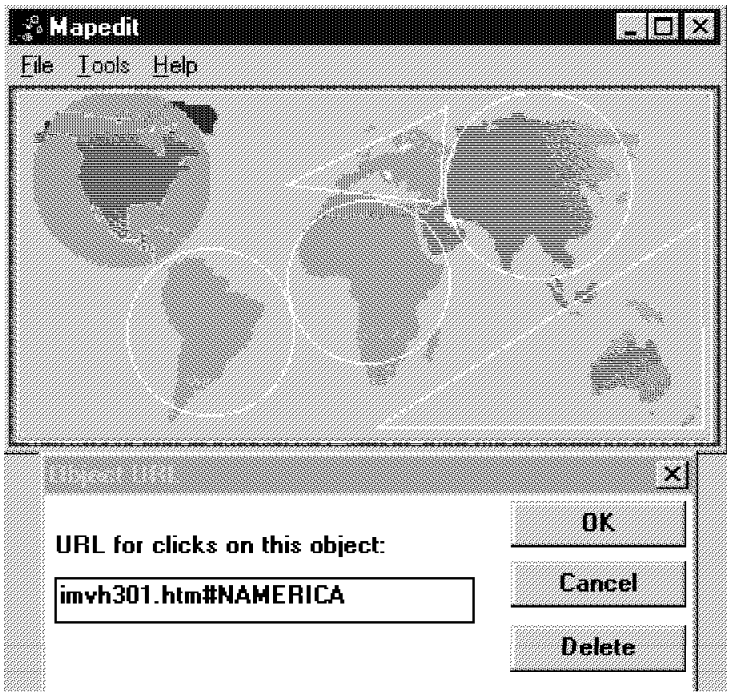


Figure 10. Defining Hotspots and Associated URLs. This figure shows North America in a circle shape mapped to its associated URL.

This map is made up of circles and triangular shaped polygons. A rectangular shape was mapped last for the entire image as a default area.

Hotspots are defined by the following shapes:

- A circle - for a circle (a pair of coordinates for the center and a single value for the radius).
- A polygon - for a polygon (the number of coordinates depend on the number of sides defined for the polygon).
- A rectangle - for a rectangle (with two coordinate pairs: upper-left and lower-right).

Coordinates are x,y pairs counting in pixels from the upper left-hand corner of the image. The URL can be either an actual URL pointing to any resource on the World Wide Web, or it can point to another image or document on your server.

```
      *************** Beginning of data ********************************
001.00 circle (101,123) 42 /ITSOIC.400/imvh301.htm#SAMERICA
002.00 circle (56,46) 45 /ITSOIC.400/imvh301.htm#NAMERICA
003.00 circle (180,98) 41 /ITSOIC.400/imvh301.htm#AFRICA
004.00 circle (265,50) 47 /ITSOIC.400/imvh301.htm#ASIA
005.00 poly (218,10) (215,58) (138,49) /ITSOIC.400/imvh301.htm#EUROPE
006.00 poly (346,68) (184,170) (347,170) /ITSOIC.400/imvh301.htm#OCEANIA
007.00 rect (4,3) (351,177) /ITSOIC.400/imvh301.htm#OCEANIA
008.00 default /ITSOIC.400/imvh301.htm#OCEANIA
      ****************** End of data **********************************
```

Figure 11. Image Map Coordinates. This example shows circular, polygon, and rectangular shapes and their defined URLs.

The server takes the coordinates from a user′s mouse click and steps through the map file to determine if the click is within any hotspots. As soon as the first match is found, the corresponding URL is used to redirect a document to the user′s Web browser. If no matches are found, a default URL (that you specify in the image map file) is returned.

Notice that the last line in the file is a default statement. This might be helpful so that if a user clicks on a part of the image that has not been mapped, the user is sent to a defined default area.

It is normal to mix and match circles, polygons, and rectangles in the same map file. Although you should try to minimize overlapping hotspots in the image map, if there are any, the first match is the one used.

2.10.1.3 Referencing Your Clickable Image Map in HTML

The final step in creating an image map is to tell the Web browser accessing the HTML document that the inline image it displays is a clickable image map. In practice, what you do is create a link where this image is the link trigger and the URL invokes the image map script and passes it the map name. For example:

```
<A HREF="http://servername/imagemap.pgm/imagemap.file">
<IMG SRC="image.gif"> ISMAP></A>
```

2.10.1.4 Configuring Your Clickable Image Map in the HTTP Server

To configure the HTTP server on the AS/400 system, we reference the previous example. Our imagemap.pgm is qtcp/qtmhimag and our image maps are found in the imagemap.file. Here is how our URL looks using the AS/400 system to serve image maps:

```
<A HREF="http://servername/qsys.lib/qtcp.lib/qtmhimag.pgm 1
/qsys.lib/itsoic400.lib/imagemap.file/mapk.mbr"> 2
<IMG SRC="imvg300.gif" ISMAP></A> 3
```

1 This section of the URL represents the server name and also where the image mapping program resides on the AS/400 system.

2 This section tells the server where the actual image map file is located on the AS/400 system.

3 This section shows the source of the image or GIF file that is to be mapped and ISMAP tells the server this is an image map.

We can shorten and simplify the URL by taking advantage of the HTTP mapping rules. For example, we use the following URL to do the same thing:

```
<A HREF="/cgi-bin/imagemap/map1.mbr">
<IMG SRC="world.gif" ISMAP></A>
```

Where you see /cgi-bin/imagemap, we used the HTTP Map configuration statement to reference the longer URL. The following lines must be added to the HTTP configuration to make it work:

```
Map /cgi-bin/imagemap/*  /qsys.lib/qtcp.lib/qtmhimag.pgm
/qsys.lib/itsoic400.lib/imagemap.file/*

Pass /qsys.lib/itsoic400.lib/imagemap.file/*

Exec /qsys.lib/qtcp.lib/*
```

To access the HTTP configuration file, type WRKHTTPCFG on an AS/400 command line. Add the Map, Pass, and Exec statements noted previously. By using the Map directive in our HTTP configuration, we are able to use shorter URLs in our HTML documents. The Pass statement is there to pass the image map files from the server. There must also be an Exec statement to call the qtmhimag program from the qtcp library.

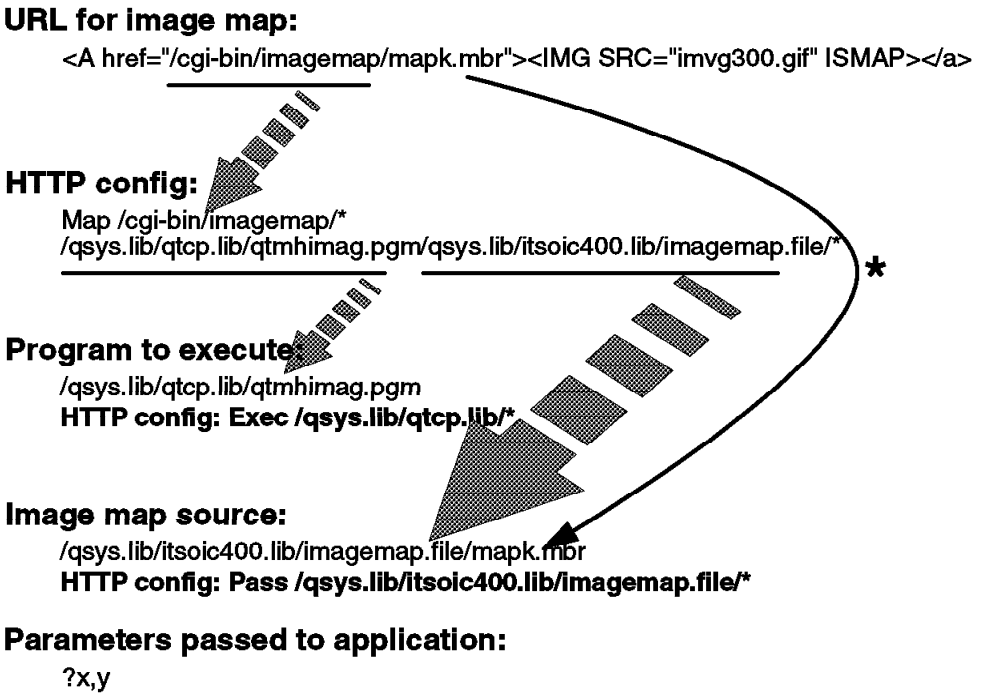


Figure 12. Mapping of Image Map URL to HTTP Configuration

Note

More information on image mapping and image map requests on the AS/400 system is available in the *TCP/IP Configuration and Reference, Version 3*, SC41-3420.

2.10.2 Server Side Image Maps Considerations

More and more browsers do support client side image maps today, so the urgency for providing this on the server is almost gone. However, a server such as the AS/400 system, especially some of the faster server models (50S, 53S), can provide a much faster solution for image maps than a typical PC can do. On the other hand, if there are many of these kinds of requests, the server is quite busy doing these operations and it might be worthwhile to move this to the browsers.

2.11 Netscape, Microsoft® IE, IBM Web Explorer, Spyglass

Currently the most popular browser for the World Wide Web is Netscape′s Navigator developed by Netscape Communications Corporation. Netscape has become so popular that using Netscape and using the Web have become synonymous to many people. However, despite the fact that Netscape has the lion′s share of the market, it is not the only browser on the Web.

It is important for you as an Internet application programmer to make a distinction between the information on the Web and the browser used to view it. Assuming Netscape is the only browser in use on the Web, designing your pages accordingly limits the audience you can reach with the information you want to present.

A wide array of Web browsers is available for just about every platform you can imagine, including graphical user-interface based systems (Mac, Windows, X11) and text-only for UNIX dial-up UNIX connections. Most browsers are freeware or shareware (try before you buy) or, the same as Netscape, have a lenient licensing policy (Netscape allows you to use its browser for personal use for an evaluation time after which you are expected to buy it). Usually all you have to do is download them from the net. If you get your Internet connection through a commercial online service such as America Online or CompuServe, there are usually Web browsers built into the software for that system as well.

Although we urge the application programmer to consider the differences of browsers, we advise the programmers to design their applications to be independent from browsers as much as possible. That means, stay away from specific functions available only on some browsers (although they might have real nice functions).

2.12 ILE Integrated Language Environment® Programming Considerations

This section concentrates on aspects of the ILE architecture that must be addressed from a design perspective. The contents should be read by anyone involved in either designing migrations from Original Program Model (OPM) to ILE or designing new ILE applications.

2.12.1 Overview of ILE Concepts

ILE provides new concepts to support both a run-time environment with fire walls built around an application and a development environment supporting the production of highly modularized, reusable code. These new concepts are summarized later. For full details of each of these facilities, please refer to *Integrated Language Environment Concepts*, SC41-3606.

- **Procedures**

A procedure is a set of self-contained HLL statements that perform a particular task and return to the caller. In ILE RPG/400® and ILE CL, there is one procedure per source member and one procedure per module. The procedure name is always the same as the containing *MODULE name for RPG IV and ILE CL. In ILE C/400® there may be multiple procedures (called functions in C) within a module. Refer to Program Entry Procedure on page 38 for this special form of a procedure.

- **Modules and programs**

  A module object (*MODULE) is the result compilation using the new ILE compile commands. A module cannot be run. To be run, a module must be bound into a program object (*PGM).

  An ILE program object (*PGM) is the result of binding one or more modules together. You run programs on the system just as you did in OPM.

- **Static binding**

  In OPM, all calls to programs are dynamic, involving system overhead to perform authority checking and find the address of the program. Binding is the process of creating a program by packaging ILE modules and resolving symbols passed between those modules.

  When a dynamic call is made to an ILE *PGM object, the program is activated. This activation involves the initialization of all static storage required by variables used by the modules within the object. In the case of RPG IV, all variables are stored in static storage.

  There are two types of static bind available within ILE. Once a program and any related service programs have been activated, there is no difference in the system code that is run to perform a bound call by reference or a bound call by copy.

  1. Bind by copy

     This is the process of copying modules directly into a program object during the binding phase. Thus, the modules specified to be bound by copy are contained directly within the program object.

  2. Bind by reference

     This is the process of making procedures indirectly available to a program (*PGM) through a service program (*SRVPGM). Modules bound in a *PGM by reference to a *SRVPGM are not copied into the *PGM object.

     Optional service programs (*SRVPGMs) is associated with the *PGM at bind time. The activation of these associated service programs involves the initialization of all static storage in all modules within the service programs.

- **Service Programs**

  Service programs cannot be run through a dynamic call. They act as a container for related procedures that are used in many different *PGMs. Thus, in order to easily maintain these popular procedures, they are stored in one place, a service program.

  Since activation of a *SRVPGM causes initialization of all static service programs, it involves the initialization of all static storage in all modules within the service programs. Even if your *PGM only uses one module from

a service program, if the service program contains N>1 modules, the static storage for all N>1 modules is initialized at activation of your *PGM.

- **Binding Language**

  This is a simple language that controls which procedures and variables are available from a service program to other programs or other service programs.

- **Activation Groups**

  An activation group enables partitioning of resources within a job.

  Therefore, an activation group is only seen by the job in which it was created. An activation group consists of static storage needed for variables belonging to activated programs, open files (open data paths), commitment control definitions, and application programs. When a new job is started, the system automatically creates two activation groups. These are collectively referred to as the default activation group but are split into activation group number 1 and number 2.

  All programs (system or user) run in an activation group. The activation group in which an application program runs is determined at program creation. When you create ILE programs and service programs, you specify the activation group in which your program runs using the ACTGRP keyword. You can choose one of the following options:

  1. NAMED activation group

     An activation group with a name you specified at the time of creating ILE programs.

  2. *NEW activation group

     Every time the program is dynamically called, a new activation group is created. As soon as the program returns control to its caller, the *NEW activation group is deleted. Frequent use of this choice provides the worst performance within your application.

     If You Want ANSI Semantics...

     Activation group of anything other than *NEW may result in the program's run-time behavior not following ANSI C semantics.

  3. *CALLER activation group

     The program is run in the activation group of its calling program.

- **ILE Program Activation**

  When a dynamic call to an ILE program is issued within a job, the system performs the following tasks known as program activation:

  1. Identify what activation group the *PGM should run in.

     If the activation group is NAMED and does not exist, create it. If the activation group is *NEW, create a new activation group in which to run the *PGM (this is expensive in terms of CPU). If the *PGM has not been called in this activation group before (always the case with *NEW activation groups), initialize all static storage for all associated modules whether they are bound by copy or bound by reference.

  2. Activate all service programs that have been bound to the *PGM. Identify which activation group the *SRVPGM should run in.

If the activation group is NAMED and does not exist, create it. If the *SRVPGM has not been called in this activation group before, initialize all static storage for all modules in the service program.

3. Pass control to the Procedure Entry Point (PEP) in the first procedure specified in the MODULE list at *PGM creation time.
4. If the entry procedure is RPG IV without using the logic cycle, control passes to the first executable statement in the calculation specifications after full opens of required files and resolution of passed parameters exactly as in OPM.

   When the program is deactivated and it runs in a *NEW activation group, the activation group is deleted and all associated resources are returned to the system.

- **Binding Directory**

  A list of modules or service programs. The contents of a binding directory are only used by the binder if unresolved imports exist during either the bind of modules specified on the MODULE list or modules exported from service programs specified on the BNDSRVPGM list.

- **Program Entry Procedure (PEP)**

  A PEP is system generated and the first procedure placed on the call stack following a dynamic call. This procedure is always given control first following a dynamic call. The PEP ensures that the procedure you specified as the Entry module on the CRTPGM is given control following a dynamic call.

  Service programs never have PEPs on the call stack. This is because any procedure in a bound service program may be called (multiple entry points (MEP)); there is no concept of a first procedure to always be run (PEP) in a service program. (Remember that service programs cannot be run by a dynamic call.) The name of the PEP on the call stack depends on the ILE HLL used for the entry module (ENTMOD) of the program. These names are:

  ```
  _C_pep for ILE C
  _Q_QRNP_PEP for ILE RPG/400
  _CL_PEP for ILE CL
  ```

2.12.2 ILE Compile and Bind Commands

There are two categories of ILE compile commands provided:

1. Full-function ILE compile and bind commands:

   The Create Module command and Create Program command provide access to all of the features available within ILE. Greater design flexibility is provided as a result of splitting the compile and bind into two separate commands. The Create Module commands are used to compile a module object, hence the command is ILE compiler-dependent. CRTRPGMOD is used to create an RPG IV module and CRTCLMOD is used to create an ILE CL module. To obtain an executable *PGM object within ILE, it is necessary to perform a compile of source code into a module object (*MODULE) followed by a bind of the *MODULE object (or objects) into a program object (*PGM). Either the Create Program (CRTPGM) command is used to bind *MODULE objects into a program, or the CRTBNDxxx command is used to perform a one step compile-and-bind to create a program.

The Create Service Program (CRTSRVPGM) is used to bind modules into a service program object. As previously stated, you cannot directly run a service program; you can only indirectly run procedures in a service program through bound calls from the *PGM object that was created with references to the service program.

2. Restricted ILE compile and bind commands:

   The Create Bound Program commands CRTBNDRPG and CRTBNDCL provide access to some but not all of the ILE facilities. Thus, these commands are designed to be simple to use and, consequently, do not provide the flexibility of the full-function ILE commands. Use of these commands enables you to take advantage of RPG IV. If you elect to run in OPM compatibility mode, you cannot use the call bound (CALLB) operation code; the compile part of the process fails with the following message:

   RNV5378 severity 30 CALLB cannot be used when DFTACTGRP(*YES) is specified for CRTBNDRPG If you specify DFTACTGRP(*YES), your program is only run in the default activation group; hence, you are unable to take full advantage of ILE named activation groups and resource scoping.

2.12.3 OPM Compatibility Mode

Compatibility mode is an ILE program attribute that, when enabled, makes an ILE program behave in a manner compatible with OPM program behavior. This facility is available for the ILE RPG/400 and ILE CL programming languages; it is not available for ILE C/400.

The way to enable OPM compatibility mode for an ILE program is to use the CRTBNDRPG or CRTBNDCL command and specify DFTACTGRP(*YES).

Use this command if you want to migrate all or part of your application from RPG III to RPG IV, but you do not want to take advantage of ILE bound calls, service programs, or activation groups at this time.

If you enable compatibility mode (specify DFTACTGRP(*YES) on the CRTBNDxxx command), the following ILE facilities are **not** available to you:

1. Bound calls
2. EXPORT/IMPORT keywords and specifications
3. Service programs
4. Use of named or *NEW activation group

2.12.4 Activation Groups

All OPM programs run in the same activation group (the system-provided default activation group).

Within ILE, activation groups are used to logically partition applications that are used concurrently within the same OS/400 job. Activation groups exist for the life of the job that created them. Once the job is terminated, all activation groups associated with that job are deleted.

Creation of a new ILE program or service program always involves the specification of the activation group in which the program runs. Thus, resources such as the order entry (OE) and accounts receivable applications are separated such that there is no conflict over files common to both applications, commitment control scoping, and variables used within programs common to both applications.

This separation of applications and application resources within a job is implemented in ILE through the use of activation groups.

2.12.4.1 Default Activation Group

OPM programs and ILE programs created with OPM compatibility mode always run in the default activation group. Specifically, these programs always run in default system activation group number 2, which is the user-portion of the default activation group available for your applications to use.

2.12.4.2 User-Named Activation Group

Upon a dynamic call to an ILE program that specifies a named activation group, if the activation group does not already exist within the job, it is created. An important difference between named activation groups and the other types of activation groups is that a new activation group is only deleted when the Reclaim Activation Group (RCLACTGRP) command is issued.

Specify the name of an activation group using the ACTGRP keyword on the CRTPGM or CRTBNDxxx commands.

Named Activation Group...

You do not want to populate a table with activation groups. They should be used by heavy-duty applications that require significant startup costs.

2.12.4.3 Activation Group of Caller

We expect that most ILE programs are created to run in the same activation group as their calling program. To specify this, you should use ACTGRP(*CALLER) on the CRTPGM command. Control of where your program is run is determined by the application start-up program or programs that explicitly specify your application′s activation group name.

You can also specify that an ILE service program be run in the same activation group as the ILE program to which it is bound by using the default of ACTGRP(*CALLER) on the CRTSRVPGM command.

2.12.4.4 System-Named Activation Group (*NEW)

This is the default on the CRTPGM command, and is deliberately not available on the CRTSRVPGM command. Whenever a program created with ACTGRP(*NEW) is called dynamically, a new activation group is created, the program is activated and run, and when the program returns control to its caller (through RETURN, LR, or a hard leave), the activation group is deleted.

IMPORTANT

Avoid using ACTGRP(*NEW) even though it is the default on CRTPGM. Using this activation group option is the worst choice for performance.

We recommend that this option is not used within an ILE application. It is expensive in terms of system resource to keep creating and deleting *NEW activation groups.

We recommend that you run ILE programs in a named activation group rather than allowing them to run in the default activation group.

2.12.5 Activation Group Recommendations

Application start-up program or programs should be created to run in NAMED activation group or groups such that programs subsequently called can safely specify ACTGRP(*CALLER). Since activation group names can only be changed by re-creating the program, this approach minimizes your effort in case of a change.

Consider changing the default value to *CALLER using the Change Command Default (CHGCMDDFT) command.

Your application design should ensure that ILE programs created with ACTGRP(*CALLER) are not called from the default activation group. When an ILE program is run in the default activation group, its files may be closed by the RCLRSC command. The RCLRSC command may be issued by some other application running in the default activation group and, therefore, outside the control of your application.

When an ILE program is run in a named activation group, its files may only be closed by the RCLACTGRP command or by the use of the RPG IV SETON LR operation just as it functions today for OPM.

2.12.6 Differences Between Default and Non-Default Activation Groups

Any ILE application design must consider whether programs should be allowed to run in the default activation group or not.

The cause for an ILE program running in the default activation group is ONE of the following reasons:

- The program was created with the ACTGRP(*CALLER) attribute, and its caller is running in the default activation group.
- The program is running in compatibility mode.

The default activation group is provided for running OPM programs and ILE programs with OPM compatibility mode. You should **not** design new ILE programs to run in the default activation group; for a new application you should use a named activation group.

We now summarize the main differences between running an ILE *PGM created with ACTGRP(*CALLER) (thus not created with OPM compatibility mode) in the default activation group versus running it in a non-default activation group (for example, user-named or *NEW).

1. Exception handling

   If you do not handle exceptions in your application, an extra CEE9901 message is generated when you run the *PGM in the default activation group.

2. Scoping

   OVRSCOPE and OPNSCOPE on the OVRDBF and OPNDBF commands default to *ACTGRPDFN. Thus, when you open your files as SHARE(*YES), the file is opened scoped to the call level in the default activation group; it is opened scoped to the activation group level in a non-default activation group.

3. RCLRSC

This command cannot be run in a non-default activation group. It causes files to be closed for ILE procedures run in the default activation group. This command also works differently for ILE compatibility mode programs to ILE non-compatibility mode programs.

4. Static Storage

   All RPG variables are kept in static storage. Once an ILE procedure has been run in the default activation group, the static storage associated with its variables are not returned to the system until end-of-job (the program is not deactivated until end-of-job).

2.12.7 Migration to ILE from the Original Programming Model

To take advantage of the enhancements provided with the RPG IV language definition, you might consider converting your existing RPG application. As you can appreciate, RPG IV gives you more flexibility and better readability, and as a result of these advantages, you can improve your development environment and productivity.

An important point is that the new design of RPG IV opens the door for future enhancements and functions that were impossible to implement on the previous compiler. The reason was the restrictions in the layout on the different specifications statements. Increasing the length for some of the entries, introducing keywords, and free format operations in the extended calculation specifications lifted these limitations.

IBM offers you a simple control language command to convert your RPG/400 or RPG III source members to RPG IV source members.

Before you start migrating your RPG source members, you might want to create a new source file to store the RPG IV sources. The default name for the RPG IV source file is QRPGLESRC. It is recommended that you use this name because all of the CL commands use this name as the default when they refer to the RPG IV source file.

When you are creating the target source file using the command CRTSRCPF for RPG IV members, specify 112 for record length. Twelve characters are for sequence numbers and date fields. The additional 100 characters receive your source code. The length of the statement field has been increased to include the new size of the entries in the specifications layouts. If you leave the default size, you lose the commenting text entries.

If you are using variant characters in your program, make sure that the CCSID of the source file is specified correctly. The CCSID of a new source file created on V3R1 is taken from the default CCDID job attribute unless a CCSID is specified on the CRTSRCPF command.

There is also a new source type, RPGLE. The SEU and PDM recognize the new type as an RPG IV source. If the source type is RPGLE, SEU performs the appropriate syntax check for RPG IV statements.

In the Work with Members Using PDM display, there is a new option (15=Create module). Using this option, if the source type is RPGLE, PDM submits to job queue to run the CRTRPGMOD command to create an ILE RPG module. Using option 14, if the source type is RPGLE, PDM submits to job queue the

CRTBNDRPG command to create a bind RPG program with only one module. More about these commands appears later in this chapter.

RPGLE is also the attribute type for an ILE RPG program and for module objects.

Note

The conversion command from RPG/400 to RPG IV source members automatically renames the source type to RPGLE.

2.13 IFS - Integrated File System Considerations

This section discusses the IFS (Integrated File System) support available on the AS/400 system. The IFS is a base support for Internet application development on the AS/400 systems.

2.13.1 Short Overview of Various File Systems in IFS

A file system provides the support that allows users and applications to access specific segments of storage that are organized as logical units. These logical units are files, directories, libraries, and objects.

Each file system has a set of logical structures and rules for interacting with information in storage. These structures and rules may be different from one file system to another. In fact, from the perspective of structures and rules, the OS/400 support for accessing database files and various other object types through libraries can be thought of as a file system. Similarly, the OS/400 support for accessing documents (which are really stream files) through the folders structure may be thought of as a separate file system.

The integrated file system does indeed treat the library support and folders support as separate file systems. Other types of file management support that have differing capabilities are also treated as separate file systems. The file systems are:

"root"	The / file system. This file system takes full advantage of the stream file support and hierarchical directory structure of the integrated file system. The root file system has the characteristics of the Disk Operating System (DOS) and OS/2 file systems.
QOpenSys	The open systems file system. This file system is compatible with UNIX-based open system standards such as POSIX and XPG. Similar to the root file system, it takes advantage of the stream file and directory support that is provided by the integrated file system. In addition, it supports case-sensitive object names.
QSYS.LIB	The library file system. This file system supports the AS/400 library structure. This file system provides access to database files and all of the other AS/400 object types that the library support manages. This is also the only file system where AS/400 executable objects can be stored.

QDLS	The document library services file system. This file system supports the folders structure. It provides access to documents and folders.
QLANSrv	The LAN Server file system. This file system provides access to the same directories and files that are accessible through the LAN Server licensed program. It allows users of the OS/400 file server and AS/400 applications to use the same data as LAN Server clients.
QOPT	The QOPT file system. This file system provides access to stream data that is stored on optical media.
QFileSvr.400	The QFileSvr.400 file system. This file system provides access to other file systems that reside on remote AS/400 systems.
UDFS	The user-defined file system. This file system resides on the Auxiliary Storage Pool (ASP) of the user's choice. The user creates and manages this file system.
NFS	The Network File System. This file system provides the user with access to data and objects that are stored on a remote NFS server. Network file systems can be exported from an NFS server and dynamically mounted by NFS clients.
QNetWare	The QNetWare file system. This file system provides access to local or remote data and objects that are stored on a server that runs Novell NetWare 3.12 or 4.10. A user can dynamically mount NetWare file systems over existing local file systems.

Users and application programs can interact with any of the file systems through a common integrated file system interface. This interface is optimized for the input/output of stream data in contrast to the record input/output provided through the data management interfaces. A set of user interfaces (commands, menus, and displays) and application program interfaces (APIs) is provided for interacting with the file systems through this common interface.

2.13.2 Which AS/400 File System Works Best?

IBM's HTTP server can serve documents from the following file systems:

- The AS/400 library system (QSYS.LIB)
- QDLS
- The integrated file system "root" directory
- QOpenSys

You can also serve files from all other Integrated files systems installed on your AS/400 system (for example, QLANSrv). However, these file systems are more sophisticated to configure and we do not recommend them unless you have a burning need to do so.

We cannot provide a simple answer to this question but we can give you a few things to think about. The good news is that you are not stuck with any one file

system. You can use a combination of file systems to apply the best solution to each situation. The following list contains some of the primary considerations:

- **Performance** is usually a primary consideration. Our tests show that in a typical configuration, serving from the root directory is the fastest. Here are the relative numbers (your mileage may vary):

  1. Integrated File System - ″root″ directory = 1
  2. Integrated File System - QOpenSys directory = 1
  3. QDLS = .75
  4. AS/400 file system = .45
  5. QLANSrv = .22

- **Environment** is also important. How do you intend to create and maintain your HTML files? With Client Access, it is easy to move files from your PC to a directory in the root or QDLS. You can take advantage of the cool PC tools for maintaining HTML files. On the other hand, you can use SEU and create your files right on a ″green screen″ terminal. It is not a lot of fun but it allows you to utilize data entry people that are familiar with editing file members.
- **Flexibility** is an issue if you want to transport your pages from one platform to another. A typical situation is that your boss wants to load your Web pages on a lap top and show it off to people higher on the food chain. If you keep file names to eight characters with three-character extensions, use relative path names for links, and store the files in a directory structure under the Integrated File System root, you can easily replicate the structure on a PC. If you foresee any potential for needing to run your site in demo mode (customer visits, trade shows, Mom and Dad come for a visit), we highly recommend you consider the Integrated File System.
- **Applications** may also affect your decision. For example, if you have a CGI-BIN application that interacts with an HTML page, you may want to keep that page in the same library as the program. This keeps the entire application in one place.

  We use several library systems. Most of our HTML files are in a directory structure under the Integrated File System root (performance and file maintenance). We keep all of our CGI-BIN programs and related HTML files in a single library (simplify moving things from our test system to the production server).

The following figure shows the complete Integrated File System structure.

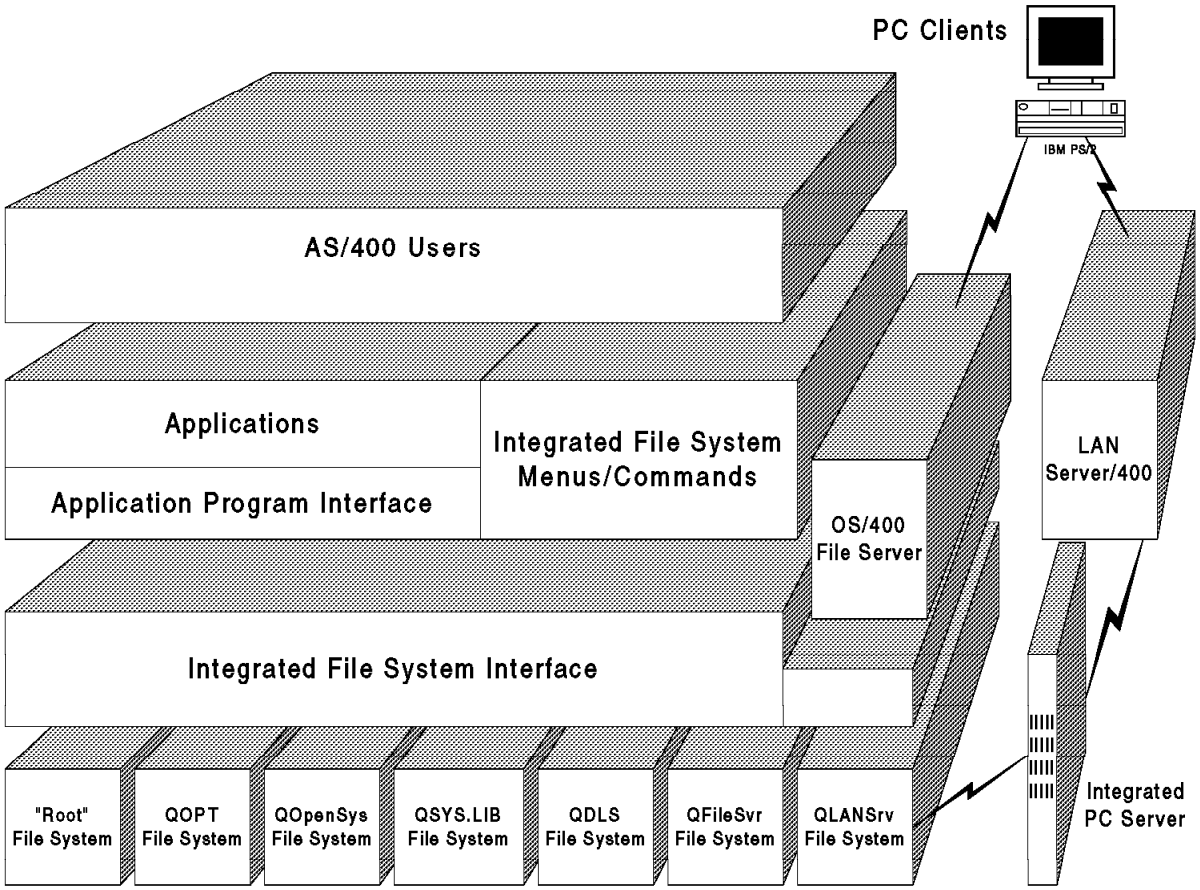


Figure 13. AS/400 File Systems, File Server, and Integrated File System Interface

The OS/400 file server also uses part of the Integrated File System interface. This file server provides file serving capabilities equivalent to shared folders, but allows PC clients to access information in any of the file systems. The PC clients use their own interfaces to interact with the file systems rather than the Integrated File System user interfaces and APIs.

PC clients using the LAN Server requestor or Novell NetWare requestor interact with the Integrated PC Server (File Server I/O Processor) directly rather than the Integrated File System interface. The Integrated PC Server provides even higher rates of access to its files.

2.13.3 How to Exchange Data Between the Root or QOpenSys and QSYS.LIB

There are two commands available to transfer data between the AS/400 library file system and any other IFS file system:

- CPYFRMSTMF - Copy from Stream File
- CPYTOSTMF - Copy to Stream File

However, you can also use your file manager in OS/2 or in Windows95 and Client Access to copy, edit, and manage your files in the IFS on the AS/400 system.

For more information, refer to the following topics and publications:

- OS/400 file server:

  OS/400 Server Concepts and Administration, SC41-3740

- LAN Server:

  LAN Server for OS/400 Administration, SC41-3423

- NetWare:

  OS/400 NetWare Integration Support, SC41-4124.

2.14 Java Programming Considerations

This section discusses how different Java programming techniques can further enhance your AS/400 HTML pages.

2.14.1 Java Scripts being Served from AS/400 System

Any Java Scripts created on any platform can be stored and served from the AS/400 system to any client requesting the script.

There are no special things to take into consideration for serving Java Scripts. The following information is a Java Script test page found on the Internet at:

`http://www.software.ibm.com/data/net.data/demos/demopage.html#jsl`

This test page can be used to explore the capabilities of JavaScript and you can copy the code and implement it into your application. Please give it a try!

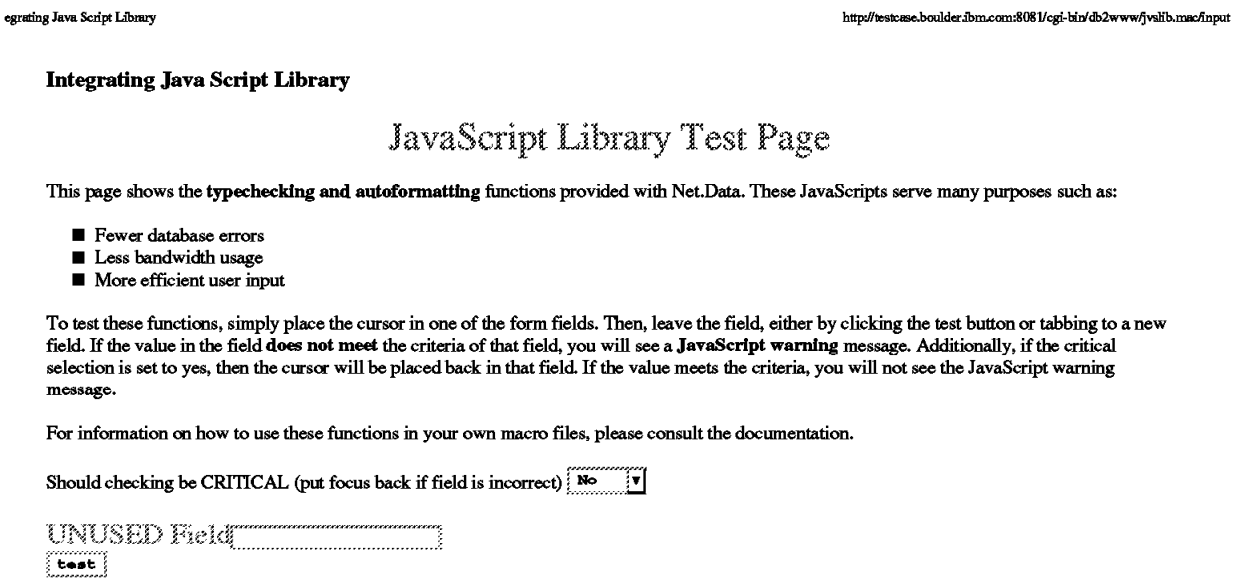


Figure 14. JavaScript Library Test Page (Part 1)

Note

Although it says in the preceding text that these functions are provided with Net.Data, these JavaScripts can be used in conjunction with CGI.bin programs as well.

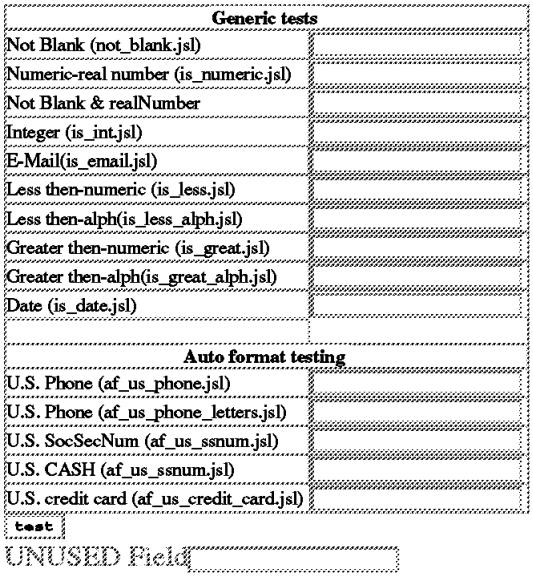


Figure 15. JavaScript Library Test Page (Part 2)

Reference

For more Information on Java Scripts, see Chapter 6, “Further Enhancing Your AS/400 HTML Pages” on page 221.

2.14.2 Java Applets being Served from AS/400 System

Any Java applet created on any platform can be stored and served from the AS/400 system to any client requesting the applet.

There are no special things to take into consideration for serving Java applets.

Reference

For more information on Java Applets, see Chapter 6, “Further Enhancing Your AS/400 HTML Pages” on page 221.

2.14.3 Java Applications being Executed on AS/400 Java Virtual Machine

A prototype of the Java virtual machine for the AS/400 system is available as a technology preview that you can download from:

`http://www.ibm.com/java/`
`http://ncc.hursley.ibm.com/javainfo/download/index.html`

2.14.3.1 What Is a Technology Preview?

This technical preview is an early demonstration of Java technology on the AS/400 system. It is intended to give developers a chance to ″try-out″ Java on an AS/400 system. This technology preview is a port of Sun’s JDK 1.0.2 and can be used for prototyping or testing Java applications on the AS/400 system. This technology preview does **not** include a JIT (Just-In-Time) compiler.

2.14.3.2 Capabilities of the Current Technology Preview

The technology preview provides run-time support for Java on the AS/400 system. Development of Java applications can be done on any Java workstation, and be ported onto the AS/400 system without modification. Alternatively, the javac compiler is also being made available in this release of the technology preview. This allows the compilation of Java code on the AS/400 system.

Even though the Java byte code compiler is included in Technology Preview T1, the recommended way to develop Java code for the AS/400 system is to create and compile the code on a PC based client machine and transfer it to OS/400. The best way of doing this is to use Client Access/400 because it provides an easy way of transferring files between your client and OS/400. It also provides the ability to use long file names in mixed case using the Integrated File System. You cannot access the QOpenSys file system when using Client Access/400 for Windows.

2.14.3.3 Performance

This technology preview does not include any Just-in-Time (JIT) or other optimizing compiler technology. The performance characteristics are similar to other initial Java offerings based on the SUN JDK 1.0.2. Furthermore, the JDK runs on top of OS/400 and is not integrated into the operating system.

Therefore, it is expected that the performance of the Technology Preview release of Java for the AS/400 system will be slower than what is seen on comparable platforms with JIT compilers or other optimizing technology. But as is always true when performance measurements are made, ″your mileage may vary″. Performance is highly dependant on the application being run.

Read the Java for the AS/400 White Paper to learn more about planned performance improvements.

2.14.3.4 White Paper

Reference

A white paper on Java for the AS/400 system is available that provides information on the future strategy and direction for Java on the AS/400 system. Please see Chapter 7, “AS/400 Internet Technology Preview” on page 225.

2.14.4 Using NetRexx to Create Java Applets

This section discusses the support provide by NetRexx for creating Java Applets.

2.14.4.1 What Is NetRexx?

IBM is actively investigating alternatives to the Java language for programming the Java environment. The most advanced of these is NetRexx, an experimental dialect of REXX that can be as efficient as languages such as Java while preserving the low threshold to learning and the ease of use of REXX and Object REXX. It is anticipated that the technology developed during the research of NetRexx can be applied to some future version of Object REXX.

Object REXX can also play its role in network-centric computing; it is well positioned to be used for developing so-called intelligent agents. Intelligent agents are programs that make decisions, filter information, or obtain knowledge for their owners based on information ranging from a simple user profile to

complex, systematized rules and directives. Agents are especially suited for things such as stock brokering, shopping in ″electronic marketplaces″, and other tasks that require rummaging around networked environments.

NetRexx is a new human-oriented programming language designed as an effective and simple alternative to the Java language. With NetRexx, you can create programs and applets for the Java environment faster and more easily than by programming in Java. Using Java classes is especially easy in NetRexx as the different types of numbers and strings that Java expects are handled automatically by the language.

Inspired by two different programming languages, Rexx and Java, NetRexx blends the easy-to-learn syntax of Rexx with the robustness and portability of the Java environment. The result is a language that is tuned for both scripting and application development and is, therefore, truly general-purpose.

The initial implementation of the language is a compiler that first translates the NetRexx source code into Java source code; a Java compiler is used to generate the Java byte codes (class files) for execution. NetRexx classes and Java classes are entirely equivalent (NetRexx can use any Java class and vice versa).

Initial measurements using the current implementation suggest that the Java source for a typical class has approximately 35% more lexical tokens and requires 20% more keystrokes than the equivalent in NetRexx.

The NetRexx compiler (NetRexxC) is written in NetRexx and should run on any Java platform that supports the Java toolkit and compiler (javac). By default, NetRexxC automatically calls the javac compiler to create class files but you can use other Java compilers if you want (the generated Java source is accessible).

For samples (and examples of using Java classes from NetRexx) and for more formal details of the language, please see the other NetRexx documents at

`http://www2.hursley.ibm.com/netrexx/`

You can find the NetRexx packages to download there also.

2.14.5 Using Perl to Create Java Applets

IBM and Mortice Kern Systems (MKS) have released a beta of version 5.003 of PERL for the AS/400 system. PERL is a popular scripting language used to create Web Common Gateway Interface (CGI) applications running on a variety of operating systems, including UNIX, Windows 95, and Windows NT™. As part of IBM′s commitment to Web-enable the AS/400 system, IBM contracted with MKS to port PERL to the AS/400 system.

Although IBM could charge for PERL, the original PERL authors guaranteed in their licensing agreement that anyone can freely distribute any version of PERL (and the PERL source code). This is a boon to users, of course, although it usually means that once the initial port is accomplished, users are left on their own for technical support. There are many resources available to PERL programmers from news groups and books.

2.14.5.1 What Is PERL?

PERL's author, Larry Wall, describes PERL this way:

PERL is an interpreted language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information.

It is also a good language for many system management tasks. The language is intended to be practical (easy to use, efficient, and complete) rather than beautiful (tiny, elegant, and minimal). It combines (in the author's opinion, anyway) some of the best features of C, sed, awk, and sh so people familiar with those languages should have little difficulty with it. (Language historians also note some vestiges of csh, Pascal, and even BASIC-PLUS.) Expression syntax corresponds quite closely to C expression syntax. Unlike most UNIX utilities, PERL does not arbitrarily limit the size of your data (if you have the memory, PERL can slurp in your entire file as a single string). Recursion is of unlimited depth. And the hash tables used by associative arrays grow as necessary to prevent degraded performance. PERL uses sophisticated pattern matching techniques to scan large amounts of data quickly. Although optimized for scanning text, PERL can also deal with binary data and can make dbm files look the same as associative arrays (where dbm is available).

Setuid PERL scripts are safer than C programs through a data flow tracing mechanism that prevents many stupid security holes. If you have a problem that ordinarily uses sed or awk or sh, but it exceeds their capabilities or must run a little faster, and you do not want to write the silly thing in C, PERL may be for you. There are also translators to turn your sed and awk scripts into PERL scripts.

The PERL beta run time and source code is distributed as AS/400 save files and should be available for downloading at:

```
ftp://ftp.cs.colorado.edu/pub/perl/CPAN/ports/as400/
```

The public domain software is now available for downloading that allows you to run PERL scripts on the AS/400 system.

The PERL distribution for the AS/400 system is packaged as three SAVF format files:

```
The PERL source, libraries, and test suites (perlsrc.savf) (4.3MB)
A precompiled perl executable with the PERL libraries
and test suites (perlpgm.savf) (4.9MB)
A precompiled perl executable with the PERL libraries
and test suites (perlrisc.savf) for RISC systems (9MB)
```

These files are available at CPAN sites in the directory: ports/as400. Please see the installation notes and the readme which is included in the save file for complete unpacking and installation instructions and any known problems and limitations.

Note:

This is public domain software.

IBM does not provide service and support for running PERL scripts on the AS/400 system. Do not call the IBM support line for assistance. Comments and questions should be directed to:

comp.lang.perl.misc or comp.sys.ibm.as400.misc newsgroups.

Note

For more information about PERL, see:

http://perl.com/perl/

2.14.6 Using VisualAge for Java to Create Applets and Applications

The VisualAge family of software application development tools allows developers to construct applications visually by connecting prefabricated, reusable software components from an expansive library of predefined classes and parts from IBM and other vendors. IBM′s unique visual-construction-from-parts technology will be extended to the Java programming environment, enabling developers to visually build Java ″applets″. ″The new IBM Java tools will extend VisualAge′s powerful capabilities to the World Wide Web, thus accelerating a developer′s path to Java.″

Java is an unparalleled cross-platform programming environment for the Internet, allowing widespread distribution of ″applets″. An ″applet″ created in Java can be downloaded from the Internet to anywhere in the enterprise, immediately delivering critical information to the user. Java ″applets″ also support user interaction, providing for delivery of ″live″ content (that is, immediate computation of data) and manipulation of displays.

VisualAge for Java is the newest member of IBM′s acclaimed VisualAge family of application development tools. Taking advantage of the hottest new programming language in years, VisualAge for Java combines visual construction from parts with the Java language to create a fully scalable, end-to-end, object-oriented, rapid application development environment.

The promise of Java... and the power of VisualAge!

- Take advantage of an object-oriented, rapid application development environment including incremental compile capabilities.
- Build secure, live Web applications using existing enterprise data and perform transactions across multiple architectures and open protocols.
- Implement live logic interpreter tests for your Java logic on the same display.
- Eliminate client configuration. Create ″Ready for Production″ applications or applets.
- Write it once and run it on multiple platforms including Windows 95, Windows NT, OS/2, AIX, the AS/400 system, or MVS.
- Enjoy true team-based environment with versioning for each developer.
- Use standard parts definition including Java beans.
- Get portable applications and reduced application development costs.
- Use an incremental compiler for faster development.

Reference

For more information about VisualAge Web directions, go to:
www.software.ibm.com/ad/visage/webwp.htm.

2.15 Application Security Considerations

The number of servers connecting to the Internet is growing at a phenomenal rate going from a few thousand systems in 1994 to over 252 000 in June 1996. It is estimated that there are over 50 million Web pages on the Internet. This explosive growth of the Internet makes it increasingly an attractive way of doing business. However, this vast quantity of information also makes it an attractive environment for hackers.

Many systems are attacked. The Computer Security Institute released a survey of corporate security specialists in May 1996 where 42% of them indicated they had knowledge of unauthorized use of their systems in the preceding year. It is to be assumed that the remaining 58% of the companies neither had or detected any attack.

A break in, publicly known, can be damaging to your company′s reputation. Would you want to do business with a bank that had reported a system break-in?

Certainly, large and well-known companies are a favorite target for intruders. Being a small company should be of no consolation. Intruders know that small businesses are less likely to be Internet savvy. Tools widely available on the Internet enable new systems to be discovered quickly.

The threat is serious and no company should attach a system to the Internet without understanding the risks involved.

Most of the Internet servers run on UNIX based systems. And it is widely accepted that the AS/400 system has strong security features. Do these capabilities make the AS/400 system immune from attacks from the Internet? Unfortunately, networks in general and TCP/IP protocols and applications used by the Internet in particular have inherent security issues that are problematic to all kind of systems.

Technical insufficiency is one source that makes an Internet server vulnerable. However, bear in mind that another source of threats is human errors such as incorrect configuration or careless handling of passwords. And those risks are independent from any hardware platform.

2.15.1 Related Publications

Securing Your AS/400 from Harm on the Internet, SG24-4929, discusses security considerations when attaching an AS/400 system to the Internet.

Cool Title About the AS/400 and Internet, SG24-4815, helps you to understand how to use and implement functions and features available with OS/400 V3R2 and V3R7 and the TCP/IP Connectivity Utilities/400, also known as Internet Connection for AS/400.

IBM SecureWay™, AS/400 and the Internet, G325-6321, provides general security information when you are thinking about connecting the AS/400 system to the Internet.

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this document.

Manuals

- *AS/400 TCP/IP Configuration and Reference,* SC41-3420
- *Tips and Tools for Securing Your AS/400,* SC41-3300
- *AS/400 Internet SecureWay,* G325-6321

ITSO Publications

- *Cool Title About the AS/400 and Internet,* SG24-4815
- *Building a Firewall with the IBM Internet Connection Secured Network Gateway,* SG24-2577
- *www.security, How to Build a Secure WWW Connection,* SG24-4564
- *TCP/IP Tutorial and Technical Overview,* GG24-3376

Paperbacks (can be ordered through IBM)

- *Implementing Internet Security,* Frederic J. Cooper. Published by New Riders 1995, SR28-5744
- *Internet Firewalls and Network Security,* Kranajit Sijan. Published by New Riders 1995, SR28-5685
- *Network and Internet Security,* Vijay Ahuja. Published by Academic Press 1996, SR23-7465

2.16 Internet URL References for More Information

Do you want to get more information?

For the latest detailed, product-specific information, visit the VisualAge for Smalltalk Web site at URL:

```
http://www.software.ibm.com/software/ad/vastub.html
```

or the VisualAge for C++ site at:

```
http://www.software.ibm.com/ad/visualage_c++
```

or the IBM Internet Connection Family Web site at:

```
http://www.ibm.com/Internet
```

or the IBM AS/400 Internet Workshop Web site at:

```
http://www.as400.ibm.com/workshop/webbuild.htm
```

or the IBM Java Web site at:

```
http://ncc.hursley.ibm.com/javainfo
```

For more information on Lotus products, visit the Lotus site at:

    http://www.lotus.com

Background on the Internet:
     The Internet, Complete Reference, Second Edition;
     by Harley Hahn
     Spinning The Web; by Ford

Demographics & Statistics:
     http://www.commerce.net/information/surveys/

TCP/IP Stuff:
     Networking Personal Computers with TCP/IP; by Craig Hunt;
     O'Reilly & Associates, Inc.

A Beginner's Guide to HTML
     http://www.ncsa.uiuc.edu/demoweb/html-primer.html

A few good books on HTML stuff:
     Teach Yourself Web Publishing with HTML in 14 Days; by Laura Lemay
     HTML for Dummies; by Tittel & James
     HTML Publishing on the Internet for Windows; by Heslop & Budnick

HTML Quick Reference
     http://kuhttp.cc.ukans.edu/lynx_help/HTML_quick.html

AS/400 Reference Manuals:
     AS/400 Security Tips and Techniques, GC41-0615
     AS/400 TCP/IP Configuration and Reference Manual, SC41-3420
     Cool Title About the AS/400 and Internet, SG24-0372
     AS/400 Security Reference, SC41-3302

2.17 Road Map for Internet Application Design

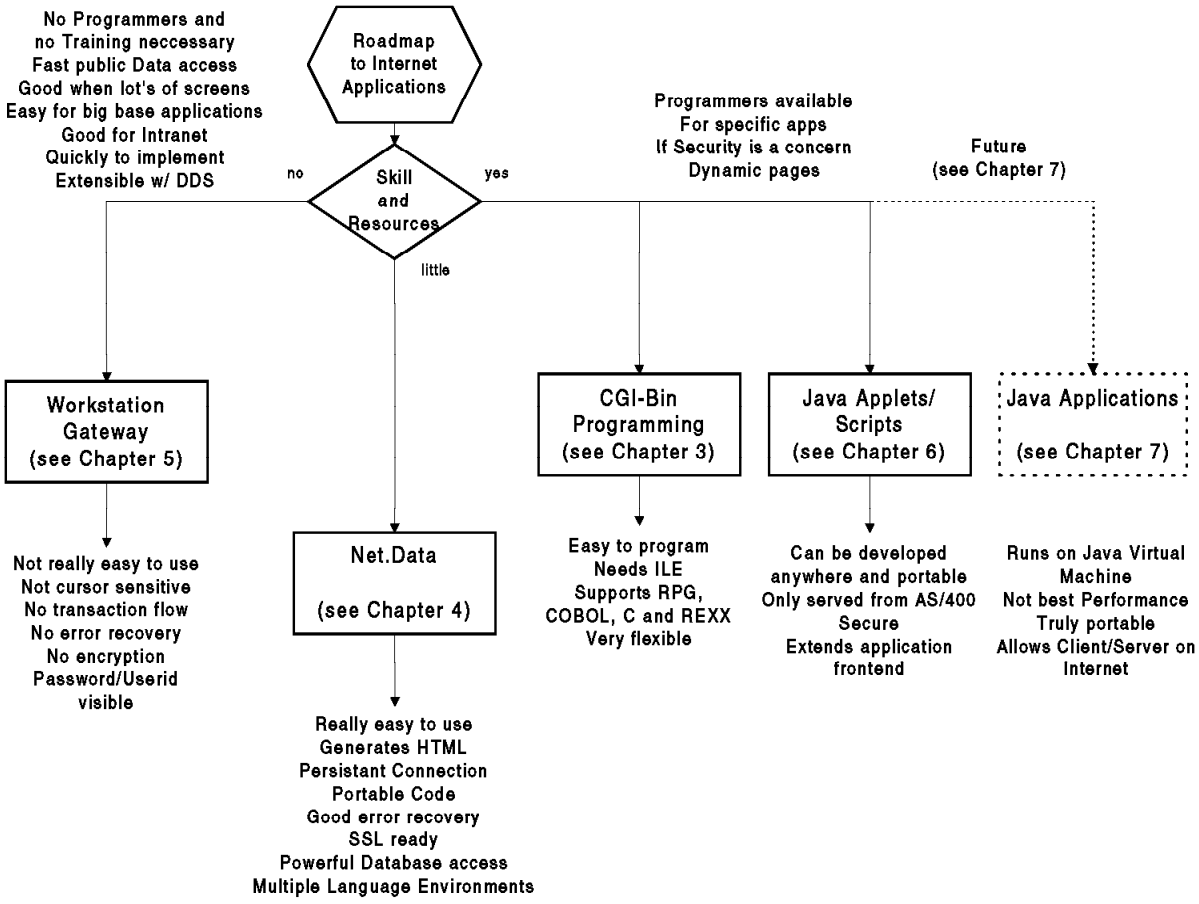


Figure 16. Roadmap for Internet Application Design

The preceding chart can be used as a road map for choosing the right approach for developing your Internet application.

Chapter 3. Common Gateway Interface (CGI-BIN) Implementation

As an owner of an HTTP server on the Internet, you may sometimes see an advantage in allowing the readers of your documents (HTML documents) to return information back to your server, or let the readers retrieve certain kinds of information from your server through the execution of special programs.

The mechanism for doing these or other similar functions is called the Common Gateway Interface (CGI).

A programmer creates a gateway program and places it in the /CGI-BIN directory of the server. A user of a Web page selects the gateway program URL as part of using a Web page link, FORM, or ISINDEX query box. In response to the user request, the gateway program executes and returns the results to the client.

3.1 HTTP CGI Programming Model

The HTTP Common Gateway Interface (CGI) programming model provides a simple interface for running programs external to the HTTP server in a platform-independent manner. This interface has been in use by the World Wide Web since 1993. A good document that describes this interface can be found at URL:

http://hoohoo.ncsa.uiuc.edu/cgi/interface.html

The purpose of CGI is to extend the capability of an HTTP server by providing framework in which the HTTP server can interface with a program that is specified on a URL. The format of the URL allows parameters to be passed to the CGI program. On the server side, the interface describes how the program is started by the HTTP server and how parameters for the program are passed using a combination of standard-input and environment variables. It also describes how output information (usually HTML elements) are passed back to the HTTP server using standard output. Thus, in its simplest form, a CGI program can be defined as a program that:

1. Can be called as an executable and run as a child process of the HTTP server (this does not preclude the use of interpretive languages such as REXX and PERL. In this case, their respective interpreters run as child processes of the HTTP server).

   Note: The CGI program on the AS/400 system runs within the same job as the HTTP server.

2. Is able to read from the standard input.
3. Is able to access environment variables.
4. Is able to write to the standard output.
5. Is able to access ″command line″ arguments passed to main()

CGI programs can be as simple or as complex as the programmer wants to make them, but clearly, the intent of the interface is to allow simple programs or scripts to be developed quickly.

Given this definition of the CGI interface, the following characteristics are for CGI programs:

- They can be written in compiled programming languages or interpreted scripts.
- They can be simple five or six-line programs that serve a specialized function or generalized business applications that do complex calculations and database transactions (for example, order processing applications).
- They are portable. As with any programming model, portability is a matter of degree. To the extent that the program uses any operating system unique interfaces or naming semantics, portability is reduced.
- Since they run as a child process of the HTTP server, they can benefit from the services provided by the server:
  - Communication over the HTTP protocol
  - Security
  - Firewall transparency. HTTP is usually allowed to pass through firewalls (through socks or application gateways).
  - Resource mapping and name indirection provided by HTTP server directives (program can be moved without affecting client URLs).
  - Error logging and audit logging provided by HTTP server
  - Course grained navigation provided by HTML documents
- They provide process isolation. Since each CGI program runs in its own child process, it has the security and integrity advantages that separate processes offer.
- They run in a connectionless environment. The HTTP server starts the CGI program, it does its thing, and terminates. The output is sent to the browser and the connection is broken. Subsequent requests for that same program result in a new connection and a new instance. No state information is maintained unless the CGI program had stored previous state information in fields in the requesting HTML form or parameters in the requesting URL. This characteristic comes about as a consequence of the Web′s hypertext model that allows users to hop around from machine-to-machine across the world at will. Forcing the browser to maintain persistent connections to all of these machines is ridiculous.
- Process start time often gates performance. The price to pay for process isolation is to suffer the costs of process start time. In a connectionless environment, this becomes an even bigger problem as the process must get started for each request. The AS/400 HTTP model minimizes this penalty by providing a pool of pre-started server jobs that minimize a majority of the start-up costs. Add to that the ability to exploit named activation groups that also eliminate program initialization costs, and the AS/400 CGI model starts to look better than what is provided on other platforms.
- They use short-running transactions. This is a consequence of the connectionless model. The length of a transaction is limited to a single URL request since that is the length of a process. This problem brings about the requirement for the persistent CGI model described later.
- The logic is almost entirely on the server. The CGI model is basically a distributed presentation model where the logic is on the server and the client is responsible for window presentation. The advantage of having the logic on the server is apparent when the number of window interactions is far smaller than the number of data accesses or the number of calculations performed by the server.

Although the CGI model is simple and universal, it has some definite shortcomings. This is primarily due to its connectionless nature.

- CGI program parameter data (passed as part of the URL) has portions encoded with escape sequences. It is cumbersome for CGI programs to decode these sequences after they have been converted from ASCII to EBCDIC.
- It is difficult for the CGI programs to determine and control what CCSIDs to use when handling ASCII and EBCDIC data.
- Multiple conversions from ASCII to EBCDIC of input and output data presents some performance problems.

The basic idea of the CGI is illustrated in Figure 17.

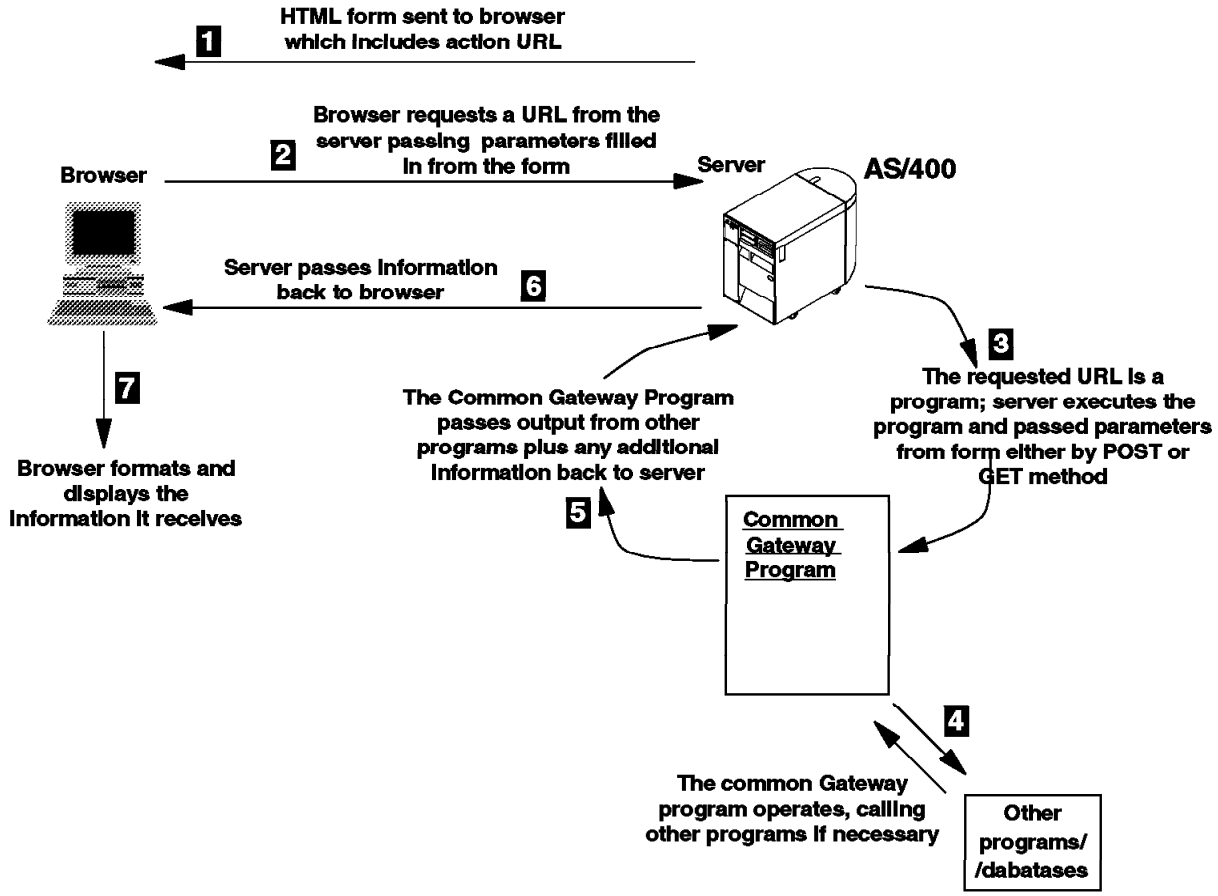


Figure 17. The Basic Idea of How the CGI is Working

1. An HTML form is sent from the server to the Web client. Along with the form is an action URL that points back to the CGI-bin program located someplace on the network.
2. The browser is sending a request for a URL (Uniform Resource Locator) resource from the server. This URL contains the name and path of the CGI-bin application along with all of the form's parameters.
3. The requested URL is a program - a Common Gateway Interface program. The parameters are passed to the application program by one of two ways depending on the form's method of either post or get.

4. The server is executing the CGI-program. While running, the CGI-program is calling other programs or databases.
5. The CGI program passes the output from the other programs or databases plus any additional information back to the server. This is done by writing to the standard output of the application (stdout).
6. The server passes the information back to the browser. This information is probably more HTML or a redirection indicator.
7. It is the browser's job to format the information it receives and display it.

URL

The URL specifies the address or location of a resource, such as CGI programs, on the Internet. The URL looks similar to this:

```
protocol://host_name:port/pathname (?parameters)
```

To do all of this, a set of standards has been set on how to implement CGI programs on your server so every possible Web browser that exists can access your server and execute the CGI program.

3.2 Considerations Before You Start

The CGI program must be in the QSYS.LIB file system, which is the only file system from which you can execute programs on the AS/400 system. The format for a program call is:

Example:

```
/QSYS.LIB/library.LIB/program.PGM
```

The HTTP server does not execute a user-defined CGI program unless the HTTP server administrator has explicitly enabled it by coding an Exec directive. The HTTP server administrator can, for example, limit CGI requests to a specific library in QSYS.LIB.

Example:

If you want the actual library specified where the program is stored:

```
Exec  /QSYS.LIB/CGISAMPLE.LIB/*

That is specified in the URL:
http://hostname/QSYS.LIB/CGISAMPLE.LIB/SAMPLE.PGM
```

If you want the URL mapped into the library where the program is stored:

```
Exec  /cgipgm/*   /QSYS.LIB/CGISAMPLE.LIB/*

That is specified in the URL:
http://hostname/cgipgm/SAMPLE.PGM
```

Note!

The minimum directives that you need to execute a CGI program are a Pass and Exec directive.

The enable and disable directives cause the HTTP server to accept or reject incoming request URLs based on the method coded in the URL. These methods are described in the HTTP/1.0 specification. The AS/400 HTTP server supports the GET, POST, and HEAD methods.

Example:

```
Enable  GET
Disable POST
```

The AS/400 system runs CGI programs under the QTMHHTP1 user profile. The QTMHHTP1 user profile must have authority to access all of the objects accessed by the CGI programs (*PGM).

3.2.1 Required Configuration

You need:

- TCP/IP configuration for the TCP/IP HTTP Server
- Configuration of the AS/400 HTTP Server directives

HTTP Configuration

1. Use the WRKHTTPCFG command to change the HTTP server configuration or CHGHTTPA command to change the HTTP server attributes.
2. Refer to the following books to help you in configuring the HTTP server:
   - *OS/400 TCP/IP Configuration and Reference V3*, SC41-3420
   - *Cool Title About the AS/400 and Internet*, SG24-4815

3.2.2 Overview of the Communication Between CGI Programs and the Server

The AS/400 HTTP server provides the means for a CGI program to read post data from environment variables and standard input stream (stdin). The server also provides the means for a CGI program to return data in the standard output stream (stdout).

Necessary APIs (Application Program Interfaces) are available on the AS/400 system to support the environment variables, the standard input stream, the standard output stream for RPG and COBOL programs, and also for C language programs using SAA® C language library APIs.

The server and the CGI program communicates in four major ways, all supported by the AS/400 system:

- Environment variables
- The command line
- Standard input
- Standard output

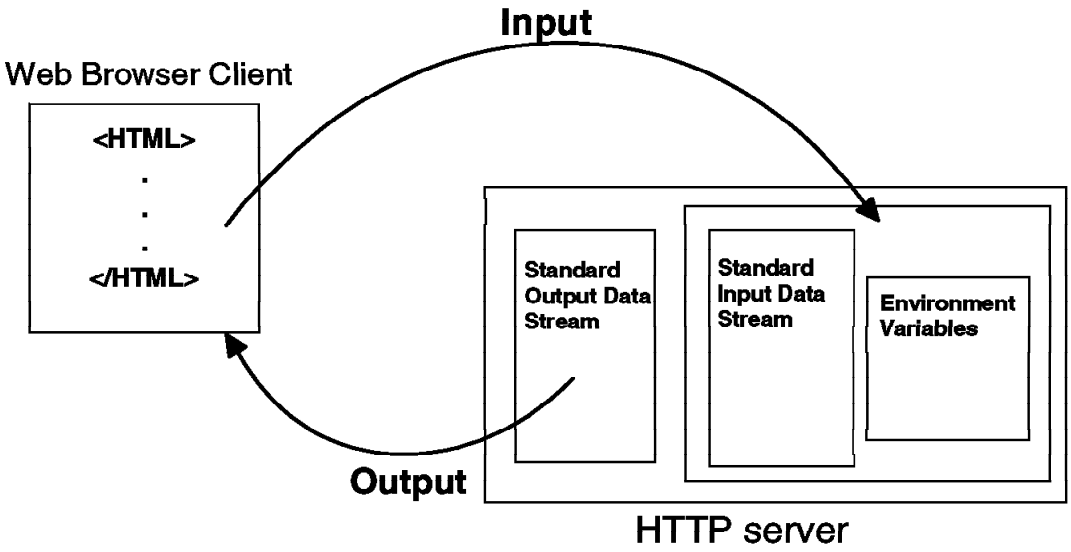


Figure 18. Ways of Communication for CGI Programs

3.3 CGI-BIN Programming

Usually the FORM tag is used to start the CGI program, based on variables selected by the end user. Figure 19 shows an example on how to write an HTML document with the FORM tag.

```
<html>
<head>
<title>Ordering an AS/400 Method=GET</title>
</head>
<body bgcolor="#F8F8FF">
<img align=middle src="as400.gif">
<b>    Ordering an IBM System AS/400 </b>
<p>
<b>Please fill in the following :</b>
<form method="GET" action="/BonusCGI/ORDAS400G.PGM">
<INPUT type="hidden" name="MBR" value="ORDAS400E " size=10>
Name :
<input type=text name="NAME" size="30" maxlength="40" clear="all">
<p>
Address :
<textarea name="ADDRESS" cols=30 rows=2 VALUE=" ">
</textarea>
<p>
<b>Which AS/400 would you like to order ?</b>
<br>
<input name="TYPE" value="P1" type=radio checked>Portable
<input name="TYPE" value="P2" type=radio>Server
<input name="TYPE" value="P3" type=radio>System
<p>
<b>Do you want the Support Line Service ?</b>
<br>
<input name="SERV" value="T" type=radio checked>Yes
<input name="SERV" value="F" type=radio>No
<p>
Please order :
<p>
<input type=submit VALUE="Order">
<input type=reset VALUE="Clear Form">
</form>
</P>
<A HREF="/web/DEMO2E.MBR">Back</A> Demo-Menue
```

Figure 19. The HTML Source using the FORM Tag

When the Web browser displays the HTML document, it is similar to the following figure.

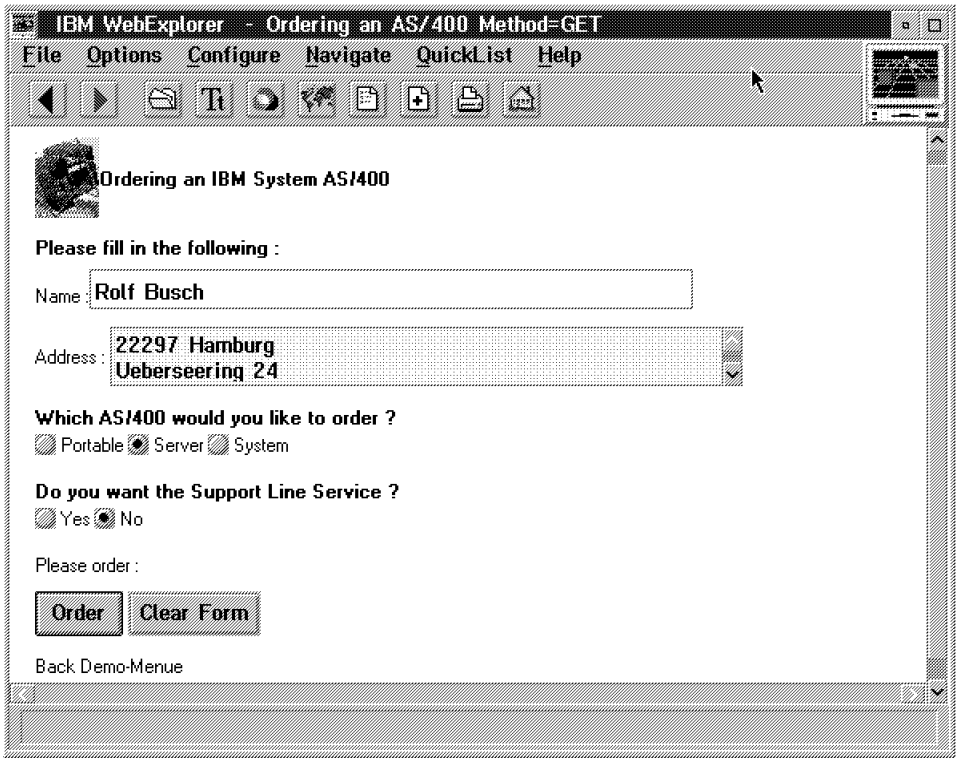


Figure 20. The HTML Document Using the FORM Tag Displayed by the Browser

A Form basically consist of three elements:

1. The Form method:

   ```
   <FORM METHOD="GET" ACTION="/BonusCGI/ORDAS400G.PGM">
   ```

   or

   ```
   <FORM METHOD="POST" ACTION="/BonusCGI/ORDAS400P.PGM">
   ```

2. The Input tags:

   `text, textarea, checkbox, radiobutton, options...`

3. Submit button

All three elements are used in Figure 19 on page 63. Note that the *action* keyword contains the URL address to send the input data to when the Submit button is pressed. In this case, *action="/BonusCGI/ORDAS400x.PGM"*.

A match with an EXEC directive in our HTTP configuration enables the HTTP server to execute a CGI program on behalf of the client.

In our case, action="/BonusCGI/ORDAS400x.PGM" is mapped with:

```
EXEC /BonusCGI/*  /QSYS.LIB/ITSCOIC400.LIB/*
```

Note This!

Please note that only programs in the QSYS.LIB library may be the target for the input data.

This is because only programs in the QSYS.LIB File system may be executed on the AS/400 system.

The following figure shows the HTML output after the execution of the CGI-BIN program.

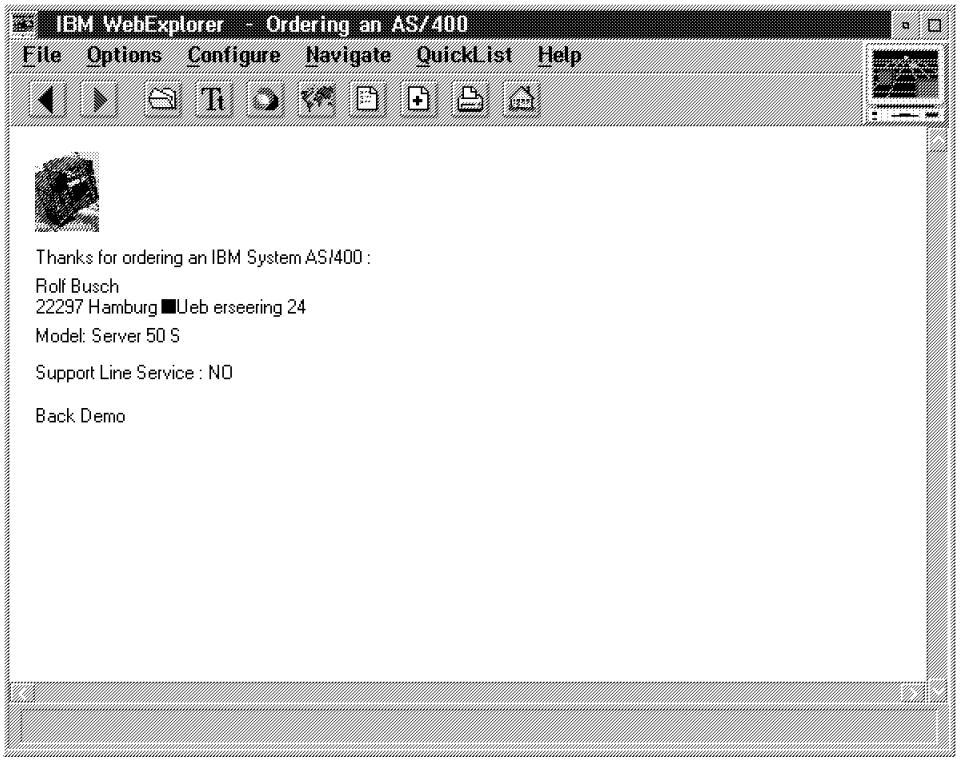


Figure 21. The HTML Output Information Back on the Browser

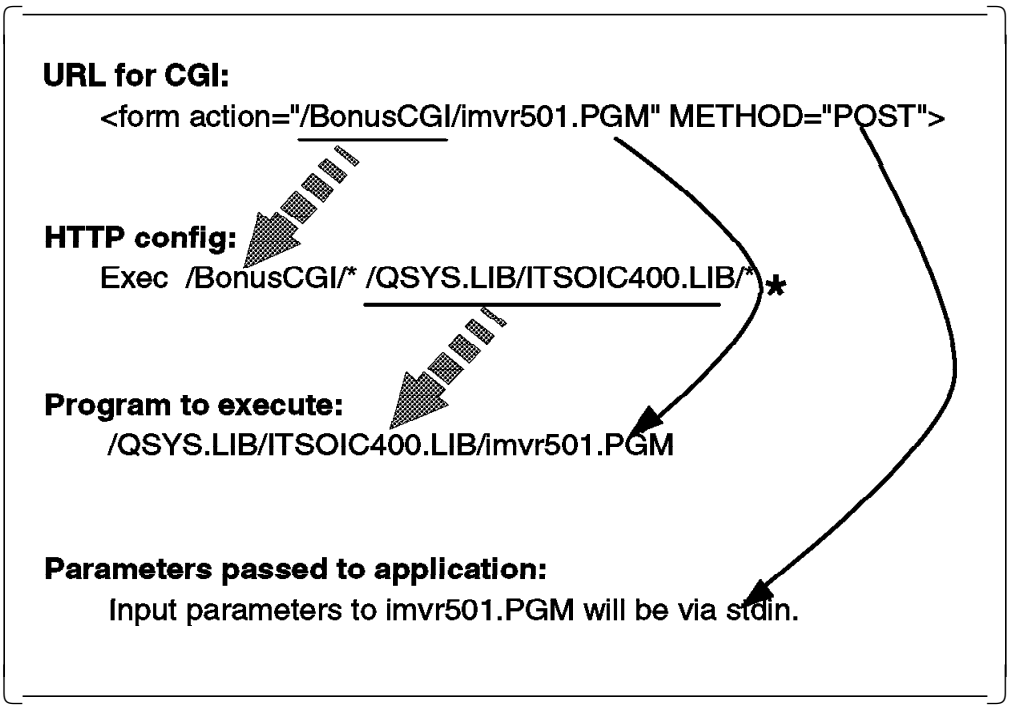


Figure 22. Mapping of CGI URL to HTTP Configuration

There are two *methods* that can be used to access your forms. Depending on which method you use, you receive the encoded results of the form in a different way. The two methods are:

Method	Description
GET	If your form in the HTML document has METHOD="GET" in its FORM tag, your CGI program receives the encoded form input in the environment variable QUERY_STRING.
POST	If your form in the HTML document has METHOD="POST" in its FORM tag, your CGI program receives the encoded form input on stdin. The server does *not* send you an End of File (EOF) on the end of the data. Instead, you must use the environment variable CONTENT_LENGTH to determine how much data you should read from stdin.

Note This!

POST is often the preferred method of operating with CGI-BIN programming.

The reason why POST is preferred is that GET uses the environment variable QUERY_STRING.

Some server platforms are limited in the length of environment variables.

A form with a lot of data can cause a loss of data.

In the POST method, there are no limits (all date is passed through STANDARD INPUT, a way of transferring information in a "stream" without limits).

As you can see, the form method determines the way of passing the encoded input data from the browser to the CGI program through the server. The server and the CGI program communicates in four major ways, all supported by the AS/400 system:

Environment variables	Used for the GET method.
The command line	Used in the case of an ISINDEX query.
Standard input	Used for the POST method.
Standard output	Used to produce output data.

The AS/400 HTTP server provides the means whereby called CGI programs can access environment variables and standard input stream (stdin). The server also provides the means for a CGI program to return data in the standard output stream.

Request for programs in an HTML document can be made using either the GET or the POST method. See Figure 23 on page 69 and Figure 24 on page 71.

Note This!

The AS/400 system runs CGI programs under the QTMHHTP1 user profile. The QTMHHTP1 user profile **must** have authority to access all the objects accessed by the CGI programs.

3.3.1 HTML Form Tags

The following table contains a quick summary of tags and attributes you can use in your HTML documents inside FORM tag.

Table 2 (Page 1 of 2). HTML Form Tags

Tag	Use
<FORM>...</FORM>	A form. You can have multiple forms within a document but forms cannot be nested.
METHOD	An attribute of the <FORM> tag indicating the method with which the form input is given to the script that processes the form input. Possible values are GET and POST
ACTION	An attribute of the <FORM> tag indicating the script to process the form input. Contains a relative path or URL to the CGI program
<INPUT>	A form element
TYPE	An attribute of the <INPUT> tag indicating the type of form element. Possible values are SUBMIT, RESET, TEXT, RADIO, CHECKBOX, PASSWORD, and HIDDEN. SUBMIT creates a button to submit the form to the script that processes the input. RESET creates a button that resets the default values of the form, if any. TEXT creates a single-line text field. RADIO creates a radio button. CHECKBOX creates a check box. PASSWORD creates a single-line text field, but the text is shown in encrypted form. HIDDEN creates a form element that is not shown but has a name and a value that can be passed onto the script that processes the form input.
NAME	An attribute of the <INPUT>, <SELECT>, and <TEXTAREA> tags. Indicates the name of the variable that holds the eventual value of this element as submitted to the script.
VALUE	An attribute of the <INPUT> tag indicating the default value for the form element, if any, or the value submitted with the NAME of the script. For SUBMIT and RESET buttons, VALUE indicates the label of the button.
SIZE	An attribute of the <INPUT> tag used only when TYPE is TEXT. Indicates the size of the text field in characters.
MAXLENGTH	An attribute of the <INPUT> tag used only when TYPE is TEXT. Indicates the maximum number of characters this text field accepts.
CHECKED	An attribute of the <INPUT> tag used only when TYPE is CHECKBOX or RADIO. Indicates that this element is selected by default.
<SELECT>	A menu or scrolling list of items. Individual items are indicated by the <OPTION> tag.
NAME	An attribute of the <SELECT> tag. Indicates the name of the variable that holds the eventual value of this element as submitted to the script.
MULTIPLE	An attribute of the <SELECT> tag indicating that multiple items in the list can be selected.

Table 2 (Page 2 of 2). HTML Form Tags	
Tag	**Use**
SIZE	An attribute of the <SELECT> tag that causes the list of items to be displayed as a scrolling list with the number of items indicated by SIZE visible.
<OPTION>	Individual items within a <SELECT> element.
SELECTED	An attribute of the <OPTION> tag indicating that this item is selected by default.
<TEXTAREA>	A text-entry field with multiple lines.
NAME	An attribute of the <TEXTAREA> tag. Indicates the name of the variable that holds the eventual value of this element as submitted to the script.
ROWS	An attribute of the <TEXTAREA> tag indicating the height of the text field in rows.
COLS	An attribute of the <TEXTAREA> tag indicating the width of the text field in characters.

3.4 GET Method

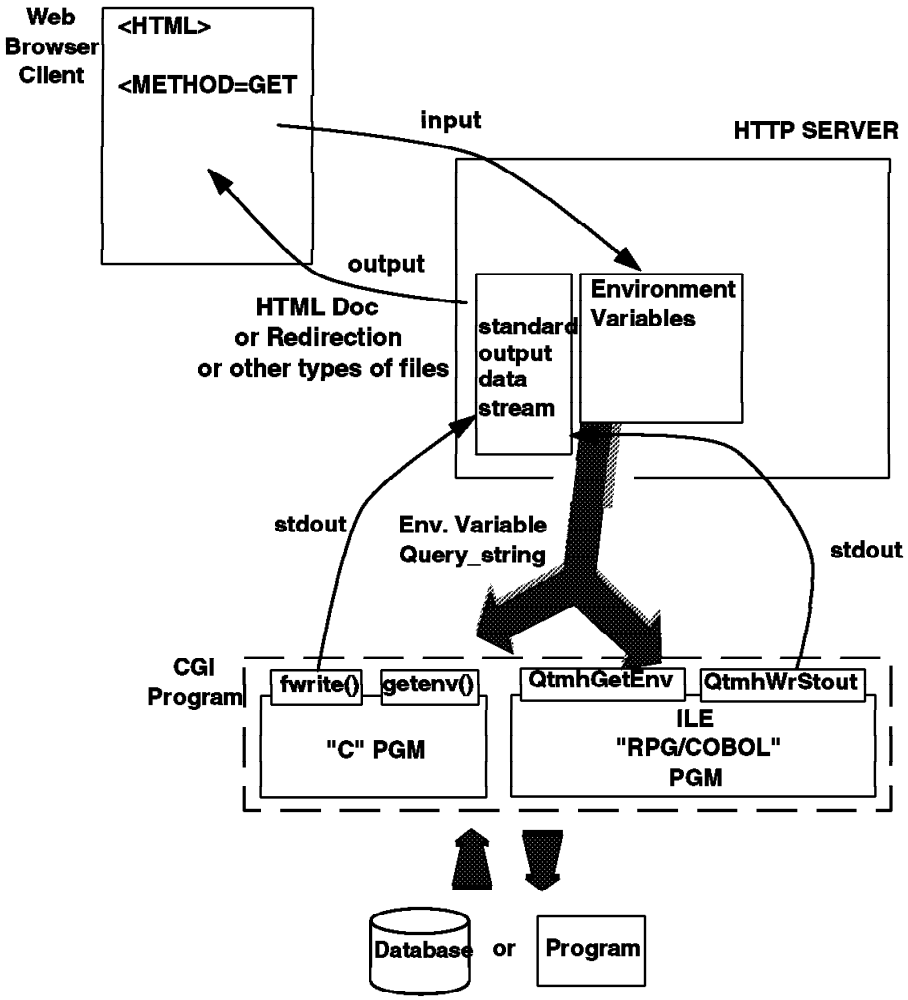


Figure 23. AS/400 CGI Using the GET Method

As you can see in Figure 23, the flow is the same as in the basic chart for CGI Figure 17 on page 59. The HTML document displayed by the browser uses <FORM METHOD="GET".

When the GET method is used, the input parameters are passed to the called CGI program in one of two ways:

1. In command line parameters (program call parameters):

   Parameters are passed to the CGI program using command line arguments (program call parameters) only when the data in the request URL beyond the first question mark (?) contains *no* equal signs (=). This is called an ISINDEX. In the URL for an ISINDEX request, the CGI parameters are positional (being separated by plus (+) signs). The server parses the parameters and puts each parameter, according to its position in the string, into corresponding parameters of the CGI program call.

An example of a URL is:

```
http://sysnm003.sysnmaa.ibm.com/CGILIB/INDEX.PGM?value1+value2+value3
```

2. In the environment variable QUERY_STRING:

   If the parameter string of the URL beyond the first question mark (?) contains an equal sign (=), the entire parameter string is put into the QUERY_STRING environment variable. This is the most normal case.

   An example for the URL is:

   ```
   http://sysnm003.sysnmaa.ibm.com/CGILIB/FORM.PGM?kwd1=value1&kwd2=value2&kwd3=value3
   ```

Note

The assumption is made that CGILIB has been mapped to a library in QSYS.LIB containing CGI programs with the MAP and Exec directives for the AS/400 HTTP server.

The mapping directives allow the Web Administrator on the AS/400 system to define a virtual hierarchy of Web resources that is presented by the AS/400 HTTP server. This virtual hierarchy is independent of the physical file system (or systems) on which the resources are actually stored. This is useful since:

1. It allows resources to be physically relocated without changing the apparent hierarchy and its implied associations.
2. The details of underlying physical file systems can be hidden from client applications that are accessing the AS/400 HTTP server.

The second reason is especially important for the AS/400 HTTP server because of the differences between some of the AS/400 file systems and the tree structured hierarchical UNIX file systems upon which the HTTP protocol and the URL scheme are based.

The mapping directives, MAP and Exec, are used to specify a set of rules that define a process for translating and processing the path (and file) information contained in a URL that is received in a client request URL.

Each of the mapping directives may be specified multiple times and in any combination, and may be placed anywhere in the AS/400 HTTP server configuration. When the AS/400 HTTP server configuration is read during initialization, the mapping directives are stored in order of appearance in a mapping rule table.

The incoming URL is compared against each of the specified mapping rules in turn. If a directive applies, the specified translation is carried out. Depending on the rule that was matched, rule processing is either suspended or continues with the next rule using the newly translated URL.

The mapping and the exec definitions are created through the WRKHTTPCFG command.

3.5 POST Method

When the POST method is being used, the QUERY_STRING environment variable contains a null string. The null string makes the CGI program look for its input in the standard input stream. See Figure 24. The CGI program can determine the method, GET or POST, from the REQUEST_METHOD environment variable.

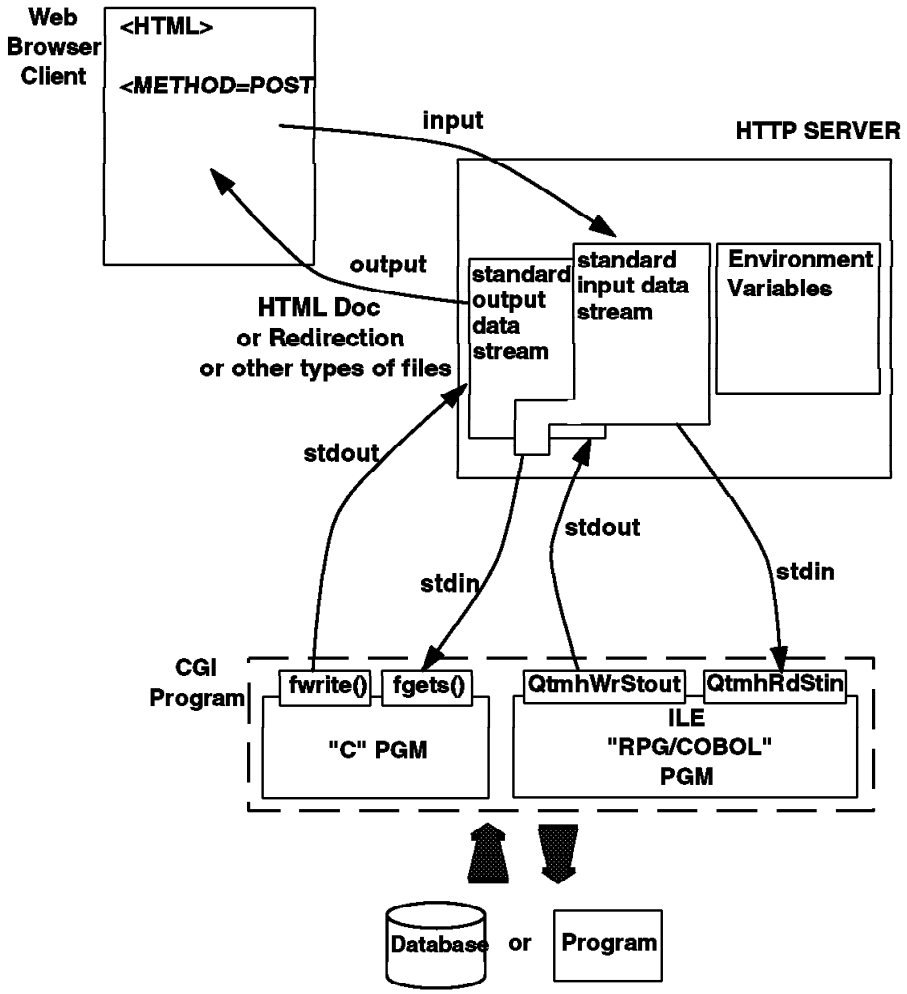


Figure 24. AS/400 CGI using the POST Method

When the HTML document uses the <FORM METHOD="POST", the form is submitted with the standard input data stream, stdin. The CGI program inspects the request method to determine if post data is available by reading the REQUEST_METHOD environment variable. The CGI program can obtain other environment variables set by the HTTP server regardless of the request method.

To support environment variables, the AS/400 HTTP server creates an environment variable user index that contains the name and value for each environment variable set by the server. The index is named QTMHENVI in the QTEMP library.

The reason why it is created in the QTEMP library is that it has to be unique for each child job that defines it. So each child job creates the index that is accessible to the CGI programs that the server child job calls.

Each time a CGI program is called, a new index is created and new environment variables are set (preventing one CGI program from using the environment variables intended for another CGI program). Creation of the index is not dependent on the use of either method, GET or POST. The index is always created.

The QTMHENVI index contains the following environmental variables:

Table 3. CGI Environment Variables in the QTMHENVI Index File

Environment Variable	**Contain**
QUERY_STRING	Information that follows the first question mark (?) in the URL referencing the CGI program.
SERVER_SOFTWARE	The name and version of the information server software answering the request (and running the gateway).
SERVER_NAME	Host name, DNS alias, or IP address of the server.
GATEWAY_INTERFACE	The version of the CGI specification to which the server complies.
SERVER_PROTOCOL	The name and revision of the information protocol this request came in with.
SERVER_PORT	The port number to which the request was sent.
REQUEST_METHOD	The method with which the request was made. For HTTP, this is ″GET″ or ″POST″.
PATH_INFO	The extra path information following the path information required to identify the CGI program name.
PATH_TRANSLATED	The server provides a translated version of PATH_INFO, which takes the path and does any virtual-to-physical mapping to it.
SCRIPT_NAME	A virtual path to the program being executed.
REMOTE_HOST	The host name making the request.
REMOTE_ADDR	The IP address of the host name making the request.
REMOTE_USER	This is the user name making the request.
CONTENT_TYPE	For queries that have attached information, such as HTTP POST, this is the content type of the data.
CONTENT_LENGTH	The length of data in the attached HTTP POST from the client.
IBM_CCSID_VALUE	The CCSID under which the current server job is running.
HTTP_ACCEPT	MIME content types the browser accepts.
HTTP_USER_AGENT	String identifying the Web client. Includes name and version of the browser request made through a proxy and other information.
HTTP_REFERER	Name of HTML document

When the CGI program has finished executing, it is expected to return an HTML document or a redirection to an HTML document.

The CGI program passes output from other programs or databases plus any additional information back to the server. To do that, the CGI program uses the standard output data stream, stdout.

3.6 AS/400 Programming Languages Supported

CGI programs are often called CGI scripts, but you can develop your own CGI programs in many languages (not only in scripting languages). To select your programming language, choose the one to use according to:

- The task your application has to perform
- Your programming skills

There is support for CGI programs written in:

- C language
- ILE RPG
- ILE COBOL
- REXX (See Section 3.10, “Programming CGI with REXX Language” on page 93 for considerations and restrictions.)

Necessary APIs are available to support the environment variables, the standard input stream, and the standard output stream.

The C language supports the environment variables through the `getenv()` API. The standard input stream and the standard output stream are supported through many APIs. Examples are `fgets()` and `fwrite()`.

For the C programs, the APIs are found in the `QTMHCGI *SVCPGM`. A C language header file member named `QTMHCGI` is provided in `H` file in library `QSYSINC`.

The APIs to support ILE RPG and ILE COBOL are provided in source files. The source files are:

- QCBLLESRC for ILE COBOL
- QRPGLESRC for ILE RPG

Four APIs are provided:

Table 4. The APIs Provided for ILE RPG, ILE COBOL, and C Programs

Type of API	ILE RPG	ILE COBOL	C Language
Get Environment Variable	QtmhGetEnv	QtmhGetEnv	getenv()
Read from Stdin	QtmhRdStin	QtmhRdStin	Many, for example fgets()
Write to Stdout	QtmhWrStout	QtmhWrStout	Many, for example fwrite()
Parse CGI input	QtmhCvtDb	QtmhCvtDb	#pragma mapinc

Note This for RPG and COBOL!

For the **CRTPGM**, you **must** always specify the BNDSRVPGM **QTCP/QTMHCGI** to ensure the module import for QtmhGetEnv, QtmhRdStin, QtmhWrStout, and QtmhCvtDb are satisfied. If you forget to include the SRVPGM, you receive a message similar to the following example:

```
Definition not found for symbol 'QtmhRdStin'.
```

Case Sensitive Alert!

The service program APIs of QtmhGetEnv, QtmhRdStin, QtmhWrStout, and QtmhCvtDb are case sensitive. If you misspell one of them or if the case is wrong for just one of the letters, you receive the following message after your CRTPGM (the binding step) fails:

```
Message ID . . . . . :   CPD5D1D       Severity . . . . . :   20
Message type . . . . . :   Diagnostic
Date sent  . . . . . :   10/30/96      Time sent  . . . . . :   14:33

Message . . . . :   *SRVPGM object QZDMMDTA in library QSOC not found.
Cause . . . . . :   *SRVPGM object QZDMMDTA in library QSOC was specified
  binding directory QUSAPIBD in library *LIBL, but was not found for bind
Recovery  . . . :   Contact your application provider or service
  representative.
```

This message is seemingly not related at all to the source of the problem.

The APIs are provided to simplify parsing form data from either stdin or from the environment variable QUERY_STRING. The CGI program is expected to provide the name of the DDS file specification that identifies the anticipated form variables and their attributes.

The server times the operation CGI programs and terminates child server jobs exceeding the time limit. There are three timeout keywords for input timeout, output timeout, and script timeout.

Also, the server protects itself from CGI program exceptions by providing exception handling that reduces any escape messages resulting from CGI program failures to diagnostic messages.

3.7 Decoding the Parameters from the Remote Web Client

You now know that depending on the HTML form's method, which can be either POST or GET, the input string to your application is made available to your CGI application running on the AS/400 system in either standard in or the QUERY_STRING environment variable. If your application is in ILE C/400, you have native I/O statements to code to retrieve the parameter string. If your application is in ILE COBOL/400 or ILE RPG/400, IBM provides a service program to do the job of reading the parameter string and placing it in a local program variable.

What we have not covered is how this parameter string looks and what exactly you must do to parse it into something that you can use in your application. How to parse the string is dependant upon if you are writing your CGI application

in C/400 (see Section 3.8.2, “CGI Parameter Parsing with ILE C/400” on page 77), or in either RPG/400 or COBOL/400 (see Section 3.9.2, “CGI Parameter Parsing with ILE COBOL/400 or RPG/400” on page 86).

But first, let’s spend a little bit of time to understand the basic syntax of the parameter string that we receive from a Web client.

3.7.1 CGI Parameter String Syntax

When you write the HTML form, each of your input items has a NAME tag associated with it. As an example, take a look at the FORM tag (see Figure 25). The first input field is a hidden field, which simply means that the end user does not see this field nor is the user given the opportunity to modify the value associated with it. The second named field is ″querydata″, which is presented to the end user with the possibility to update. The input that is entered by the end user for each input field is called the value.

```
<FORM METHOD=”GET” ACTION
  =”/BonusCGI/CGIENVGET.PGM”>
<INPUT type=”hidden” name=”MBR” value=”CGIENVGETE” size=10>
<INPUT TYPE=text NAME=”querydata” MAXLENGTH=”20” VALUE=” ”>
<BR>
<BR>
<INPUT TYPE=”SUBMIT” VALUE=”Send Request”>
<INPUT TYPE=”RESET” VALUE=”Clear Your Input”>
</FORM>
```

Figure 25. Two Named Input Fields Defined in the HTML Form

It is this stream of name=value pairs that flow back to your CGI application running on the AS/400 system when the user selects the Submit button with the mouse. These are the name=value pairs that you are reading either from standard in or the QUERY_STRING environment variable.

One more thing must be mentioned. A well-behaved Web client can modify the format of the parameter string. Each name=value pair is separated by the ampersand (&) character. Also, each name=value pair is URL encoded, which means spaces are changed into pluses (+) and some characters are encoded into hexadecimal.

Again, using the same form shown in Figure 25, let’s assume that the end user enters a value for querydata of:

```
AaBbCc, First three
```

The content of QUERY_STRING, as echoed back to the client by CGI program CGIENVGET, is shown as:

```
MBR=CGIENVGETE&querydata=AaBbCc%2C+First+three
```

Let’s parse this string ourselves:

```
MBR=CGIENVGETE&querydata=AaBbCc%2C+First+three
|              |              |  |     |
|              |              |  Note : Spaces become plus (+) signs
|              |              Special characters, like the comma (,)
|              |              are represented as three ASCII character
|              |              escape sequences representing a hexadecimal
|              |              entry into an ASCII code page.
```

```
|             Second name=value pair.  The querydata name is case
|             sensitive.  The value follows the equal (=) sign.
|
First name=value pair.  The name is MBR and is case sensitive.  The
value follows the equal (=) sign.
```

For more information about the format of the name=value pairs, we suggest that you look on the Web. A good place to start is

`http://hoohoo.ncsa.uiuc.edu/cgi/`

To learn how to parse the name=value pairs coming from the Web client, keep reading.

3.8 Programming CGI-BIN with ILE C/400

The C language supports the environment variables through the `getenv()` API. The standard input stream and the standard output stream are supported through many APIs. Examples are `fgets()` and `fwrite()`.

For the C programs, the APIs are found in the `QTMHCGI *SVCPGM`. A C language header file member named `QTMHCGI` is provided in `H` file in library `QSYSINC`.

To get the necessary prototypes, include the header file as follows:

```
#include <qtmhcgi.h>
```

Important Compiling Information!

Make sure QSYSINC is on your system before compiling programs that use these header files because the QSYSINC library is optionally installable.

V3R2 Only

If you use the getenv() function, you must include the following header file:

```
#include <qp0z1170.h>
```

The version of getenv() that is in the stdlib.h header file does not work.

3.8.1 Structure of C Program with POST Method

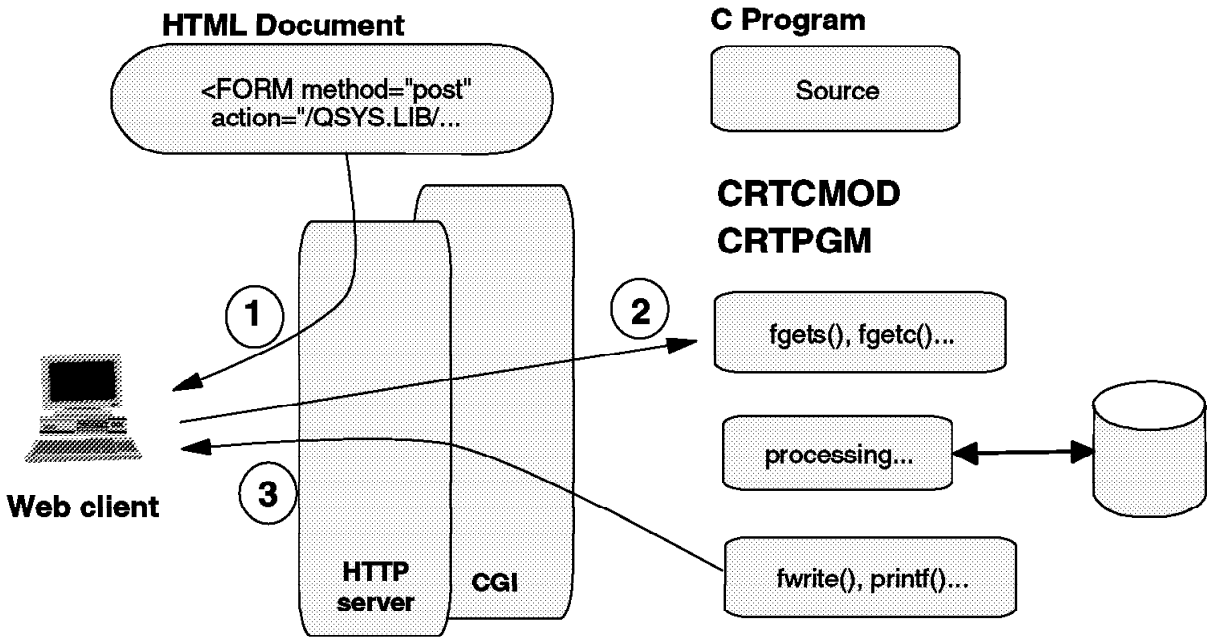


Figure 26. CGI Programming with C Language using POST Method

1. The HTML form is sent to the client. This form includes all of the named fields that the user can click, select, pull down, or type in the answers. The form tag also contains the method (in this case, POST) and the action (which is the URL to which all the input parameters are sent). In the case of the AS/400 system, this URL is your CGI application that runs in QSYS.LIB.
2. The user selects the Submit button. The HTTP server routes the data to the CGI program, which places the named variables and values in Stdin. It calls your CGI application, which can:
   - Read data from stdin using SAA C language library APIs such as fgets() or fread().
   - Processing (anything that you want to do including access to DB2®/400 data).
   - Your CGI application must generate the HTML output that is going back to the client.
   - Write data to stdout using SAA C language library APIs such as fwrite() or printf().
3. When your application ends, whatever you wrote to stdout is sent back to the Web client.

3.8.2 CGI Parameter Parsing with ILE C/400

If you are using ILE C/400 as your CGI application on the AS/400 system, you have one advantage. And that advantage is that many other platforms, namely UNIX based, also use C as a tool for writing CGI. And, if you have not figured it out yet, you soon realize that the CGI interface was born and develop on UNIX styled systems first. So, for the job of parsing the name=value pairs that come from the Web client, we can go to the Web to pull down public domain C code

examples. This is what we did. For this example, please see Section 3.12.3, “Source Code C Program PARSECGIP” on page 108.

We did need to modify the CGI parsing algorithm to work on the AS/400 system. The problem with porting these routines came when the three-character escape sequences (such as %2C that represents the comma (,) character) where being translated into a single character. The code that we found on the Web assumed that the native character set on the system it was running was ASCII. The AS/400 system is EBCDIC, so we needed to modify the routine once we understood the problem.

This, as we see in Section 3.9.2, “CGI Parameter Parsing with ILE COBOL/400 or RPG/400” on page 86, is a problem in more than just this one place.

3.8.3 ILE C Sample Programs using POST and GET Methods

The following ILE C programs are examples of CGI programming using the POST method and the GET method.

3.8.3.1 ILE C Program that Reads from Stdin (POST Method)

In this example, we use a function to read from the standard input stream (stdin) and put the var=value pairs into the entries structure.

```
/*****************************************************
/*           SAMPLE USING POST METHOD                *
/* Define struct for name = value pairs              *
/*****************************************************/
typedef struct {
    char *name;
    char *val;
} entry;
entry entries[10000];

/*****************************************************
/* Read CONTENT_LENGTH environment variable so know *
/* the length of data from the client                *
/*****************************************************/
cl = atoi(getenv("CONTENT_LENGTH"));
for(x=0;cl && (!feof(stdin));x++) {
    m=x;
    /*********************************************
    /* Keep in entries struct name and value pairs *
    /*********************************************/
    entries[x].val = fmakeword(stdin,'&',&cl);  1
    plustospace(entries[x].val);   2
    unescape_url(entries[x].val);  3
    entries[x].name=makeword(entries[x].val,'='); 4
}
```

1 Fmakeword function

This function reads a var=value pairs from stdin.

```
char *fmakeword(FILE *f, char stop, int *cl) {
    int wsize;
    char *word;
    int ll;

    wsize = 102400;
    ll=0;
    word = (char *) malloc(sizeof(char) * (wsize + 1));

    while(1) {
        word[ll] = (char)fgetc(f);
        if(ll==wsize) {
            word[ll+1] = '\0';
            wsize+=102400;
            word = (char *)realloc(word,sizeof(char)*(wsize+1));
        }
        --(*cl);
        if((word[ll] == stop) || (feof(f)) || (!(*cl))) {
            if(word[ll] != stop) ll++;
            word[ll] = '\0';
            return word;
        }
        ++ll;
    }
}
```

2 Plustospace function

This function changes the plus characters to space characters.

```
void plustospace(char *str) {
    register int x;

   for(x=0;str[x];x++) if(str[x] == '+') str[x] = ' ');
}
```

3 Unescape_url function

When you write a special character in the browser, the HTTP server decodes it in the %xx form. The unescape_url() function looks for the special characters and if it finds them, calls the x2c() function to decode them.

```
void unescape_url(char *url) {
    register int x,y;

    for(x=0,y=0;url[y];++x,++y) {
        if((url[x] = url[y]) == '%') {
            url[x] = x2c(&url[y+1]);
            y+=2;
        }
    }
    url[x] = '\0';
}

char x2c(char *what) {
    register char digit;

    digit=(what[0] >= 'A' ? ((what[0] & 0xdf) - 'A')+10 : (what[0]-'0'));
    digit*=16;
    digit+=(what[1] >= 'A' ? ((what[1] & 0xdf) - 'A')+10 : (what[1]-'0'));
    return(digit);
}
```

4 Makeword function

At the end, separate the name of the value.

```
char *makeword(char *line, char stop) {
    int x = 0,y;
    char *word = (char *) malloc(sizeof(char) * (strlen(line) + 1));

    for(x=0;((line[x]) && (line[x] != stop));x++)
        word[x] = line[x];

    word[x] = '\0';
    if(line[x]) ++x;
    y=0;

    while(line[y++] = line[x++]);
    return word;
}
```

3.8.3.2 ILE C Program that Reads from QUERY_STRING (GET Method)

In this example, the GET method reads from QUERY_STRING using a getword() function.

```
    /*****************************************************
    /*           SAMPLE USING GET METHOD                 *
    /* Define struct for name = value pairs              *
    /*****************************************************/
    typedef struct {
        char name[128];
        char val[128];
    } entry;
    entry entries[10000];

    /*****************************************************
    /* Read QUERY_STRING environment variable            *
    /*****************************************************/
    cl=getenv("QUERY_STRING");
    if( cl!=NULL ){
        for(x=0 ; cl[0]!='\0' ; x++){
            m=x;
            getword(entries[x].val, cl, '&'); 1
            plustospace(entries[x].val);  2
            unescape_url(entries[x].val); 3
            getword(entries[x].name,entries[x].val,'=');
        }
    }
```

1 Getword function

This function reads from a string and looks for a character (in our sample, ampersand to equal characters). If the character is found, the function stops and returns the string.

```
void getword(char *word, char *line, char stop) {
   int x = 0,y;

   for(x=0;((line[x]) && (line[x] != stop));x++)
       word[x] = line[x];

   word[x] = '\0';
   if(line[x]) ++x;
   y=0;

   while(line[y++] = line[x++]);
}
```

2 Plustospace funtion

This function is the same as the POST method. The data format is the same for POST and GET methods; only the way it is received is different.

3 Unescape_url funtion

This function is the same as the POST method.

3.9 Programming CGI-BIN with ILE RPG/400 and ILE COBOL/400

APIs are provided to allow programs written in COBOL and RPG to read stdin, write to stdout, and read the environment variables. These APIs are found in the QTMHCGI service program in the QTCP library.

The APIs to support ILE RPG and ILE COBOL are provided in source files. The source files are:

- QCBLLESRC for ILE Cobol
- QRPGLESRC for ILE RPG

3.9.1 Structure of RPG Program Using POST Method

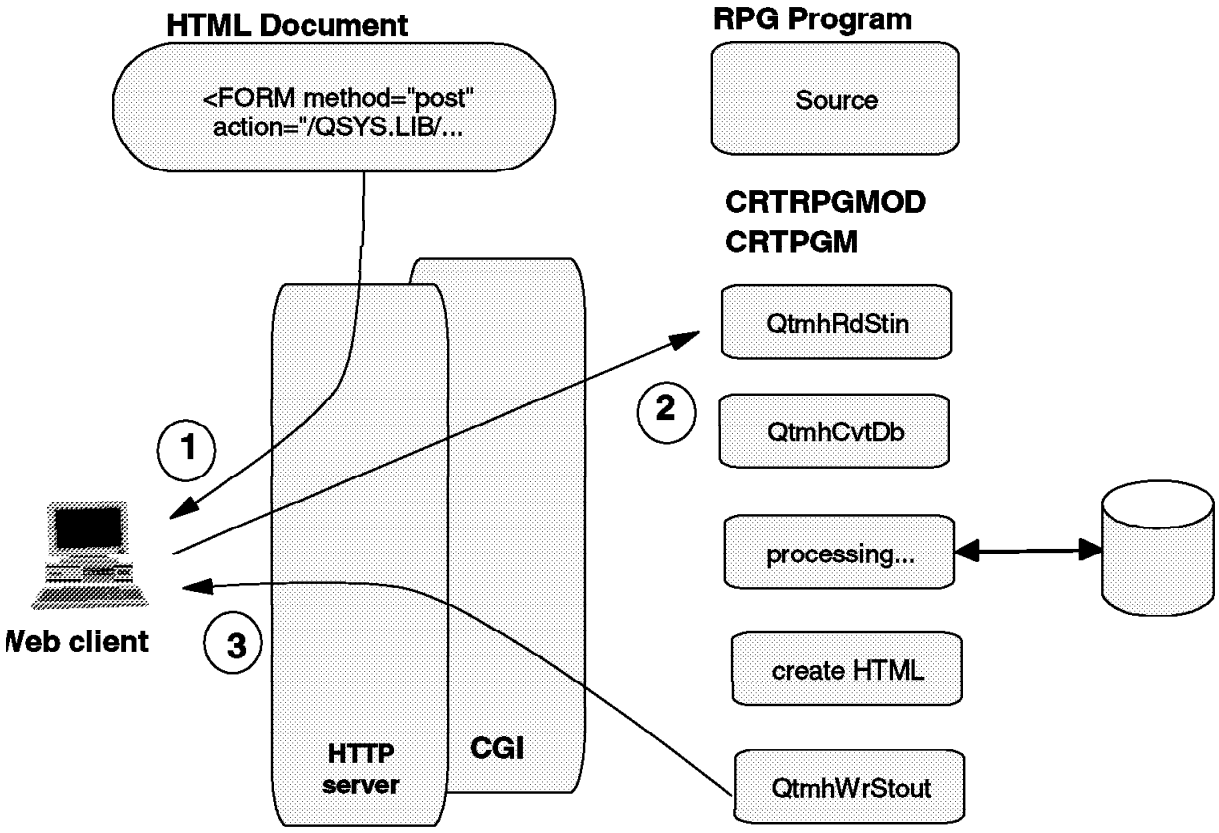


Figure 27. CGI Programming with RPG Language using POST Method

1. It is the same structure as the C program. The HTML form is sent to the client.
2. It is the same structure as the C program. The user selects the Submit button.
   - Call QtmhRdStin, which places the entire input string read from stdin into a receiver variable of type pointer to character.
   - Call QtmhCvtDb, which parses the string of named variables and values and places all of the data into a database file formatted by DDS.
   - Processing (anything that you want to do including access to DB2/400 data).

- The processing of your CGI application must generate the HTML output that is sent back to the client.
- Call QtmhWrStout to write the HTML to standard out (stdout).

3. When your application ends, whatever you wrote to stdout is sent back to the Web client.

Note This for RPG and COBOL!

For the **CRTPGM**, you **must** always specify the BNDSRVPGM **QTCP/QTMHCGI** to ensure the module import for QtmhGetEnv, QtmhRdStin, QtmhWrStout, and QtmhCvtDb are satisfied. If you forget to include the SRVPGM, you receive a message similar to the following example:

```
Definition not found for symbol 'QtmhRdStin'.
```

All APIs use a data structure for error reporting. This data structure is in:

- QSYSINC/QRPGLESRC(QUSEC) and QSYSINC/QRPGSRC(QUSEC) for RPG programs
- QSYSINC/QCBLLESRC(QUSEC) and QSYSINC/QLBLSRC(QUSEC) for COBOL programs

```
DQUSEC            DS
D*                                        Qus EC
D QUSBPRV                1      4B 0
D*                                        Bytes Provided
D QUSBAVL                5      8B 0
D*                                        Bytes Available
D QUSEI                  9     15
D*                                        Exception Id
D QUSERVED              16     16
D*                                        Reserved
D*QUSED01               17     17
D*
D*                                   Varying length
```

Figure 28. QUSEC Data Structure for Error Reporting

- The QtmhGetEnv API allows you to get the value set by the server for a particular HTTP environment variable.

Table 5. Get Environment Variable (QtmhGetEnv) API

Description	I/O	Type	Example
Receiver variable	Output	Char(*)	name=fred&address=center+street
Llength of receiver variable	Input	Binary(4) **1**	1024 max.
Length of response	Output	Binary(4)	31
Request variable	Input	Char(*)	QUERY_STRING
Length of request variable	Input	Binary(4)	12
Error code	I/O	Char(*)	QUSEC data structure (see Figure 28)

Note: Reference at number of bytes, not at number of characters.

The following display shows a piece of RPG code using the QtmhGetEnv API.

```
 * Copy QUSEC structure
 /copy qsysinc/qrpglesrc,qusec
D*QUSEDO1               17    116
 * Variables for QtmhGetEnv
Denvrcvr          S           32767
Denvrcvrln        S             10i 0 inz(%size(envrcvr))
Denvrspln         S             10i 0
Denvrqsnm         S             30
Denvrqsln         S             10i 0
 *
C                   eval      qusbprv = 16
 * Get environment variable
C                   checkr    envrqsnm      envrqsln
C                   CALLB     'QtmhGetEnv'
C                   parm                    envrcvr
C                   parm                    envrcvrln
C                   parm                    envrspln
C                   parm                    envrqsnm
C                   parm                    envrqsln
C                   parm                    QUSEC
```

- The QtmhRdStin API allows CGI programs to read from standard in (stdin).

Table 6. Read from Stdin (QtmhRdStin) API

Description	I/O	Type	Example
Receiver variable	Output	Char(*)	name=fred&address=Center+street
Length of receiver variable	Input	Binary(4)	1024 max.
Length of response	Output	Binary(4)	31
Error code	I/O	Char(*)	QUSEC data structure (see Figure 28 on page 83)

The following display shows a piece of RPG code using the QtmhRdStin API.

```
 * Copy QUSEC structure
 /copy qsysinc/qrpglesrc,qusec
D*QUSEDO1               17    116
 * Variables for QtmhRdStin
DInDataLn         S             10I 0
DInActLn          S             10I 0
Dbuffer           S           9999
 *
C                   eval      qusbprv = 16
 * Standard input subroutine
C                   CALLB     'QtmhRdStin'
C                   parm                    buffer
C                   parm                    INDataLn
C                   parm                    INActLn
C                   parm                    QUSEC
```

- The QtmhWrStout API provides the ability for CGI programs to write to standard out (stdout).

Table 7. Write from Stdout (QtmhWrStout) API

Description	I/O	Type	Example
Data variable	Input	Char(*)	Content-type: text/html
Length of data variable	Input	Binary(4)	25
Error code	I/O	Char(*)	QUSEC data structure (see Figure 28 on page 83)

The following display shows a piece of RPG code using the QtmhWrStout API.

```
 * Copy QUSEC structure error
 /copy qsysinc/qrpglesrc,qusec
D*QUSEDO1               17    116
 * Variables for QtmhWrStout
DOUTBuffLn        S             10i 0
DOUTbuff          S           9999
 *
C                   eval      qusbprv = 16
 * Standard output subroutine
C                   callb     'QtmhWrStout'
C                   parm                    OUTBuff
C                   parm                    OUTBuffLn
C                   parm                    QUSEC
```

- The QtmhCvtDb API provides an interface for CGI programs to parse CGI input defined as a series of keywords and their values into a buffer that is formatted according to a DDS file specification.

Table 8. Convert to DB (QtmhCvtDb) API

Description	I/O	Type	Example
Qualified database file name	Input	Char(20)	'FILESAMPL LIBLSAMPLE'
Input string	Input	Char(*)	name=fred&addrees=center+street
Length of input string	Input	Binary(4)	31
Response variable	Output	Char(*)	FILESAMPL
Length of response variable	Input	Binary(4)	length of FILESAMPL
Length of response available	Output	Binary(4)	Size required to contain the entire response
Response code	output	binary(4)	0=All keywords translated -1=The database file contains extra fields -2=The CGI input contains extra fields -3=Combination of -1 and -2 -4=Different data types
Error code	I/O	Char(*)	QUSEC data structure (see Figure 28 on page 83)

The following display shows a piece of RPG code using the QtmhCvtDb API.

```
 * Copy QUSEC structure
 /copy qsysinc/qrpglesrc,qusec
D*QUSEDO1               17    116
 * Variables for QtmhCvtDb
DDbFileName       S
DInData           S          32767    options(*varsize)
DInActLn          S             10i 0
DDSBuffer         S          32767    options(*varsize)
DDBDSLn           S             10i 0
DDBActLn          S             10i 0
DDBRespCd         S             10i 0
 *
C                   eval      qusbprv = 16
 * Format the input data into the externally described data structure
C                   CALLB     'QtmhCvtDb'
C                   parm                    DDbFileName
C                   parm                    DInData
C                   parm                    DInActLn
C                   parm                    DDSBuffer
C                   parm                    DDBDSLn
C                   parm                    DDBActLn
C                   parm                    DDBRespCd
C                   parm                    QUSEC
```

3.9.2 CGI Parameter Parsing with ILE COBOL/400 or RPG/400

IBM has provided a service program routine by the name QtmhCvtDb to automatically parse the name=value string and place the results in an externally defined physical file (externally defined by DDS, of course). This makes the job of parsing this rather complex string easy for ILE COBOL/400 and ILE RPG/400 programmers as it is trivial to define and read in the fields associated with the DDS file in RPG (as an example). This is the good news.

The bad news is that the QtmhCvtDb as shipped with V3R2 of the TCP/IP Connectivity Utilities/400 has a bug in it (at the time of the writing of this redbook) that causes it to make a mistake in the translation of the special escape sequences (such a %2C) into the EBCDIC equivalent. The root cause of the mistake lies in the same problem described in Section 3.8.2, "CGI Parameter Parsing with ILE C/400" on page 77.

Here are some circumventions to the problem that can be used until IBM fixes QtmhCvtDb:

1. Use the C sample program as mentioned in Section 3.8.2, "CGI Parameter Parsing with ILE C/400" on page 77 to parse the string of name=value pairs. This code is useful if you need to make some modification to the code for your environment. You may find, for example, the translation table that we use to convert ASCII to EBCDIC may not work for your environment.
2. You can always write the parsing code yourself in any language you want.
3. PID (IBM's Partners In Development) has made available a code snippet (small section of source code) that you can place right before the call to QtmhCvtDb. This code snippet modifies the string *before* the call by changing the hexadecimal characters of the escape sequence that point to a position in an ASCII code page into the EBCDIC representative. This causes QtmhCvtDb to properly convert the escape sequences into a single EBCDIC character.

   When IBM fixes the problem with QtmhCvtDb, you simply remove the code snippet.

3.9.3 ILE RPG Program Using the POST and GET Methods

The following RPG program example shows pieces of RPG code fully functional that serves as a model for building new CGI programs. This example is from a more complex RPG program that you can find in the following URL:

http://www.as400.ibm.com/workshop/webbuild.htm

The functions used in this example are in a prototypes file (included in the program by the /copy command), and the program calls them by the callp command.

The program is called by an HTML page, reads the standard input stream and the SCRIPT_NAME environment variable, and finally writes the HTML output.

```
 ****************************************************************************
 * THIS SAMPLE CODE IS PROVIDED BY IBM FOR ILLUSTRATIVE
 * PURPOSES ONLY.  IT HAS NOT BEEN FULLY TESTED. IT IS
 * PROVIDED AS-IS WITHOUT ANY WARRANTIES OF ANY KIND,
 * INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE.
 ****************************************************************************
 * This sample use a two external functions:
 ****************************************************************************
 * Prototype definitions for gethtml and wrtsection functions
 /copy prototypes
 * Data structure for error reporting
 /copy qsysinc/qrpglesrc,qusec
 * Constants
DBufSize          C                   32767
DOutBufSize       S              5i 0 inz(30000)
DDtaStrSize       C                   55
 * Variables for gethtml. Is the name of the file containing output
 * HTML code with sections & variables (see fig x).
Dfn               S             10    INZ('HTMLSRC')
Dlib              S             10    INZ('CGIDEV')
Dmbr              S             10    INZ('OUTRPG')
 * Names, values and lengths arrays for writing standard output.
 * Used in setvardata subroutine and wrtsection function.
Dsection          S             10
Dvarnm            S             10    dim(50)
Dvarval           S            500    dim(50)
Dvarln            S              5I 0 dim(50)
Dv1               S              5i 0
 * Variables for getenvf subprocedure
Denvrcvr          s          32767
Denvrcvrln        s             10i 0 inz(%size(envrcvr))
Denvrspln         s             10i 0
Denvrqsnm         s             30
Denvrqsln         s             10i 0
 * Variables for getinf subprocedure
DInData           S          32767A
DInDataLn         S             10I 0 INZ(BufSize)
DInActLn          S             10I 0
DInDataType       S              5
 * Variables for cvtdbf subprocedure. DbFileName MUST be UPPERCASE!
DDBFileName       S             20A   INZ('DATASTRYCTCGIDEV    ')
DDBDSLn           S             10I 0 INZ(DtaStrSize)
DDBActLn          S             10I 0 INZ
DDBRespCd         S             10i 0 INZ
DDATASTRUCT     E DS
 * Read environment variables
Dprogram          S             20A
```

Figure 29 (Part 1 of 2). RPG Source Code

```
 ****************************************************************************
 * Mainline
 ****************************************************************************
C                   eval      qusbprv = 16
 * Read output html page
C                   callp     gethtml(fn:lib:mbr)  1
 * Get input data from POST or GET method
C                   callp     getinput(indata:indataln: inactln:indatatype) 2
* Format the input data into the externally described data structure
C                   callp     CvtDb(DBFileName:InData:InActLn:datastruct:  3
C                                   DBDSLn:DBActLn:DBRespCd:QUSEC)
 * Use getenvp to get the environment variables
C                   eval      envrqsnm='SCRIPT_NAME'
C     ' '           checkr    envrqsnm      envrqsln
C                   exsr      getenvf     4
C                   eval      program=%subst(envrcvr:1:envrspln)
 * Setup data for writing standard output.
C                   exsr      setvardata  5
 * Write sections of HTML.
C                   eval      section='top'
C                   callp     wrtsection(section:varnm:varval:varln)       6
C                   eval      section='variab'
C                   callp     wrtsection(section:varnm:varval:varln)
C                   eval      section='*fini'
C                   callp     wrtsection(section:varnm:varval:varln)
C                   eval      *inlr = *on
 ****************************************************************************
 * Setvardata - Sets values of variable names, contents, lengths
 ****************************************************************************
C     setvardata    begsr
C                   eval      v1=1
C                   eval      varnm(v1)='NAME'
C                   eval      varval(v1)=name
C     ' '           checkr    varval(v1)    varln(v1)
C                   eval      v1 = v1 + 1
C                   eval      varnm(v1)='PROGRAM'
C                   eval      varval(v1)=program
C     ' '           checkr    varval(v1)    varln(v1)
C                   endsr
 ****************************************************************************
 * GETENVF. Call the QtmhGetEnv API
 ****************************************************************************
C     getenvf       begsr
C                   callb     'QtmhGetEnv'
C                   parm                    envrcvr
C                   parm                    envrcvrln
C                   parm                    envrspln
C                   parm                    envrqsnm
C                   parm                    envrqsln
C                   parm                    QUSEC
C                   endsr
 ********************************************************************** *****
```

Figure 29 (Part 2 of 2). RPG Source Code

1 Gethtml procedure

This procedure reads records from the HTML file CGIDEV/HTMLSRC/OUTRPG and puts them into a dynamic array.

```
  * Variables for gethtml (Is the name of the file containing HTML page output)
 Dfn               S             10    INZ('HTMLSRC')
 Dlib              S             10    INZ('CGIDEV')
 Dmbr              S             10    INZ('OUTRPG')
 * Read output html page
C                   callp     gethtml(fn:lib:mbr)
```

The HTML page output is a special page and looks similar to the following display:

```
/$top
Content-type: text/html
<HTML>
<HEAD><TITLE>Demo POST method</TITLE></HEAD>
<BODY>
<H1>Environment variables</H1>
<HR>
<P>Welcome <B>/%name%/</B>,
/$variab
<TABLE Border Width=45% center>
<CAPTION><B>Here are some environment variables</B></CAPTION>
<TR><TH ALIGN="left">Variable</TH><TH ALIGN="left">Content</TH></TR>
<TR><TD ALIGN="left">SERVER_NAME</TD><TD ALIGN="LEFT">/%program%/</TD>
</TR>
</TABLE>
<HR>
</BODY>
</HTML>
```

This page has two special words that show the program where it should write the sections and where it should write the variables identified by:

- /$section
- %variable%

2 Getinput procedure

In our sample, we call the getinput procedure with these parameters:

```
  * Variables for getinput subprocedure
 DInData           S          32767A
 DInDataLn         S             10I 0 INZ(BufSize)
 DInActLn          S             10I 0
 DInDataType       S              5
  * Call to getinput subprocedure
 C                   callp     getinput(indata:indataln:inactln:indatatype)
```

This function gets the REQUEST_METHOD environment variable to determine the method used and reads input data from stdin (POST method) or using a QtnhRdStin API, or reads from QUERY_STRING environment variable (GET method) using a QtnhGetEnv API. This program also changes the escape sequences and special characters (+) using the cgiparse procedure and keeps the data in the getinput variables.

For a GET method, the subprocedure calls the getenvf procedure to read the QUERY_STRING environment variable.

```
 ****************************************************************************
 * THIS SAMPLE CODE IS PROVIDED BY IBM FOR ILLUSTRATIVE PURPOSES ONLY
 * IT HAS NOT BEEN FULLY TESTED. IT IS PROVIDED AS-IS WITHOUT ANY
 * WARRANTIES OF ANY KIND, INCLUDING BUT NOT LIMITED TO THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 ****************************************************************************
Hnomain
 /copy prototypes
 /copy qsysinc/qrpglesrc,qusec
 *
Pgetinput         b                   export
Dgetinput         pi
DInData                       32767    options(*varsize)
DInDataln                        10i 0
DInActLn                         10I 0
DInDataType                       5
 * Work variables
DDummy            s                   like(InActLn)
 * Variables for getenvf subprocedure
Denvrcvr          s             32767
Denvrcvrln        s                10i 0 inz(%size(envrcvr))
Denvrspln         s                10i 0
Denvrqsnm         s                30
Denvrqsln         s                10i 0
C                   eval      qusbprv = 16
 * Get request_method and put into indatatype
C                   eval      envrqsnm='REQUEST_METHOD'
C     ' '           checkr    envrqsnm      envrqsln
C                   clear                   envrcvr
C                   exec      getenvf
C                   eval      indatatype = %subst(envrcvr:1:envrspln)
 * If requst method is post, Get content_length, convert to numeric,
 * and put into inactln
C                   if        indatatype='POST'
C                   eval      envrqsnm='CONTENT_LENGTH'
C     ' '           checkr    envrqsnm      envrqsln
C                   clear                   envrcvr
C                   exsr      getenvf
C                   eval      inactln=c2n(envrcvr)
C                   endif
 * If CONTENT_LENGTH is 0 and method = POST, set indata to blanks
 * and return
C                   if              indatatype='POST' and inactln = 0
C                   eval      indata = *blanks
C                   return
C                   endif
 * Handle GET
C                   select
C                   when      indatatype='GET'
C                   eval      envrqsnm='QUERY_STRING'
C     ' '           checkr    envrqsnm      envrqsln
C                   clear                   envrcvr
C                   exsr      getenvf
C                   eval      indata = %subst(envrcvr:1:envrspln)
C                   eval      inactln = envrspln
 * Handle POST
C                   when      indatatype='POST'
C                   CALLB     'QtmhRdStin'
C                   parm                    INData
C                   parm                    INDataLn
C                   parm                    dummy
C                   parm                    QUSEC
C                   endsl
 * Handle escape sequences, + signs, etc.
C                   callp     cgiparse(indata:inactln)
C                   return
 ****************************************************************************
```

Figure 30. Getinput Subprocedure Sample

3 CvtDb procedure

The CvtDb function moves the values read (by getinput function) to the DB file fields (remember to code the external DS because the QtmhCvtDb API returns

the data in this structure). In this sample, the data structure looks similar to the following display:

```
    A          R SAMPLEREC
    A*  This field is the same as a field in the HTML input page
    A*    <INPUT NAME="NAME" SIZE="40" MAXLENGTH="40">
    A*
    A            NAME          15
```

The program calls the CvtDb procedure.

```
 * Variables for cvtdbf subprocedure. DbFileName MUST be UPPERCASE!
 * and 20 characters long
 *                  10 characters for the file name
 *                  10 characters for the library name
DDBFileName       S             20A   INZ('DATASTRUCTCGIDEV    ')
DDBDSLn           S             10I 0 INZ(DtaStrSize)
DDBActLn          S             10I 0 INZ
DDBRespCd         S             10i 0 INZ
DDATASTRUCT     E DS
 * Format the input data into the externally described data structure
C                callp     CvtDb(DBFileName&colonInData:
                                 InActLn&colondatastruct:
                                 DBDSLn&colonDBActLn&colonDBRespCd&colonQUSEC)
```

The CvtDb procedure calls the QtmhCvtDb API.

```
CvtDb            pr                  extproc('QtmhCvtDb')
DbFileName                     20
InData                      32767    options(*varsize)
InActLn                        10i 0
DSBuffer                    32767    options(*varsize)
DBDSLn                         10i 0
DBActLn                        10i 0
DBRespCd                       10i 0
qusec                          16
```

4 Getenvf subroutine

The subroutine returns in the envrcvr variable, the SCRIPT_NAME environment variable.

```
 * Variables for getenvf subprocedure
Denvrcvr          s          32767
Denvrcvrln        s             10i 0 inz(%size(envrcvr))
Denvrspln         s             10i 0
Denvrqsnm         s             30
Denvrqsln         s             10i 0
 * Read environment variables
Dprogram          S             20A
 * Use getenvp to get the environment variables
C                   eval      envrqsnm='SCRIPT_NAME'
C     ' '           checkr    envrqsnm      envrqsln
C                   exsr      getenvf
C                   eval      program=%subst(envrcvr:1:envrspln)
 ***********************************************************************
 * GETENVF. Call the QtmhGetEnv API
 ***********************************************************************
C     getenvf       begsr
C                   callb     'QtmhGetEnv'
C                   parm                    envrcvr
C                   parm                    envrcvrln
C                   parm                    envrspln
C                   parm                    envrqsnm
C                   parm                    envrqsln
C                   parm                    QUSEC
C                   endsr
```

5 Setvardata procedure

This procedure puts the content of variable = value pairs into arrays that the wrtsection procedure uses later.

```
 * Names, values and lengths arrays for writing standard output.
 * Used in setvardata subroutine and wrtsection function.
Dsection          S             10
Dvarnm            S             10    dim(50)
Dvarval           S            500    dim(50)
Dvarln            S              5I 0 dim(50)
Dv1               S              5i 0
 * Setup data for writing standard output.
C                   exsr      setvardata
 ***********************************************************************
 * Setvardata - Sets values of variable names, contents, lengths
 ***********************************************************************
C     setvardata    begsr
C                   eval      v1=1
C                   eval      varnm(v1)='NAME'
C                   eval      varval(v1)=name
C     ' '           checkr    varval(v1)    varln(v1)
C                   eval      v1 = v1 + 1
```

6 Wrtsection procedure

At the end, the wrtsection subroutine uses the variables that have been prepared by the setvardata subroutine and writes sections of HTML.

Wrtsection scans the delimiters for the sections (for example, "/$top") and the delimiters for the variables (for example, "/%program%/") in the arrays, creates the gethtml subroutine, and replaces the "/%variable%/" for the data.

```
  * Names, values and lengths arrays for writing standard output.
  * Used in setvardata subroutine and wrtsection function.
Dsection          S             10
Dvarnm            S             10    dim(50)
Dvarval           S            500    dim(50)
Dvarln            S              5I 0 dim(50)
Dv1               S              5i 0
  * Write sections of HTML.
C                   eval      section='top'
C                   callp     wrtsection(section:varnm:varval:varln)
C                   eval      section='variab'
C                   callp     wrtsection(section:varnm:varval:varln)
C                   eval      section='*fini'
C                   callp     wrtsection(section:varnm:varval:varln)
```

The *fini section is not included in the HTML page because it is used only for the wrtsection procedure to ensure that all output HTML that has been buffered gets output.

This procedure calls the QtmhWrStout API to write in the standard output.

For RPG Programmers

We recommended using a debug in all of your programs. In the address:

http://www.as400.ibm.com/workshop/webbuild.htm

there is a good sample using a debugger.

3.10 Programming CGI with REXX Language

REXX programs cannot be used directly because REXX and the AS/400 Internet Connection HTTP server do not handle standard input and standard output compatibly. The REXX pull and say commands should **not** be used in CGI scripts. If they are used, unpredictable results can occur, including function checks in the HTTP server.

Because REXX procedures are not program objects, they cannot be invoked as CGI programs. A CL program is invoked with a parameter containing the name of the REXX procedure to be run in your form tag:

```
<form method="POST' action="/CGILIB/CLREXX.PGM?SAMPLE">
```

The CL program CLREXX.PGM uses the STRREXPRC command to call the REXX procedure. In this case, the REXXPROC variable is changed by SAMPLE:

```
            PGM        PARM(&REXXPROC)
            DCL        VAR(&REXXPROC) TYPE(*CHAR) LEN(10)
          /* This command runs a REXX Procedure */
            STRREXPRC  SRCMBR(&REXXPROC)
            MONMSG     MSGID(CPF0000) /* Ignore any errors */
            ENDPGM
```

To enable compatibility with the HTTP server, the REXX program can call to ILE-RPG programs that use QtmhRdStin, QtmhWrStout, and QtmhGetEnv APIs for standard input and standard output operations.

3.10.1 Structure of REXX Program Using POST Method

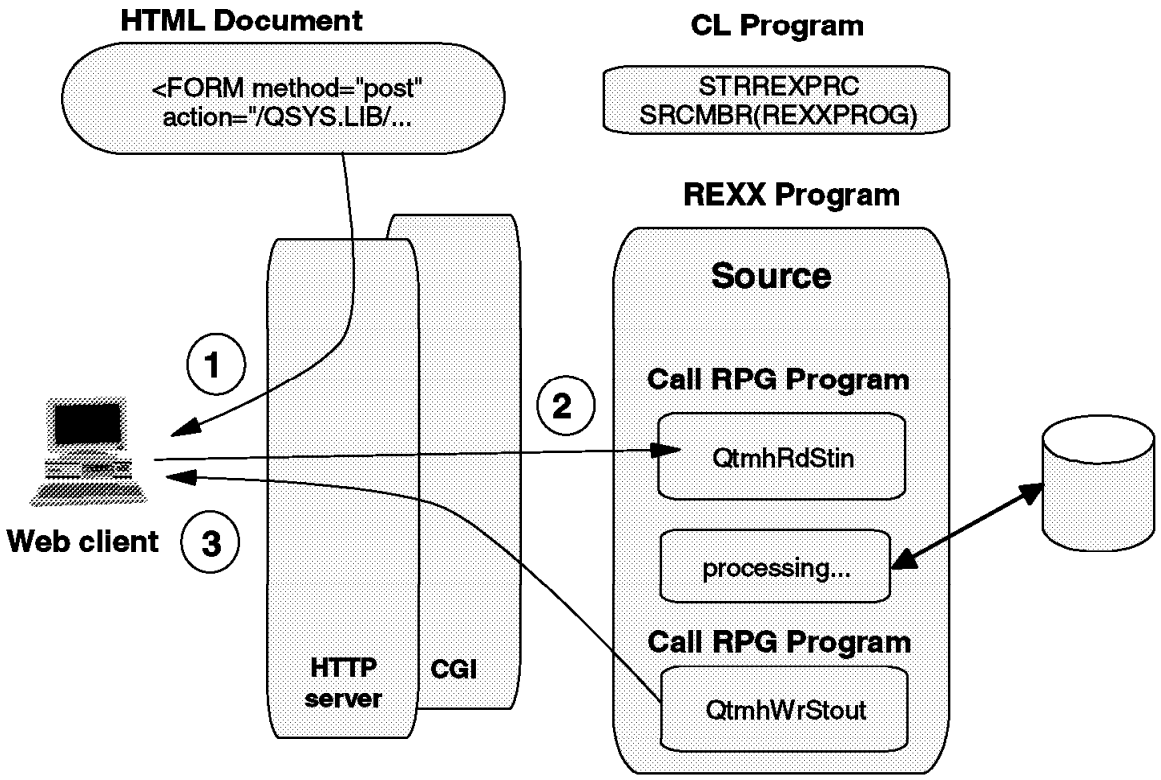


Figure 31. CGI Programming with REXX Language using POST Method

1. This point is the same as C and RPG languages. The HTML form is sent to the client.
2. It is the same structure as the C and RPG programs. The user selects the Submit button.
   - Call the CL program with the name of the REXX program.
   - Call QtmhRdStin through an RPG program. The QtmhRdStin API places the entire input string read from stdin into a receiver variable of type pointer to character.
   - Processing (anything that you want to do). This processing must generate the HTML output.
   - Call QtmhWrStout through an RPG program. This API writes the HTML code to standard output.
3. When your application ends, whatever you wrote to stdout is sent back to the Web client.

3.10.1.1 REXX Program Sample using POST Method

The next sample shows pieces of REXX code. The complete sample is in the Internet address:

```
http://www.as400.ibm.com/workshop/webbuild.htm
```

In this REXX program, we read from stdin by the getparse procedure, and write to the stdout by the sayh procedure.

```
/*******************************************************
* Initialization of variables                          *
*******************************************************/
call initialize
/*******************************************************
* Read and parse data from stdin                       *
*******************************************************/
call getparse      1
if alldataln > 0
   then
      do
        /*************************************************
        * Load translate table for use by handleescape *
        *************************************************/
        call gettable
        /****************************
        * Handle escaped characters *
        ****************************/
        call handleescape
        /****************************
        /* Creates stem variables   *
        ****************************/
        call donames
      end
/*******************************************************
* HTML output                                          *
*******************************************************/
title='This is the title'
call sayh 'Content-type: text/html'    2
call sayh '<HTML>'
call sayh '<HEAD>'
call sayh '<TITLE>'title'</TITLE>'
call sayh '</HEAD>'
call sayh '<BODY>'
call sayh 'Sample REXX procedure'
call sayh '</BODY>'
call sayh '</HTML>'
call sayh '(((((flush'
exit
```

Figure 32. REXX Source Code

Now we try explain the most important subroutines used in this sample:

1 Getparse subroutine

This subroutine reads from standard input (uses a call to STDIN RPG program that uses the QtmhRdStdin API) until no more input and concatenates all of the input into variables.

When finished, it converts any plus (+) characters to blanks and separates variable/value pairs.

```
getparse:
if g.1jobtype='1'
  then
    do
      say 'Enter standard input data'
      parse pull alldata
      alldataln = length(alldata)
    end
  else
    do
      alldata=''
      alldataln=0
      g.1content_length=getenv('CONTENT_LENGTH')
      g.1recvd = 0
      if g.1content_length = 1 then return

      /* Read from standard input
      ***************************/
      do while g.1content_length > 0
        if g.1content_length > g.1maxparmlength
          then
            g.1toget = g.1maxparmlength
          else
            g.1toget = g.1content_length
      /* Calls the RPG program
      ************************/
     'CALL PGM(STDIN) PARM(&g.1parm &g.1toget &g.1recvd)'
        alldataln = alldataln + g.1recvd
        alldata = alldata || substr(g.1parm,1,g.1recvd)
        g.1content_length = g.1content_length - g.1toget
      end
    end
/* If the stdin is empty
************************/
if alldata = '' | alldata = ' '
  then
    do
      alldata=''
      alldataln=0
      return
    end
/* Translate '+' to blank
*************************/
alldata = translate(alldata,' ','+')
g.1work = alldata                          /* copy input data to work */
data.0 = 0
/* Separate the variable/value pairs (ampersand character)
   and separate the variable and value (equal character)
************************************************************/
do i = 1 while g.1work <> ''
    parse var g.1work data.i '&' g.1work  /* parse va/value pairs */
    parse var data.i varname.i '=' value.i   /* parse var. and  value */
    data.0 = data.0 + 1                    /* increment variable count */
end
return
```

2 Sayh subroutine

This subroutine writes the data to standard out by calling the subroutine stdout. First, it checks the line length and if it is larger than the maximum number of characters allowed in a line without a newline, the sayh subroutine cuts the line and calls the stdout function to insert a newline character at the end of the line and write the string.

If the option (((((FLUSH appears in the argument list, see the stdout subroutine (called by sayh). This option is a sample; you can use any string for identify the end of the HTML document.

```
sayh: procedure expose g.
parse arg x                              /* read string argument */
do until x = ''
  /* Checks the lenght of the line
  *********************************/
  if length(x) <= g.1maxnonl
    then
      do
   call stdout x
        x = ''
      end
  else
    /* If the data has more than the maximum number allowed
    ******************************************************/
    do
      i = lastpos(' ',x,g.1maxnonl)
      if i = 0
        then i = g.1maxnonl /* couldn't find blank, break at maximum */
      y = left(x,i)
      call stdout y               /* write first i bytes */
      x = delstr(x,1,i)                 /* delete first i bytes */
      x = strip(x)                /* strip leading and trailing blanks */
    end
end
return
```

The sayh subroutine calls the stdout subroutine.

Stdout inserts a newline character at the end of the line to make the data conform to the ″newline character rule″ and writes the line in standard output by a call to the STDOUT RPG program that uses a QtmhWrStout API.

If the option (((((FLUSH appears in the argument list, it forces any pending output to standard output before processing the data.

```
stdout: procedure expose g.
parse arg data '((((('options
dataln = length(data)
if g.1jobtype='1' /* an interactive job */
  then
    do
      say data
    end
else
  do
    /* Flush buffer if option = 'FLUSH'.
     * This case is similar to write a EOF file. The
     * browser needs to know when finish the output.
     ***********************************************/
    if wordpos('FLUSH',translate(options)) > 0
      then
        do
          /* Calls the QtmhWrSout API.
           *************************/
          'CALL PGM(STDOUT) PARM(&g.1stdout &g.1stdoutl)'
          g.1stdout=''
          g.1stdoutl=0
          if dataln = 0
            then
              return
        end
    /* If no newline charapter, appends at the end of the line.
     ***********************************************************/
    if right(data,1) <> g.1newline then
      data = data || g.1newline
    dataln = length(data)
    if (dataln + g.1stdoutl) > g.1maxparmlength
      then
        do
          /* Calls the QtmhWrSout API.
          *************************/
          'CALL PGM(STDOUT) PARM(&g.1stdout &g.1stdoutl)'
          g.1stdout = data
          g.1stdoutl = dataln
        end
      else
        do
          g.1stdout = g.1stdout || data
          g.1stdoutl = g.1stdoutl + dataln
        end
  end
return
```

This program takes parameters and calls the API to write them in standard stdout.

```
 /copy qsysinc/qrpglesrc,qusec
 * Variables for QtmhWrStout
DOutBuffLn        S             10i 0
 * Variables for entry parameter list.  Note: 9999 is maximum size of C
DOutbuff          S           9999
DOutBuffPk        S             15P 5
Dnewline          C                   x'15'
 * Mainline calculations
C     *entry        plist
C                   parm                    outbuff
C     OutBuffln     parm                    outbuffPk
 *
C                   eval      qusbprv = 16
 * Standard output subroutine
C                   callb     'QtmhWrStout'
C                   parm                    OUTBuff
C                   parm                    OUTBuffLn
C                   parm                    QUSEC
C                   return
```

REXX

We recommend not using PULL and SAY procedures in REXX CGI programs. You can make a function that calls RPG programs that use a QtmhRdStin and QtmhWrStout.

For REXX Programmers

We recommend using a debug method in all of your programs. In the address:

`http://www.as400.ibm.com/workshop/webbuild.htm`

there is a good sample using a debugger.

3.11 Examples for Environment Variables

On the following pages, there are some inquiry examples for environment variables.

Environment Variables with GET

Figure 34 on page 100 shows the inquiry results with the Method=GET.

The environment variable QUERY_STRING contains the name/value pairs!

Environment Variables with POST

Figure 36 on page 101 shows the inquiry results with the Method=POST.

The environment variable QUERY_STRING is **empty**.

POST always reads from **STDIN**!

The following figure shows the form for program CGIENVGET: Environment Variables with METHOD=GET.

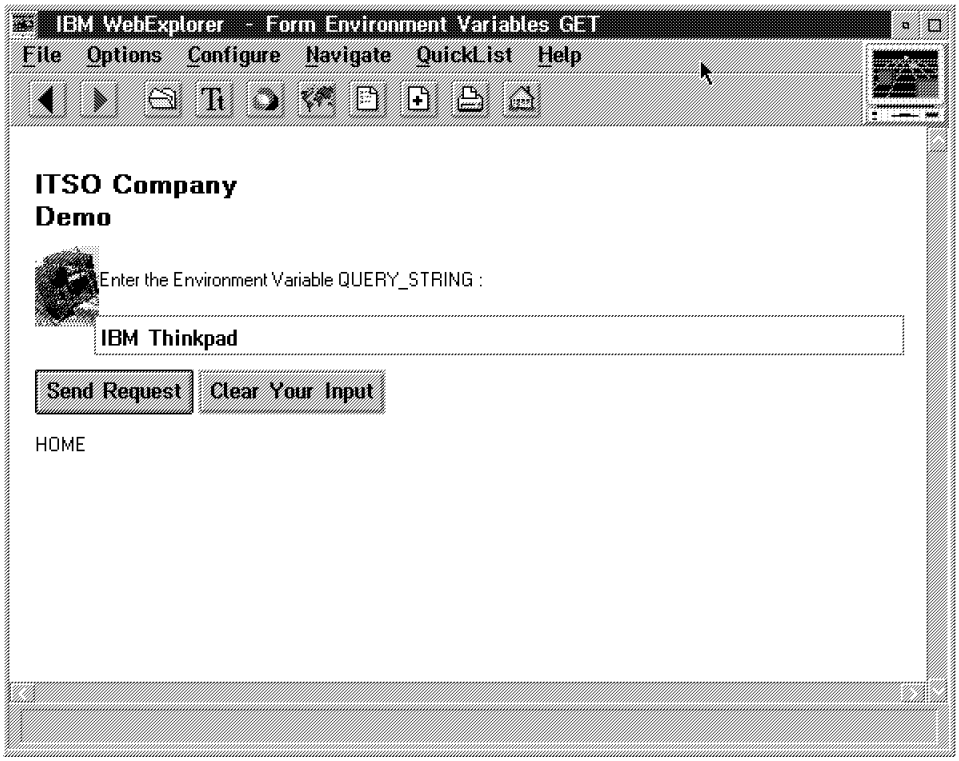


Figure 33. The HTML Document using the FORM Tag with Method=GET

The following figure shows the HTML output from program CGIENVGET.

Environment Variables with GET

QUERY_STRING :	MBR=CGIEN VGETE&querydata=IBM+Thinkpad+
SERVER_SOFTWARE :	IBM-AS/400-HTTP-Server/1.0
SERVER_NAME :	internut.r chland.ibm.com
GATEWAY_INTERFACE :	CERN-PreParsed
SERVER_PROTOCOL :	HTTP/1 .0
SERVER_PORT :	80
REQUEST_METHOD :	GET
PATH_INFO :	N/A
PATH_TRANSLATED :	N/A
SCRIPT_NAME :	/BonusCGI/CGIENVGET.PGM
REMOTE_HOST :	w3proxy-b. rchland.ibm.com
REMOTE_ADDR :	9.5.100.112
REMOTE_USER :	N/A
CONTENT_TYPE :	application/x-www-form-urlencoded
CONTENT_LENGTH :	45
IBM_CCSID_VALUE :	N/A

Back Demo

Figure 34. Document Returned from the Program CGIENVGET

The following figure shows the form for program CGIENVPOST: Environment Variables with METHOD=POST.

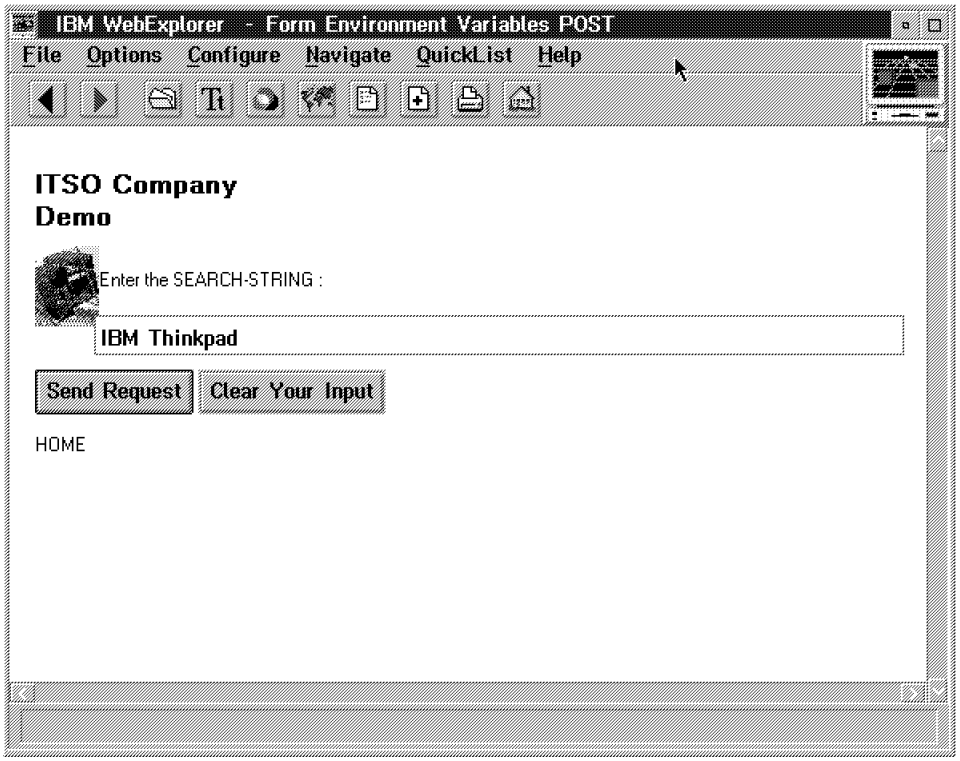


Figure 35. The HTML Document using the FORM Tag with Method=POST

The following figure shows the HTML output from program CGIENVPOST.

Environment Variables with POST

QUERY_STRING :	N/A
SERVER_SOFTWARE :	IBM-AS/400-HTTP-Server/1.0
SERVER_NAME :	internut.r chland.ibm.com
GATEWAY_INTERFACE :	CERN-PreParsed
SERVER_PROTOCOL :	HTTP/1 .0
SERVER_PORT :	80
REQUEST_METHOD :	POST
PATH_INFO :	N/A
PATH_TRANSLATED :	N/A
SCRIPT_NAME :	/BonusCGI/CGIENVPOST.PGM
REMOTE_HOST :	w3proxy-b. rchland.ibm.com
REMOTE_ADDR :	9.5.100.112
REMOTE_USER :	N/A
CONTENT_TYPE :	application/x-www-form-urlencoded
CONTENT_LENGTH :	38
IBM_CCSID_VALUE :	N/A

Back Demo

Figure 36. Document Returned from the Program CGIENVPOST

3.12 ITSO Company Example

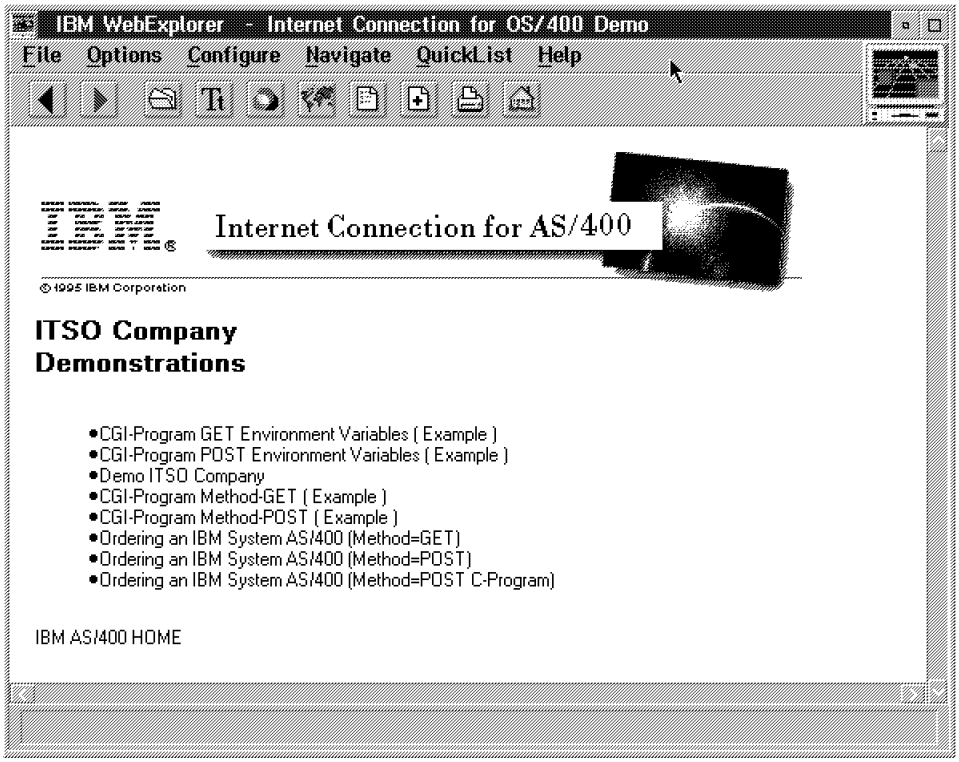


Figure 37. Welcome to the ITSO Company Example

Table 9. Overview - Programs HTMLFILE=INPUT

Program	Function	Files	HTML-Mbr	HTMLO-Mbr	Language
CGIENVGET	Environment Variables GET	ENVFILE ENVGETDS HTMLO	FORMGETE	CGIENVGETE	ILE RPG
CGIENVPOST	Environment Variables POST	ENVFILE ENVPOSTDS HTMLO	FORMPOSE	CGIENVPOSE	ILE RPG
CGIGET	Search Online Catalog	INVFILE QUERYDS	L04P040		ILE RPG
CGIPOST	Feedback Talk To Us	TALKTOUS TALKDS	L04P060		ILE RPG
ORDAS400G	Ordering an AS/400 GET	ORDAS00F ORDASDS HTMLO	AS4FORMGE	ORDAS400E	ILE RPG
ORDAS400P	Ordering an AS/400 POST	ORDAS00F ORDASDS HTMLO	AS4FORMPE	ORDAS400E	ILE RPG
PARSECGIP	Ordering an AS/400 POST		RESIFORM		ILE C

3.12.1 Source Code RPG Program ORDAS400G

ILE RPG Program Example for the Method=GET

The information from the URL (value/name pairs) is assigned to the environment variable QUERY_STRING.

This gateway program picks out the name/value pairs of the variables from this QUERY_STRING through the API for ILE RPG:

The following example shows the Get Environment Variable QtmhGetEnv (subroutine GETENV).

```
     ****************************************************************************
     *  Simple ILE RPG program ORDAS400G to test CGI Method=GET
     *
     *
     * 1. Compile this source member as module ORDAS400G ( PDM Option=15 )
     *
     * 2. Create program ORDAS400G from module ORDAS400G  ( PDM Option=26 )
     *    with PROMPT(PF4) and BNDSRVPGM(QTCP/QTMHCGI)
     *
     * Define your files here
     ****************************************************************************
     * Order Database file
     ****************************************************************************
    FORDASOOF  O  A E             DISK
     ****************************************************************************
     * HTMLFILE Input file   ( read FORM input )
     ****************************************************************************
    FHTMLFILE  IF   E             DISK
     ****************************************************************************
     * HTMLO   Output file   ( prepared HTML Output in SRC-PF HTMLO  )
     *                       ( MBR = hidden field in HTML Input Form )
     ****************************************************************************
    FHTMLO     IF   E             DISK    USROPN  RENAME(HTMLO:HTMLOUT)
     ****************************************************************************
A
|    *Variables for the CGI interface APIs
|    *These are for the APIEnVar
|   DENBuff           S           2048A   INZ
|   DENBuffLn         S              9B 0 INZ(2048)
|   DENActLn          S              9B 0 INZ
|   DENVarName        S             64A   INZ
|   DENVarLn          S              9b 0 INZ
A
     *These are for the APICvtDB  Datastructure INPUT fields
     ****************************************************************************
    DDBFileName       S             20A   INZ('ORDASDS   *LIBL     ')
     ****************************************************************************
    DDBBuff           S           2048A   INZ
    DDBBuffLn         S              9B 0 INZ(2048)
    DDBDSLn           S              9B 0 INZ
    DDBActLn          S              9B 0 INZ
    DDBRespCd         S              9B 0 INZ
     *These are used for APIStdOut
    DOutBuff          S           2048A   INZ
    DOutBuffLn        S              9B 0 INZ(2048)
     ****************************************************************************
     *Externally described data structure.  Used for Parsing
     *Need a different one in each CGI-BIN you write
    DORDASDS        E DS
```

Figure 38 (Part 1 of 5). ILE RPG Program ORDAS400G Method=GET

```
 ****************************************************************************
 * Data structure for error reporting.  Copied from QSYSINC/QRPGLESRC(QUSEC
DQUSEC            DS
D*                                              Qus EC
D QUSBPRV                1      4B 0 INZ(16)
D*                                              Bytes Provided
D QUSBAVL                5      8B 0
D*                                              Bytes Available
D QUSEI                  9     15
D*                                              Exception Id
D QUSERVED              16     16
D*                                              Reserved
D*QUSED01               17    116
D*
D*                                    Varying length
 ****************************************************************************
 *Constants for names of CGI APIs
DAPIStdIn         C                   'QtmhRdStin'
DAPIStdOut        C                   'QtmhWrStout'
DAPICvtDB         C                   'QtmhCvtDb'
DAPIEnVar         C                   'QtmhGetEnv'
 *Compile-time array for OVRDBF
DOVRDBF           S             80    DIM(2) PERRCD(1) CTDATA
DOVRARR           S              1    DIM(80)
D****************************************************************************
D* Define NewLine
DNewLine          C                   x'15'
D****************************************************************************
D* Define break
DBreak            C                   '<BR>'
```

Figure 38 (Part 2 of 5). ILE RPG Program ORDAS400G Method=GET

```
     ************************************************************
B
     * Get the Environment Variable called QUERY_STRING
     * Set the ENVarName to QUERY_STRING
     *You must count the length of the Var Name!
     *Set the ENVarLn to this length (12 In This Case)
    C                   MOVEL     *BLANKS       ENBuff
    C                   MOVEL     'QUERY_STRING'ENVarName
    ****************************************************************
    C                   Z-ADD     12            ENVarLn
    ****************************************************************
    C                   EXSR      GETENV
     *  Upon return, your Query_String data is in ENBuff with the len
     *  of the data returned in ENActLn
     *  Move this data to the DBCvt parms
     ******************************************************************************
    C                   Z-ADD     103           DBDSLn
     ******************************************************************************
    C                   MOVEL     ENBuff        DBBuff
    C                   Z-ADD     ENActLn       DBBuffLn
B
     ******************************************************************************
     * Circumvention for HIDDEN fields  ( find out MEMBER-NAME for HTMLO )
     ******************************************************************************
    C                   MOVEL     ENBuff        M14             14
    C                   MOVE      M14           M10             10
    C     '+':' '       XLATE     M10           MBR             10
     ********************************************************************
     * END Circumvention
     ************************************************************************
     * Parse using the CvtDB API
    C                   EXSR      PARSE
     * The field names in your Ext DS now
     * contain  the Values passed in the POST data
     * Move them to the DB file fields
     ******************************************************************************
     * OVR HTMLOUT with MEMBER ORDAS400 and open file
    C                   MOVEA     OVRDBF(1)     OVRARR
    C                   MOVEA     MBR           OVRARR(24)
    C                   MOVEA     OVRARR        CMD             80
    C                   Z-ADD     80            L               15 5
    C                   CALL      'QCMDEXC'
    C                   PARM                    CMD
    C                   PARM                    L
    C                   open      HTMLO
     ********************************************************************
     * Move FORM Input to Database fields
     ******************************************************************************
    C                   MOVEL     NAME          NAME_X
    C                   MOVEL     ADDRESS       ADDRESS_X
    C                   MOVEL     TYPE          TYPE_X
    C                   MOVEL     SERV          SERV_X
     ********************************************************************
     * Write Database record file ORDASOOF
     ******************************************************************************
    C                   WRITE     ORDASR
     * If you had multiple values for the same field, you would
     * have lost all but the first. You need another technique for
     * this situation
```

Figure 38 (Part 3 of 5). ILE RPG Program ORDAS400G Method=GET

```
 ****************************************************************************
 * Create the HTML Output
 * Write HTML Required control records
 * ADD NewLine append after 80 to 120 characters to OutBuff
 ****************************************************************************
C                   DO        9             I                5 0
C                   READ      HTMLOUT                                  98
C     OutBuff       cat       SRCDTA:0      OutBuff
C     OutBuff       CAT       NewLine:0     OutBuff
 * Prepare variable OUTPUT
C     I             ifeq      9
C     OutBuff       CAT       Break:0       OutBuff
C     OutBuff       CAT       Break:0       OutBuff
C     OutBuff       cat       NAME:0        OutBuff
C     OutBuff       CAT       Break:0       OutBuff
C     OutBuff       cat       ADDRESS:0     OutBuff
C     OutBuff       CAT       NewLine:0     OutBuff
C     TYPE          ifeq      'P1'
C     OutBuff       CAT       Break:0       OutBuff
C     OutBuff       CAT       Break:0       OutBuff
C     10            CHAIN     HTMLOUT                          98
C     OutBuff       CAT       SRCDTA:0      OutBuff
C     OutBuff       CAT       NewLine:0     OutBuff
C                   endif
C     TYPE          ifeq      'P2'
C     OutBuff       CAT       Break:0       OutBuff
C     OutBuff       CAT       Break:0       OutBuff
C     11            CHAIN     HTMLOUT                          98
C     OutBuff       CAT       SRCDTA:0      OutBuff
C     OutBuff       CAT       NewLine:0     OutBuff
C                   endif
C     TYPE          ifeq      'P3'
C     OutBuff       CAT       Break:0       OutBuff
C     OutBuff       CAT       Break:0       OutBuff
C     12            CHAIN     HTMLOUT                          98
C     OutBuff       CAT       SRCDTA:0      OutBuff
C     OutBuff       CAT       NewLine:0     OutBuff
C                   endif
C     OutBuff       CAT       Break:0       OutBuff
C     OutBuff       CAT       Break:0       OutBuff
C     SERV          ifeq      'T'
C     13            CHAIN     HTMLOUT                          98
C     OutBuff       CAT       SRCDTA:0      OutBuff
C     OutBuff       CAT       NewLine:0     OutBuff
C                   else
C     14            CHAIN     HTMLOUT                          98
C     OutBuff       CAT       SRCDTA:0      OutBuff
C     OutBuff       CAT       NewLine:0     OutBuff
C                   endif
C                   endif
C                   ENDDO
C                   MOVE      *OFF          *IN98
C     15            setll     htmlout
C     *IN98         DOWEQ     *OFF
C                   read      htmlout                              98
C     *IN98         CABEQ     *ON           EndHTM
C     OutBuff       cat       SRCDTA:0      OutBuff
C     OutBuff       CAT       NewLine:0     OutBuff
C     EndHTM        TAG
 * End variable OUTPUT
C                   ENDDO
```

Figure 38 (Part 4 of 5). ILE RPG Program ORDAS400G Method=GET

```
     * Read HTMLFILE until EOF
    C                   MOVE      *OFF          *IN99
    C     *IN99         DOWEQ     *OFF
    C                   READ      HTMLREC                               99
    C                   ENDDO
     * Send OutBuff to standard output
    C     OutBuff       CAT       NewLine:0     OutBuff
    c     ' '           CHECKR    OutBuff       OutBuffLn
    C                   EXSR      STDOUT
     * End program
    C                   CLOSE     HTMLO
    C                   MOVEA     OVRDBF(2)     OVRARR
    C                   MOVEA     OVRARR        CMD             80
    C                   CALL      'QCMDEXC'
    C                   PARM                    CMD
    C                   PARM                    L
    C                   MOVE      *ON           *INLR
     * These are the APIs used in subroutines to keep the main processing
     * simple.  They do not need to be SUBRs!
C
     * Subroutine to Get Environment Variable
    C     GETENV        BEGSR
    C                   CALLB     APIEnVar
    C                   parm                    ENBuff
    C                   parm                    ENBuffLn
    C                   parm                    ENActLn
    C                   parm                    ENVarName
    C                   parm                    ENVarLn
    C                   parm                    QUSEC
    C                   ENDSR
C
    C*  Parse subroutine
    C     PARSE         BEGSR
    C                   CALLB     APICvtDB
    C                   parm                    DBFileName
    C                   parm                    DBBuff
    C                   parm                    DBBuffLn
    ** Remember to code your External DS name.  The API returns your data
    ** in this structure. The field names are in the structure
     ******************************************************************************
    C                   parm                    ORDASDS
     ******************************************************************************
    C                   parm                    DBDSLn
    C                   parm                    DBActLn
    C                   parm                    DBRespCd
    C                   parm                    QUSEC
    C                   ENDSR
     * This is the STD OUT SUBR
    C     STDOUT        BEGSR
    C                   callb     APIStdOut
    C                   parm                    OUTBuff
    C                   parm                    OUTBuffLn
    C                   parm                    QUSEC
    C                   ENDSR
**CTDATA OVRDBF
OVRDBF FILE(HTMLO) MBR(1234567890) LVLCHK(*NO) OVRSCOPE(*JOB)
DLTOVR FILE(HTMLO) LVL(*JOB)
```

Figure 38 (Part 5 of 5). ILE RPG Program ORDAS400G Method=GET

3.12.2 Source Code RPG Program ORDAS400P

ILE RPG Program Example for the Method=POST

The logic of this program is identical with the program in Figure 38 on page 103.

Only three parts are replaced:

- **A** by **1**
- **B** by **2**
- **C** by **3**

This CGI program receives the name/value pairs from the form as encoded form input from STDIN.

The environment variable CONTENT_LENGTH determines how much data **must** be read from STDIN.

The following example shows the Read from STDIN QtmhRdStin (subroutine STDIN).

1

```
     ******************************************************************************
     *Variables for the CGI interface APIs
     *These are used for APIStdIn
    DInData           S           2048A   INZ
    DInDataLn         S              9B 0 INZ(2048)
    DInActLn          S              9B 0
     ******************************************************************************
```

2

```
     ******************************************************************************
     * Get the Input parameters from the POST from STDIN
    C                   MOVE      *BLANKS       OutBuff
    C                   EXSR      STDIN
     *  Upon return, your POST data is in INData and its length is in
     *  INActLn  It is in the FLD=VAR format at this time
     *  Move this data to the DBCvt parms
     * Set up the parameters before CALLB
     *  This includes the length of your Ext DS (103 is correct)
     ******************************************************************************
    C                   Z-ADD     103           DBDSLn
     ******************************************************************************
    C                   MOVEL     INData        DBBuff
    C                   Z-ADD     INActLn       DBBuffLn
     ******************************************************************************
```

3

```
     ******************************************************************************
     * Subroutine to read STD IN
    C     STDIN         BEGSR
    C                   CALLB     APIStdIn
    C                   parm                    INData
    C                   parm                    INDataLn
    C                   parm                    INActLn
    C                   parm                    QUSEC
    C                   ENDSR
     ******************************************************************************
```

Figure 39. ILE RPG Program ORDAS400P Method=POST

3.12.3 Source Code C Program PARSECGIP

ILE C Program Example for the Method=POST

This is the program for Ordering an AS/400 system in the C-Language.

C Programs #include qp0z1170.h

The **qp0z1170.h** must included to get the getenv() and putenv() functions, which are not part of the **stdio.h** at this time.

The **qp0z1170.h** is a member in the file **H** in the library **QSYSINC**.

```
/*********************************************************************
/*                                                                   *
/*  Simple ILE C program PARSECGIP to test CGI Method=POST           *
/*                                                                   *
/*********************************************************************
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <qp0z1170.h>

#define MAX_ENTRIES 10000

typedef struct {
    char *name;
    char *val;
} entry;

char *makeword(char *line, char stop);
char *fmakeword(FILE *f, char stop, int *len);
char x2c(char *what);
void unescape_url(char *url);
void plustospace(char *str);


main(int argc, char *argv[]) {
    entry entries[MAX_ENTRIES];
    register int x,m=0;
    int cl;

    printf("Content-type: text/html\n\n");

    if(strcmp(getenv("REQUEST_METHOD"),"POST")) {
        printf("This script should be referenced with a METHOD of POST.\n");
        exit(1);
    }
    if(strcmp(getenv("CONTENT_TYPE"),"application/x-www-form-urlencoded")) {
        printf("This script can only be used to decode form results. \n");
        exit(1);
    }
    cl = atoi(getenv("CONTENT_LENGTH"));
    for(x=0;cl && (!feof(stdin));x++) {
        m=x;
        entries[x].val = fmakeword(stdin,'&',&cl);
        plustospace(entries[x].val);
        unescape_url(entries[x].val);
        entries[x].name = makeword(entries[x].val,'=');
    }

    printf("<html>\n");
    printf("<body>\n");
    printf("<H1>Query Results</H1>");
    printf("You submitted the following name/value pairs:<p>");
    printf("<ul>");

    for(x=0; x <= m; x++)
        printf("<li> <code>%s = %s</code>",entries[x].name,
                entries[x].val);
    printf("</ul>");
    printf("</body>\n");
    printf("</html>\n");
}
```

Figure 40. ILE C Program PARSECGIP Method=POST

Chapter 4. Net.Data Implementation

With the explosion of intranets and the Internet, there is a growing demand for access to dynamic Web pages.

Net.Data allows people with little programming knowledge to easily transform static HTML Web pages into dynamic Web applications using ″Net.Data Macros″. Macros created for Net.Data have the simplicity of HTML with the functionality of CGI applications, making it easy to add live data to static Web pages. Live data can include information stored in Net.Data compatible databases locally or on other systems, REXX programs, C and C++ programs, RPG programs, COBOL programs, Java applets, or any other source.

4.1 An Overview of Net.Data for AS/400 System

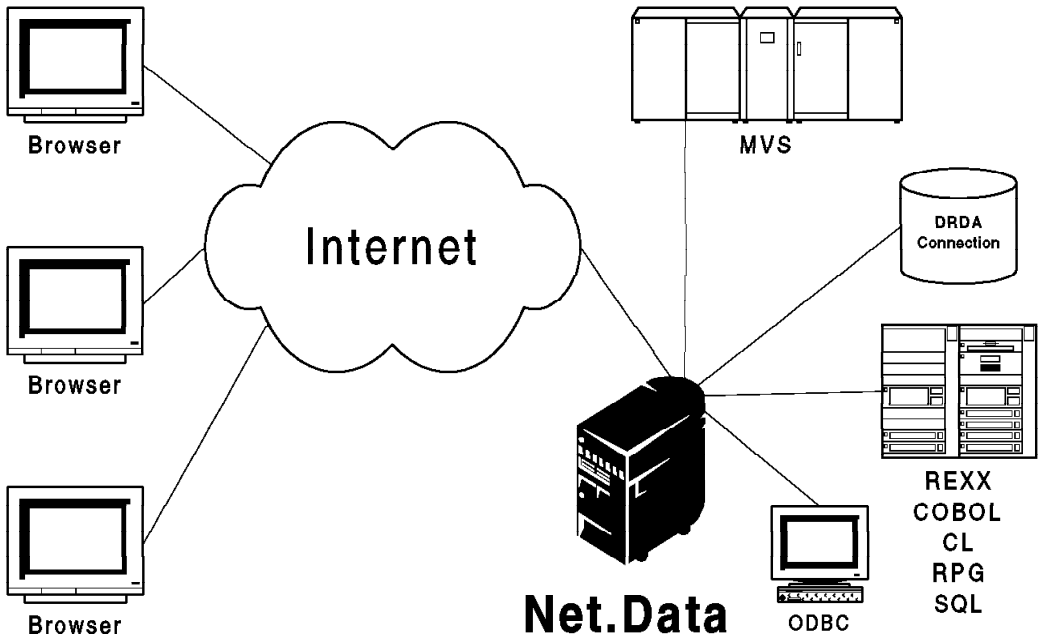


Figure 41. Net.Data Overview

Net.Data enables the application developer to build Web applications on the AS/400 system using a simple macro language that includes HTML forms, dynamic SQL, REXX, and high-level language programs. End users of Net.Data Web macros see only the forms for their input and reports showing the results. A user fills out the forms, points and clicks to submit the form, and Net.Data processes the request as determined by the macro. The complete HTML page is dynamically built by Net.Data and the result is returned to the browser.

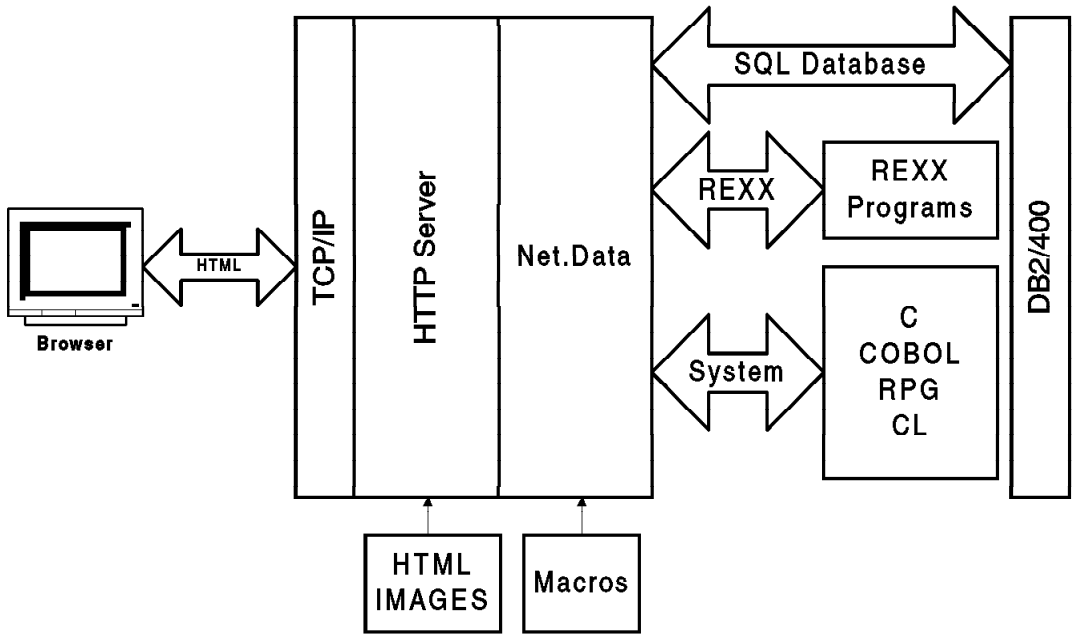


Figure 42. Net.Data General Overview

The *Macro language* was developed to be a simple and flexible language, enabling the application developer to use the full capabilities of HTML to create forms and reports. The application developer creates HTML source, SQL commands, and calls to other language environments and stores them in macro files at the Web server. Variables are used to link the Macro language commands and the HTML source within the macro file. The macro files are processed by Net.Data and the results are passed to the HTTP server.

4.1.1 Beyond DB2WWW Connection

Net.Data is an upwardly-compatible follow-on version of the DB2 World Wide Web Connection (DB2WWW) that was first shipped with OS/400 Version 3 Release 2 (V3R2), and builds on the strong database access and report creation abilities of that version.

By providing significantly increased data access and manipulation capabilities, Net.Data has become a comprehensive, general-purpose Web application development environment for the creation of interactive, data-centric Web applications.

4.1.2 Features and Functions

Net.Data was designed with the following objectives:

- To not require extensive programming by the application developer other than the use of HTML to create forms, SQL for queries and updates against the database, REXX for string manipulation and more advanced functions, and the Advanced Macro Language for ease-of-use functionality. For more information about REXX, see:

  `http://rexx.hursley.ibm.com/`

- To be flexible for a variety of Web applications that can grow from simple to complex.
- To be portable to multiple server platforms.

- To be usable with existing Web HTML editors and database query tools.

Net.Data has the following features:

- Net.Data applications can use native HTML, SQL, and REXX, thus exploiting the expressive power of these languages without artificial limitations.
- Net.Data is small and efficient, and has been ported to multiple platforms.
- Visual tools may be used to partially generate Net.Data macro files. Third party vendors' visual HTML editors can be used to generate the HTML sections, while various visual query tools such as *IBM's Visualizer Query* may be used to generate the SQL sections.

4.1.3 Generalized Data Sources

Net.Data, with its enhanced functionality, can use a variety of data sources to build dynamic Web pages:

- DB2, through SQL
- REXX programs
- C programs
- COBOL, RPG programs
- Built-in functions

4.1.4 Advanced Macro Language

Net.Data Web macros combine things you already know such as HTML, SQL, and REXX with a simple macro language. Capabilities of the macro language include:

- Multiple, named HTML sections in one macro
- Ability to include files
- If-Then-Else
- Function definitions
- Large library of predefined functions
- Access to environment variables
- Ability to disable the default report
- Table variables
- Ability to specify multiple paths for macros, executable, and includes
- Persistent database connections
- An external interface for user-supplied language environments that can provide new sources to data

Examples of the previous functions can be found in Section 4.2, "Writing Net.Data Macro Files" on page 114. To find out more information about the Net.Data macro language, see the online *Net.Data Reference Guide* at:

`http://www.software.ibm.com/data/net.data/docs/`

4.1.5 Net.Data and Internet Security

The following section is a general description of Net.Data and security. To use these functions with the AS/400 system, you need IBM′s V4R1 secure server Net.Commerce or I/NET′s Commerce server.

Net.Data works with the DB2 database, the Web server, and the firewall products to provide secure data access over the Internet. When used with one or more of these other products, the types of security provided are as follows:

- *Authentication:* Net.Data can take advantage of two types of authentication (user login and password), one provided by the Web server and the other by DB2. They are as follows:
  - Web servers can typically be configured to protect certain directories on the server. When a URL accesses a file in that directory, login and password pop-up windows appears on the end user′s Web browser to request for authentication.
  - DB2 has a login/password authentication mechanism for database access. This mechanism can be used to restrict access to tables and columns by certain users.
- *Encryption:* Net.Data, when used with a secure Web server that has support for Secure Sockets Layer (SSL) or Secure Hypertext Transfer Protocol (SHTTP), or both, benefits from the public key encryption scheme provided by the secure Web server and secure Web client. The user login and password used for authentication are encrypted for transmission, as well as all user inputs to the forms and the query results that are sent back to the user. A secure Web server/client combination is a must for protection of sensitive data.
- *Firewall:* Net.Data may be used with firewall products such as NetSP for IBM, which protects both the Net.Data machine and the corporate network from external Internet attacks.

For more information on protecting your assets, see the Internet security list of frequently asked questions (FAQ) at this Web site:

`http://www-genome.wi.mit.edu/WWW/faqs/www-security-faq.html`

For the most secure access to DB2 data, Net.Data should be used in conjunction with both a secure Web server and a firewall.

4.2 Writing Net.Data Macro Files

When Net.Data is used as a CGI application, it calls the executable DB2WWW and one or more language environments.

Language environments are Net.Data′s interface to your data and applications. Each language environment provides a specific interface. For example, Net.Data provides language environments to access DB2 databases, REXX, C, C++, RPG, and COBOL applications. Net.Data also lets you build your own language environments.

The Web macro contains a series of macro language, HTML, and language environment-specific statements. Language environment-specific statements can be from languages such as SQL and REXX.

When a URL is received by the Web server that refers to Net.Data, the Web server passes essential information to it, including the name of the Web macro and the HTML section to process. Net.Data reads and parses through the Web macro and interprets the statements. When a Net.Data function call statement is encountered, it loads the service program of the requested language environment and passes language-specific information to the language environment for processing. The language environment processes the information and returns the results to Net.Data. After all parsing is done and the language environment processing is completed, all that remains is HTML text that any Web browser can interpret. You have complete control over the HTML that is used. The HTML text is passed back to the Web server. The server passes it to the Web browser where the user interacts with it. Further requests from this or any other user results in the process being repeated over again.

Net.Data Macro files consist of multiple logical sections. The most common sections are the *Define section*, *Function section*, *HTML Input section*, and the *HTML Output section*. The following sample is a Net.Data Macro file.

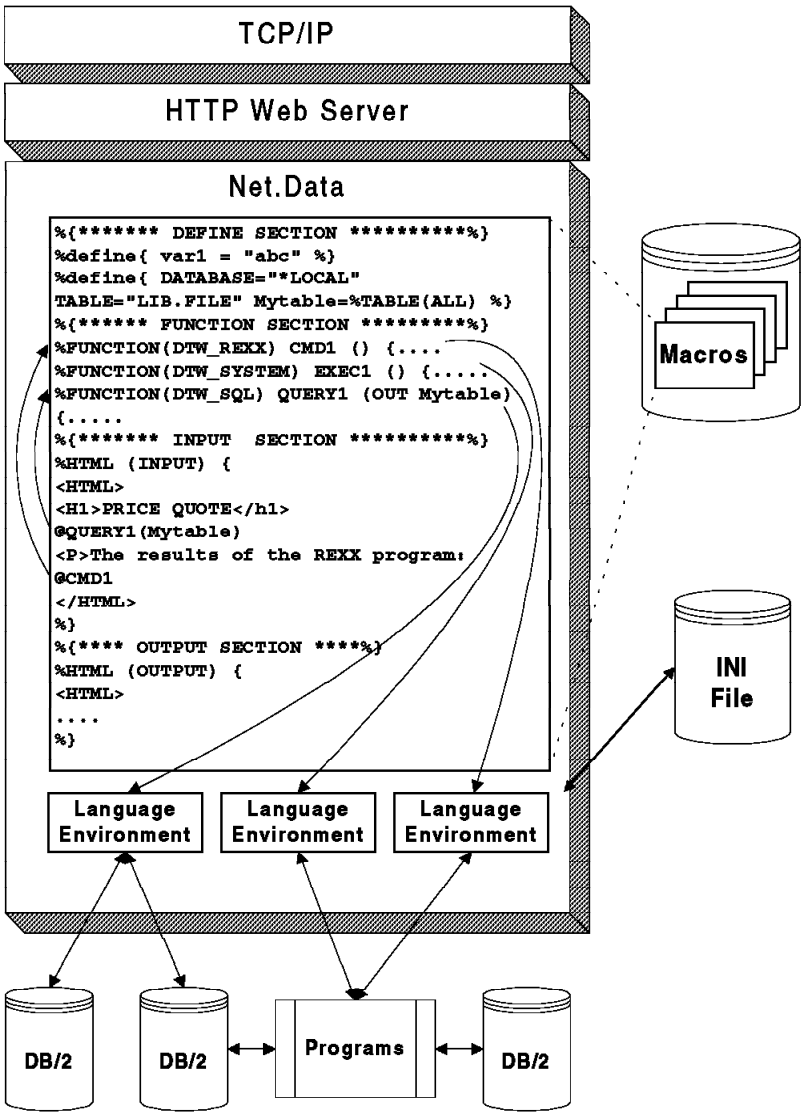


Figure 43. Net.Data Macro Files

Notice that the following macro consists of four major logical sections: the Define section, the Function Definition section, the HTML Input section, and the HTML Output section. You can have several HTML sections, but this simple example only has two.

Also notice that the two HTML sections contain familiar HTML tags, which makes writing Web macros easy. If you are familiar with HTML, building a macro simply involves adding *macro statements* to be processed dynamically at the server.

Although the macro file looks similar to an HTML document, the Web server accesses it through Net.Data as a CGI program. Net.Data expects two parameters: the name of the macro to process and the HTML section in that macro to display.

To understand what happens when a macro is executed, let's go through the following example macro section-by-section and see what each statement means.

```
%{ ********       Define Section        *********%}
%DEFINE {
   page_title="Web Macro Template"
%}
%DEFINE { DATABASE="*LOCAL" TABLE="LIB.FILE"
Mytable=%TABLE(ALL) %}


%{ ******** Function Definition Section *********%}
%FUNCTION(DTW_REXX) rexx1 (IN input) returns(result)
    { %EXEC{/qsys.lib/netdatawww.lib/qrexsrc.file/ompsamp.mbr %}
%}

%FUNCTION(DTW_REXX) today () RETURNS(result)
  {
      result = date()
%}

%{ *******      HTML Section: Input    **********%}
 %HTML (INPUT) {
 <html>
 <head>
 <title>$(page_title)</title>
 </head>
 <body>
 <h1>Input Form</h1>
 <hr>
 Today is @today()

 <FORM  METHOD="post" ACTION="output">
 Type some data to pass to a REXX program:
 <INPUT NAME="input_data" TYPE="text" SIZE="30">
 <p>
 <INPUT TYPE="submit" VALUE="Enter">
 </FORM>
 <hr>
 <p>
 [
 <a href="/">Home page</a>
 ]
```

```
</body>
</html>

%}

%{ ******    HTML Section: Output     **********%}
%HTML (OUTPUT) {
<html>
<head>
<title>$(page_title)</title>
</head>
<body>
<h1>Output Page</h1>
<hr>
<p>@rexx1(input_data)
<p>
<hr>
<p>
[
<a href="/">Home page</a> |
<a href="input">Previous page</a>
]
</body>
</html>

%}
```

4.2.1 Define Section

The Define section allows you to set up the variables that are used in the macro.

```
%{ ********       Define Section        **********%}
%DEFINE { page_title="Web Macro Template"
%}
%DEFINE { DATABASE="*LOCAL" TABLE="LIB.FILE"
Mytable=%TABLE(ALL) %}
```

Figure 44. Define Section of Web Macro

The first line is a comment. The format of a comment is *%{ ...one or more comment lines... %}*. Comments usually appear before each macro section. They cannot appear in a macro section. The next statement starts a DEFINE section. You can define multiple variables in one define section. In this example, only one variable, "page_title", is defined. Once defined, this variable can be referenced anywhere in the macro using the syntax, $(page_title). Using variables makes it easy to make global changes to your macro later. The second DEFINE establishes what database to connect to (it can be local or remote). The database must be defined in the "Relational Database Directory using the WRKRDBDIRE command. The last line of this section, "%}", identifies the end of the DEFINE section.

4.2.2 Function Definition Section

The function definition section allows you to define the language environments you use and what parameters are passed to the language environment.

```
%{ ********************** Function Definition Section ************************%}
%FUNCTION(DTW_REXX) rexx1 (IN input) returns(result)
                          ↑This function accepts one parameter
                          and returns a result which is substituted
                          for the associated function call
     { %EXEC{/qsys.lib/netdatawww.lib/qrexsrc.file/ompsamp.mbr %}
                          ↑ This function executes an
                          external REXX program called "ompsamp.mbr"
%}

%FUNCTION(DTW_REXX) today () RETURNS(result) {
      result = date()     ←The single source statement for this function is
                          contained inline.
%}
```

Figure 45. Function Definition Section

This function definition section contains two function declarations. The first, "rexx1", is a REXX function declaration that, in turn, executes an external REXX program called "ompsamp.cmd". One input variable, "input", is accepted by this function and automatically passed to the external REXX command. The REXX command also returns one variable called "result". The contents of the "result" variable in the REXX command replaces the invoking @rexx1() function call contained in the OUTPUT section. The variables "input" and "result" are directly accessible by the REXX program as you can see in the following source for "ompsamp.cmd":

```
/* REXX */
result = 'The REXX program received "'input'" from the macro.'
```

The code in this function echoes the data that was passed to it. You can format the resulting text any way you want by enclosing the requesting @rexx1() function call in normal HTML style tags (such as <b> or). Rather than using the "result" variable, the REXX program can have written HTML statements to standard out using REXX SAY statements. Although this is acceptable, it minimizes the degree of separation between program logic and screen presentation.

The second function declaration, "today", also refers to a REXX program. However, the entire REXX program (one entire line, in this case) is contained in the function declaration itself. An external program is not needed. Inline programs are allowed for REXX functions because it is an interpreted language that can be parsed and executed dynamically. Inline programs have the advantage of simplicity by not requiring a separate program file to manage. The first REXX function could also just as well have been handled inline.

A function must be defined before it is called.

4.2.3 HTML INPUT Section

```
%{ **********************      HTML Section: Input    ************************%}
%HTML (INPUT) {                ←Identifies the name of this HTML section
<html>
<head>
<title>$(page_title)</title>  ←Note the variable substitution from
                                    the define section.
</head>
<body>
<h1>Input Form</h1>
<hr>
Today is @today()              ←This line contains a call to the
                                    REXX function today.
Type some data to pass to a REXX program:
<FORM METHOD="post" ACTION="output">  ←When this form is
                                        submitted, the "output"
                                        HTML section is called.
<INPUT NAME="input_data"       ←"input_data" is now an implicitly defined
      TYPE="text" SIZE="30">        variable that can be referenced elsewhere in
                                    this macro. It is initialized to whatever
                                    the user types into the input field.
                                    We will use it in the output section .
<p>
<INPUT TYPE="submit" VALUE="Enter">
</FORM>
<hr>
<p>
[
<a href="/">Home page
]

</body>
</html>

%}                             ←Closes the HTML section
```

Figure 46. HTML Section: Input

This section contains the HTML for a simple form with one input field. The entire HTML section is surrounded by the HTML section identifier, %HTML (INPUT) { ... %}. INPUT identifies the name of this section. You can give it any name *(an enhancement from DB2WWW Connection, which only allowed INPUT and REPORT).* The HTML <title> statement contains an example of macro variable substitution. The contents of the variable "page_title" are substituted into the title of the form.

This section also has an example of a function call. The expression @today() is a call to function "today". This function is defined in the FUNCTION definition section that is described in Section 4.2.2, "Function Definition Section" on page 118. The result of the "today" function (today's date) is inserted into the HTML text in the same location that the @today() expression is located.

The ACTION parameter of the FORM statement is an example of navigation between HTML sections or between macros. Referencing the name of another section in an "ACTION" parameter accesses that section when the form is submitted. Any input data is passed to the new section as implicit variables. This is true of the single input field defined on this form. When the form is

submitted, data typed into this field is passed to the ″output″ section in the variable ″input_data″.

You can access sections in other macros with a relative reference. For example, ACTION=″../othermacro.mbr/main″ accesses the ″main″ HTML section in a macro file called ″othermacro.mbr″ on the AS/400 system. Again, any data typed into the form is passed to this macro in the implicit variable ″input_data″.

Notice again the simplicity of writing a macro. There is no need to deal with environment variables to receive input data as you do with CGI-BIN programs. That processing is handled for you automatically by Net.Data. You only need to reference the variable names.

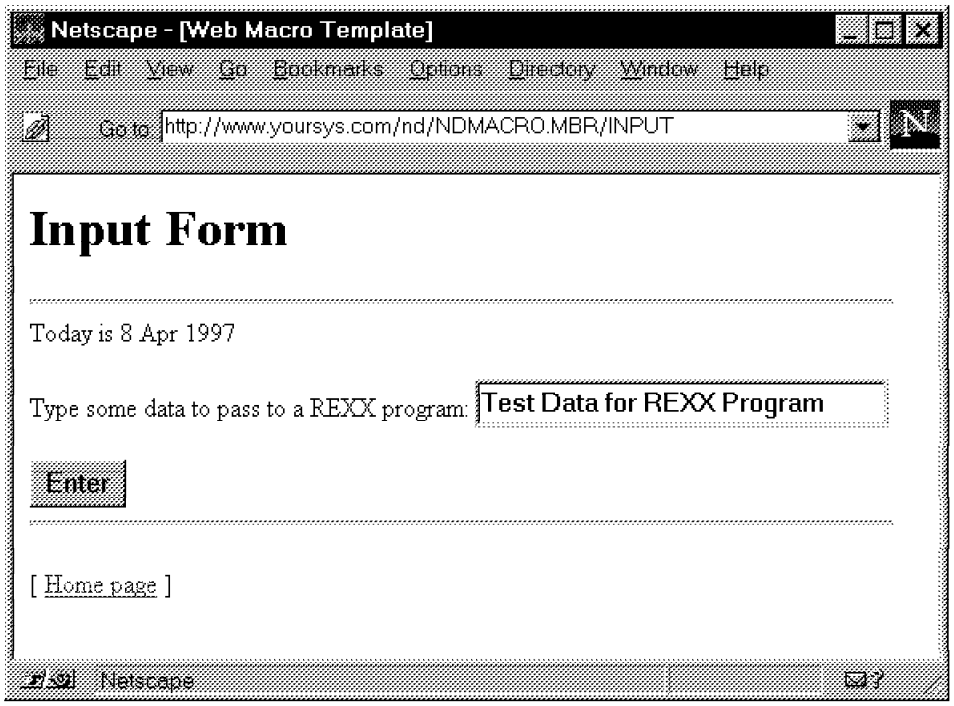


Figure 47. Browser View of Input HTML

4.2.4 HTML OUTPUT Section

```
%{ **********************    HTML Section: Output     ************************%}
%HTML (OUTPUT) {
<html>
<head>
<title>$(page_title)</title>  ←More substitution
</head>
<body>
<h1>Output Page</h1>
<hr>
<p>@rexx1(input_data)  ←This line contains a call to function rexx1
                            passing the argument ″input_data″
<p>
<hr>
<p>
[
<a href=″/″>Home page |
<a href=″input″>Previous page
]
</body>
</html>
%}
```

Figure 48. HTML Section: Output

The same as the INPUT section, this section is standard HTML enhanced with macro statements to substitute variables and a function call. Again, the ″page_title″ variable is substituted into the title statement. And, as before, this section contains a function call. In this case, it calls the function ″rexx1″ and passes to it the contents of the variable ″input_data″ that it received from the INPUT form. You can pass any number of variables to and from a function. The function definition determines the number of variables passed and their type. This is the output as displayed in the browser.

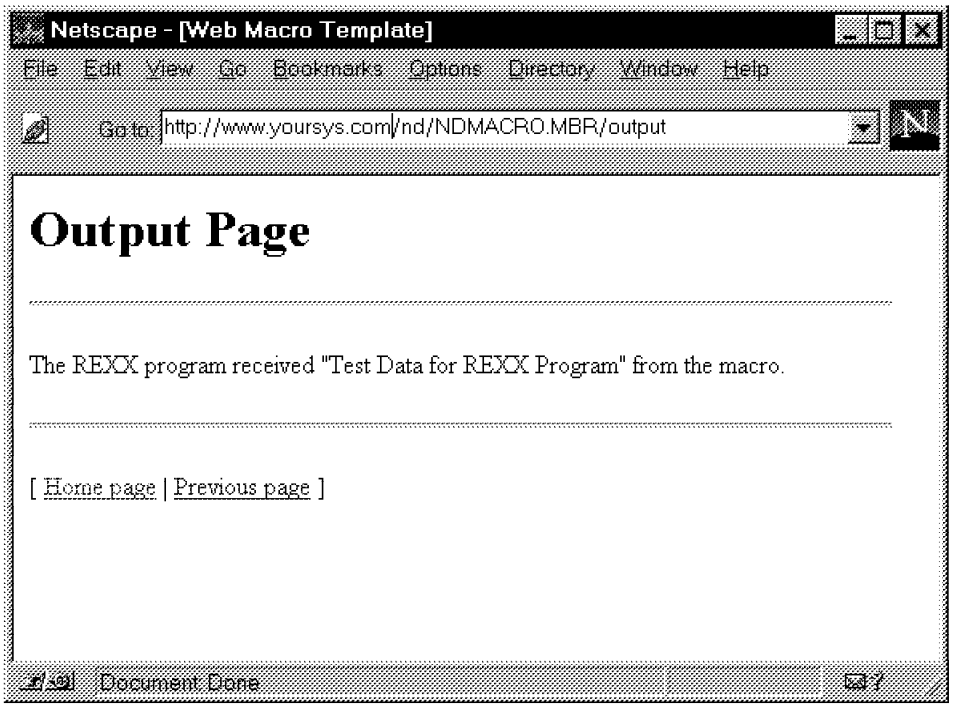


Figure 49. Browser View of HTML Output

4.3 Generating HTML in a Web Macro

The ability to access data and applications and to present that information as HTML on a Web browser is what makes Net.Data so powerful.

4.3.1 HTML Blocks

The %HTML block and the functions that get invoked from the %HTML block are the sections of the Web macro that generate HTML output to the browser. Whenever Net.Data is invoked, an HTML block must be specified. What is contained in this block controls the rest of the Net.Data invocation.

Any valid HTML may appear in an %HTML block. In addition, %INCLUDE statements, function calls, and variable references can be in an %HTML block. A common use of %HTML blocks in a Web macro is shown by this sample Web macro:

```
%{ **********************      Define Section         ************************%}
%DEFINE{ DATABASE="*LOCAL"
         TABLE="ASOLIB.HARDWARE"
 %}

%{ **********************      HTML Section: Input    ************************%}
%HTML(INPUT){
<H1>Hardware Query Form</H1>
<HR>
<FORM METHOD="POST" ACTION="OUTPUT">
<dl>
<dt>What hardware do you want to list?
<dd><input type="radio" name="hdware" value="'MON'" checked>Monitors
<dd><input type="radio" name="hdware" value="'PNT'">Pointing devices
<dd><input type="radio" name="hdware" value="'PRT'">Printers
```

```
<dd><input type="radio" name="hdware" value="'SCN'">Scanners
</dl>
<input type=submit value="Submit">
</FORM>
%}
%FUNCTION(DTW_SQL) queryHardware() {
SELECT MODNO, COST, DESCRIP FROM $(TABLE) WHERE TYPE=$(hdware)
   %REPORT{
<B>Here is the list you requested:</B><BR>
  %ROW{
<HR>
$(N1): $(V1)    $(N2): $(V2)
<P>
$(V3)
    %}
  %}
%}
%{ **********************    HTML Section: Output     ************************%}
%HTML(OUTPUT){
<H1>Hardware Query Results</H1>
<HR>
@queryHardware()
<HR>
<A href="http://www.ibm.com/cgi-bin/db2www/equiplst.mbr/input">List of hardware</A>
%}
```

The Web macro might initially be invoked from an anchor reference such as the following example:

```
<A href="http://www.ibm.com/cgi-bin/db2www/equiplst.mbr/input">List of hardware</A>
```

When the application user clicks on this reference, Net.Data is invoked and Net.Data parses the Web macro file. When it gets to the %HTML block specified on the invocation (in this case, the %HTML(INPUT) block), it begins to process the text inside the block. Anything that is not recognized by Net.Data as a Web macro language construct is assumed to be HTML and is sent to the browser to be displayed. After a selection is made and the Submit button pressed, the **ACTION** part of the HTML **<FORM>** element is executed, which specifies a call to the Web macro's %HTML(OUTPUT) block. The %HTML(OUTPUT) block is processed just as the %HTML(INPUT) block was. All data up to the *@queryHardware()* function call is output to the browser as HTML.

The *queryHardware()* function call is processed, which, in turn, invokes the SQL %FUNCTION block. After the *$(hdware)* variable reference is replaced in the SQL statement with the value returned in the input form, the query is executed. At this point, Net.Data again starts sending HTML to the browser, displaying the results of the query according to the HTML specified in the %REPORT block.

After the %REPORT block processing is done, we return again to the %HTML(OUTPUT) block and finish processing by sending out the remaining HTML specified after the *@queryHardware()* function call.

Only one %HTML block is processed for each Net.Data invocation. However, by using HTML anchor references and forms, it becomes easy to let the end-user start another invocation of Net.Data on another %HTML block, all controlled by you.

4.4 Web Macro Functions

This section discusses the Net.Data macro function definitions.

4.4.1 Define Functions

You can define your own functions or use Net.Data's library of built-in functions. For functions that are not built-in, use a %FUNCTION block to define the function in the Web macro. The syntax is shown here:

```
%FUNCTION(type) function-name([usage parameter, ...]
 ) [RETURNS(return-var)] {
  executable-statements
  [report-block]
  [message-block]
 %}
```

Table 10 (Page 1 of 2). Net.Data Function Definitions

Implicit Function Definitions	
Type	Identifies a *language environment* that is configured in the initialization file. The language environment invokes a specific language processor (which processes the executable statements) and provides a standard interface between Net.Data and the language processor.
Function name	This is the name of the %FUNCTION block. The %FUNCTION block is executed by referencing this name elsewhere in the Web macro preceded by an *at* (@) sign. Multiple %FUNCTION blocks can exist with the same name. They must all have identical parameter lists. When the function is called, all %FUNCTION blocks with the same name are executed in the order that they are defined in the Web macro.
Usage	IN, OUT, or INOUT. This indicates whether the parameter is passed into or received back from the %FUNCTION block, or both. The usage type applies to all of the following parameters in the parameter list until changed by another usage type. The default type is IN.
Parameter	The name of a locally scoped variable that is replaced with the value of a corresponding argument specified on a function call. Parameter references (for example, *$(parm1)*) in the executable statements or report section are replaced with the actual value of the parameter. In addition, parameters are passed to the language environment and are accessible to the executable statements using the natural syntax of that language or as environment variables. Parameter variable references are not valid outside the %FUNCTION block. You can also pass implicit parameters on a function call of a type you specify. You must define the parameters in the ENVIRONMENT statement in the initialization file. Parameters can only be specified once in the parameter list.
Return var	Specify this parameter after the *RETURNS* keyword. It identifies a special OUT parameter. The value of the return variable is assigned to the function call and replaces the function call in the Web macro processing. If you do not specify the RETURNS clause, the value of the function call is the null string if the return code from the call to the language environment is 0 or the value of the return code otherwise.

Table 10 (Page 2 of 2). Net.Data Function Definitions	
Implicit Function Definitions	
Executable statements	After the variables are substituted and the function calls are processed, these statements are passed to the specified language environment for execution. Each language environment processes the statements differently.
Report_block	See Section 4.5, “Report Blocks” on page 134.
Message block	See Section 4.5.1, “Message Blocks” on page 136.

You need to define functions at the outermost Web macro layer before they are called in the Web macro.

Besides letting you define your own %FUNCTION blocks, Net.Data also has a library of built-in functions. You do **not** need to define these functions before you reference them. Just call them from a Web macro anywhere a function call can be made. See Section 4.4.3, “Net.Data Built-In Functions” on page 129 for a list of these functions.

4.4.2 Calling Functions

You invoke a function from a Web macro using the at (@) character followed by a %FUNCTION block name:

```
@function_name([ argument,... ])
```

Table 11. Net.Data Calling Functions	
Function_name	This is the name of the %FUNCTION block to invoke. The function must already be defined in the Web macro unless this is a built-in function.
Argument	This is the name of a defined variable or a literal character string. Arguments on a function call are matched up with the parameters on a %FUNCTION block and each parameter is assigned the value of its corresponding argument for the duration of the %FUNCTION block. The arguments must be the same number and type as the corresponding parameters.

When a %FUNCTION block is invoked, Net.Data proceeds this way:

1. Net.Data matches up all of the arguments on the function call with the parameters from the %FUNCTION block. If the number or usage type of the variables do not match, an error occurs.
2. A set of variables is built from all of the function parameters specified in the %FUNCTION block and the arguments specified in the ENVIRONMENT statement in the initialization file. There are two distinct sets of variables at this step: the *global* set, consisting of all variables that have been defined in the Web macro so far and the *local* set just built for the function.
3. Variable references in the executable text of the %FUNCTION block are replaced with the actual value of the variable by looking for the variable in the local variable set and then in the global variable set. If the variable exists in both sets (for example, it was specified as a function parameter and on a previous %DEFINE), the local set takes precedence.
4. Function calls in the executable text of the %FUNCTION block are processed. The context of the function call does not matter to Net.Data within executable text. For example, if your executable text contains conditional logic, and a

function call exists within a particular leg of that conditional logic, it is processed by Net.Data regardless of the logic. This is because Net.Data processes the function call before the executable text.

5. The variables in the local set are passed to the language environment along with the text of the executable statements. The language environment is responsible for passing the IN and INOUT variables to the language processor, interpreting the executable statements or invoking the language processor to execute the statements and retrieving any OUT or INOUT variables back from the language processor when the program completes.
6. When the language environment returns to Net.Data, any OUT or INOUT parameters are obtained from the parameter list and their values are used to replace the values of their corresponding arguments in the local and global variable sets. If there is a RETURNS clause, the value of the return variable is saved with the %FUNCTION block information so that it can be used to replace the function call in the Web macro expansion.

The rules for using and modifying variables from a %FUNCTION block in a function call can be summarized as follows:

1. All variables, both globally-defined and function parameters, are used to perform variable substitution on the executable statements before invoking the function.
2. Only IN or INOUT parameters are passed to the language environment.
3. Only OUT or INOUT parameters can be modified by the function.
4. All variables, both globally-defined and function parameters, are used to perform variable substitution on the report section of the %FUNCTION block after invoking the function.

When %MESSAGE block and %REPORT block processing is complete, the value of the function call is used to replace the function call in the Web macro.

4.4.2.1 Calling High-Level Language Programs

This example shows you how to call an RPG program from Net.Data. The same method is used to call other high-level language programs on the AS/400 system. To pass parameters from Net.Data to your high-level language program, you need to use the same API (Application Programming Interface) you use for CGI-BIN programs. This means using the API Get Environment Variables (QtmhGetEnv) to get Variables from Net.Data and Write to Standard out (QtmhWrStout) to return a text buffer to Net.Data. For a more detailed discussion about these APIs, see Section 3.9, “Programming CGI-BIN with ILE RPG/400 and ILE COBOL/400” on page 82.

Returning Variables to Net.Data

Currently it is not possible to return environment variables to Net.Data from RPG or COBOL. One solution for this is to call a C program to return the environment variables to Net.Data. A new API, QtmhPutEnv, is planned for V4R1 and will allow RPG or COBOL programs to return varibles to Net.Data.

This is the listing of the RPG program that we call from the Net.Data macro. This simple program uses the API QtmhGetEnv to read the input from the browser. We add a little constant text to the output buffer and use the API QtmhWrStout to return the modified string to the browser.

```
0001.00  * Data structure for error reporting.  Copied from QSYSINC/QRPGLESRC(QUSEC)
0002.00 deol              c                   x'15'
0003.00 dtxt              c                   ' Text added to the buffer'
0004.00 DQUSEC            DS
0005.00 D*                                            Qus EC
0006.00 D QUSBPRV                 1      4B 0 INZ(16)
0007.00 D*                                            Bytes Provided
0008.00 D QUSBAVL                 5      8B 0
0009.00 D*                                            Bytes Available
0010.00 D QUSEI                   9     15
0011.00 D*                                            Exception Id
0012.00 D QUSERVED               16     16
0013.00  ******************************************************************************************
0014.00  *These are for the APIEnVar
0015.00 DENBuff           S           1024A   INZ                                 Return area for API
0016.00 DENBuffLn         S              9B 0 INZ(1024)                           Ln of return area
0017.00 DENActLn          S              9B 0 INZ                                 Act Ln of ret data
0018.00 DENVarName        S             64A   INZ                                 Name of Env Var
0019.00 DENVarLn          S              9b 0 INZ                                 Ln of EnVar name
0020.00  *These are used for APIStdOut
0021.00 DOutBuff          S            100A                                       Area for output
0022.00 DOutBuffLn        S              9B 0 INZ(100)                            Length of Output ar
0023.00  ******************************************************************************************
0024.00 C                   MOVEL     'parm1'       ENVarName
0025.00 C                   Z-ADD     05            ENVarLn
0026.00 C                   CALLB     'QtmhGetEnv'
0027.00 C                   parm                    ENBuff                       Input buffer
0028.00 C                   parm                    ENBuffLn                     Input buffer max ln
0029.00 C                   parm                    ENActLn                      StdIn actual length
0030.00 C                   parm                    ENVarName
0031.00 C                   parm                    ENVarLn
0032.00 C                   parm                    QUSEC
0033.00  ******************************************************************************************
0034.00  *  Upon return, your Query_String data is in ENBuff with the length
0035.00  *  of the data returned in ENActLn
0036.00  *  Move this data to the DBCvt parms
0037.00 C                   eval      outbuff = %subst(enbuff:1:enactln)+txt+eol
0038.00 C     ' '           checkr    outbuff       outbuffln
0039.00 C                   callb     'QtmhWrStout'
0040.00 C                   parm                    OUTBuff                       Output buffer
0041.00 C                   parm                    OUTBuffLn                     Output buffer len h
0042.00 C                   parm                    QUSEC
0043.00  ******************************************************************************************
0044.00 C                   eval      *inlr = *on
0045.00 C                   return
```

Figure 50. RPG Program Called from Net.Data

This macro calls the RPG program and returns the result to the browser.

```
%{********* Function **********%}
%FUNCTION (DTW_SYSTEM) rpgpgm (in parm1) {
 %EXEC{ /QSYS.LIB/NDCGI.LIB/rpgpgm.pgm %}
%}
%{********* INPUT *************%}
%HTML (input) {
<HTML>
<FORM METHOD="POST" ACTION="OUTPUT">
Enter input for the RPGPGM here:<br>
<INPUT TYPE=text NAME=prod>
<INPUT TYPE=submit>
</FORM>
</HTML>
%}
%{********* REPORT ************%}
```

```
%HTML (OUTPUT) {
<HTML>
We will now call the RPG program and pass the parameter: <P>
$(prod) <P>
<HR>
The output of the RPG program is:<P>
@rpgpgm(prod)
</HTML>
%}
```

This is the input window where you enter the value to be passed to the RPG program.

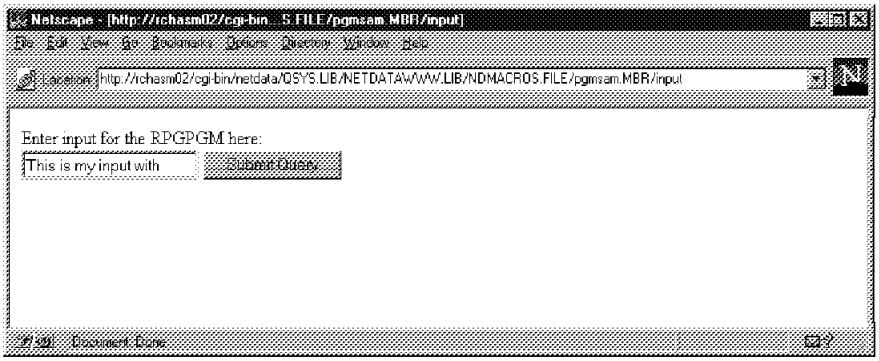


Figure 51. RPG Program Input Window

When the user presses the ″Submit Query″ button, Net.Data parses the ″Report″ HTML section and runs the program RPGPGM.PGM, passing the value entered in the ″INPUT″ HTML section and returns the modified value to the browser.

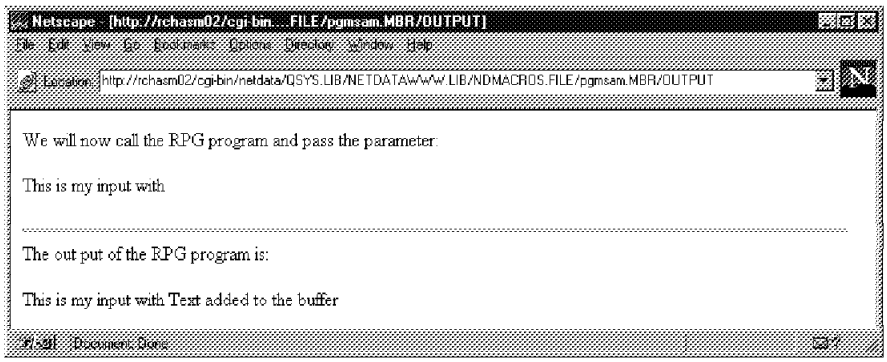


Figure 52. RPG Program Report Window

An alternative way to call a high level program and pass parameters is through the REXX language environment as demonstrated with the following example that calls the ″count″ RPG program from the code snippets page. To download the count program, follow the link from the AS/400 Web Builders pages:

`http://205.217.130.15/workshop/snippets/snippets.htm`
or
`http://www.as400.ibm.com/workshop/webbuild.htm`

The ″count″ program allows you to use one program to add counters to multiple pages within a Net.Data macro by passing a unique ″countname″ for each page you want the counter on.

```
%{ **************       Define Section        ********************%}
%DEFINE { countvalue="0" %}
%DEFINE { countname="PGCNTR1" %}
%{ ************** Function Definition Section ********************%}
%FUNCTION(DTW_REXX) CNTR2 (IN countname, INOUT countvalue)
    { 'CALL PGM(CGIDEV/COUNT) PARM(&countname &countvalue)' %}
%{ **********      HTML Section: Input    ************************%}
%HTML (INPUT) {
<html>
<head>
<title>Counter</title>
</head>
<BODY>
@CNTR2(countname, countvalue)
This page has been viewed <b>$(countvalue)</b> times!
<p>
@DTW_ASSIGN (countname, "PGCNTR2")
@CNTR2(countname, countvalue)
This page2 has been viewed <b>$(countvalue)</b> times!
</HTML>
%}
```

This macro puts two counters on one page. This may not be what you want! However, it illustrates using the built-in function @DTW_ASSIGN to give the value "countname" a new value that adds a record to the data file CGICOUNT and returns the "countvalue" to the browser. Typically, you use the "%DEFINE { countname="PGCNTR1" %} to change this value.

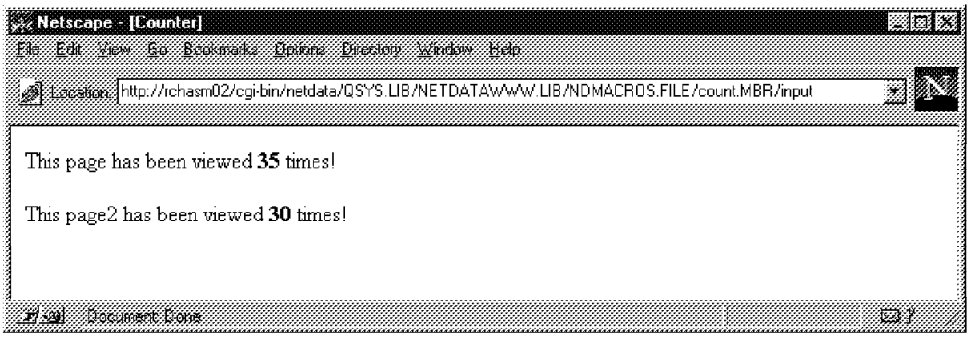


Figure 53. REXX Calls RPG Program COUNT

4.4.3 Net.Data Built-In Functions

Net.Data provides a large library of predefined functions for use within your macros. The use of these functions can simplify the construction of your dynamically created Web pages. The built-in functions fall into seven basic groups.

1. *General functions* are a group of functions that Web builders can use to develop Web pages that can change data and give access to some system services (that is, date and time).

   .

   - DTW_ADDQUOTE / DTW_rADDQUOTE / DTW_mADDQUOTE
   - DTW_DATE / DTW_rDATE
   - DTW_GETENV / DTW_rGETENV
   - DTW_GETINIDATA / DTW_rGETINIDATA

- DTW_HTMLENCODE / DTW_rHTMLENCODE
- DTW_QHTMLENCODE / DTW_rQHTMLENCODE
- DTW_SETENV / DTW_rSETENV
- DTW_TIME / DTW_rTIME
- DTW_URLESCSEQ / DTW_rURLESCSEQ

2. *Math functions* allow you to do math functions within the macro without the need to call a high level language program. This simplifies the writing of the Net.Data macro. The following math functions are currently supported.

   - DTW_ADD / DTW_rADD
   - DTW_DIVIDE / DTW_rDIVIDE
   - DTW_DIVREM / DTW_rDIVREM
   - DTW_FORMAT / DTW_rFORMAT
   - DTW_INTDIV / DTW_rINTDIV
   - DTW_MULTIPLY / DTW_rMULTIPLY
   - DTW_POWER / DTW_rPOWER
   - DTW_SUBTRACT / DTW_rSUBTRACT

3. *String manipulation functions* provide functions to modify character strings. Net.Data supports the following functions:

   - DTW_ASSIGN
   - DTW_CONCAT, DTW_rCONCAT
   - DTW_DELSTR, DTW_rDELSTR
   - DTW_INSERT, DTW_rINSERT
   - DTW_LASTPOS, DTW_rLASTPOS
   - DTW_LENGTH, DTW_rLENGTH
   - DTW_LOWERCASE, DTW_rLOWERCASE, DTW_mLOWERCASE
   - DTW_POS, DTW_rPOS
   - DTW_REVERSE, DTW_rREVERSE
   - DTW_STRIP, DTW_rSTRIP
   - DTW_SUBSTR, DTW_rSUBSTR
   - DTW_TRANSLATE, DTW_rTRANSLATE
   - DTW_UPPERCASE, DTW_rUPPERCASE, DTW_mUPPERCASE

4. *Word manipulation functions* allow you to manipulate words in a character string. The following functions are supported:

   - DTW_DELWORD, DTW_rDELWORD
   - DTW_SUBWORD, DTW_rSUBWORD
   - DTW_WORD, DTW_rWORD
   - DTW_WORDINDEX, DTW_rWORDINDEX
   - DTW_WORDLENGTH, DTW_rWORDLENGTH
   - DTW_WORDPOS, DTW_rWORDPOS
   - DTW_WORDS, DTW_rWORDS

5. *Table manipulation functions* allow you to manipulate tables and easily build Web forms with dynamic data. These functions allow you to build select blocks, radio buttons, and so on, and fill them automatically with data from the AS/400 database. The table manipulation functions supported by Net.Data are:

   - DTW_TB_DLIST
   - DTW_TB_DUMPH
   - DTW_TB_DUMPV
   - DTW_TB_HTMLENCODE
   - DTW_TB_INPUT_CHECKBOX
   - DTW_TB_INPUT_RADIO

- DTW_TB_INPUT_TEXT
- DTW_TB_LIST
- DTW_TB_SELECT
- DTW_TB_TABLE
- DTW_TB_TEXTAREA

6. *Flat file interface functions* are not currently supported on the AS/400 system. It is planned that support for these functions will be included in a future release of the operating system. As defined by Net.Data, the flat file interface functions are:

   - DTWF_APPEND
   - DTWF_CLOSE
   - DTWF_DELETE
   - DTWF_INSERT
   - DTWF_OPEN
   - DTWF_READ
   - DTWF_REMOVE
   - DTWF_SEARCH
   - DTWF_UPDATE
   - DTWF_WRITE

7. *Web registry functions* are not currently supported on the AS/400 system. It is planned that support for these functions will be included in a future release of the operating system. As defined by Net.Data, the Web registry functions are:

   - DTWR_ADDENTRY
   - DTWR_DELENTRY
   - DTWR_DELREG
   - DTWR_LISTREG
   - DTWR_LISTSUB
   - DTWR_RTVENTRY, DTWR_rRTVENTRY
   - DTWR_UPDATEENTRY

For detailed information on any of the built-in functions, you should see the online *Net.Data Reference Guide* at

`http://www.software.ibm.com/data/net.data/docs/dtwref.htm`

4.4.4 Table Variables

The table variable defines a collection of related data within Net.Data. It contains an array of identical records, or rows, and an array of column names describing the fields in each row. A table is defined in the Web macro with a statement such as this:

```
%DEFINE myTable=%TABLE(30)
```

The number following %TABLE is the limit on the number of rows this table can contain. To specify a table with no limit on the number of rows, use the default or specify ALL as shown in these examples:

```
%DEFINE myTable2=%TABLE
%DEFINE myTable3=%TABLE(ALL)
```

A table can be passed between functions by referring to the table variable name. The individual elements of a table can be referred to in a %REPORT block of a function. Table variables are usually used for output from an SQL function and input to a report, but you can pass them as IN, OUT, or INOUT parameters to any

non-SQL function. Tables can only be passed to SQL functions as OUT parameters. A table can be represented graphically:

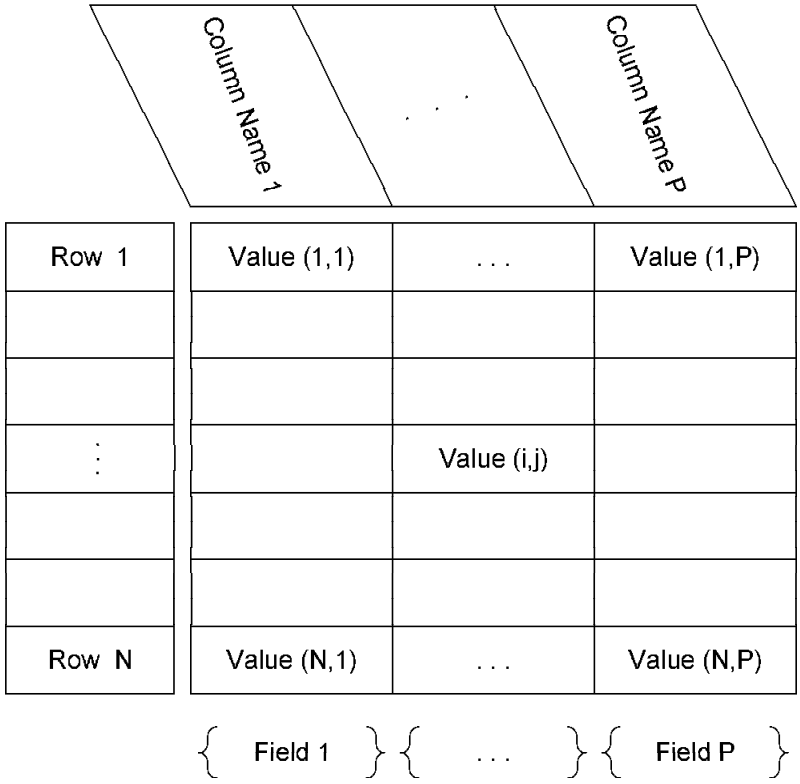


Figure 54. Net.Data Table View

The column names and field values in a table are addressed as array elements with an origin index of 1 rather than the standard C and C++ language convention of starting arrays at an index of 0.

4.4.5 Implicitly Defined Variables

Defining a table variable causes Net.Data to implicitly define two sets of variables that you can use to refer to the column names and field contents of the table. One set of these implicit variables is referred to in the %REPORT block of a %FUNCTION block in a Web macro, and the other set is referred to in programs called from language environments. You cannot refer to these variables in any other section of the Web macro.

Table 12 (Page 1 of 2). Net.Data %REPORT Block

Implicit %REPORT Variables	
N1,...N*p*	Where *p* is the column number. These variables contain the column names.
N_*column-name*	Where *column-name* is the value of one of the N*p* variables. If it exists, this variable has the value *column-name*.
NLIST	This LIST variable is created by concatenating all of the individual column names N*p*. The default separator for the list variable is a space character, but this can be changed by specifying it on a LIST define such as: %define %LIST ″;] ″; NLIST.

Table 12 (Page 2 of 2). Net.Data %REPORT Block	
Implicit %REPORT Variables	
V1...V*p*	Where *p* is the column number. This variable contains the value of the pth column in the current row. These variables are only valid in a %ROW block of a %REPORT block and are set to new values as each row is printed.
V_*column-name*	Where *column-name* is the value of one of the N*p* variables. This variable contains the value of the field in the current row whose column name is *column_name*. These variables are only valid in a %ROW block of a %REPORT block and are set to new values as each row is printed.
VLIST	This %LIST variable is created by concatenating all of the individual field values V*p* in the current row. The default separator for this list variable is a space character, but this can be changed by specifying it on a %LIST define. These variables are only valid in a %ROW block of a %REPORT block and are set to new values as each row is printed. This is most useful for creating tables in HTML 3.0.
ROW_NUM	Contains the row number of the current row. After the %ROW block has been processed, ROW_NUM contains the total number of rows in the table. This variable is only valid in a %ROW block of a %REPORT block and is set to a new value as each row is printed.
NUM_COLUMNS	Contains the number of columns in the table.
TOTAL_ROWS	Contains the number of rows in the table.

The following variables can be used in a REXX program called by a function call to a REXX %FUNCTION block.

Table 13. Net.Data REXX Language Environment Variables	
Implicit REXX Language Environment Variables	
*T*_ROWS	Where *T* is the table name. This variable contains the current number of rows in the table.
*T*_COLS	Where *T* is the table name. This variable contains the current number of columns in the table.
*T*_N.*p*	*T* is the table name and *p* is the column number. This variable contains the name of the pth column.
*T*_V.*i.p*	*T* is the table name, *i* is the row number, and *p* is the column number. This variable contains the value of the pth column of the ith row in the table.

You can use the following variables in programs invoked by a function call to a SYSTEM %FUNCTION block. These table variables are accessible as environment variables.

Table 14 (Page 1 of 2). Net.Data SYSTEM Language Environment Variables	
Implicit SYSTEM Language Environment Variables	
*T*_ROWS	*T* is the table name. This variable contains the current number of rows in the table.
*T*_COLS	*T* is the table name. This variable contains the current number of columns in the table.

Table 14 (Page 2 of 2). Net.Data SYSTEM Language Environment Variables	
Implicit SYSTEM Language Environment Variables	
*T*_N_*p*	*T* is the table name and *p* is the column number. This variable contains the name of the pth column.
*T*_V_*i*_*p*	*T* is the table name, *i* is the row number, and *p* is the column number. This variable contains the value of the pth column of the ith row in the table.

4.5 Report Blocks

The %REPORT block is used to format and display data output from a %FUNCTION block. This output is typically table data, although any valid combination of HTML tags, macro variable references, and function calls may be specified. A table name may be specified on the %REPORT block but is not required. If a table name is not specified, the table data used is that of the first output table in the parameter list of this %FUNCTION block. If no table was specified on the %FUNCTION block, the default table data is used.

The %REPORT block is composed of three parts, each of which is optional:

1. **Header:** Contains HTML data displayed before table row data.
2. **%ROW block:** Contains HTML and table variables displayed once per row of the table.
3. **Footer:** Contains data displayed after table row data.

If you do not want to display any table output from the %ROW block, just leave it empty.

When Net.Data processes a %FUNCTION block, a call is made to a language environment and data is returned. Net.Data processes the %REPORT block.

Inside the %REPORT block, several implicitly defined variables are made available to you to access the data in the Web macro table that is used in the report. These variables are described in Table 12 on page 132.

Header and footer information is not explicitly specified as such in a %REPORT block. Net.Data simply assumes that everything it finds before a %ROW block is header information and everything it finds after the %ROW block is footer information. As with the %HTML block, the Web macro processor treats everything in the header, %ROW block, and footer sections as HTML and sends that data to the browser unless it recognizes the data as a Web macro construct.

1. The header information is processed and displayed once.
2. %ROW block information is processed once for each row in the table.
3. The footer information is processed and displayed once.

You can decide not to display any data by specifying an empty %REPORT block this way:

```
%REPORT {
%}
```

Or you can display a default report by omitting the %REPORT block. The following example shows how a default report might look for a table with five rows and five columns:

```
 COL1    | COL2    | COL3    | COL4    | COL5    |
-------------------------------------------------
 ( 1 1 ) | ( 1 2 ) | ( 1 3 ) | ( 1 4 ) | ( 1 5 ) |
-------------------------------------------------
 ( 2 1 ) | ( 2 2 ) | ( 2 3 ) | ( 2 4 ) | ( 2 5 ) |
-------------------------------------------------
 ( 3 1 ) | ( 3 2 ) | ( 3 3 ) | ( 3 4 ) | ( 3 5 ) |
-------------------------------------------------
 ( 4 1 ) | ( 4 2 ) | ( 4 3 ) | ( 4 4 ) | ( 4 5 ) |
-------------------------------------------------
 ( 5 1 ) | ( 5 2 ) | ( 5 3 ) | ( 5 4 ) | ( 5 5 ) |
-------------------------------------------------
```

You can use the *DTW_DEFAULT_REPORT* special variable to enable or disable default reporting when no %REPORT block is specified. If *DTW_DEFAULT_REPORT* is set to *"NO"*, default reporting is disabled. If *DTW_DEFAULT_REPORT* is set to *"YES"*, default reporting is enabled. The default is "YES".

DTW_DEFAULT_REPORT only applies if no %REPORT block is defined.

Also, you have the choice of having the default report generated using HTML tables. To enable this, the *DTW_HTML_TABLE* special variable must be set to *"YES"*. The default is not to use HTML table tags.

This example shows how you can customize report formats using special variables and HTML tags. It displays the names, phone numbers, and Fax numbers from the table *CustomerTbl*.

```
%FUNCTION(DTW_SQL) custlist() {
  SELECT Name, Phone, Fax FROM CustomerTbl
%REPORT{
Phone Query Results:<BR>
==================== <BR>
%ROW{
Name: <B>$(V1):</B><BR>
Phone: $(V2)<BR>
Fax: $(V3)<BR>
---------------------------- <BR>
%}
Total records retrieved: $(ROW_NUM)
%}
%}
```

The resulting report looks similar to this:

```
Phone Query Results:
====================
Name: Doen, David
Phone: 422-245-1293
Fax: 422-245-7383
----------------------------
Name: Williamson, Jack
Phone: 955-768-3489
Fax: 955-768-3974
```

```
------------------------------
Total records retrieved: 2
```

Net.Data generated the report by:

1. Printing *Phone Query Results*, once at the beginning of the report.
2. Giving the variables *$(V1)*, *$(V2)*, and *$(V3)* the values for Name, Phone, and Fax respectively for each row as it is retrieved.
3. Drawing a line after each row retrieved to help readability.
4. Printing the string *Total records retrieved:* and the value for *$(ROW_NUM)* once at the end of the report.

4.5.1 Message Blocks

The %MESSAGE block lets you determine how to proceed after a function call based on the success or failure of the function call, and lets you display information to the caller of the function.

Net.Data sets RETURN_CODE, an implicit variable, for each function call. RETURN_CODE is set to the return code of the call to the language environment the function calls. When the function call is completed, the %MESSAGE block uses the value of RETURN_CODE to determine how to proceed. A %MESSAGE block consists of a series of message statements, each statement specifying a return code value, message text, and an action to take. The syntax of a %MESSAGE block is shown here:

```
%MESSAGE {
  [return_code : message_text [ : action]]
        .
        .
        .
%}
```

Table 15 (Page 1 of 2). Message Block Definitions

return_code	Any positive or negative number. If the RETURN_CODE variable value matches this value, the remaining information in the message statement is used to process the function call. There are three special values that can be specified for *return_code*: • *+default:* If RETURN_CODE is greater than 0 and an exact match is not specified, the information in this message statement is used to process the function call. • *-default:* If RETURN_CODE is less than 0 and an exact match is not specified, the information in this message statement is used to process the function call. • *default:* If RETURN_CODE does not equal 0 and an exact match is not specified and the *+default* (for RETURN_CODE greater than 0) or *-default* (for RETURN_CODE less than 0) value is not specified, the information in this message statement is used to process the function call.
Message_text	This string is sent to the Web browser if the RETURN_CODE matches the *return_code* value in this message's message statement.

Table 15 (Page 2 of 2). Message Block Definitions	
Action	This determines what action Net.Data takes if the RETURN_CODE matches the *return_code* in this message statement. There are two valid values: • *exit:* Net.Data exits immediately. • *continue:* Net.Data continues processing the Web macro in the function block it was calling. This is optional. If no action is specified, the default is *exit*.

A %MESSAGE block can have a global or a local scope. If the %MESSAGE block is defined in a %FUNCTION block, it is scoped locally to that %FUNCTION block. If it is specified at the outermost macro layer, it has global scope and is active for all function calls executed in the Web macro. If you defined more than one global %MESSAGE block, the last one defined is active.

Net.Data uses these rules to process a RETURN_CODE from a function call:

1. Check local %MESSAGE block for an exact match; exit or continue as specified.
2. If RETURN_CODE is not 0, check local %MESSAGE block for *+default* or *-default*, depending on the sign of RETURN_CODE; exit or continue as specified.
3. If RETURN_CODE is not 0, check local %MESSAGE block for *default*; exit or continue as specified.
4. Check global %MESSAGE block for an exact match; exit or continue as specified.
5. If RETURN_CODE is not 0, check global %MESSAGE block for *+default* or *-default*, depending on the sign of RETURN_CODE; exit or continue as specified.
6. If RETURN_CODE is not 0, check global %MESSAGE block for *default*; exit or continue as specified.
7. If RETURN_CODE is not 0, issue Net.Data internal default message and exit.

Here is an example. Assume the following %MESSAGE blocks are defined in your Web macro:

```
%{ global message block %}
%MESSAGE {
   -100     : "return code -100 message"   : exit
    100     : "return code 100 message"    : continue
   +default : { This a long message that spans more
               than one line. You can use HTML tags, including
               anchors and forms, in this message also. %}   : continue
%}

%{ local message block inside a %FUNCTION block %}
%FUNCTION(DTW_REXX) my_function() {
  %EXEC { my_command.cmd %}
  %MESSAGE {
     -100     : "return code -100 message"   : exit
      100     : "return code 100 message"    : continue
     -default : { This is a second long message that spans more
                  than one line. You can use HTML tags, including
                  anchors and forms, in this message. %}   : exit
  %}
%}
```

Figure 55. Message Block Macro

If *my_function()* returns with RETURN_CODE set to 50, Net.Data processes in this order:

1. Check for an exact match in the local %MESSAGE block. (There is none.)
2. Check for a *+default* in the local %MESSAGE block. (There is none.)
3. Check for a *default* in the local %MESSAGE block. (There is none.)
4. Check for an exact match in the global %MESSAGE block. (There is none.)
5. Check for a *+default* in the global %MESSAGE block. (There is one.)

Now that Net.Data found a match, it sends the message text to the Web browser and checks the requested action. Because *continue* is specified, Net.Data continues to process the Web macro after printing the message text.

For example, if error 100 is found during processing with the %MESSAGE block in the example, output from a program can look similar to this:

```
.
.
.
11 November 1996                  $245.45
13 November 1996                  $623.23
19 November 1996                  $ 83.02
return code 100 message
22 November 1996                  $ 42.67

Total:                            $994.37
```

4.6 Language Environments

Net.Data is designed to allow new language and database interfaces to be added in a ″pluggable″ fashion. These language environments are accessed as service programs. The name of the service program is configured in the Net.Data initialization file and associated with a language environment name. Each language environment must support a standard set of interfaces defined by Net.Data.

Net.Data for OS/400 provides the following language environments:

REXX Allows external REXX programs or inline (REXX statements in a Web macro function block) to be interpreted by the REXX interpreter.

SQL Allows SQL statements to be processed by DB2.

System Allows external programs (C, RPG, COBOL) to be run.

The following sections describe the language environments previously listed.

4.6.1 REXX (DTW_REXX) Language Environment

The REXX language environment can interpret internal REXX programs that are specified in a %FUNCTION block of the Web macro, or it can execute external REXX programs stored in a separate file. Some of the characteristics of the REXX language environments are:

- The QREXX() REXX application program interface (API) is used by the language environment to start the REXX interpreter for a REXX program.
- The QREXVAR() REXX API is used by the REXX language environment to pass data to a REXX program and to read data from a REXX program. Thus, REXX programs can directly manipulate the Net.Data parameter variables specified in a %FUNCTION block.
- The REXX language environment starts the REXX interpreter in the default command environment, COMMAND (the CL command environment). If you want to run another command environment, such as EXECSQL (SQL environment), CPICOMM (CPI communications environment), or a user-defined language environment, you need to specify the language environment in the REXX program using the ADDRESS built-in REXX function.
- The REXX language environment requires external REXX programs to reside in the QSYS.LIB file system.
- The QTMHHTP1 user profile must have the proper authority to access and read the file that contains the external REXX program in addition to any resources that a REXX program uses.

Calls to external REXX programs are identified in a %FUNCTION block by a statement of the form:

```
%EXEC{ REXX-file-name [optional parameters] %}
```

The following simple example shows a macro named REXXM with both an internal REXX program and a reference to an external REXX program:

```
%define a = "3"
%define b = "0"
%function(DTW_REXX) func1(IN inp1, OUT outp1) {
%EXEC{
/QSYS.LIB/REXX.LIB/REXXSRC.FILE/TREXX.MBR
%}
%}
%function(DTW_REXX) func2(IN inp1, OUT outp1) {
   outp1 = 2*inp1
%}
%HTML(REPORT) {
@func1(a, b)
b=$(b)
@func2(a, b)
b=$(b)
%}
```

Figure 56. Macro REXXM

In the example, @func1 results in the REXX program TREXX.MBR being interpreted by the REXX interpreter, and @func2 results in the REXX interpreter interpreting the statement "outp1 = 2*inp1". In both cases, the REXX variable pool is set so that the REXX interpreter can access variables "a" and "b".

This is an example URL that references the macro, assuming that:

- The macro is located in the /WWW/macro directory.
- The HTTP server has been configured to call the Net.Data CGI-BIN program DB2WWW.
- The user profile that CGI-BIN programs run under (QTMHHTP1) has been given authority to access the macro file and the file containing the REXX program.

```
http://hostname/cgi-bin/db2www/WWW/macro/REXXM/report
```

If you run this macro with the contents of the TREXX.MBR as shown:

```
 Columns . . . :    1  80
 SEU==>
 FMT **  ...+... 1 ...+... 2 ...+... 3 ...+... 4 .
        *************** Beginning of data ********
0001.00 outp1 = 3*inp1
        ****************** End of data ***********
```

Figure 57. REXX Source of TREXX.MBR

The output to the browser is:

```
b=9 b=6
```

If you did not create a Net.Data initialization file, the REXX language environment is enabled by default. If you create an initialization file and you want to use the REXX language environment, the following configuration statement must be in the initialization file:

```
ENVIRONMENT(DTW_REXX) /QSYS.LIB/QTCP.LIB/QTMHREXX.SRVPGM ( )
```

4.6.2 SQL (DTW_SQL or SQL) Language Environment

The SQL language environment is used to execute SQL statements using DB2. Some of the characteristics of the SQL language environments include:

- The SQL language environment requires that a directory entry for the local database is in the relational database directory (that is, a directory entry with a remote location of *LOCAL). An entry can be added by using the Add Relational Database Directory Entry (ADDRDBDIRE) command.
- Any valid SQL statement can be passed to the SQL language environment. The SQL statements must be valid DB2 for OS/400 commands.
- SQL or system naming mode can be used. SQL naming mode is the default. SQL naming mode uses a period when naming the tables.

  ```
  Library.Table
  ```

  System naming mode uses the standard slash.

  ```
  Library/Table
  ```

  If system naming mode is desired, insert the following line in the Net.Data initialization file:

  ```
  DTW_SQL_NAMING_MODE=SYSTEM_MODE
  ```

  You can also set DTW_SQL_NAMING_MODE to SQL_MODE, which is the same as the default of SQL naming mode.

  When a connection is made to a remote AS/400 system, the SQL Call Level Interface looks for an *SQLPKG object in library QGPL with the name of QSQCLIPKG. If it exists, it is used, but if it does not exist it is created. This SQL package contains all the rules by which the native SQL is accessed. Therefore, the first connection′s attributes set the rules that all subsequent connections must follow. The contents of the QGPL/QSQCLIPKG contains the following information.

```
5716SS1 V3R6M0 950929     Print SQL information
Object name...............QGPL/QSQCLIPKG
Object type...............*SQLPKG
  CRTSQL***
       PGM(QGPL/QSQCLIPKG)
       SRCFILE(*)
       SRCMBR(*)
       COMMIT(*CHG)
       OPTION(*SQL *PERIOD)
       TGTRLS(*PRV)
       ALWCPYDTA(*OPTIMIZE)
       CLOSQLCSR(*ENDPGM)
       RDB(*NONE)
       DATFMT(*ISO)
       TIMFMT(*ISO)
       ALWBLK(*ALLREAD)
       DLYPRP(*NO)
       DYNUSRPRF(*USER)
       SRTSEQ(*HEX)
       LANGID(          )
       RDBCNNMTH(*RUW)
       TEXT('                                        ')
DECLARE SQLCURSOR000000001 CURSOR FOR SQLSTATEMENT000001
```

Figure 58. Contents of QGPL/QSQCLIPKG

Naming Mode Considerations

If you set the DTW_SQL_NAMING_MODE to conflict with the naming mode in an existing QGPL/QSQCLIPKG on a remote AS/400 system, the SQL statement in the Web macro ends with an SQLCODE of -5016. To avoid this, select a naming mode and stick with it. If the object QGPL/QSQCLIPKG conflicts with the naming mode you have chosen, delete it and issue the Net.Data request again. A new QGPL/QSQCLIPKG is created with the naming mode option you require.

- Concurrent connections to the same remote database are not allowed. If a connection exists to a remote database using one user ID (the LOGIN SQL language environment parameter) and another request is made to connect to the same remote database using a second user ID, the SQL language environment must first disconnect the existing connection, do a commit, and reestablish the connection using the ″new″ user ID and password. The reason that the commit is required is that if the connection is broken, there is no way that a rollback can be accomplished in case of an error later in the macro.

  This example shows the parameters that need to be defined in the macro to connect to a remote database. The database that you are connecting to must be defined in the ″Relational Database Directory″ using the WRKRDBDIRE (Work with Relational Database Directory Entries) command to add, change, or remove entries in the directory.

  ```
  %define{
   DATABASE=″SYSTEM05″
   LOGIN=″USERID2A″
   PASSWORD=″HX1IMU″
   TABLE=″AS0219R.BRANDS″
   Mytable=%TABLE(ALL) %}
  ```

  The database ″SYSTEM05″ has been added to the Relational Database directory on the local AS/400 system. We can connect with the remote database ″SYSTEM05″ (in this case, it is another AS/400 system). We pass the USERID of ″USERID2A″ and the PASSWORD ″HX1IMU″. We return all rows of the query from the file ″BRANDS″ in library ″AS0219R″ to the Net.Data table ″Mytable″. More explanations of all the possible ″defines″ are explained later in this chapter.

  This does not pose a problem if you are using ″TRANSACTION_SCOPE=SINGLE″. If you use ″TRANSACTION_SCOPE=MULTIPLE″ (the default) and the ″new″ user ID differs from the one used to establish the database connection to the remote system, the SQL language environment automatically rolls back and an SQL_CODE of -752 is returned, which indicates that the connection cannot be changed.

- Up to 50 databases can be accessed either local or remote. The connections to the databases are kept active by the SQL language environment for the life of the HTTP server job that Net.Data is running under. This results in fast database access after the initial connection to the database.

- When the SQL language environment establishes a connection to a remote system, it associates a user ID with the connection. If, on a subsequent Net.Data query, the user ID does not match those associated with the connection, the connection is ended and a new connection to the database is established (this only occurs if transaction scope is SINGLE).

- If a language environment is created that uses the database access class library or the SQL call level interface and the language environment is referenced in a macro, the SQL language environment cannot be used.
- The QTMHHTP1 user profile must have the proper authority to access the database on the machine where the HTTP server resides. For remote databases, the user ID and password is used to determine what database resources can be accessed.

  Performance Tip

  If performance is a concern, Web macro writers should hard code or use the same user ID when issuing SQL statements to a remote database. For local database access, the user ID and password are ignored.

- SQL statements cannot be passed to the SQL language environment on an %EXEC statement.

The following simple example shows a macro named SQLM that issues a single SQL command:

```
%define DATABASE="HOSTNAME"
%FUNCTION(DTW_SQL) sql1 () {
select * from custinfo.customer
%}
%HTML(REPORT) {
@sql1()
%}
```

Figure 59. Simple SQL Command Function

This is an example URL that references the macro, assuming that:

- The macro is located in the /WWW/macro directory.
- The HTTP server has been configured to call the Net.Data CGI-BIN program DB2WWW.
- QTMHHTP1 has been given authority to access the macro file and table that the SQL command reads.

```
http://hostname/cgi-bin/db2www/WWW/macro/SQLM/report
```

If you did not create a Net.Data initialization file, the SQL language environment is enabled by default. However, if you create an initialization file, and you want to use the SQL language environment, the following configuration statement must be in the initialization file:

```
ENVIRONMENT(DTW_SQL)  /QSYS.LIB/QTCP.LIB/QTMHSQL.SRVPGM
  ( IN DATABASE, LOGIN, PASSWORD, TRANSACTION_SCOPE, SHOWSQL,
    DB_CASE, OUT DTWTABLE, SQL_CODE )
```

The text of this environment statement must all be on one line in the initialization file. It is shown here divided between multiple lines for readability.

The SQL language environment parameters in the preceding configuration statement are passed to the language environment and are described in the following list:

- DATABASE: The SQL language environment establishes a connection (if a connection has not already been established) to the database specified in this variable. The following example shows how this variable can be set in a Web macro:

  ```
  %DEFINE DATABASE="HOSTNAME"
  ```

- *LOGIN* and *PASSWORD*: If the DATABASE parameter references a remote database, the user ID and password specified in the LOGIN and PASSWORD variables are used when connecting to a database. These parameters are ignored when accessing the local database, and the user profile that CGI-BIN programs run under (TMHHTP1) must be given access to any files that are accessed.

  ```
  %DEFINE LOGIN="MYUSERID"
  %DEFINE PASSWORD="DB2WWW"
  ```

- *TRANSACTION_SCOPE*: Specifies the transaction scope for SQL commands. If the variable is not defined, the default action is "MULTIPLE", which means to COMMIT only after all SQL commands in an %HTML block complete successfully. An unsuccessful SQL command causes all previously executed SQL commands in that block to be rolled back. Specifying "SINGLE" means that a COMMIT is made after each successful SQL command in an %HTML block.

  ```
  %DEFINE TRANSACTION_SCOPE="SINGLE"
  ```

- *SHOWSQL:* Hide or display the SQL command that was executed. The default is not to display the SQL command that was executed.

  ```
  %DEFINE SHOWSQL="YES"
  ```

- *DB_CASE:* Specifies what case the SQL command should be in before executing the command. The default is to do no conversion. Specify "UPPER" or "LOWER" to force all characters in the SQL command to upper or lower case.

  ```
  %DEFINE DB_CASE="UPPER"
  ```

- *DTW_TABLE:* The result of a SQL query is stored in this table if there are no user-defined tables passed in.

- *SQL_CODE:* This variable contains the SQL warning or error code. Successful SQL queries result in an SQL_CODE of zero.

  ```
  Query completed with SQL code $(SQL_CODE).
  ```

Commitment Control Considerations

Currently, Net.data for the AS/400 system uses commitment control so files that are accessed for update through it must be journaled. Files that are accessed through the SQL SELECT statement are automatically journaled for you through the native SQL support on the AS/400 system so you do not need to start journaling for them.

4.6.3 SYSTEM (DTW_SYSTEM) Language Environment

The SYSTEM language environment supports calls to external programs identified in an %EXEC statement in the %FUNCTION block.

The SYSTEM language environment interprets the %EXEC statement by passing the specified program name and parameters to the operating system for execution using the C language system() function call. This method does not

allow Net.Data variables to be directly passed or retrieved to the executable statements as the REXX language environment does, so the SYSTEM language environment passes and retrieves variables in the following manner:

- Input parameters are passed as system ″environment variables″ using the putenv() function and can be retrieved by the executing program in a language-specific manner.
- Output parameters are passed back to the SYSTEM language environment by using the language-specific equivalent of the putenv() function.

The SYSTEM language environment expects the executable to be a command or a program. The QTMHHTP1 user profile must have the proper authority to run the executable in addition to any resources that the executable uses.

The following simple example shows a macro named SYSM that specifies a program as the executable, passing it one Net.Data parameter:

```
%define var1 = ″OriginalValue″
%FUNCTION(DTW_SYSTEM) test(INOUT parm1) {
%EXEC{ /QSYS.LIB/PGM.LIB/TSYS0001.PGM %}
%}
%HTML(REPORT) {
<PRE>
Value of var1 before function call: $(var1)
@test(var1)
Value of var1 after function call: $(var1)
</PRE>
%}
```

Figure 60. System Function Macro

This is an example URL that references the macro, assuming that:

- The macro is located in the /WWW/macro directory.
- The HTTP server has been configured to call the Net.Data CGI-BIN program DB2WWW.
- The user profile that CGI-BIN programs run under (QTMHHTP1) has been given authority to access the macro file and the file containing the REXX program.

```
http://hostname/cgi-bin/db2www/WWW/macro/SYSM/report
```

If you chose not to create a Net.Data initialization file, the SYSTEM language environment is enabled by default. However, if you create an initialization file and you want to use the SYSTEM language environment, the following configuration statement must be in the initialization file:

```
ENVIRONMENT(DTW_SYSTEM) /QSYS.LIB/QTCP.LIB/QTMHSYS.SRVPGM ( )
```

4.7 Net.Data Advanced Macro Language Examples

This section explains a few of the many functions that Net.Data has to offer. For the complete documentation about Net.Data, please refer to the online documentation available in several languages from the AS/400 Net.Data home page:

```
http://www.as400.ibm.com/netdata
```

4.7.1 Multiple HTML Sections

Net.Data allows you to have multiple named HTML sections in one macro. This lets you call new HTML pages to be displayed at the browser. Using conditional logic, your macro can branch to different pages in the same macro or to a different macro file. To display a new page in the same macro file, you can use the ″href″ HTML tag.

```
<a href=″input″>Return to previous page</a>
<a href=″end″>Go to the last page</a>
```

By clicking on the preceding hypertext link, the user is sent to the:

```
%HTML(INPUT) {....       %}
or
%HTML(END) {....       %}
```

section of the current macro file, depending on which link was selected.

4.7.2 Using Include Files

Include files can be served from the IFS and the QSYS file system. If you have not created an INI file that includes the INCLUDE_PATH statement, you need to fully qualify the path to the files so Net.Data can find them. This sample shows three include files that get two time stamps using the REXX function ″time″, subtract the seconds and hundredths of seconds, and show the results on the browser.

```
%{**********************FUNCTION SECTION***************%}
%FUNCTION(DTW_SQL) QUERY1 (OUT Mytable) {
select * from $(TABLE)
%report { %}
%}
%INCLUDE ″TIMERFTN.MBR″
%{********* INPUT  SECTION **********************%}
%HTML (INPUT) {
<HTML>
%INCLUDE ″TIMERSTR.MBR″
...
...
...
%INCLUDE ″TIMEREND.MBR″
```

Figure 61. Using Include Files

The contents of the first %INCLUDE file TIMERFTN sets up the REXX command time. It is included from the FUNCTION SECTION of the macro.

```
%function(dtw_rexx) stime1(out t1) { t1 = time('l')
%}
%function(dtw_rexx) etime1(OUT t2) returns(t2){ t2 = time('l')
%}
```

Figure 62. Include File TIMERFTN

The contents of the second %INCLUDE file TIMERSTR starts the timer. Notice that the first function ″@stime1″ does not have a ″returns″ parameter so the start time is not displayed at the browser, but is stored in the variable $(t1) to print later in the macro. We are using the Net.Data built-in function @DTW_DELSTR to

start at the first position of the string, delete the next six positions, and return the value of the deleted string to $(d1).

```
@stime1(t1)
@DTW_DELSTR(t1,"1","6",d1)
```

Figure 63. INCLUDE File TIMERSTR

The contents of the third %INCLUDE file TIMEREND gets another time stamp, and parses the time stamp to remove the hours and the colons from the time string $(t2). We use the built in Net.Data math functions to return the elapsed time in seconds to the browser.

```
<table>
<TR><TD>The end time was<TD>@etime1(t2) </TR>
@DTW_DELSTR(t2,"1","6",d2)
@DTW_SUBTRACT(d2,d1,d3)
<TR><TD>The start time was<TD>$(t1)</TR>
<TR><TD>The total time was<TD align=right>$(d3)</TR>
</table>
```

Figure 64. INCLUDE File TIMEREND

The output on the browser display looks similar to this:

```
The end time was   15:16:43.506000
The start time was 15:16:42.726000
The total time was        0.780000
```

Notice that in this example we are only subtracting the seconds and tenths of seconds. Parsing the string can be changed to include the minutes and hours.

4.7.3 Conditional Logic

The %IF / %ELIF / %ELSE / %ENDIF block using conditional logic in your Net.Data macros is one of the more powerful functions of the macro language. This allows the macro writer to dynamically change the HTML in the macro that is returned to the browser. When you think of it, imagine all the stuff you can do with this function.

```
%IF ($(ROW_NUM) < $(MAX_ROWS))
<P>The table is not full yet.<BR>
%ELIF ($(ROW_NUM) == $(MAX_ROWS))
<P>The table is now full.<BR>
%ELSE
<P>The table is too small.<BR>
%ENDIF
```

4.7.4 Maintaining State using "HTML" Hidden Variables

The following example shows one way to keep track of the HTML pages and their associated variables. This example uses REXX functions to push and pop values from a stack and pass the values to the Net.Data macro. Using the "nextButton" and the "backButton", the field values of the form are remembered for the user so they can change an entry on the form easily. Notice the HTML Meta tag <META HTTP-EQUIV="Expires" CONTENT="Mon, 01 Jan 1990 13:00:00 GMT">. This makes sure that when a user selects the "backButton", the variables are not coming from the user's cached pages.

```
%{
   Source File Name: itsoexmp.mbr
   Source File Description:
   This source file contains macro code that
   pushes an HTML screen onto a
   stack and also pops it off the stack.     %}
%{ *********       Define Section        *****************%}
%DEFINE {
  FIRSTNAME = ""
  LASTNAME = ""
  MAINQ = ""
  TEMPQ = ""
%}
%{ *************** Assign Function  **********************%}
%FUNCTION(DTW_REXX) assign(IN from, OUT to) {
  to = from    /* assigns the input variable to the output variable */
%}
%{ ************* Stack Push Function *********************%}
%FUNCTION(DTW_REXX) myPush(IN x, INOUT y) {
  y = x' 'y         /* pushes 'x' onto stack 'y'     */
%}
%{ ************* Stack Pop Function **********************%}
%FUNCTION(DTW_REXX) myPop(INOUT x, INOUT y) {
  parse var y x y   /* pops 'x' off of stack 'y'     */
%}
%{ *********      HTML Section: main0    *****************%}
%HTML (main0) {
<html>
<head>
<META  HTTP-EQUIV="Expires" CONTENT="Mon, 01 Jan 1990 13:00:00 GMT">
<TITLE>First Screen</TITLE>
</head>
<body bgcolor="#ffffff">
<b>First Screen</b>
@assign(mainQ,tempQ)
<p>
This is the first screen.
<p>
<form method=POST action="main1">
<!-- Push current panel onto my queue in case we --!>
<!-- need to find our way back --!>
@myPush("main0" , mainQ)
<input type="hidden" name="mainQ" value="$(mainQ)">
<!-- POST hidden variables -->
%INCLUDE "itsoexm2.mbr"
<input type="submit" name="nextButton" value="Next">
</form>
</body>
</html>
%}
%{ ***********      HTML Section: main1    *************** %}
%HTML (main1) {
<html>
<head>
<META  HTTP-EQUIV="Expires" CONTENT="Mon, 01 Jan 1990 13:00:00 GMT">
<TITLE>Second Screen</TITLE>
</head>
<body bgcolor="#ffffff">
<b>Second Screen</b>
```

```
@assign(mainQ,tempQ)
<p>
This is the second screen.  Enter your first name.
<p>
<!-- This is the next form -->
<form method=POST action="main2">
<!-- Push current panel onto my queue in case --!>
<!-- we need to find our way back --!>
@myPush("main1" , mainQ)
<input type="hidden" name="mainQ" value="$(mainQ)">
Your First Name<input type="text" name="firstName"
maxlength=12 size=12 value="$(firstName)">
<p>
<!-- This table horizontally aligns the buttons -->
<TABLE>
<TR>
<TD>
<!-- POST hidden variables -->
<input type="hidden" name="lastName" value="$(lastName)">
<input type="hidden" name="Serial" value="$(Serial)">
<input type="submit" name="nextButton" value="Next">
</form>
</TD>
<TD>
<!-- This is the back form -->
@myPop(prevPanel,tempQ)
<form method=POST action="$(prevPanel)">
<input type="hidden" name="mainQ" value="$(tempQ)">
<!-- POST hidden variables -->
%INCLUDE "itsoexm2.mbr"
<input type="submit" name="backButton" value="Back">
</form>
</TD>
</TR>
</TABLE>
</body>
</html>
%}
%{ *************      HTML Section: main2    *************** %}
%HTML (main2) {
<html>
<head>
<META  HTTP-EQUIV="Expires" CONTENT="Mon, 01 Jan 1990 13:00:00 GMT">
<TITLE>Last Screen</TITLE>
</head>
<body bgcolor="#ffffff">
<b>Last Screen</b>
@assign(mainQ,tempQ)
<p>
This is the last screen.  Your first name is $(firstName).
<p>
<!-- This table horizontally aligns the buttons -->
<TABLE>
<TR>
<TD>
<!-- This is the back form -->
@myPop(prevPanel,tempQ)
<form method=POST action="$(prevPanel)">
<input type="hidden" name="mainQ" value="$(tempQ)">
```

```
<!-- POST hidden variables -->
%INCLUDE "itsoexm2.mbr"
<input type="submit" name="backButton" value="Back">
</form>
</TD>
</TR>
</TABLE>
</body>
</html>
%}
```

The contents of the %INCLUDE "itsoexm2.mbr" are shown in the following example:

```
<input type="hidden" name="firstName" value="$(firstName)">
<input type="hidden" name="lastName" value="$(lastName)">
<input type="hidden" name="Serial" value="$(Serial)">
```

In this example, we are only using the "firstName" Hidden field. However, the idea is to understand how to move from one HTML section to another and bring forward or backward the field values that the user has input to the forms.

4.7.5 Net.Data Hidden Variables

Hidden variables can be used to conceal the actual name of a variable from people who choose to view your HTML source with their Web browser.

1. Define a variable for each string you want to hide. Put the %DEFINE statement for these variables after the %HTML block where you reference the variables and before the %HTML block where they are to be used.
2. In the %HTML block where the variables are referenced, use double dollar signs instead of a single dollar sign to reference the variables (for example, $$(name) instead of $(name)).

```
%HTML(INPUT) {
<FORM ...>
<P>Select fields to view:
<SELECT NAME="Field">
<OPTION VALUE="$$(name)"> Name
<OPTION VALUE="$$(addr)"> Address
....
....
....
</FORM>
%}

%DEFINE{
name="customer.name"
addr="customer.address"
%}

%FUNCTION(DTW_SQL) mySelect() {
  SELECT $(Field) FROM customer
%}
....
....
....
```

When the HTML form is displayed on a Web browser, $$(name) and $$(addr) are replaced with $(name) and $(addr) respectively, so the actual table and column names never appear on the HTML form and there is no way to tell that the true

variable names are hidden. When the customer submits the form, the %HTML(REPORT) block is called. When the @mySelect() statement calls the %FUNCTION block, $(Field) is substituted in the SQL statement with customer.name or customer.addr (depending on what the user selected) in the SQL query.

4.7.6 Net.Data Predefined Variables

Predefined variables are supplied by Net.Data with a value set by Net.Data. You cannot change predefined variables. A predefined variable does not have to be explicitly defined before being referenced in a Net.Data web macro. The following sample macro uses currently available predefined variables. Notice that all of the variables start with "DTW_".

```
%{********* REPORT SECTION **********************%}
%HTML (REPORT) {
<HTML>
<TITLE>Environment Variables</TITLE>
<P>
HERE ARE THE RESULTS OF ENVIRONMENT REQUESTS
<HR>
<P><TABLE BORDER>
<TR><TD>This Web macro was last modified<TD> $(DTW_MACRO_LAST_MODIFIED)
</TR>
<TR><TD>File name is<TD>          $(DTW_MACRO_FILENAME)</TR>
<TR><TD>CURRENT FILE NAME<TD>     $(DTW_CURRENT_FILENAME)</TR>
<TR><TD>Current last modified<TD>$(DTW_CURRENT_LAST_MODIFIED)</TR>
<TR><TD>Path   <TD>               $(DTW_MP_PATH)</TR>
<TR><TD>Version <TD>              $(DTW_MP_VERSION)</TR>
</TABLE>
</HTML>
%}
```

Figure 65. Net.Data Environment Variables Macro

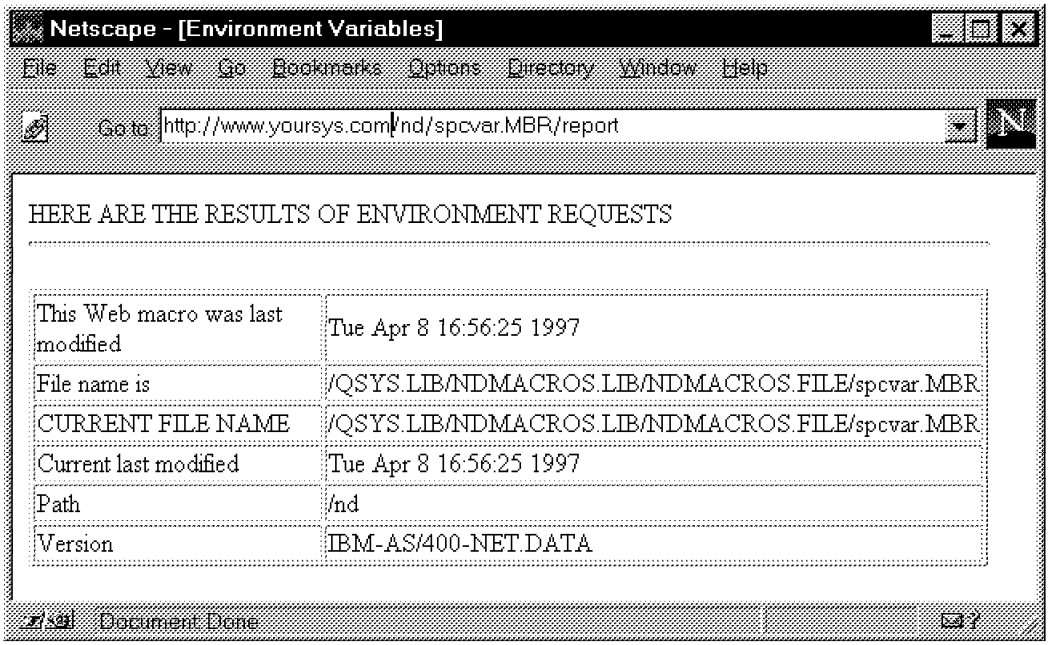


Figure 66. Sample of Net.Data Environment Variables

4.7.7 Net.Data Environment Variables

Net.Data environment variables allow you to reference any environment variable that exists in the process under which Net.Data is running. To define a Net.Data environment variable, use the actual environment variable name. You need to define Net.Data environment variables before you use them.

```
%DEFINE environment_variable_name=%ENVVAR
```

When the variable is referenced in the Web macro, Net.Data simply gets the environment variable's current value. Net.Data replaces the variable reference with the current value. For example, the following figure displays the current values of the environment variables.

```
%{************** DEFINE SECTION ****************%}
%DEFINE SCRIPT_NAME=%ENVVAR
%DEFINE HTTP_USER_AGENT=%ENVVAR
%DEFINE SERVER_SOFTWARE=%ENVVAR
%{********* REPORT SECTION **********************%}
%HTML (REPORT) {
<HTML>
<TITLE>Environment Variables</TITLE>
<P>
HERE ARE THE RESULTS OF ENVIRONMENT REQUESTS
<HR>
<P><TABLE BORDER>
<TR><TD>The script name is<TD> $(SCRIPT_NAME)</TR>
<TR><TD>The requesting browser was <TD>$(HTTP_USER_AGENT)</TR>
<TR><TD>The HTTP Server Software is<TD> $(SERVER_SOFTWARE)</TR>
</table>
</HTML>
%}
```

Figure 67. Macro to Get Environment Variables

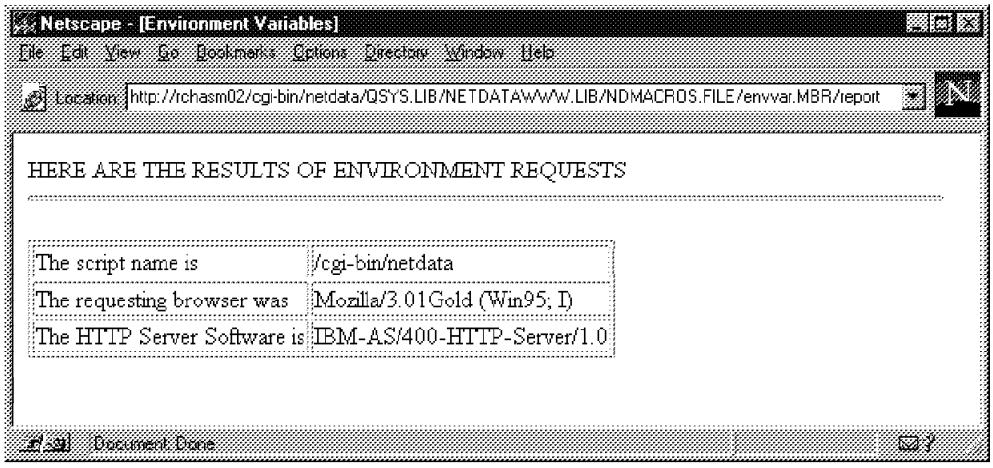


Figure 68. Sample Output of Environment Variables

4.7.8 Net.Data Default Report

Net.Data gives you the ability to disable the default report. The default report is built from text characters. Net.Data can automatically build HTML tables for you if you include the following in your macro.

```
%define DTW_HTML_TABLE="YES"
```

Once this is defined in the macro, you can turn the value on or off by including the following statement before the table is to be displayed with this statement.

```
@DTW_ASSIGN(DTW_HTML_TABLE,"NO")
or
@DTW_HTML_TABLE="YES"
```

In some cases, you may want to display your table data using the built-in table functions provided by Net.Data rather than displaying the table data using a default report. These table functions produce select, radio, list, and check box HTML objects from your Net.Data table. The easiest way not to produce a report is to include a blank report block, %report{ %} within the function define as illustrated in this example, which creates a select box where the "@QUERY1(Mytable)" causes Net.Data to execute the "%FUNCTION(DTW_SQL) QUERY1..." SQL command.

```
%{********* DEFINE SECTION **********************%}
%define{
 DATABASE="*LOCAL"
 TABLE="ASO219R.BRANDS"
 Mytable=%TABLE(ALL)
%}
%{***************FUNCTION SECTION***************%}
%FUNCTION(DTW_SQL) QUERY1 (OUT Mytable) {
select * from $(TABLE)
%report { %}
%}
%{********* INPUT  SECTION **********************%}
%HTML (FORM1){
<HTML>
<h1>Net.Data Select Box</h1>
<FORM METHOD="POST" ACTION="FORM2">
<h3>Vehicle Selection Box</h3> <BR>
@QUERY1(Mytable)
@DTW_TB_SELECT(Mytable,"brand","","10","N","","1")
<BR>
<INPUT TYPE="SUBMIT" VALUE="Submit it">
</FORM>
</HTML>
%}
%{********* REPORT SECTION **********************%}
%HTML (FORM2) {
<HTML>
<TITLE>Selct Box Results</TITLE>
<P>
You selected the value <B>"$(brand)"</B> from the select box
</HTML>
%}
```

Figure 69. Net.Data Select Box Macro

When this macro is called, it produces the following browser output. Notice that 10 [@DTW_TB_SELECT(Mytable,"brand","",▌"10"▌,"N" ,"","1")] rows from the table are displayed and that the first option

[@DTW_TB_SELECT(Mytable,″brand″,″″,″10″,″N″,″″,[″1″])] was selected as the default selection.

Debug Tip

The HTML section FORM2 prints the environment variable that is passed by the macro in the ″ACTION=″ parameter of the FORM METHOD=″POST″. By including the **$(brand)**, you can see the value that was passed when the ″Submit it″ button was selected.

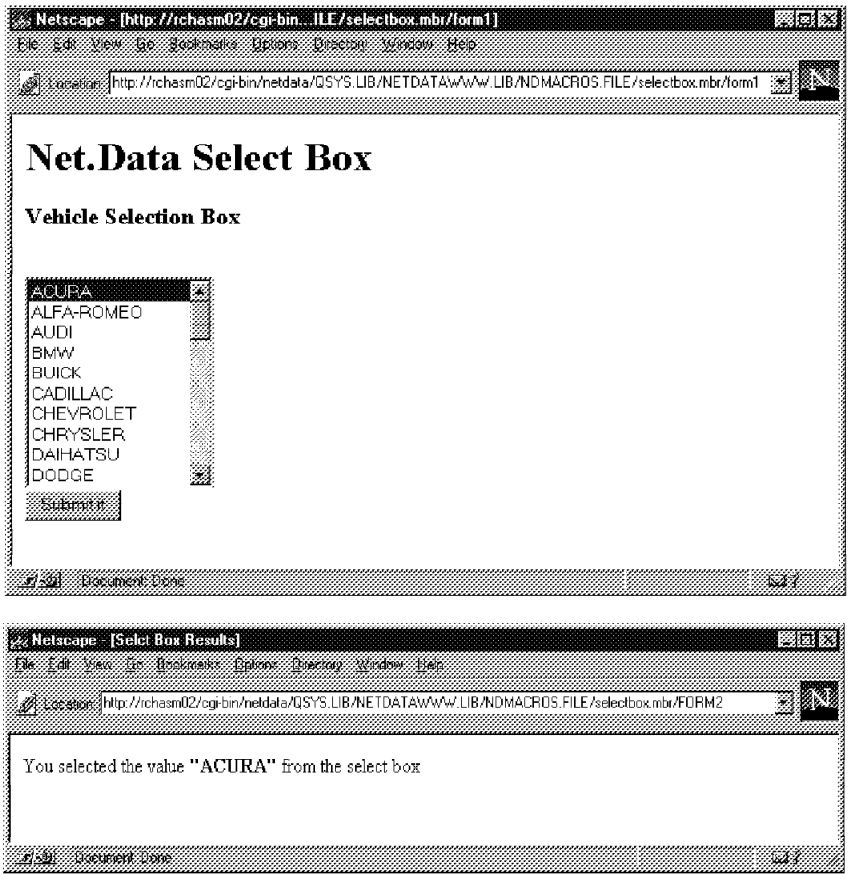


Figure 70. Sample Select Box

When the select button ″Submit it″ is activated by the user, it calls the macro section HTML (FORM2) and returns this display to the browser.

4.8 Additional Tips

The following list contains distinguishing features of the OS/400 implementation of Net.Data and some helpful hints:

- Web macros can be stored in any file system. You can copy Web macros to and from the QSYS.LIB file system to another file system by using the Copy To Stream File (CPYTOSTMF) command and Copy From Stream File (CPYFRMSTMF) command.

Tip

You can use the Copy To Stream File (CPYTOSTMF) command to copy your macros into the IFS. The IFS gives you the best performance when serving your macro.

Net.Data expects the macros in the Integrated File System to be in ASCII format. If you have the AS/400 system create the file during the copy, you need to use the conversion table QASCII.TBL in library QSYS.LIB as shown in the following example. By using the QOpenSys directory, you can use long file names as shown in this example by using the .html extension.

```
CPYTOSTMF FROMMBR('qsys.lib/netdatawww.lib/ndmacros.file/tim
                  erend.mbr')
          TOSTMF('/qopensys/timerend.html')
          CVTDTA(*TBL)
          TBL('/qsys.lib/qascii.tbl')
```

Figure 71. Copy to Stream File with ASCII Conversion

- The APIs to support the Net.Data language environment interface are in the QTMHLE service program in library QTCP. An ILE C language header file provides the prototypes required to call the Net.Data APIs in ILE C. Documentation about the interface can be found at "http://www.as400.ibm.com/netdata". To get the necessary prototypes, include the header file as follows:

  ```
  #include <dtwle.h>
  ```

  The QTMHLE *SRVPGM object is included in the QTCP library.

  The QSYSINC library contains header files and is optionally installable. Make sure QSYSINC is on your system before compiling programs that use these header files.

- If you have problems getting a language environment to function properly, look in the job log of the HTTP server job to see if any messages were issued that may indicate the cause of the problem. When debugging newly created language environments, it is best to have the minimum and maximum number of server jobs set to two so that Net.Data requests are funneled through one HTTP server job. This allows you to easily look at the job log of the server to see if any messages are being logged. In addition, every time a language environment service program is updated, you must end the HTTP server and restart it so that Net.Data can activate the updated service program.

- The OS/400 implementation of Net.Data did not support the following Net.Data features when this was published. They may be supported on future releases:

  – %INCLUDE_URL statement. It ignores this statement if it encounters it.

  – Live connectivity using Common Gateway Interface thread management. However, connections to databases are kept persistent across URL invocations so that there is no performance degradation when accessing remote databases.

  – Application programming interface support for the IBM Internet Connection servers (ICS), NetScape servers (NS), and Microsoft's Internet Information server (IIS).

4.8.1 Net.Data Error Messages

Net.Data returns the following messages to alert you when a problem is found.

Return Code	Message Text	Error Condition
-1002	″function function_name: Unable to allocate memory.″	The server could not process a request for storage from Net.Data.
-1001	″function function_name: Internal code code.″	A call to an internal function failed. This is a Net.Data internal error. Report the problem to your software service representative.
1000	″function function_name: Function not found.″	The function requested on a function call is not a supported Net.Data built-in function.
1001	″function function_name: Parameter parm_name contains a null value.″	An input parameter contained a NULL value. This may occur if the parameter passed on the function call has not been previously defined in the Web macro.
1002	″function function_name: Parameter parm_name contains a null string.″	An input parameter contained a string value which consisted of the null-terminating character.
1003	″function function_name: The number of parameters passed is not correct.″	The number of parameters passed on a function call either exceeded the maximum number allowed, or was less than the minimum number required by the function being called.
1004	″function function_name: Parameter parm_name is not a table.″	A parameter was passed on a function call which was required to be a Web macro table variable, but was instead a string variable.
1005	″function function_name: Parameter parm_name is not a string.″	A parameter was passed on a function call which was required to be a Web macro string variable, but was instead a table variable.
1006	″function function_name: Parameter parm_name is not an output parameter.″	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	″function function_name: Parameter parm_name contains a value which is not valid.″	One of these conditions exists: • A value was passed that exceeded the maximum supported value. • A value was passed that was less than the minimum supported value. • A value was passed that was not one of the supported choices. • A table row or column value was passed that was less than or equal to zero.

Return Code	Message Text	Error Condition
1008	″function function_name: Value parm_value is outside of table bounds.″	One of these conditions exists: • A program attempted to modify a table′s row or column value, but the row or column value received was less than 0 or greater than the maximum number of rows allowed in the table. • A row or column value was received as input to a built-in function, but the value received was less than 0 or greater than the current number of rows or columns in the table.
1009	″function function_name: Variable string is not in the correct format.″	The syntax of the data returned by a system or Perl program is not correct. One of these conditions exists: • An equal sign was not found. • A beginning quote was not found. • An ending quote was not found. • A space separator between values was not found.
1010	″function function_name: Not all requested data could be returned.″	A table was specified as an output parameter, but the number of rows of data returned by the language environment was greater than the maximum number of rows allowed for the table. Data was written to the table until it was full, and the remainder of the data was discarded.
2000	″function function_name: The requested file filename was not found.″	A flat file interface built-in function could not find the specified file in the directories it was allowed to search.
2001	″function function_name: The requested file filename could not be opened in the specified mode.″	A flat file interface built-in function could not open the specified file because it was in use by this or another process, and could not be shared in the specified mode.
2002	″function function_name: The requested file filename was not opened.″	A flat file interface built-in function could not close the specified file because it was not opened by this macro invocation.

Return Code	Message Text	Error Condition
2003	″function function_name: Attempted to read a row of data that exceeded the maximum supported number of bytes.″	A flat file interface built-in function could not read a row of data into a table variable because the number of bytes in the row exceeded the maximum supported number of bytes.
2004	″function function_name: A path specified in FFI_PATH exceeded the maximum supported number of bytes.″	A flat file interface built-in function was attempting to find a file, but encountered a path in the FFI_PATH configuration file variable that was longer than the maximum supported number of bytes.
2005	″function function_name: System call error text″	A call to a system function failed. This is an internal error reported to Net.Data that may require user interaction or it may be a temporary system error that is not appropriate for Net.Data to handle. If this problem persists, report the problem to your software service representative.
2006	″function function_name: The requested file name could not be accessed in the specified mode.″	A flat file interface built-in function could not access the specified file because it was in use by this or another process and could not be shared in the specified mode.
3001	″function function_name: Registry registry_name already exists.″	A Web registry built-in function could not create a Web registry because the specified registry already existed.
3002	″function function_name: Registry registry_name is in use by another process or does not exist.″	A Web registry built-in function could not delete the specified registry: • The registry was in use by another process. • The registry could not be found.
3003	″function function_name: Registry entry registry_entry already exists.″	A Web registry built-in function could not add an entry to the specified registry because the specified entry already existed.
3004	″function function_name: Registry entry registry_entry cannot be found.″	A Web registry built-in function could not remove or retrieve an entry from the specified registry because the specified entry did not exist.
3005	″function function_name: Registry registry_name cannot be found.″	A Web registry built-in function could not use the specified registry because it could not be found.

Return Code	Message Text	Error Condition
3006	″function function_name: Path in registry registry_name does not exist.″	A Web registry built-in function could not create the specified registry because a path in the registry name did not exist.
3007	″function function_name: You are not authorized to perform the requested registry operation.″	A Web registry built-in function could not complete the specified operation because the requestor did not have the proper authority to the specified registry.
3008	″function function_name: Registry registry_name failed to create.″	A Web registry built-in function could not create the specified registry for unknown reasons.
4000	″function function_name: Parameter parm_name is not a whole number or is too large.″	One of these conditions exists: • An input parameter contained a value that was not a whole number. • An input parameter contained a value that was greater than the supported maximum of 999,999,999. • An output cannot be expressed as a whole number.
4001	″function function_name: Parameter parm_name is not a valid number.″	One of these conditions exists: • An input parameter contained a value that was not a valid format for a number. • An input parameter contained a value that specified an exponent outside the supported range of -999 999 999 to +999 999 999.
4002	″function function_name: Arithmetic overflow or underflow.″	The result of an arithmetic operation had an exponent that was outside the supported range of -999 999 999 to +999 999 999.
5000	″function function_name: EXEC statement is empty.″	The string specified in the %EXEC statement of a function block contained only space characters.
6000	″function function_name: EXEC was not specified in the function block.″	A %EXEC statement was not specified in the function block for the function being called.

4.9 Getting Net.Data Up and Running

To enable Net.Data, you must do the following steps:

1. Copy the Net.Data program object to your CGI-BIN library. (This is not absolutely necessary; see the following Security Alert message box.)
2. Configure the HTTP server by adding specific directives to the configuration file.
3. Create a Net.Data initialization (INI) file (optional).

4. Grant authority to the QTMHHTP1 user profile to objects that are accessed by the Net.Data program.

Let's take an in-depth look at each of the preceding steps to enable Net.Data on the AS/400 system.

1. Copy the Net.Data Program Object to a CGI-BIN library.

   Systems Management Issue

   If you move the DB2WWW program out of the QTCP library, it is not updated automatically by the PTF process or when you install a new release. You need to have a good system management process in place to update the DB2WWW program when needed. (You can use the following untested suggestions as an alternative to moving DB2WWW program out of QTCP library.)

   Security Alert

   If you choose to leave the DB2WWW program in the QTCP library, you need to configure an EXEC statement to the QTCP library. This can open a security hole that allows any browser to run any program in the library QTCP if a MAP or PASS statement to the QTCP library is in the HTTP configuration file. One way to avoid this is shown in "Alternative Setup to Leave DB2WWW in QTCP" on page 161.

   The Net.Data program (DB2WWW) should be copied to a CGI-BIN library. The DB2WWW program is located in the QTCP library. You can copy the program object using the Create Duplicate Object (CRTDUPOBJ) command.

   The DB2WWW program object's authority for *PUBLIC users is set to *EXCLUDE. Change the DB2WWW program object in the CGI-BIN directory so that the user profile that CGI programs run under (QTMHHTP1) has access to the program object. You can do this by changing the program object's authority for *PUBLIC users to *USE, or by specifically giving the QTMHHTP1 user profile *USE access to the DB2WWW program object.

2. Add Net.Data Directives to the HTTP configuration file:

   Use option 1 (Add) or option 13 (Insert) of the Work with HTTP Configuration (WRKHTTPCFG) command to do the following steps:

   a. Ensure that the Enable GET and Enable POST directives are in the configuration file.

   b. Add Map and Exec directives for Net.Data.

Alternative Setup to Leave DB2WWW in QTCP

One solution is to temporarily remap the incoming URL to a bogus one, Fail /QSYS.LIB/QTCP.LIB/*, and remap the temporary bogus URL back to /QSYS.LIB/QTCP.LIB/*. This looks similar to the following:

```
Map   /QSYS.LIB/QTCP.LIB/DB2WWW.PGM/* /DB2WWWBOGUS/*
Fail  /QSYS.LIB/QTCP.LIB/*
Map   /DB2WWWBOGUS/* /QSYS.LIB/QTCP.LIB/DB2WWW.PGM/*
Exec  /QSYS.LIB/QTCP.LIB/*
```

Examples:

- For http://QSYS.LIB/QTCP.LIB/DB2WWW.PGM/..., this is mapped to /DB2WWWBOGUS. The Fail does not happen. The next Map fixes the bogus URL. And the final Exec allows the DB2WWW.PGM to execute.
- For http://QSYS.LIB/QTCP.LIB/FTP.PGM/..., this does not do the first Map and subsequently fails with the next directive.
- For http://qsys.lib/qtcp.lib/ftp.pgm/..., this does not Map, Fail, or Exec due to any of the preceding directives because they are case sensitive. So, it falls through all of the preceding directives and eventually fails after all of the directives had been exhausted.

Carefully planning the HTTP server configuration file is important to maintain the integrity of the servers files and programs from harm.

If they are not already there, add the Enable GET **1** and Enable POST **2** directives to the section of the configuration file where you enable methods. With the directives in the file, your display is similar to the display shown in the following figure:

```
                        Work with HTTP Configuration
                                                          System:   SYSNAM01
Type options, press Enter.
  1=Add   2=Change   3=Copy   4=Remove   5=Display   13=Insert

      Sequence
Opt    Number    Entry

       00010     # * * * * * * * * * * * * * * * * * * * * * * * * * * * * #
       00020     # HTTP CONFIGURATION FOR NET.DATA TESTING                  #
       00030     #                                                         #
       00040     HostName sysnam01.location.company.com
       00050     Port 80
       00060     #-----------------------------------------------------
       00070     # Methods Enabled
       00080     #
       00090     Enable GET  1
       00100     Enable POST 2
       00110     #
       00120     #-----------------------------------------------------
                                                                      More...
F3=Exit   F5=Refresh   F6=Print List   F12=Cancel   F17=Top   F18=Bottom
F19=Edit Sequence
```

Figure 72. Get and Post Statements

Add the Map and Exec statements shown in Figure 73 on page 162. This display shows the directives after they have been added.

```
                          Work with HTTP Configuration
                                                                System:   SYSNAM01
 Type options, press Enter.
   1=Add   2=Change   3=Copy   4=Remove   5=Display   13=Insert

       Sequence
 Opt    Number    Entry

        00130     #-----------------------------------------------------
        00140     #                      Mapping/Pass Rules + Executables
        00150     #
        00160 3    Map /cgi-bin/db2www/* /QSYS.LIB/CGI.LIB/DB2WWW.PGM/*
        00170 3    Map /CGI-BIN/DB2WWW/* /QSYS.LIB/CGI.LIB/DB2WWW.PGM/*
        00180     #
        00190     #
        00200 4   Exec  /QSYS.LIB/CGI.LIB/*
        00210     Pass  /WWW/html/*
        00220     #-----------------------------------------------------
                                                                        More...
 F3=Exit   F5=Refresh   F6=Print List   F12=Cancel   F17=Top   F18=Bottom
 F19=Edit Sequence
```

Figure 73. Map and Exec Statements

The Map directives **3** map entries in the form "/cgi-bin/db2www/*" to the library where the Net.Data program resides on your system. (The asterisk (*) at the end of the string refers to anything that follows the string.) Both upper-case and lower-case map statements are included because the directives are case sensitive. In this example, both Map statements map to the same location.

The Exec directive **4** enables the HTTP server to execute any CGI programs in the CGI library. Specify the library where the program resides (not the program itself) on the directive. To prevent other *PGM objects in the CGI library from being executed, exclude *PUBLIC and QTMHHTP1 from object access.

You must restart the HTTP server for changes to the configuration file to take effect.

3. Create the Net.Data initialization file:

   The creation of the Net.Data initialization file is optional. If an initialization file is not created, Net.Data runs as if an initialization file with only the default language environment statements in the file had been configured. All macro, include, and executable references need to be fully qualified.

   If an initialization file is created and updated, you do *not* need to end or restart the HTTP server for the changes to take effect. Net.Data reads the initialization file once during the initial invocation by an HTTP server job. The configuration data is saved so that on subsequent Net.Data invocations, Net.Data does not have to read the initialization file. However, if a change is made to the initialization file, Net.Data detects the change to the initialization file and reads the initialization file again.

   Use the Create Source Physical File (CRTSRCPF) command to create the initialization (INI) file. Since the text of configuration statements must all be

on one line, it is a good idea to create the initialization file with a record length of 240. The file must be created in the library where the DB2WWW program object resides. The file name must be ″INI.″ The member name must be ″DB2WWW″. Use the Source Entry Utility (SEU) to add configuration statements to the file. A sample initialization file is shown in the following figure. The text of the environment statements must all be on one line. These statements are shown on several lines in this example for readability.

```
MACRO_PATH    /WWW/MACROS;/QSYS.LIB/WWWMACROS.LIB/MACROS.FILE
INCLUDE_PATH  /WWW/INCLUDES
EXEC_PATH     /QSYS.LIB;/QSYS.LIB/WWWPGMS.LIB

ENVIRONMENT(DTW_REXX) /QSYS.LIB//QTCP.LIB/QTMHREXX.SRVPGM ( )
ENVIRONMENT(DTW_SQL)  /QSYS.LIB/QTCP.LIB/QTMHSQL.SRVPGM (IN DATABASE,
  LOGIN, PASSWORD, TRANSACTION_SCOPE, SHOWSQL, DB_CASE,
  OUT DTWTABLE, SQL_CODE)
ENVIRONMENT(DTW_SYSTEM) /QSYS.LIB/QTCP.LIB/QTMHSYS.SRVPGM  ( )
```

Figure 74. Contents of Net.Data Initialization (INI) File

Net.Data on the AS/400 system recognizes three special path statements. It uses these statements to determine where to look for files to process:

- MACRO_PATH
- EXEC_PATH
- INCLUDE_PATH

These path statements identify a set of directories that are searched by the Web macro processor for Web macros, include files, and executable files. Each directory specified is delimited by a semi-colon (″;″). Forward slashes (″/″) and backward slashes (″\″) are treated the same. The multiple-path capability of the MACRO_PATH, EXEC_PATH, INCLUDE_PATH, and HTML_PATH statements enable different products using Net.Data to be isolated to their own directories.

The format of the MACRO_PATH statement is:

```
MACRO_PATH  /pathname[;pathname ...]
```

The MACRO_PATH statement identifies one or more directories to search in the order in which they are specified for a Net.Data macro file. Net.Data uses the PATH_INFO environment variable when resolving to a Web macro. For example, if the following URL was used to send a Net.Data request:

```
http://hostname/cgi-bin/db2www/WWW/macro/SQLM/report
```

the portion of the URL that Net.Data considers to be the Web macro name is ″/WWW/macro/SQLM″. Net.Data appends the path to the paths in the MACRO_PATH configuration statement from left to right until the Web macro is found. If the Web macro is not found, Net.Data assumes that ″/WWW/macro/SQLM″ references the Web macro.

The path statements for the MACRO_PATH configuration statement can be integrated file system directories (IFS) and QSYS.LIB directories. For example, if macros are in the IFS directory /WWW/MACRO and in the QSYS.LIB directory of /QSYS.LIB/WWW.LIB/MACRO.FILE, the MACRO_PATH statement looks similar to this:

```
MACRO_PATH /WWW/MACRO;/QSYS.LIB/WWW.LIB/MACRO.FILE
```

The format of the EXEC_PATH statement is:

```
EXEC_PATH  pathname[;pathname ...]
```

The EXEC_PATH statement identifies one or more directories to search in the order in which they are specified to find an external program invoked using the %EXEC statement. If the program is found, the external program name is appended to the path specification resulting in a qualified file name which is passed to the language environment for execution.

For example, if programs are stored in a library called WWWPGMS, the EXEC_PATH statement looks similar to this:

```
EXEC_PATH  /QSYS.LIB/WWWPGMS.LIB
```

The format of the INCLUDE_PATH statement is:

```
INCLUDE_PATH  pathname[;pathname ...]
```

The INCLUDE_PATH identifies one or more directories that are searched in the order in which they are specified to find a file specified on the %INCLUDE statement in a Net.Data macro. When found, the include file name is appended to the path specification to produce the qualified include file name. You can specify IFS and QSYS.LIB directories.

The ENVIRONMENT statement is used to configure a language environment. Net.Data′s language environment allows you to directly run REXX programs, query the database, and access high level programs on the AS/400 system. Further setup considerations for the language environments are explained in Section 4.6, “Language Environments” on page 139. The syntax of the language environment configuration statement is:

```
ENVIRONMENT(type) srvpgm-name ([usage argument, ...])
```

A language environment configuration statement consists of the following data:

- A type name for associating it with the type field of a %FUNCTION block.
- A service program name.
- A list of optional arguments that are passed to the service program′s entry points.

4. Grant QTMHHTP1 user profile authority to objects.

The user profile that CGI programs run under (QTMHHTP1) must have the proper access to any objects that are referenced in a Web macro and to the macro that a URL references. One way to grant authority to the QTMHHTP1 user profile is to use the Change Authority (CHGAUT) command, which allows you to grant authority to objects that reside in the QSYS.LIB file system in addition to objects residing in the integrated file system that are not in the QSYS.LIB file system. For example, if Web macro files reside in the /WWW/macro directory and in the /QSYS.LIB/WWW.LIB/MACRO.FILE directory, the following commands need to be issued:

```
CHGAUT OBJ('/WWW') USER(QTMHHTP1) DTAAUT(*RX)
CHGAUT OBJ('/WWW/macro') USER(QTMHHTP1) DTAAUT(*RX)
CHGAUT OBJ('/WWW/macro/*') USER(QTMHHTP1) DTAAUT(*RX)

CHGAUT OBJ('/QSYS.LIB/WWW.LIB') USER(QTMHHTP1) DTAAUT(*RX)
CHGAUT OBJ('/QSYS.LIB/WWW.LIB/MACRO.FILE') USER(QTMHHTP1) DTAAUT(*RX)
```

4.9.1 Example URL Calling Net.Data Macro

This is how the AS/400 HTTP server configuration and the Net.Data configuration work together to build the path to the Macro and which HTML section the Macro will execute.

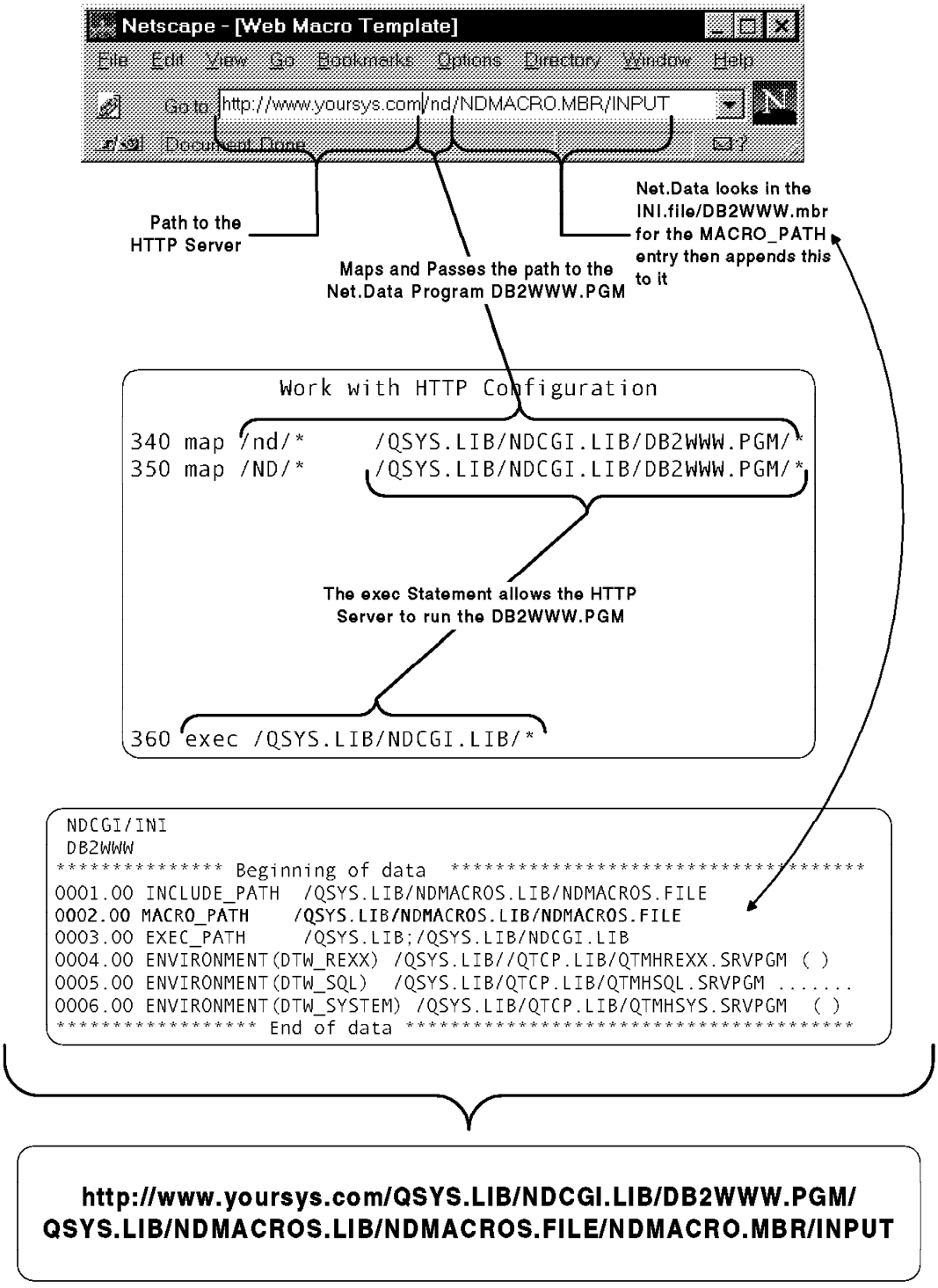


Figure 75. Sample Net.Data URL

4.10 Migrating from DB2 WWW Connection to Net.Data

If you are using DB2 WWW Version 1 and want to migrate to Net.Data (DB2WWW Version 2), read Section 4.9, “Getting Net.Data Up and Running” on page 159. After reading the section, you may need to create a Net.Data initialization file. As explained in the section *Getting Net.Data Up and Running*, you need to create an initialization file for URLs that reference Web macros that are not fully qualified in the Web macro. If you need to create a configuration file, you need to add the SQL language environment statement in the configuration file *SQL (DTW_SQL or SQL) Language Environment*. For information on the syntax of the SQL language environment statement, see Section 4.6.2, “SQL (DTW_SQL or SQL) Language Environment” on page 141.

With DB2 WWW Connection Version 1, you needed an initialization file in every library in which macro files resided. This is no longer true with Net.Data. If you choose to create a Net.Data initialization file and you use the MACRO_PATH configuration statement in your DB2 WWW Version 1 initialization files, you need to combine the path statements into one path statement and insert the combined path statement in the initialization file in the library where the DB2WWW program object resides. Once Net.Data is serving Web macros, you should delete all the DB2 WWW Version 1 initialization files that are not being used.

4.11 Debug HTTP Server Setup for Net.Data

For the following problems, assume that the Net.Data CGI-BIN program object, DB2WWW, has been moved to a library named WWWCGI where your CGI-BIN programs reside.

SYMPTOM

Error 500 (text of message follows)

```
Bad script request -- '/QSYS.LIB/WWWCGI.LIB/DB2WWW.PGM/
  QSYS.LIB' not executable
```

CAUSE: Incorrect Exec rule.

```
Exec /QSYS.LIB/WWWCGI.LIB/DB2WWW.PGM/*
Exec /qsys.lib/wwwcgi.lib/db2www.pgm/*
```

SOLUTION: Specify an Exec rule that only supplies the path to the DB2WWW program. For example:

```
Exec /QSYS.LIB/WWWCGI.LIB/*
Exec /qsys.lib/wwwcgi.lib/*
```

SYMPTOM

Error 404 (Text of message follows)

```
Not found - file doesn't exist or is read protected  [even tried
multi]
```

CAUSE: There is an Exec rule missing.

SOLUTION: Specify an Exec rule that supplies the path to the DB2WWW program in both upper and lower case. For example:

```
Exec /QSYS.LIB/WWWCGI.LIB/*
Exec /qsys.lib/wwwcgi.lib/*
```

SYMPTOM

Error 403 (Text of message follows)

Forbidden - by rule

CAUSE: A Map or Exec rule is missing or incorrect.

SOLUTION: Specify a Map and Exec rule for the DB2WWW program in both upper and lower case. For example:

```
Map /cgi-bin/db2www/* /QSYS.LIB/WWWCGI.LIB/DB2WWW.PGM/*
Map /CGI-BIN/DB2WWW/* /QSYS.LIB/WWWCGI.LIB/DB2WWW.PGM/*
Exec /QSYS.LIB/WWWCGI.LIB/*
```

SYMPTOM

Configuration Statements Correct but Net.Data Not Working

This problem occurs when all configurations are present and correct but Net.Data is still not serving data or documents correctly (or at all). There are several possible causes and solutions:

CAUSE SOLUTION

Map, Exec, or Pass rules are not in the correct order.
When a URL is evaluated against a Map, Exec, or Pass rule, it is acted on based on the first matching rule. Therefore, you must be careful to ensure that the statement to be evaluated is not remapped or altered prior to reaching the desired rule. Also, make sure that you do not have a ″Pass /*″ in the configuration file.

User profile QTMHHTP1 does not have proper authority to access Web macros.
All CGI-BIN programs run under the user profile QTMHHTP1. The QTMHHTP1 user profile must be granted authority to all objects that Net.Data accesses while processing a Web macro.

Path statements in the Net.Data initialization file are not correct.
Net.Data uses the path statements in the initialization file (if there is one) to resolve to any Web macro or executable references in the Web macro that is being processed. If the object references are not fully qualified and the path statements in the initialization file are not correct, Net.Data indicates that the object being referenced is not found. Therefore, you must make sure that either the object references are fully qualified or the Net.Data initialization file has the proper path statements.

4.12 Service and Support

Net.Data is not a separate product in OS/400. The Net.Data function has been integrated directly into OS/400's TCP/IP software; see *IBM TCP/IP Connectivity Utilities/400 Version 3 Release 7* (Program 5716-TC1). V3R2 users can obtain the Net.Data function by requesting the correct PTFs as shown in the following list:

To receive IBM AS/400 service and support for a Net.Data issue, you should treat it the same as you do any other *IBM OS/400* software issue and report it to IBM through your normal procedures.

PTFs and known problems can be found at the Net.Data page:

`http://www.as400.ibm.com/netdata`

For the examples in this chapter to work, you need to have the following PTFs applied to your system.

	V3R7M0	**V3R2M0**
Net.Data	SF38425 **1**	SF38428 **1** **2**
Net.Data (Include File)	SF34468	SF36743

Note:

1. You should always load the latest PTFs.
2. It overlays DB2 WWW Connection if you have not already applied PTF SF36975. Note that even if you do not load this PTF, any HTTP server PTF you load after the release date of PTF SF36975 results in Net.Data being enabled on your system.

The following table lists PTFs that impact Net.Data. Note that the PTFs may have been superseded by other PTFs.

	V3R7M0	**V3R2M0**
HTTP Server	SF37633 **1**	N/A
C/C++ Runtime	SF35079 **1**	N/A
Database	SF35074 **1**	SF38554 **1**
Database (ACL)	SF38031 **1**	SF38030 **1**
REXX	SF34680	SF34679
NLS	N/A	MF14090 **2**
Other	N/A	N/A

Note:

1. Must be loaded for Net.Data to function properly.
2. Must be loaded to enable DBCS support.

Chapter 5. HTML Gateway Implementation

Web browsers can directly access many Internet Information Services, but the presentation of some complex data formats and commercial applications often need some help. For instance, application programs written using CGI-bin are needed in addition to the basic Web server to read relational databases, dynamically build Web pages containing the database records, and transmit them to the Web browser. We have seen the use of such implementations for the AS/400 system using CGI-bin and Net.Data in the previous two chapters.

HTML gateways allow Web clients to access your database, middleware, or applications through the Web server without you having to make any changes to the existing applications, or even to write any additional code.

Most Web servers today require that you write scripts or programs to create interactive forms and applications for the Internet. For most software providers, this can mean learning new tools and procedures if they want to this enhanced Internet functionality. This is not true for existing AS/400 customers. By using an HTML gateway function, your current development tools work for creating Internet or Intranet applications. Once your AS/400 applications are created, you can start using the Internet′s worldwide reach to open new marketing opportunities. Existing AS/400 applications can run over the Web without modifying any code. There is no conversion program to run. Just install and configure the gateway and the applications on your AS/400 system are ready to go!

Why should a company use an HTML gateway?

Take the case of an typical company, as shown in Figure 76 on page 170, the existing business situation can fully utilize existing applications and data on the Web with no coding required. The HTML gateway provides a rapid way of taking advantage of the Internet.

This chapter describes the two HTML gateways that are available for the AS/400 system, how to utilize them, and describes some of the differences between them. The final sections of this chapter provide pointers to additional sources of information on HTML gateways and discuss some of the pros and cons of application enablement versus the use of CGI-bin and Net.Data programming.

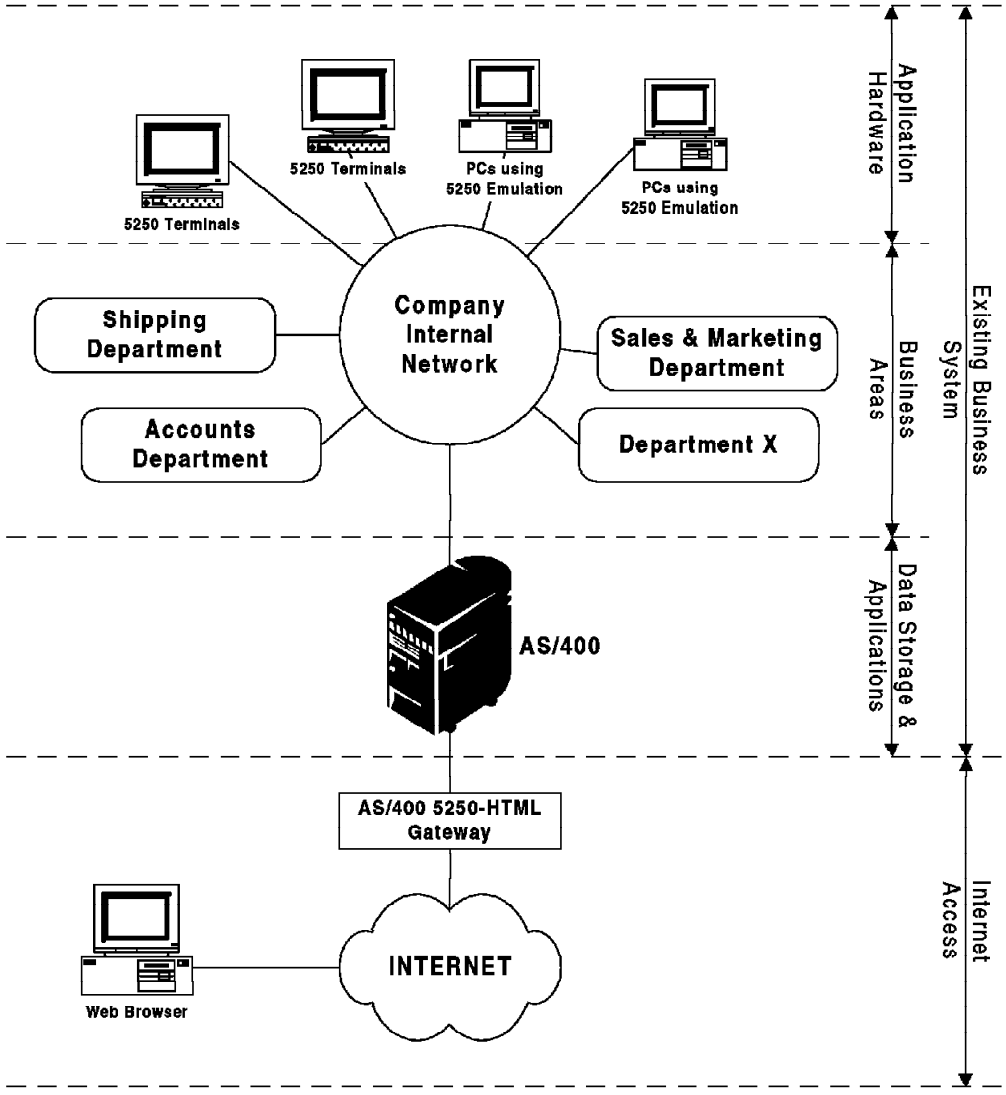


Figure 76. A Typical Company Showing the HTML Gateway Link to the Internet

5.1 What is an HTML Gateway?

An HTML gateway provides automatic translation between 5250 data streams and HTML. The 5250 data stream that is normally sent to end users′ displays is intercepted. The display layout is evaluated including variable data fields and static text. By performing efficient translation between 5250 data and HTML, the user interface (terminal) to your AS/400 application becomes platform-independent.

In addition, the application can be made globally accessible with a more visually appealing user interface. It means that your AS/400 application can have a much improved user interface, using the Web browser as an alternative to the 5250 interface. No AS/400-specific software is needed on the users′ end system.

Currently, two HTML gateways exist for the AS/400 system, one provided by IBM and the other by I/NET. IBM′s product is called Workstation Gateway and is provided as part of Internet Connection for AS/400, which is provided at no cost

with versions V3R2 and V3R7 of OS/400. I/NET's product is called Webulator/400 and operates in conjunction with I/NET's AS/400 Web server products, either Web Server/400 or Commerce Server/400. I/NET's products are available at a cost but can be used with earlier versions of OS/400.

So how is it done?

AS/400 applications are inherently display-oriented. That means, each application creates a series of displays for use in its application. These displays are normally sent out in a 5250 data stream to the workstation or emulator, which shows the text. The HTML gateway intercepts this 5250 data stream and converts it to HTML, a language the Web understands. Any Web browser used for accessing the World Wide Web can work with the application.

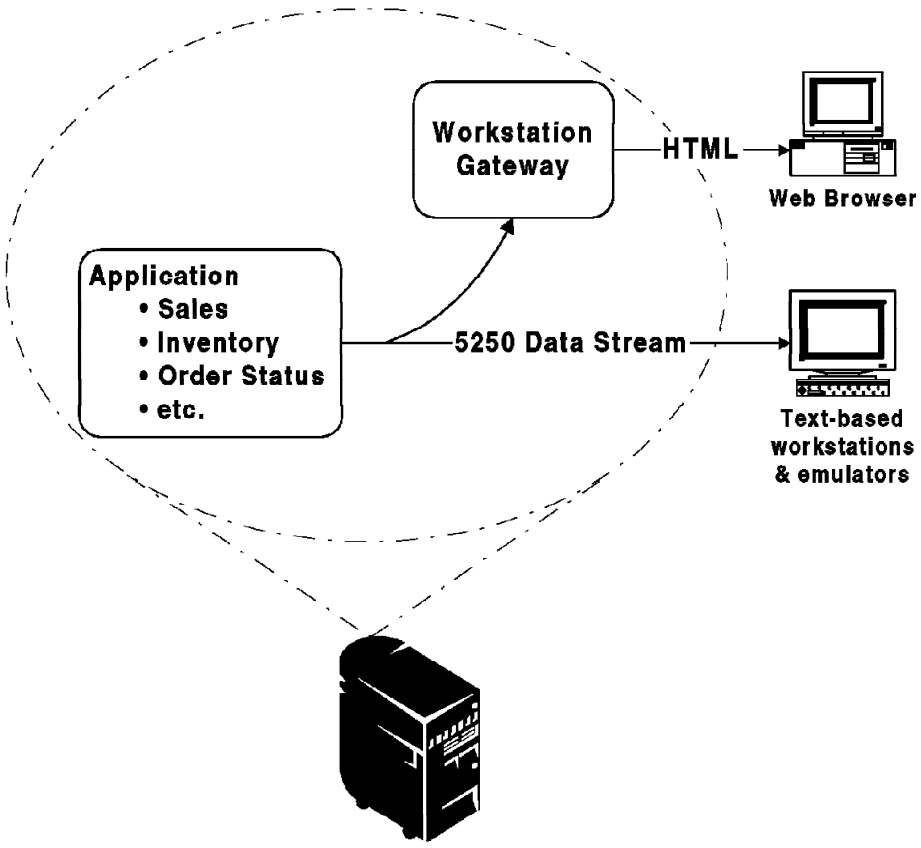


Figure 77. Workstation Gateway

An HTML gateway means your business does not need to rely on one specific client platform. Any PC that has a Web browser installed can run AS/400 applications. There is no additional connection configuration. Just point your Web browser to the AS/400 system and you are in business!

If your company writes AS/400 applications, using an HTML gateway means a wealth of new applications on the Internet. You do not need to retrain your programmers. They can continue using their existing development tools (RPG, COBOL, C, and DDS). Also, with AS/400 HTML Gateway, your programmers can enhance and jazz up your applications by adding graphics and other HTML tags.

It requires only a small change to the DDS specifications and does not affect your workstation users.

5.2 Using the HTML Gateway in Application Development

Having decided upon the use of an HTML gateway, there are various routes you might take. Figure 78 shows the implementation route that you might adopt depending on your application requirement. You must consider whether you are proposing to use your applications on the Internet or the Intranet and, consequently, what login type and security is required. In addition, you must examine the level to which you want to “enhance” your applications. You might also want to restrict the number of displays that are shown, or even develop completely new applications for the web.

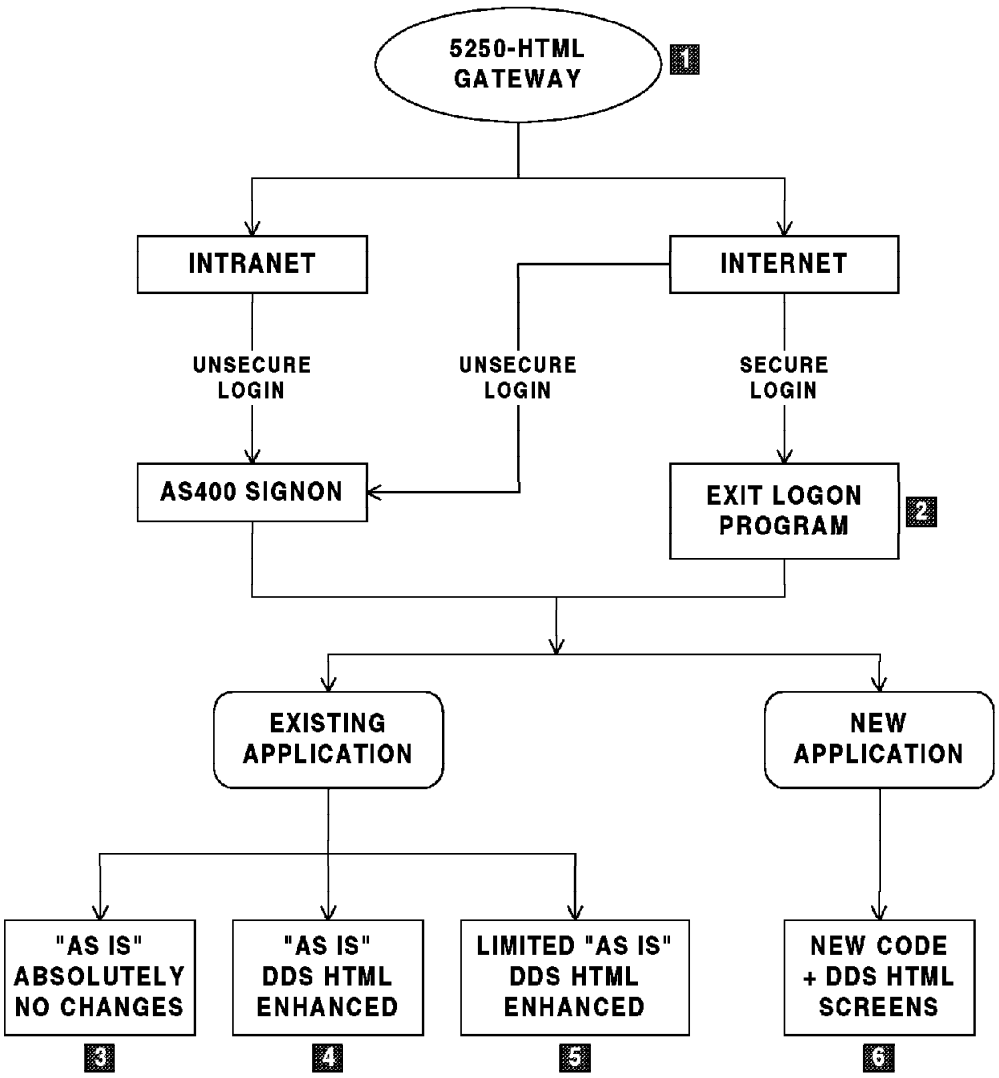


Figure 78. 5250-HTML Gateway Implementation Route

Notes:

1. At present it is only possible to implement a 5250-HTML gateway secure transaction by running I/NET’s Webulator/400 and Commerce Server/400

products. To use this functionality, you should consult the I/NET documentation available with that product.

2. If the user has an AS/400 sign-on display, the USERID and PASSWORD are transmitted as plain text over the Web. This means that unauthorized users might gain access to the AS/400 system. A preferred method is to provide a Logon Exit program whereby the USERID and PASSWORD are not transmitted over the Web. Alternatively, Webulator/400 provides the ability to encode the USERID and PASSWORD before sending the data to the server.
3. The existing application can be fully utilized depending on the user profile. There may be several hundreds of displays, all of which are available. Absolutely no code changes are required.
4. The same as the preceding item but the displays are enhanced by using the new DDS HTML keyword. This requires code changes and recompilation of the DDS files.
5. Using the existing application but providing a limited set of purpose-written HTML displays using the new DDS keyword. For example, three new screens might be provided for order status, order input, and account enquiry. Such displays access the existing routines and data. This implementation provides for the case where you want to provide limited access to your existing applications and data.
6. You might choose to design a completely new application, tailored exclusively for use over the Internet. Your application and display design can be created without having to consider ″green screen″ output.

Further details of the methods of application enhancement are discussed in Section 5.4, “Application Development with Workstation Gateway” on page 181 and Section 5.7, “Application Development with Webulator/400” on page 204

5.3 IBM Workstation Gateway (WSG)

Workstation Gateway is the 5250-HTML gateway from IBM available on the AS/400. You can use your current applications or even develop new Web applications using such tools as RPG, COBOL, DDS, ILE C, etc.

5.3.1 Getting Started

This section explores the WSG support available on the AS/400. It describes in details how to configure and use the WSG to build your Web applications.

5.3.1.1 Starting the Workstation Gateway Server

Important Note

Some of the examples discussed here are taken from the redbook *Cool Title About the AS/400 and Internet*, SG24-4815-01.. You might first want to install the ITSO Company demonstration and other AS/400 Web-based applications from the CD-ROM that came with this redbook. Please see Appendix A, ″Installing the ITSO Company Demo″ in the *Cool Title About the AS/400 and Internet* for instructions on how to get your AS/400 system up and running right away.

Important Note

Install the following PTFs for licensed program 5763-TC1 before using the Workstation Gateway:

- For V3R2, apply SF37790
- For V3R7, apply SF37791

Check the Web site listed in Section 5.11.1.1, “Support Pages on the Internet” on page 216 for the latest PTF information.

Do not confuse the Workstation Gateway with the HTTP Web server. The HTTP Web server allows the AS/400 system to act as a Web server on the Internet. The Workstation Gateway converts your 5250 data stream to HTML. Both servers can be started and function independently from each other.

The Workstation Gateway is a TCP/IP application that services requests from HTTP clients. After the initial request is received from a client, a free port is arbitrarily allocated to the client and that client is considered “active”. All future connections requests for that client occur over the same port number.

The client remains active until the session is signed off or an inactivity timeout limit is reached.

Note

The Workstation Gateway maintains the illusion that the browser is logically (or permanently) connected to the AS/400 system, even though every transaction between the browser and the AS/400 server is a once-only connection. The Workstation Gateway server maintains the virtual terminal API connection indefinitely or until the browser logs off or the inactivity timeout value is exceeded.

The Workstation Gateway server is started through the STRTCPSVR SERVER(*WSG) command and ended with the ENDTCPSVR SERVER(*WSG) command.

Note: Be sure to check the Workstation Gateway attribute to see if the Display Signon (DSPSGN) parameter is set to *YES or else you do not get the Signon display. You can use CHGWSGA command and prompt to check it.

After you start or end the Workstation Gateway, you can use WRKACTJOB SBS(QSYSWRK) command to check the QSYSWRK subsystem to see whether the QTWSGnnnn jobs are there or not (′nnnn′ is a unique numeric string that is derived from the timestamp). Alternatively, it is started through the AUTOSTART option of the STRTCP command.

The format of a link in an HTML document is called a Universal Resource Locator (URL). For HTTP, the URL identifies the protocol that the browser should use when contacting the server (for example, HTTP, FTP, WAIS, Gopher, and so on), the location of the server, and of the requested object. HTTP has the following form:

```
http://hostname:port/path
```

The port numbers for most TCP/IP applications such as FTP, Telnet, or WWW are predefined or you might say “well-known” numbers, which means everyone knows them and uses the same port numbers.

The Workstation Gateway does not have such a well-known port number such as the HTTP server has. Therefore, the port number used by the AS/400 Workstation Gateway is found by querying the local TCP/IP configuration services database. To establish a Workstation Gateway session, you must connect to it by using the form:

```
http://hostname:port/WSG
```

where port is the configured port number for that Workstation Gateway. The default is a TCP port of 5061.

Note: It appears that this port number cannot be changed.

5.3.1.2 Further Workstation Gateway Configuration

This section discusses some specific configuration and software requirements for Workstation Gateway. For more complete documentation of the detailed installation or configuration of the product, you should refer to the following publications:

1. *TCP/IP Configuration and Reference, Version 3*, SC41-3420-04
2. *Cool Title About the AS/400 and the Internet*, SG24-4815-01

5.3.1.3 Configure TCP/IP WSG (CFGTCPWSG) Main Menu

The easiest way to configure the Workstation Gateway is to use the menus. The following examples show the sequence of the configuration commands.

The following display appears if the CFGTCPWSG command is entered on the command line, or if CFGTCPAPP option 15 is selected.

```
                Configure TCP/IP Workstation Gateway
                                                          System:  SYSNM011
 Select one of the following:

     1. Change workstation gateway attributes

  Related options:
    10. Configure HTTP
    11. Work with auto configure virtual devices
    12. Work with limit security officer device access

Selection or command
===> ____________________________________________________________________

F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel
```

Figure 79. CFGTCPWSG Display

- Option 1 - Prompts the CHGWSGA CL command.
- Option 10 - Calls the CFGTCPHTTP CL command.
- Option 11 - Calls WRKSYSVAL SYSVAL(QAUTOVRT)
- Option 12 - Calls WRKSYSVAL SYSVAL(QLMTSECOFR)

5.3.1.4 Change Workstation Gateway Attributes (CHGWSGA) Command Prompt

The following display appears if the CHGWSGA command is prompted from the command line or if CFGTCPWSG option 1 is selected.

The values shown are the current values as determined by the Prompt Override Program for CHGWSGA.

```
      Change Workstation Gateway Attributes (CHGWSGA)
                                                        System:  SYSNM011
Type choices, press Enter.

Autostart  . . . . . . . . . . .   *NO           *NO, *YES, *SAME
Number of clients per server job   20           1-50, *SAME, *DFT
Inactivity timeout . . . . . . .   10           0-60 minutes, *SAME, *DFT
Data request timeout . . . . . .   10           1-1200 seconds, *SAME, *DFT
Display sign on panel  . . . . .   *YES          *SAME,  *NO,  *YES
Access logging . . . . . . . . .   *NO           *SAME,  *NO,  *YES
Top banner URL . . . . . . . . .   *NONE_____________________________________
_____________________________________________________________________________
Bottom banner URL  . . . . . . .   *NONE_____________________________________
_____________________________________________________________________________
_____________________________________________________________________________
_____________________________________

F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

Figure 80. Change Workstation Gateway Attributes Display

Since many clients can be expected to use the Workstation Gateway Server, it is important to try to always have free servers waiting for new connect requests. To stay ahead of potential load demands, jobs are “pre-started” to avoid SBMJOB latency when a new server job is close to being needed.

When we say “pre-started”, we mean that we submit a new child server with the SBMJOB when the number of available jobs (remember we are multiplexing connections within a single batch server job) goes below threshold limits. The threshold limit is determined based upon the value selected for the configured number of clients.

Number of Workstation Gateway Server Clients Per Server Example

Assume that you configured five client sessions per server job. The workstation gateway server ensures that four jobs are available for work. The four jobs allow 20 clients to use the workstation gateway server concurrently with this configuration (five client sessions times four server jobs totals 20 client sessions). The server always ensures enough jobs are running to provide at least 20 client sessions.

There are two types of timeouts for the Workstation Gateway Server:

1. Inactivity timeout (INACTTIMO) - default 10 minutes:

   Specifies the number of minutes the system allows a Workstation Gateway session to remain inactive before it is ended. When a Workstation Gateway session is inactive longer than the specified length of time, it is ended.

Note: It may take the system an additional 1 to 120 seconds to end the inactive session.

If a Workstation Gateway session is ended by the system, you see a message on the Web Browser display ″Session in use - perhaps your session expired?″ when you try to continue the operation.

2. Data request timeout (DTARQSTIMO) - default 10 seconds:

   Specifies the number of seconds from the time a browser connects until the client′s request data is received by the Workstation Gateway.

Note

What happens when the application is abruptly ended?

- Hitting the close button is the equivalent to signing off the Telnet session. Also, if a timeout occurs, the session is ended. It is expected that good programming techniques can trap for this error condition and, indeed, other errors conditions, allow the application to end gracefully. This is discussed further in Section 5.7.1.3, “Non-Programmed Sign Off” on page 205.
- Jumping to another link does not end your session, but if your history list becomes too large, you cannot get back to your session. The “keys” to your session are lost if your last Workstation Gateway window drops off your history list; your session will time out. You can bookmark the page and jump back to it (if the session has not timed out). However, the bookmark is only good for the duration of the Workstation Gateway session.

5.3.2 Workstation Gateway Server Jobs

The Workstation Gateway server is organized into:

- A single *parent* job that *listens* and *accepts* connections from HTTP browser clients. It is important to note that the port used by Workstation Gateway is different from the port of the HTTP server because the Workstation Gateway Server is a new type of server for which there is no well-known port. The parent job has only one function, to hand off connection requests to child jobs.
- One or more *child* jobs: A child job performs the actual work to satisfy the client connect request. The number of jobs actually started can vary as previously discussed.

The benefit of this technique is that it allows the AS/400 system to do a multiplexing of connections within a single batch job.

5.3.3 Using Workstation Gateway

Now that we know what an HTML gateway does, let′s see some examples of the translation from text-based 5250 panels to something a Web client can see and use. For this, we are going to show you some OS/400 displays that have been translated to HTML.

1. **Sign On:**

   Figure 81 on page 178 shows a portion of the traditional AS/400 sign-on display converted now to HTML and displayed on a Netscape client. Note

the functionality is really no different than with a normal text-based 5250 emulator.

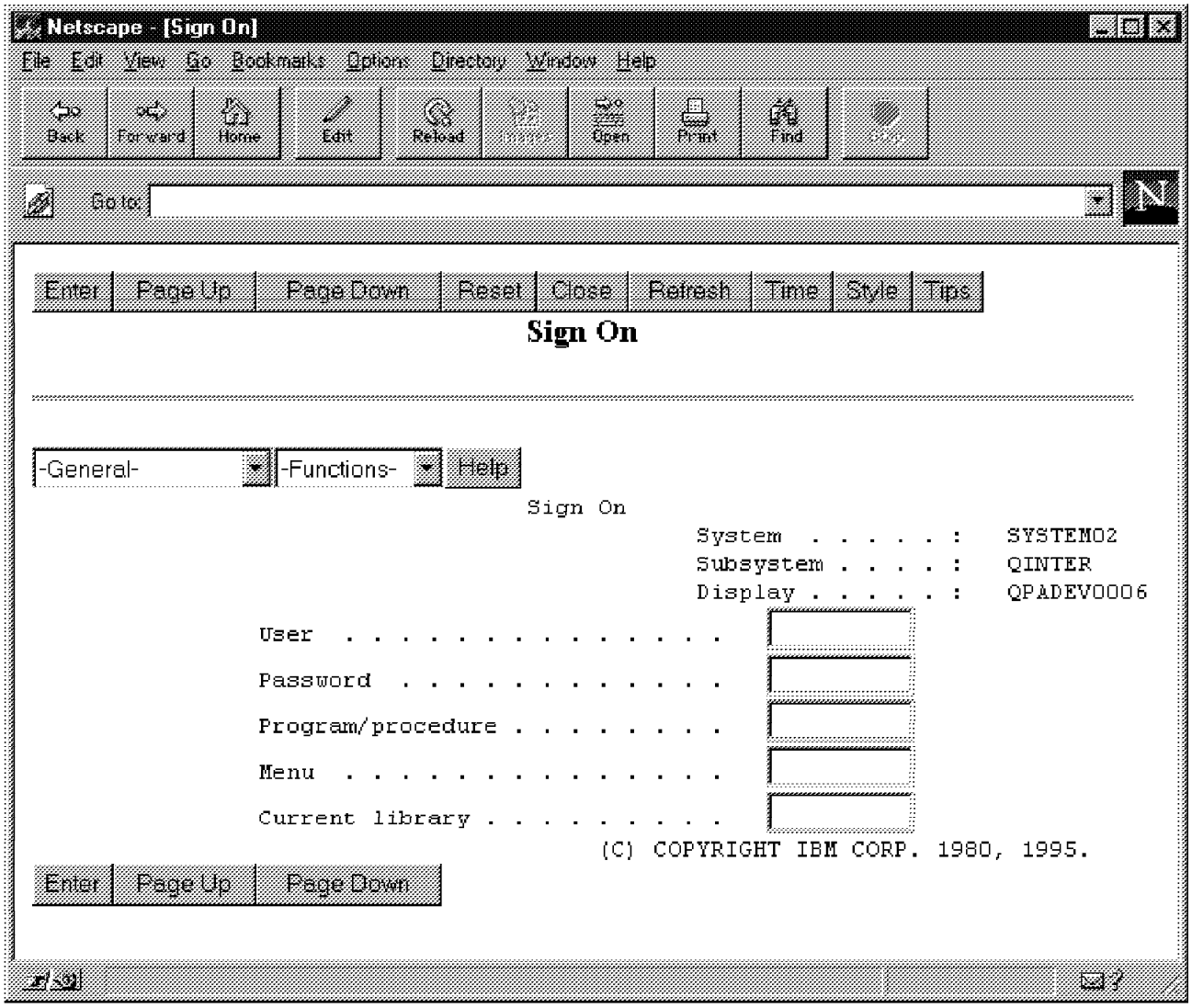


Figure 81. The AS/400 Sign-on Display Seen by the Workstation Gateway

The URL that your Web client needs to specify to evoke the Workstation Gateway support looks similar to the following example:

```
http://hostname:5061/wsg
```

Where:

HTTP: The Workstation Gateway uses the HTTP protocol.

Hostname This identifies the system where the request goes. This can be just the host name or the fully-qualified host name with domain.

:5061 5061 is the default well-known port for the Workstation Gateway server. You must specify this port because your Web client tries to connect to port 80 by default if you fail to override this.

WSG Means using the HTML Workstation Gateway function. WSG can be upper or lower case. If only uppercase works, this indicates that perhaps the latest PTF is not installed on your AS/400 system.

For more information about the logon exit programs, please see Section 5.4.4, “Logon Exit Programs” on page 191. However, if a logon exit program is being used, the URL looks similar to:

```
http://hostname:5061/wsg/QAPP0100
```

Where:

/QAPP0100 The prefix that indicates that exit point information follows. QAPP0100 can be either upper or lower case.

?exit_information
This is used for the optional parameters that can be used to pass information from the client to the Workstation Gateway server running on the AS/400 system. Characters following the /QAPP0100 are interpreted as parameters to be passed to the server job. For the initial connection, these parameters can be a USERID and password used to direct the new client directly to an AS/400 application without the need to sign on to the AS/400 system.

Save State Information

Once the session has been established, what follows after the WSG in the URL is information to allow the AS/400 system to route this display to the proper Workstation Gateway server. This is because the AS/400 system must save state while using a protocol such as HTTP that does not save state! For example, the URL used to save state looks similar to:

```
http://hostname:1384/WSG/057782/QTMTWSG/QTWSG00950
```

Where:

1384 The TCP/IP allocated for the duration of this client session.

057782 The AS/400 job number allocated to this client session.

QTWSG00950
The AS/400 job name allocated to this client session.

2. **Command Entry:**

   Figure 82 on page 180 shows the Command Entry display for the Netscape client.

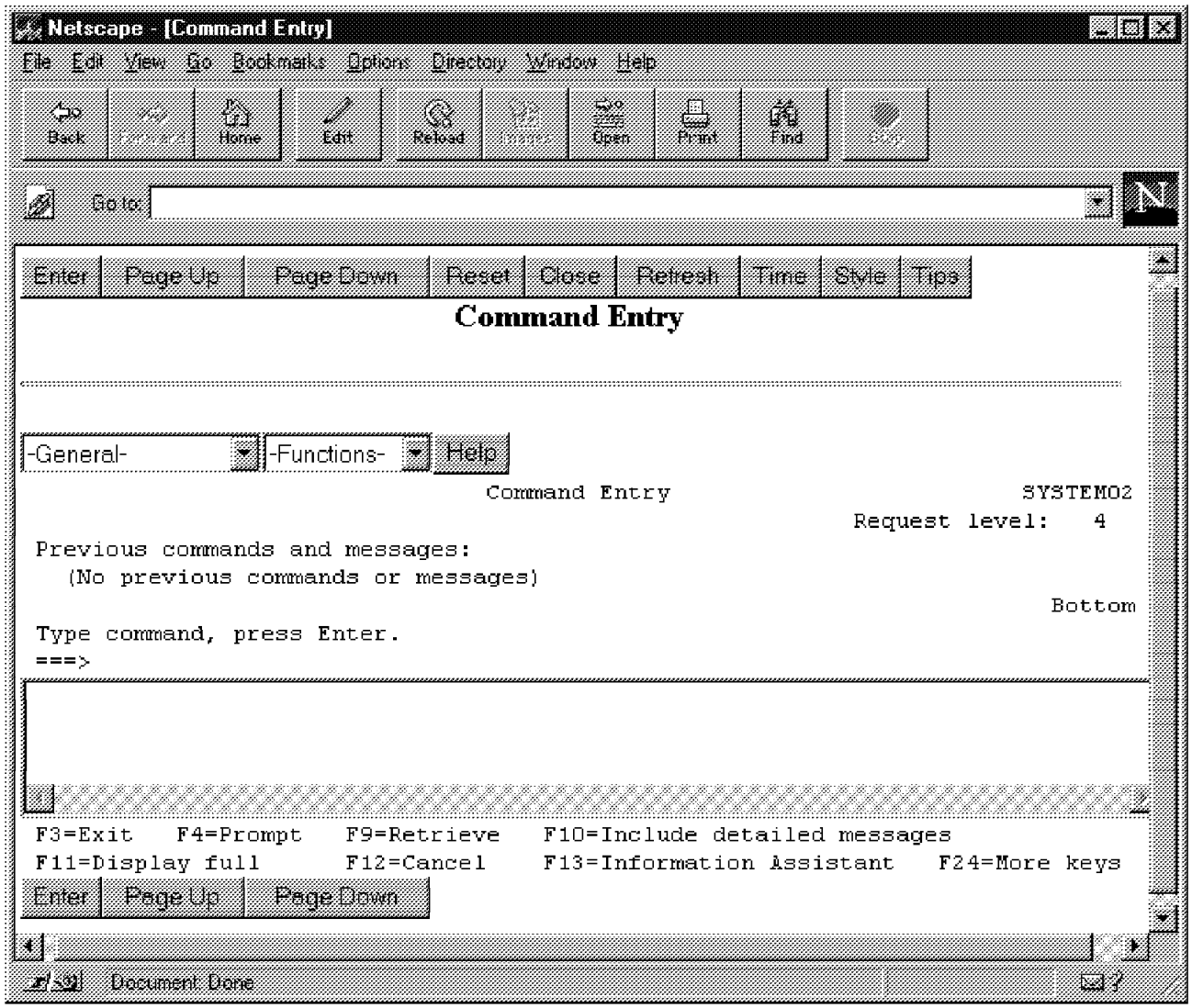


Figure 82. The Command Entry Display Seen by the Workstation Gateway

3. **Work with Active Jobs:**

   Figure 83 on page 181 shows the Work with Active Jobs display for the Netscape client.

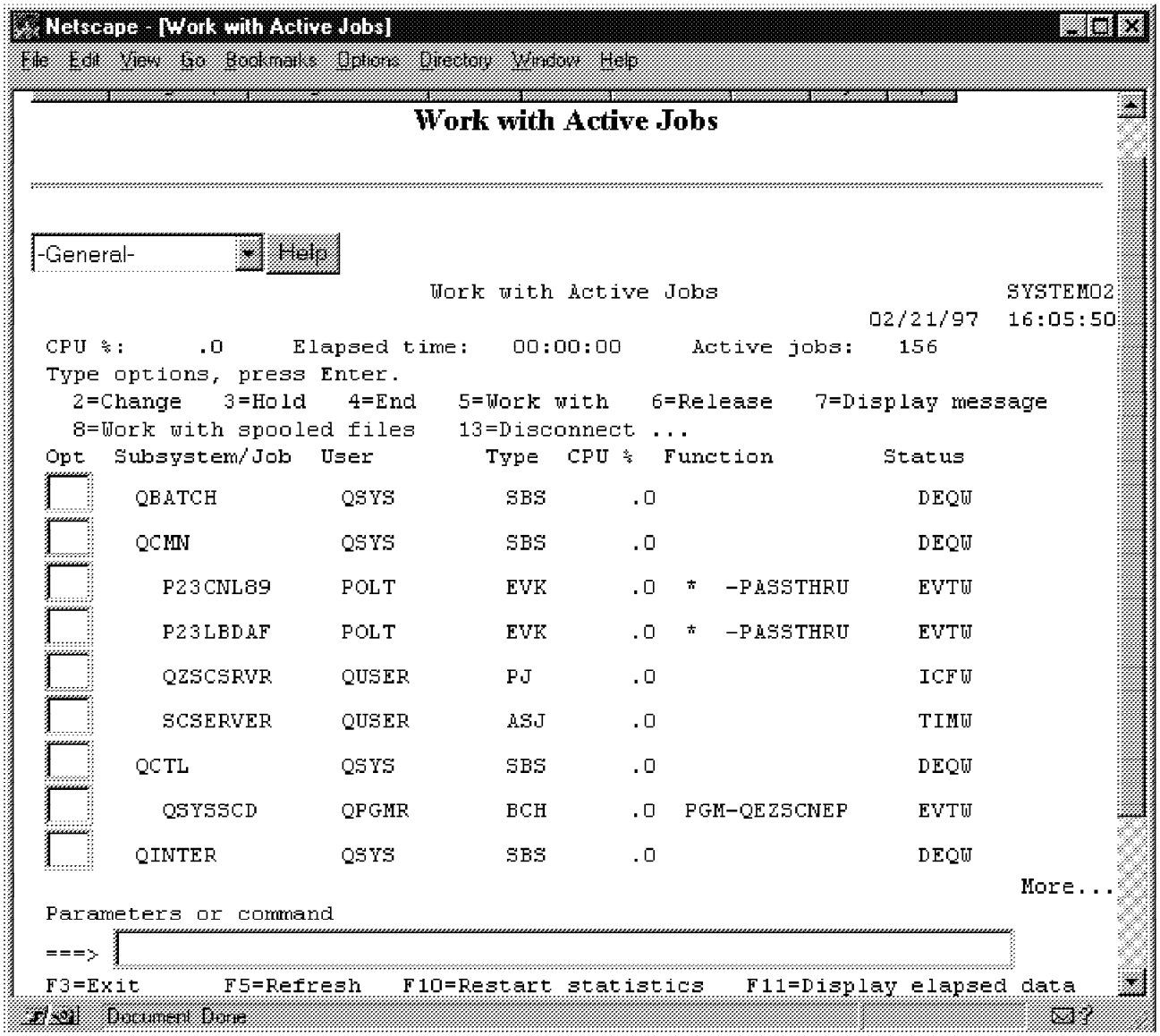


Figure 83. The Work with Active Jobs Display Seen by the Workstation Gateway

5.4 Application Development with Workstation Gateway

There are various ways of enabling or developing applications for use with Workstation Gateway:

- Simply use an existing application "as is". This means using all of the existing DDS displays, logic, and security.
- Enhance existing applications using the new DDS HTML keyword by adding some HTML to existing displays.
- Create new displays using primarily the new DDS HTML keyword. Perhaps if you only want four or five displays being used over the Internet, you can code just these displays.
- Create new applications designed specifically for use through Workstation Gateway.

- Create Logon Exit programs that bypass the AS/400 sign-on display.

When designing or using your application with Workstation Gateway, try and ensure that you keep inside of Workstation Gateway control. That is, do not go off into the ″HTML world″ by providing links to other pages. This may lead to page synchronization problems or the application being stopped by the Workstation Gateway timing out.

5.4.1 Existing Applications (Display Files)

Your existing display files need not be changed. You can use all DDS specifications as you did before. The DDS becomes (when compiled) a 5250 data stream. This means that the DDS keywords such as DSPATR(UL), BLINK, CHECK, and so on are translated in a coded string of data. In this data string, each field is preceded by one or more attribute bytes. This information makes a field such as a customer name underlined, protected, or blinking.

The AS/400 system (or more precisely, the Twinax workstation IOP (input/output processor)) sends out this generated 5250 data stream to your 5250 “green-screen”. The hardware of your display interprets this stream of data and produces a protected, underlined, or blinking field on your display.

This is how it works today. With V3R2 and V3R7, the Workstation Gateway intercepts this 5250 data stream and converts it ″on the fly″ to an HTML data stream. Let′s look at an example to make it more comprehensive.

First, we show you a simple DDS example of a display and how it looks on a 5250 workstation (″green-screen″).

Note: This DDS example is not using any new techniques or HTML keywords.

```
Columns . . . :    1  71           Edit                        ITSOIC400/DDSS
SEU==>                                                              WSGDSP
FMT DP.....AAN01N02N03T.Name++++++RLen++TDpBLinPosFunctions++++++++++++  ++++++
0012.00      A          R DSPR2
0013.00      A                                    2 34'ITSO Company'
0014.00      A                                    3 25'Customer Comment +
0015.00      A                                         Inquiry System'
0016.00      A
0017.00      A                                    6 24'Name : '
0018.00      A            FNAME_X       25   0  6 32
0019.00     A N02                                 7 22'E-mail : '
0020.00     A N02        EMAIL_X       30   0  7 32
0021.00     A N02                                 8 20'Address1 : '
0022.00     A N02        ADDRESS1_X    25   0  8 32
0023.00     A N02                                 9 20'Address2 : '
0024.00     A N02        ADDRESS2_X    25   0  9 32
0025.00     A N02                                10 24'City : '
0026.00     A N02        CITY_X        25   0 10 32
0027.00     A N02                                11 23'State : '
0028.00     A N02        STATE_X        3   0 11 32

 F3=Exit   F4=Prompt   F5=Refresh   F9=Retrieve   F10=Cursor   F11=Toggle
 F16=Repeat find       F17=Repeat change          F24=More keys
```

Figure 84. DDS Source for Customer Comment Inquiry Display

Note: If you are not familiar with DDS programming, we recommended that you consult the following publications:

- *DDS Reference*, SC41-3712-01.
- *Application Display Programming*, SC41-3715-01.

The preceding DDS looks the same as this on a 5250 display station:

```
                              ITSO Company
                        Customer Comment Inquiry System


                           Name : Vera Small
                         E-mail : vsmall@www.net.com
                       Address1 : 123 South Center Street
                       Address2 :
                           City : Rochester
                          State : MN
                        Zipcode : 12345
                        Country : USA
                          Phone : 507-2895300


                        Comment : This is only a simple test! We can use this sampl
e to show you how easily you can use Workstation Gateway in the Internet world! It
 is great for all of the AS/400 customers!!



                             Press ENTER to inquiry again
```

Figure 85. Customer Comment Inquiry DDS on the Traditional Text 5250 Display

Now let's see how the Workstation Gateway made out of our DDS specifications looks. Figure 86 on page 184 shows the result of the 5250 data stream conversion process. Note that this does not mean that you had to recompile the display file. The Workstation Gateway did this automatically "on the fly" for you. When the Workstation Gateway detected that the terminal that receives the 5250 data stream is a "virtual terminal" (that is, a Web client), the 5250 data stream was converted to the HTML data stream.

Note: To view the HTML behind a Web page, most browsers have a viewing option. For Netscape, use the View... Document Source... menu item, and for Internet Explorer use the View... Source... menu item.

```
<HTML>
<HEAD>
<TITLE>AS/400 Workstation Gateway </TITLE>
</HEAD>
<BODY>
<FORM METHOD="POST" ACTION="http://9.5.69.208:1029/WSG/003816/QTMTWSG/
<INPUT TYPE="HIDDEN" NAME="SESSION" VALUE="/A27CDCB57704125F/3B35D6E8">
VALUE="Enter"><INPUT TYPE="SUBMIT" NAME="/A27CDCB57704125F/3B35D6E8/
NAME="/A27CDCB57704125F/3B35D6E8/BUTTON.999-999=@u" VALUE="Page Up">
VALUE="Close"><INPUT TYPE="SUBMIT" NAME="SPECIALS" VALUE="Refresh">
VALUE="Style">
<CENTER><H3>AS/400 Workstation Gateway</H3></CENTER>
<HR>
<SELECT NAME="-General-" SIZE=1><OPTION SELECTED VALUE="-NONE-">-Gener
VALUE="@C">Clear<OPTION VALUE="@A@<">Record Back<OPTION VALUE="@x">PA1
VALUE="@A@C">Test Request<OPTION VALUE="@S@E">Host print screen<OPTION
VALUE="@A@H9O@E">Sign off<OPTION VALUE="@c">F12<OPTION VALUE="@3">F3</
VALUE="@H">Help</SELECT>
                                ITSO Company
                          Customer Comment Inquiry System
                            Name :  Vera Small
                          E-mail :  vsmall@www.net.com
                        Address1 :  123 South Center Street
                        Address2 :
                            City :  Rochester
                           State :  MN
                         Zipcode :  12345
                         Country :  USA
                           Phone :  507-2895300
                         Comment :  This is only a simple test" We can use
e to show you how easily you can use Workstation Gateway in the Internet
 is great for all of the AS/400 customers"
                             Press ENTER to inquiry again
<INPUT TYPE="SUBMIT" NAME="/A27CDCB57704125F/3B35D6E8/BUTTON.999-999=
NAME="/A27CDCB57704125F/3B35D6E8/BUTTON.999-999=@v" VALUE="Page Down">
Up">
<INPUT TYPE="SUBMIT" NAME="/A27CDCB57704125F/3B35D6E8/BUTTON.999-999=
NAME="/A27CDCB57704125F/3B35D6E8/BUTTON.999-999=@2" VALUE="F2 "><INPU
TYPE="SUBMIT" NAME="/A27CDCB57704125F/3B35D6E8/BUTTON.999-999=@4" VAL
VALUE="F5 "><INPUT TYPE="SUBMIT" NAME="/A27CDCB57704125F/3B35D6E8/BUT
NAME="/A27CDCB57704125F/3B35D6E8/BUTTON.999-999=@7" VALUE="F7 "><INPU
TYPE="SUBMIT" NAME="/A27CDCB57704125F/3B35D6E8/BUTTON.999-999=@9" VAL
VALUE="F10"><INPUT TYPE="SUBMIT" NAME="/A27CDCB57704125F/3B35D6E8/BUT
NAME="/A27CDCB57704125F/3B35D6E8/BUTTON.999-999=@c" VALUE="F12">
<INPUT TYPE="SUBMIT" NAME="/A27CDCB57704125F/3B35D6E8/BUTTON.999-999=
NAME="/A27CDCB57704125F/3B35D6E8/BUTTON.999-999=@e" VALUE="F14"><INPU
TYPE="SUBMIT" NAME="/A27CDCB57704125F/3B35D6E8/BUTTON.999-999=@g" VAL
VALUE="F17"><INPUT TYPE="SUBMIT" NAME="/A27CDCB57704125F/3B35D6E8/BUT
NAME="/A27CDCB57704125F/3B35D6E8/BUTTON.999-999=@j" VALUE="F19"><INPU
TYPE="SUBMIT" NAME="/A27CDCB57704125F/3B35D6E8/BUTTON.999-999=@l" VAL
VALUE="F22"><INPUT TYPE="SUBMIT" NAME="/A27CDCB57704125F/3B35D6E8/BUT
NAME="/A27CDCB57704125F/3B35D6E8/BUTTON.999-999=@o" VALUE="F24">
</PRE>
</FORM>
</BODY>
</HTML>
```

Figure 86. A Portion of the HTML Automatically Generated by the HTML Gateway

Finally, let's see how this looks on a Netscape Web browser and also an example of how a subfile window looks.

Note: The result you see on a Web browser is totally dependent upon how you configured the browser. If you choose another font, another background color, or another font size, the actual appearance of your HTML data stream on your PC might look quite different from our example.

Important

Workstation Gateway displays messages to the client using white colored text; if the user changes the browser background to white, no systems' messages are seen.

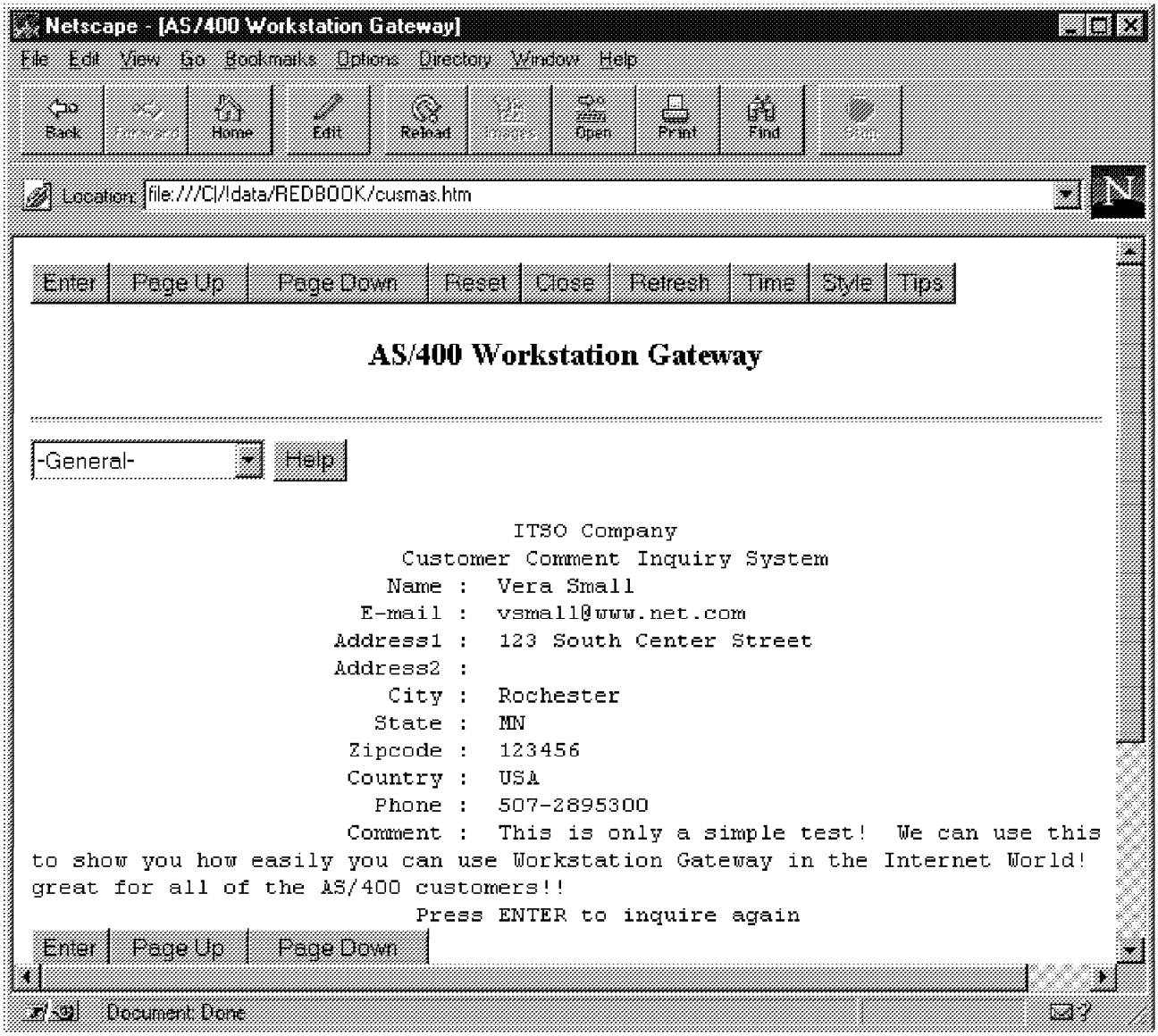


Figure 87. Display of Customer Master Record through Workstation Gateway

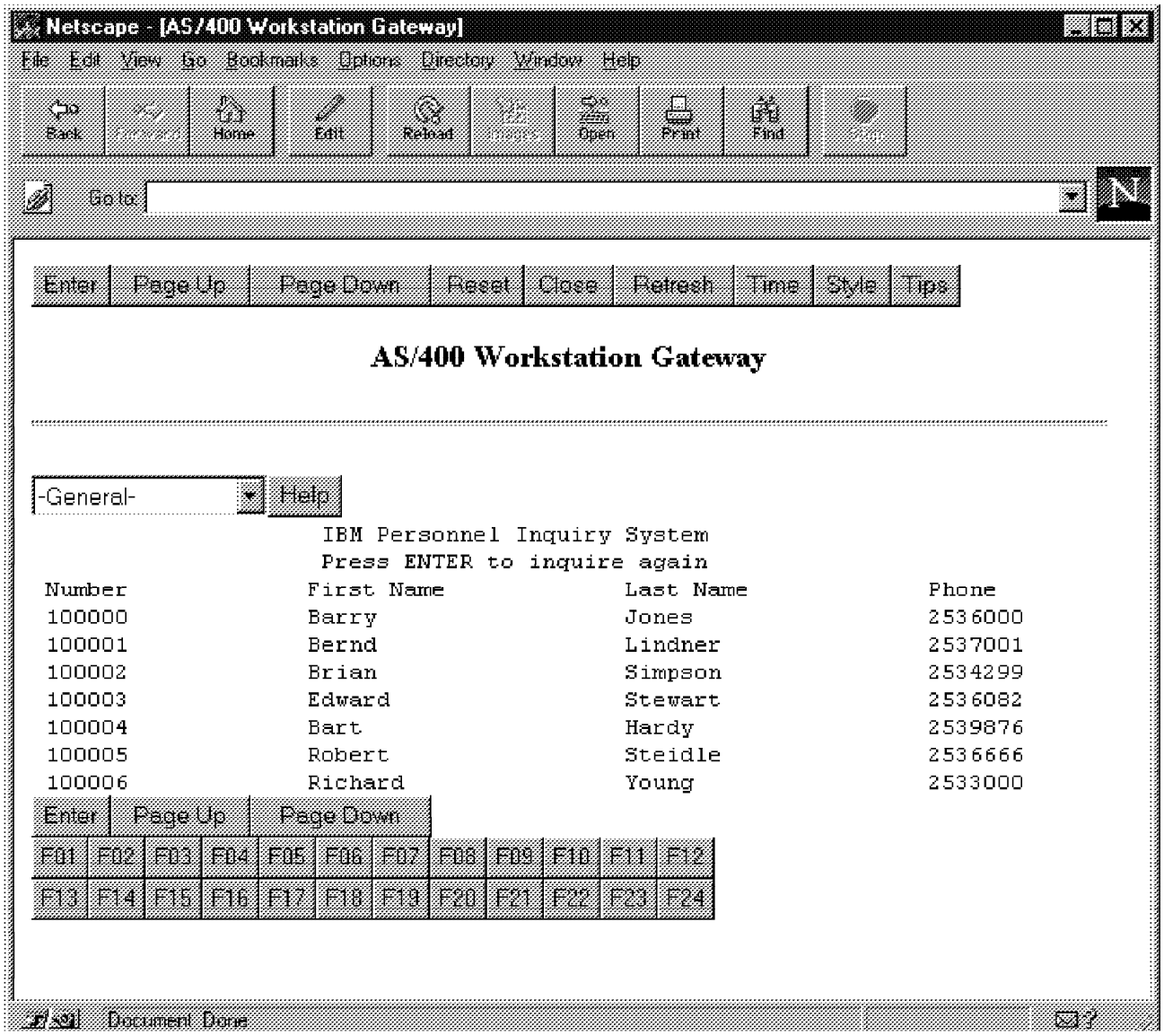


Figure 88. Display of Subfile Record through 5250-HTML Gateway

5.4.2 DDS to HTML Conversion

The Workstation Gateway formats the 5250 display according to the following basic rules:

1. The display is shown as 80-column pre-formatted text.
2. Blank lines are omitted to reduce the vertical size of the display image in the browser.
3. 5250 input fields are translated into HTML INPUT tags with the same field length as the 5250 field.
   - If an input field is longer than 160 characters, the field is translated into an HTML TEXT area that is 80 columns long and has enough lines to contain all the input text needed.

- If an input field is between 60 and 160 characters long, it is shown as an input field with 60 visible characters and a maximum length of the 5250 field size.

4. Function key tags recognized in the function key area of the display are added to the “menu” items at the top of the document.

5.4.3 DDS HTML Support

The Workstation Gateway support allows the insertion of HTML tags into the DDS of a display file. This allows us to utilize the graphic capabilities of a Web browser with only minor changes to the existing DDS. For example, a customer can add graphics through the IMG HTML tag to an existing display file and display a graphic image along with the display.

Note: These HTML tags are only inserted into the data stream that flows to a terminal if the device query indicates that the device is a PC (or more precisely, an AS/400 5250 Workstation Gateway virtual terminal). Otherwise, the HTML tags are ignored for normal displays.

This simplifies and eases the handling of display files because only *one* source is needed for Web browsers, and “green-screens”.

5.4.3.1 The New DDS Keyword

There is a new DDS keyword, HTML (HyperText Markup Language). This field-level keyword can be treated the same as a usual constant. Two things are different from a common constant. First, you have to put the new keyword HTML before the constant, and second, the “constant” itself must consist of an HTML string that must use the HTML syntax.

Let′s take a look at a DDS example with HTML statements.

```
A                                   DSPSIZ(24 80 *DS3)
A                                   CA03(03)
A                                   CA12(12)
A     R DSPREC
A                                2 22'IBM Rochester Personnel Inquiry -
A                                     System'
A                                3  4HTML('<IMG SRC="//INTERNUT/+
A                                    /ITSOIC.400/AS400.GIF">')
A                                3  4HTML('<Table BORDER>')
A                                3  4HTML('<TR>')
A                                3  4HTML('<TH COLSPAN=2></TH>')
A                                3  4HTML('<TH>Table Demo</TH>')
A                                3  4HTML('</TR>')
A                                3  4HTML('<TR ALIGN=CENTER>')
A                                3  4HTML('<TH ROWSPAN=2></TH>')
A                                3  4HTML('<TH>First</TH>')
A                                3  4HTML('<TD>Row</TD>')
A                                3  4HTML('</TR>')
A                                3  4HTML('<TR ALIGN=CENTER>')
A                                3  4HTML('<TH>Second</TH>')
A                                3  4HTML('<TD>Row</TD>')
A                                3  4HTML('</TR>')
A                                3  4HTML('</TABLE>')
A                                3  4HTML('<IMG ALIGN=TOP +
A                                    SRC="//INTERNUT/+
A                                    /ITSOIC.400/RHAND.GIF">')
A                                3  4HTML('<A HREF="http://internut/+
A                                    class/wsg000.htm">Link-Here</A>')
A                                6 20'Please Input Employee Number : '
A       EMPNUM         6S 0B  6 52
A  02                                9 26'  First Name : '
A  02        FIRSTN        20   0  9 42
A  02                               10 26'   Last Name : '
A  02        LASTN         20   0 10 42
A  02                               11 26'Phone Number : '
A  02        PHONO          7S 00 11 42
A  02                               14 25'Not found - Please press ENTER'
A  02                                     DSPATR(RI)
A                                   21  5'F03=Exit'
A  02                               21 30'Press ENTER to inquiry again'
```

Figure 89. Sample 5250 DDS Enhanced with the HTML Tag

Note: The plain text is mixed with HTML tags.

Important

The IMG SRC tag **must** have a fully qualified URL, and the HTTP server must be running for Workstation Gateway to serve images. Basically, the Workstation Gateway server re-directs all links for HTTP type objects such as images to the local server (meaning that it has to be running on the same AS/400 system as the Workstation Gateway server).

Now, let's take a look at the display output of the sample DDS with the preceding HTML tags. You can see there is a "Link-Here" which corresponds to the HREF in DDS. Also, a "Table Demo" is related to the TABLE tags.

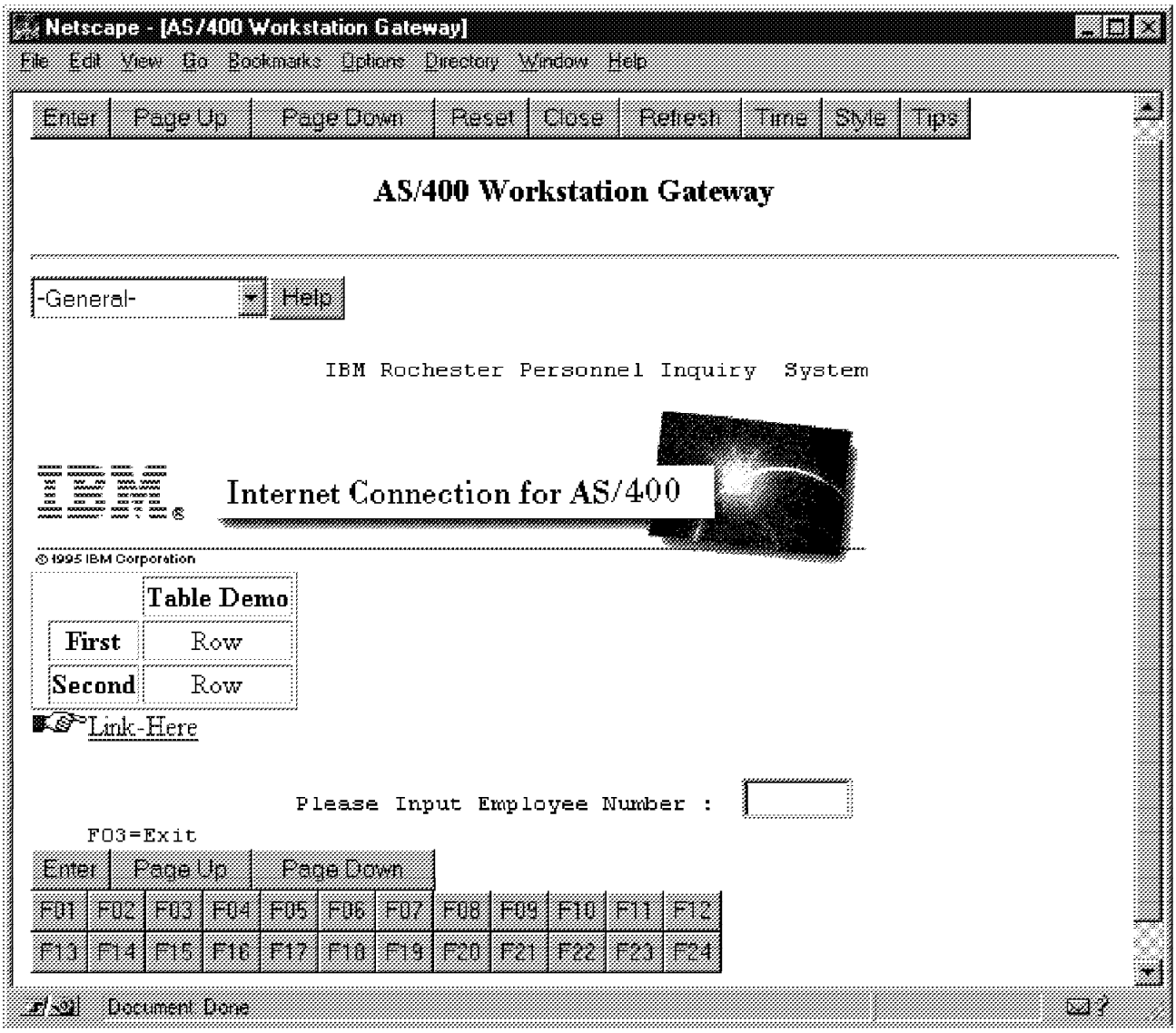


Figure 90. Link, Table, and Images Imbedded in DDS

If you have installed the sample code from the *Cool Title About the AS/400 and Internet*, this display file and the program for it are in the library ITSOIC400. The name of the file is SHTMLD and its source is in the source file QDDSSRC. The name of the program is SHTMLR and its source is in QRPGSRC.

Note

- Using CRTDSPF and setting the ENHDSP (Enhanced Display) parameter to *NO, the HTML keyword is ignored; hence, it is possible to turn off the HTML without recompiling the DDS.
- You can also use variable substitution in DDS and pull data from a DB file and merge it with the HTML keyword such as in a table.

What are HTML Tags?

HTML documents consists of plain text interspersed with markup commands called **tags**. The tags are instructions to the browser software on how to display the text. They are represented by strings enclosed in <angle brackets>. See Section 2.9.1, “Hypertext Markup Language (HTML)” on page 25 for more information about HTML.

Another thing to mention is that in the preceding example, ″normal″ DDS keywords and HTML specifications are used within one source. HTML is a procedural language where the order of the tags determines when they are processed. Row and column have no meaning in such a language. In this case, the row and column are used to determine the order in which the HTML tags are sent to the browser. For example, an HTML keyword at row 2/column 4 appears before an HTML keyword at row 2/column 6. See Appendix B, “HTML Gateway Code Examples” on page 263 for more examples of DDS usage and overlap issues.

With the HTML keyword, constant fields that have the same row and column value are processed in the order in which they appear in the DDS source.

Please notice that you can use any HTML tag except <HTML>, </HTML>, <HEAD>, </HEAD>, <TITLE>, </TITLE>, <BODY>, and </BODY> because all of the preceding tags are used by HTML Gateway to build the overall structure.

Note

- It is possible to use the <PRE>...</PRE> tags but these tags are also used by Workstation Gateway to generate the HTML page.
- It is not possible to add certain HTML tags to the header section of the generated HTML page.

5.4.3.2 The HTML Keyword Syntax

The new HTML specification can have two formats:

```
HTML('datastring with a valid HTML tag')
```

or

```
HTML(&program-to-system-field)
```

A parameter is required after an HTML keyword. This parameter can be a valid HTML tag enclosed in single quotes or a program variable. The program-to-system field can be any legal length and has to be alphanumeric (A in position 35).

Note: The syntax of the HTML tag is *not* syntax checked by the DDS compiler. The browser that receives the HTML sequence performs syntax checking. It is important, therefore, that you test the display on the browser to verify the correct HTML and DDS syntax.

5.4.3.3 Limitations and Restrictions

The following keywords are not allowed with the HTML keyword:

- COLOR
- DATE
- DFT
- DSPATR
- EDTCDE
- EDTWRD
- HLPID
- MSGCON
- NOCCSID
- OVRATR
- PUTRETAIN
- SYSNAME
- TIME
- USER

The HTML keyword is not allowed on a field in a subfile record.

5.4.4 Logon Exit Programs

An application logon exit program (QAPP0100) allows us to bypass the AS/400 Sign-on display and invoke an application program directly without the client browser having to send a user profile or password. This allows the customer the option of providing *any* application to client browsers without requiring a sign on. This has the added benefit of removing the command line for this client. This is done by calling a customer program that authenticates the client request and provides sign-on information to the Workstation Gateway Server.

The Workstation Gateway Server uses the output of the customer's user exit and performs the sign-on action on behalf of the client browser.

When the user exit is given control, it may perform any desired validation using the supplied Internet Protocol address and any of the supplied operation-specific information extracted after the "/WSG/QAPP0100" string in the URL. Setting the "Allow Operation" output determines whether the automatic logon is performed, or whether an error message is returned to the client browser.

If the operation is allowed, the user exit must return the user profile, password, current library, and program. All output must be non-NULL or else an error is returned to the client browser.

Only one exit program can be registered for the exit point QAPP0100. The *TCP/IP Configuration and Reference*, SC41-3420-03, contains information about the registration of this exit point.

Table 16. User Exit Program Extensions

Parameter	Description	In/Out	Type
1	Operation Specific Information	Input	Char(*)
2	Length of Operation Specific Information from URL	Input	Binary(4)
3	Client IP (Dotted Decimal) address	Input	Char(15)
4	CCSID	Input	Binary(4
5	Allow Operation	Output	Char(1)
6	User Profile	Output	Char(10)
7	Password	Output	Char(10)
8	Current Library	Output	Char(10)
9	Program to call	Output	Char(10)
10	Initial menu	Output	Char(10)
11	Session end URL	Output	Char(300)

Note: Do not use the Client IP address, parameter 3, as a means of identifying authorized users to your system. It is possible for IP addresses to be “spoofed” or copied.

Note: Session end URL Output parameter 11 is accepted as an extension to User Exit point QAPP0100 for Workstation Gateway. It must be a fully-qualified URL (up to 300 characters, trailing blanks stripped), for example:

```
http://httpserver/directory/some.html
```

This URL link is sent to the Web browser whenever the Workstation Gateway session is ended. If no URL is specified, the default action is to post timing statistics.

One trick you might want to do is have the session end URL arrive as part of parameter 1 (operation specific information). The exit program can simply echo parameter 1 out through parameter 11. The following example illustrates this concept:

```
<A href="http://wsgserver:5061/WSG/QAPP0100?http://httpserver/return.html">GO!</A>
```

This is useful if you want to return the HTML form that initiated the Workstation Gateway session, or if you want to jump to another HTML form when you are done with the Workstation Gateway session.

IMPORTANT!

Try to ensure that your exit programs do not allow users access to any of the following functions:

- Command Entry line
- STRPASTHR
- Attention Key
- System Request
- F3 exit key or similar (if possible, do not use F-keys, for better ease of use and better security!)

Where possible:

- Use a USER profile that has no rights or authority to sensitive information or programs.
- Prevent the sign-off/sign-on display from being shown to the user.
- Trap all errors and, if in doubt, sign off.
- Only allow sign off to the appropriate Workstation Gateway profile.
- Do not forget access through help displays.
- Code an exit URL so the client is passed back to a useful HTML page.

Above all, test your exit program and application and try to look for all the access points into your system through the Workstation Gateway.

Use the following steps to use the Exit Program:

1. Code your Exit Program. You can use the sample program provided.
2. End the Workstation Gateway Server by using the ENDTCPSVR SERVER(*WSG) command.
3. Change the workstation gateway attribute for using the Exit Program with the CHGWSGA DSPPGN(*NO) command.
4. Register your Exit Program by using the WRKREGINF EXITPNT(QIBM_QTMT_WSG) command. Use option 8 to add your exit program.
5. Start the Workstation Gateway server by using the STRTCPSVR SERVER(*WSG) command.
6. Open a URL in Web browser as `http://hostname:5061/WSG/QAPP0100?xxxxx` where xxxxx is any string that you want a user to type in so you can use your Exit Program for security checking.

Note

- Only one exit program can be registered for exit point QAPP0100.
- If you get an exception error on the browser (such as CEE9901), it might be worth checking the object authority for the exit program. Use the WRKOBJ OBJ(QUSRSYS) command, option 2, and change the *PUBLIC user authority from *USE to *CHANGE.
- Also check the job log for the Workstation Gateway jobs using WRKACTJOB and look for user profile QTMTWSG under the QSYSWRK subsystem.
- When restarting the Workstation Gateway server, allow several minutes for the stop and start process to take place!

5.4.4.1 Sample Exit Program

This sample exit program uses one of the input variables (IP address of the remote host system) from the HTML Gateway system program to check if a Web browser user has the right to use the HTML Gateway function.

You can find the source of this sample program in the QILECSRC source file in the ITSOIC400 library.

From Web browser, open the URL `http://hostname.domainname:5061/WSG/QAPP0100`. You can bypass the sign-on display and go into the command entry display.

Note: Please see Appendix B, "HTML Gateway Code Examples" on page 263 for the code listings of exit programs in C, RPG, and CL languages.

5.5 Tips for using Workstation Gateway

The following tips or hints might help you make better use of Workstation Gateway function:

1. The Workstation Gateway function is not a substitute for an original workstation connection.
2. The performance of using Workstation Gateway is not as good as using a traditional 5250 connection such as tn5250 through Telnet. Care should be given when sizing an AS/400 system, as this kind of 5250 to HTML conversion is not done without extra cost in CPU terms. See Chapter 8, "Internet Application Performance" on page 237 for more details on Workstation Gateway performance statistics and possible improvements (tuning).

   Performance Tip

   Preformatted text is rendered on the screen faster by the browser than formatted text (for example, tables). In addition, preformatted text produces a smaller HTML file that can be transferred quicker.

3. With the exception of the character input fields, always use the mouse to click on the function key you want instead of using a key on the keyboard such as Enter or F3.
4. Context sensitive help versus general help. The AS/400 system considers the ″?″ character as the **first** character of an input field to be a ″cursor move″ request. The request moves the cursor to that input field, thus allowing you to use the F1 (contextual help) or F4 (prompt) buttons. In fact, any function key button can be used, not just F1 and F4 buttons.

   The AS/400 system does not keep the ″?″ in the input field upon return from the function key button action. Please note that this help function applies only to input fields. If you have help defined for some output fields, there is no way to get to it.

   For general help, the Help menu pull-down (or button) first moves the cursor to display row 1, column 1, and invokes the help command. This is **not** the same as context sensitive help. This is because the cursor is moved to a position that is probably context insensitive before invoking help. This also means using the Help menu pull-down (or button) does not normally give the help for the home (default) input field, and the ″?″ invocation mechanism may be needed to force this help to appear.

5. Different Web browsers may have a different display output when handling the same HTML tags.

   Recommendation

   Try and test your application on as many browsers as possible, at least Netscape, Internet Explorer, and Mosaic based browsers such as IBM′s Web Explorer.

6. After you have established a session between your Web client and the AS/400 HTML gateway server function, you might want to start a Telnet or use STRPASTHR to connect to another remote host. This function was not formally tested, but it seems to work in the ITSO network.

   Note

   It is **not** recommended that you enable this functionality for public Web sites.

7. Never use previous and next page function in a Web browser as a way to jump into the application flow from that page. We suggest turning off caching on the Web client to enforce this suggestion.
8. No text assist is available.
9. You cannot use applications that automatically update the 5250 display without the client using an aid key. An example is the performance tools WRKSYSACT command when you use the automatic display refresh option.
10. Style button:

    The action bar (top row of buttons) has a style button that toggles the style used for the F1-F24 buttons. One mode shows displays F1-F24 as two rows of buttons at the bottom of every display. This takes up more display space, but has the advantage of letting you quickly submit the form.

    The other mode puts F1-F24 into a Function menu pull-down next to the General menu pull-down. This makes submitting the F1-F24 keys a two-step process (select menu item, then press Enter), but has the advantage of less display clutter and a faster display by the browser.

11. NLS code pages:

    Special characters and code pages can be requested by each individual client. Translation is usually from the EBCDIC CCSID of your AS/400 system to the ASCII CCSID specified in the CHGWSGA CCSID parameter. So, if the CHGWSGA CCSID parameter is changed, the EBCDIC CCSID default value also changes, since we get the ″best fit″ EBCDIC CCSID from the configured ASCII CCSID. This default can be overridden by each user. Examples for Sweden are:

    ```
    No user exit:
        http://as400.endicott.ibm.com:5061/WSG-SWB
        http://as400.endicott.ibm.com:5061/WSG-SWI
        http://as400.endicott.ibm.com:5061/WSG-SFI

    User exit:
        http://as400.endicott.ibm.com:5061/WSG-SWB/QAPP0100
        http://as400.endicott.ibm.com:5061/WSG-SWI/QAPP0100?any_string_data
        http://as400.endicott.ibm.com:5061/WSG-SFI/QAPP0100
    ```

Refer to Appendix C in the *National Language Support*, SC41-3101, for a table of supported keyboard strings that can be used.

Here are the rules to which the Workstation Gateway converts EBCDIC to ASCII:

- The Workstation Gateway builds the HTML in code page 037 using only invariant characters for the tags and control words.
- The 5250 application sends the data/character in the code page that it determines independent of the Workstation Gateway.
- Once the document is built, it is converted to ASCII in the CCSID specified in the following order:
  a. MIME header specified by the remote browser
  b. Overridden by the interactive subsystem (WSG-xxx)
  c. WSG attribute (CHGWSGA command)
  d. The system default CCSID

12. If the remote Web client is signing on to the AS/400 system with a unique user profile (not a shared anonymous profile), you can still use the OUTQ parameter of that user profile to direct all of the printouts to a single queue. This queue, in turn, can be a remote output queue that uses TCP/IP to route the print data back to any printer in the network (most likely, the printer sitting right next to the user's desk). Many IBM manuals and redbooks have instructions on how to configure such remote output queues. One in particular that includes sample configurations is *Printer Device Programming*, SC41-3713-01.
13. Log all accesses made by the Workstation Gateway (this is a configuration item). All accesses to the Workstation Gateway are recorded in physical file QATMTLOG in library QUSRSYS. Further information can be found in *TCP/IP Configuration and Reference*, SC41-3420-04.

5.6 I/NET's Webulator/400

Webulator/400 is the 5250-HTML gateway from I/NET, and is used in conjunction with either Web Server/400 or Commerce Server/400. You can use your current applications or even develop new Web applications using such tools as RPG, COBOL, DDS, ILE C, and so on. More information about I/NET and their AS/400 Web serving products can be found at the following URL:

`http://www.inetmi.com/`

5.6.1 Getting Started

Full installation and configuration details of the Webulator/400 product can be read in the *Webulator/400 User Guide*, which can be found at the following URL:

`http://www.inetmi.com/pubs/webulate/usrguide.htm`

Or you can read the documentation online, which may require you to start the server to access the links. In particular, the following items are discussed in more detail in the Webulator/400 documentation:

- Sign-on methods
- User profiles

- Virtual terminals
- Programming
- System security values and auditing

5.6.1.1 Configuring Webulator/400

1. Install the appropriate Web Server/400 or Commerce Server/400 and Webulator/400 code.
2. Create a Webulator/400 alias (both servers ship a default Webulator alias named /WWW5250/). To add a new alias, you can run the WRKWWWALS command.
3. Run the CHGWWWCFG command to enable Webulator/400:

   a. Set the Directory Based Configuration File Field (ACCGBLFILE):

   The easiest way to set up Webulator/400 is to use the sample Directory Based Configuration file that is shipped with the Webulator/400 product. You can use this file by setting the directory-based configuration field to /WWWSERV/CFG/WBLMACC.CFG.

   Note: If you already have an entry in the directory-based configuration field, it is recommended that you temporarily replace your current configuration file with the Webulator/400 sample Directory Based Configuration file. This allows you to test and get familiar with the functionality of Webulator/400. After you feel comfortable with its functionality, you can modify your existing Directory Based Configuration file to include the desired new entries for Webulator/400.

   Note: All Webulator/400 configuration values are set to their default values when using this file.

   b. Set the Webulator/400 User File Path Field (WBLUSRFILE):

   You must set the path of the Webulator/400 user file if you plan to take advantage of the automatic sign-on capabilities of Webulator/400. The easiest way to set up Webulator/400 to use automatic sign on is to reference the sample user file that is shipped with the Webulator/400 product. This file can be used by setting the Webulator/400 User File Path field to /WWWSERV/CFG/AUTH/WBLUSR.CFG.

   Note: This file is shipped with no entries since it is not possible to know the USERIDs or passwords on your system. You must add your own entries to this file using the WRKWBLUSR command before it is useable by Webulator/400.

   Note: It is beneficial to use the sample user file because of the authority settings that are shipped with this file.

   c. Set the Maximum Webulator/400 Sessions Field (WBLMAXSSN):

   You can optionally set the maximum number of simultaneous Webulator/400 sessions that are allowed. If you do not specify this entry, a default of 20 sessions is used.

   d. Set the Disable Webulator/400 Field (DISABLEWBL):

   You must set the Disable Webulator/400 entry to *NO to configure the server to automatically start Webulator/400 during its startup process.

4. Evaluate the Sign-on Method:

The sample Directory Based Configuration file (/WWWSERV/CFG/WBLMACC.CFG) that is shipped with Webulator/400 contains a root directory entry with a sign-on method of sign-on display.

Note: "Sign-on display" was chosen because of its ease of setup, but it does have some potential security considerations. If you are not comfortable having a sign-on display available even for a short period of time, you should change the sign-on type to a different value before restarting or reconfiguring Web Server/400. This value can be set through the sign-on method field through option 10 of the WRKWWWDIR command or directly through the CHGWBLCFG command.

Valid Sign-On Methods

Method	Result
SIGN-ON DISPLAY	The standard AS/400 sign-on display is shown.
USEAUTHENTICATION	Presents a dialog box asking for your USERID and password. These values are encoded before being sent to the server, using the base64 encoding of MIME(UUENCODED).
USER	Automatically logs onto a specific user profile. The first display for that AS/400 user profile is shown.
DISABLED	Prevents logging-in. An error message is sent to the browser when you try to access the URL.

5. Add Additional Webulator/400 Directory Entries

   The sample Directory Based Configuration file (/WWWSERV/CFG/WBLMACC.CFG) that is shipped with Webulator/400 contains only the Webulator/400 root directory entry and no child directories. You can optionally add more Webulator/400 directory entries by running the WRKWWWDIR command.

   Note: All Webulator/400 directory entries must start with /*META/WEBULATOR/. Therefore, the Webulator/400 root directory is always named /*META/WEBULATOR/. If you want to add a new directory entry below the Webulator/400 root, you can add an entry such as /*META/WEBULATOR/STATUS/.

   Note: By creating additional directories, you can have multiple URLs that have different characteristics (such as which user is automatically signed on).

6. Check AS/400 Virtual Terminals:

   Verify that the AS/400 system value QAUTOVRT is at a large enough number so that Webulator/400 can automatically create additional virtual terminal devices if needed.

7. Start your Web server:

   Start either Web Server/400 or Commerce Server/400. It starts Webulator/400 during its startup process.

   Note: If your server is already started, you can run the Set WWW Configuration Values (SETWWWCFG) command, which reconfigures the server, which, in turn, starts Webulator/400. Note that in the future when you

reconfigure Webulator/400, the new configuration values take effect for all new sessions only.

5.6.1.2 Starting Webulator/400

The following steps help you determine if Webulator/400 has been configured properly for access. Having started the server, you must also be running a Web browser on a workstation connected to the AS/400 system using TCP/IP.

1. Access the Webulator/400 Root URL through:

   `http://your.system.name/WblAliasName/`

   where:

   your.system.name Your AS/400 system's TCP/IP host name or IP address.

   WblAliasName Is an alias whose SRCTYPE is *WEBULATOR.

   Note: You can view all of your current aliases by running the WRKWWWALS command.

   Example

   Assume that your HOSTNAME is www.xyz.com and your *Webulator alias name is /WWW5250/. The name of your Webulator/400 root URL is:

   `http://www.xyz.com/WWW5250/`

2. If you added additional Webulator/400 directory entries using the WRKWWWDIR command, you should try to access their URLs also.

   Note: To determine how to access child URLs, you must look at the names of your directory entries. Since all Webulator/400 directory entries must start with the Webulator/400 root directory name of /*META/WEBULATOR/, you can ignore all entries that do not meet this criteria.

   Example

   Assume that you have the following Webulator/400 directory entries:

   /*META/WEBULATOR/

   /*META/WEBULATOR/STATUS/

   The first entry is the Webulator/400 root directory and the second directory is a child below the root directory. To determine the name of the URL to access this child directory, you need to strip off the Webulator/400 root component from the directory name. This leaves us with the name of the child directory entry (STATUS/). You must append the name of the child directory entry to the URL of the Webulator/400 root to get the correct URL for the child directory entry. Assuming that the root directory URL is:

   `http://www.xyz.com/WWW5250/`

   the URL for the child would be:

   `http://www.xyz.com/WWW5250/STATUS/`

3. Check for expected configuration values to ensure that the Webulator/400 URLs are using the configuration values you are expecting.

   Note: If any are not, they are either using a default value or are inheriting a value from one of their parent directories. If a directory entry is inheriting an undesirable configuration value, you must define a new value in the current

directory to override the inherited value. These values can be set through options 8, 9, and 10 of the WRKWWWDIR command or directly using the WRKWBLROW, WRKWBLPRS, and CHGWBLCFG commands.

5.6.2 Webulator/400 Sign-On Methods

The following section describes the detail for configuring the sign-on parameters. The parameters are:

1. **Method**. This can be one of User, Screen, UseAuthentication, or Disabled.
   - **User** causes the system to automatically sign on with a specific user name. This is the most secure way to configure Webulator/400 because you have control over what AS/400 user profiles people are allowed to sign on with. If this is specified, it must be followed by a UserName, which is described later.
   - **Screen** causes users to be presented with an AS/400 sign-on display. They may type in the AS/400 user and password they want to use to sign on. This is less secure because the AS/400 user and password are sent over the TCP/IP network between the browser and the server. It is recommended that this only be used over internal networks unless secured with SSL and Commerce Server/400. If Commerce Server/400 is configured to use SSL for Webulator/400 sessions, all data, including user IDs and passwords is encrypted.
   - **UseAuthentication** uses authentication information sent from the browser as the AS/400 user and password. This is slightly more secure than Screen because the user and password are sent UUENCODED (while UUENCODED text is not as obvious as “clear” text, it is not a form of encryption and it is easy to “decode” it). You can also combine this with access control to limit the user IDs and passwords that can be entered for a URL. This changes the meaning of Web Server/400 required entries; any users listed are expected to be valid user profiles instead of entries in a user file.
   - **Disabled** disables Webulator access in the current directory.

   Note: If no entry is provided for a directory, the parent directory′s value is inherited. If the root directory has no entry, the default, which is Disabled, is inherited.
2. **UserName**. This must be present if User was specified previously. It is the AS/400 user that is signed on. It must have a corresponding entry in the Webulator/400 user file.

   Note: Only one entry may exist in a directory section. If more than one entry is found, the last one is used.
   - **AllowSignonOverride**. This is only applicable if the method is set to User or UseAuthentication. If set, Webulator/400 allows sign-on display fields to be overridden by URL options.
   - **IgnoreSignonOverride**. This is only applicable if the method is set to User or UseAuthentication. If set, Webulator/400 does not allow sign-on display fields to be overridden by URL options.
3. **Webulator User Entry**. Specifies a user name and password that is used for automatic sign on by Webulator/400.
   - **Name**. This is the AS/400 user profile that users are signed on as.

- **Password**. This is the AS/400 password that is entered for users. Because this is not encrypted, you should protect this configuration file with OS/400 authority. Only the user who changes the file and the user who starts the server needs authority to the file. The server user profile should **not** have authority to this file.

  Note: Multiple entries may exist in the Webulator/400 user file.

5.6.3 Using Webulator/400

This section describes how to use the Webulator/400 from I/NET.

5.6.3.1 URL Syntax

You can specify options on the query string of the URL that initializes a Webulator/400 session to change the behavior of that interactive Webulator/400 session. The query string is specified on the initial URL after the complete Webulator/400 path has been specified. It starts with a ? followed immediately by the list of query string value pairs separated by an &. Assuming that your Webulator/400 session URL is:

```
www.xyz.com/www5250
```

the syntax is similar to this:

```
www.xyz.com/www5250?KEYWORD1=VALUE&KEYWORD2=VALUE
```

Spaces are not allowed inside the query string; you must substitute a + for all embedding spaces. The following query string keywords are available:

PGM Allows you to specify a value for the initial program to run.
MENU Allows you to specify a value for the initial menu to run.
LIB Allows you to specify a value for the initial library.
FIELD1 Allows you to specify the value that is returned to the AS/400 system for the first input capable field that is encountered. When this keyword is specified, Webulator/400 simulates pressing ENTER for all displays up to and including the first display with an input field. It is your responsibility to ensure that the first display with an input field can properly handle the value being passed to it.

Restrictions: The Sign-on Method value must be one of:

- USER ALLOWSIGNONOVERRIDE or
- USEAUTHENTICATION ALLOWSIGNONOVERRIDE

for Webulator/400 to recognize the PGM, MENU, or LIB query string keywords. There are no restrictions on using the FIELD1 keyword.

Considerations: Care should be taken before enabling the ALLOWSIGNONOVERRIDE feature. When this feature is enabled, it allows you to override any of the initial sign-on values based on an HTML link.

This may be useful if you want to create a series of HTML links to some of your most popular applications but you do not want to create separate user profiles that have those applications as their initial program. In this case, you can use the query string keywords to override which application is called based on the HTML link regardless of the user profile that was used to sign on.

Note: This flexibility does not come without additional security considerations. If you enable the ALLOWSIGNONOVERRIDE feature, any user that has access to that URL can also override the query string keywords to run any program that

their user profile has access to. Please keep this in mind before enabling this feature.

There are no security considerations associated with the FIELD1 query string keyword. The inclusion of this keyword on the query string is the equivalent to the user typing it in themselves. They are not able to get access to any additional AS/400 programs or functions by using this feature.

You may find this feature useful if you have a URL with a sign-on method of USER that brings up an AS/400 menu as its first display. In this case, you can create a series of HTML links to the same URL but with a different query string FIELD1 value that automatically selects the appropriate menu option for the user.

Examples

The following examples assume that your Webulator/400 URL is:

```
/www.xyz.com/www5250
```

If you want the user to automatically start the program called MYPROG in the MYLIB library, use the following URL:

```
www.xyz.com/www5250?PGM=MYPROG&LIB=MYLIB
```

If you want the user to automatically take option 1 from the menu called MYMENU in the MYLIB library, use the following URL:

```
www.xyz.com/www5250?MENU=MYMENU&LIB=MYLIB&FIELD1=1
```

You can also use the FIELD1 keyword to call an initial program that accepts dynamic parameters. Please be aware that to do this, you must give the user access to a command line that may not always be desirable for security reasons. Assume that the program MYPROG accepts parameters. You can call this program with the following URL:

```
www.xyz.com/www5250?FIELD1=CALL+MYLIB/MYPROG+PARM('PARM1'+'PARM2')
```

5.6.3.2 Keyboard Plugin

There is a browser plugin that provides keyboard support for Webulator/400. Because keyboard support is implemented as a Netscape plugin, it is platform specific and must be tested with each browser. The supported browsers are:

- Windows 3.x
  - Netscape Navigator 2.0
  - Netscape Navigator 3.0
- Windows 95/NT
  - Netscape Navigator 2.0
  - Netscape Navigator 3.0
  - Microsoft Internet Explorer 3.0

To complete the installation of the plugin, you must:

1. Ensure the server code is either Webulator 1.1a or you must install a Webulator PTF.

   Note: You can check your Webulator version by typing:

   ```
   DSPDTAARA DTAARA(WWWSERVER/VERSIONWBL)
   ```

   You can get the PTF from:

   ```
   ftp://ftp.inetmi.com/pub/webulate/wbl110001.txt
   ```

2. Download the plugin. You need to download different files for Windows 3.1 or Windows 95. The download can be found at the following URL:

   `http://www.inetmi.com/pubs/webulate/keyplug.htm`

3. Follow the instructions provided in the documentation at the preceding site and with the downloaded files.

   Note: You must close your browser for the plugin to be recognized.

4. Configure Web Server/400 as described in the documentation.

Using the Plugin: When you enter a URL to send your browser to a Webulator session, add the query string KEYBOARD=Y to activate the plugin. An example URL might be:

`host/www5250?KEYBOARD=Y`

For usability, it is recommended that you enable JavaScript for any Webulator URLs for which you plan to use the keyboard plugin. This attempts to automatically set the keyboard focus to the correct form control, reducing your need to use the mouse.

The keys F1 to F24 work as normal (press shift to use F12-F24). To cause the AS/400 system to roll up or down, hold the Control key while pressing the Page up or Page down key. Also, when in a multi-line edit control (text area), you must hold down the Control key for the Enter key to be recognized. Press the Escape key for attention and hold the Shift key down while pressing Escape for system request.

You see a combo-box, a push button and the Webulator keyboard icon when using the plugin. The combo-box contains the virtual keys you have defined for the current URL. You can still limit the keys available to a user or change descriptions through this mechanism.

Troubleshooting: There are several steps to configure and install the plugin and if any are not correct, it will not work. Fortunately, once you have the plugin correctly installed for a given browser, you should not need to worry about it in the future. Possible problems might be:

- The browser does not show the plugin in Help|About menu.
  - Make sure you *completely* shut down the browser and restart it. If you leave even one browser window open, it does not recognize the plugin.
  - Make sure you have the correct version of the DLL. If you are using a 16-bit browser, you must have the plugin for Windows 3.x. If you are using a 32-bit browser, you must have the plugin for Windows 95/NT.
  - Check to see if your browser is on the list of supported browsers. Because this plugin is platform-specific, it does not work with all browsers.
  - Make sure you copied the DLL file into the proper directory. Problems have been seen with Netscape Navigator 2.0 recognizing any plugin when installed on a network drive (in this case, install the browser to a local drive).
- The browser says ″Plugin Not Loaded″.
  - The browser may not be able to find the PLUGIN.WKY file. Make sure it is in the document root and that it is available to the browser. You can check the access log to see if the server was able to successfully send it.

- The content type may not have been set correctly. Use WRKWWDIR to check the entry.

- Everything looks fine, but pressing a key just elicits a beep.
  - You probably have loaded the beta version of the plugin, but are still running against the test version of PTF WBL110001. Get the latest PTF installed on your AS/400 system. The beta version of the plugin is only compatible with the release version of PTF WBL110001.

5.7 Application Development with Webulator/400

There are a few things you should keep in mind when preparing to include your AS/400 application on the Internet. They all involve the access and availability of your system and its objects to the general Internet public. Keep in mind that this applies only if you are planning to allow access outside your business (through the Internet, or any other means). If you are maintaining a closed Intranet system, you can follow your normal security precautions.

First, you probably want to restrict users who sign on through Webulator/400 to the applications that you have selected for their use. This means that the user should not be given a command line. The command line allows users to enter commands, including commands that you may not want them to execute, such as the CALL command to call a program or STRPASTHR to access another system. At minimum, the SNDMSG command in the wrong user′s hands can be a real nuisance. This includes the availability of the command line on some IBM provided displays. An operation as simple as the WRKJOB command (to allow the user to view and affect aspects of their individual job), while automated as part of a menu, still has a command line associated with it.

Inquiries and simple data entry that is run through a verification and authentication process are probably the best applications for Internet access. These applications allow users the ability to view information about your company and its products, as well as enter limited information to order products or request more detailed inquiries. They also allow for the entry of order requests that can easily be followed up or verified after the fact. If you have chosen to perform automatic sign on or user authentication sign on for the user, you probably do not want them to get back to a real AS/400 Sign On display. Access to the Sign On display defeats the work you have done in restricting access to the USERID that you have defined for Web access. It also gives the user the opportunity to begin “guessing” the USERIDs and passwords on your system. As a result, it is best not to include an option to sign off from your menus (usually option 90) and application displays available to the public. Keep in mind that this also includes the ability to sign off through help panels, whether they were written for your application or are supplied by IBM as generic help for the AS/400 system.

5.7.1 Signing Off

This section describes how to close a session when using Webulator/400.

5.7.1.1 Closing Confirmation

Webulator/400 allows you to determine if the user should be given a Confirmation display after they press the Close Session command button. You may want to set this value to YES if you want to protect users from accidentally pressing the Close Session button and losing their session data. The Closing Confirmation display allows the user to return back to the previous display or to continue closing. You may want to set this value to NO if you feel the likelihood of your users accidentally pressing the Close Session is remote or you find that they are less likely to close a session because of the extra step involved.

5.7.1.2 Termination URL

Webulator/400 allows you to specify a Termination URL that control is transferred to when the user ends a Webulator/400 session. This allows a transparent way for the user to be returned to a meaningful URL when they are finished with their Webulator/400 session. In addition to the URL, a description must be entered that is used as the link text to your termination URL on all Webulator/400 error messages.

5.7.1.3 Non-Programmed Sign Off

If your program does not permit a user sign off, this presents us with a reasonable question, ″How do I end the application and the job when the Webulator/400 session is closed by the user?″ Interestingly, the answer to that question is to issue the “SIGNOFF” command. This is not available to the user, but from within the application when a display error occurs identifying that the session has been closed. Basically, you should monitor for errors whenever a display is written to the 5250 display from your program. This can be done using MONMSG in CL programs following any SNDRCVF statements. The contents of the EXEC parameter are simply the command “SIGNOFF”. An illustration of this procedure, as well as RPG and COBOL logic, can be found in the following examples:

```
PGM
DCLF FILE(DSPFILE)
.
.
.
SNDRCVF RCDFMT(SCREEN1)
MONMSG MSG(CPF0000) EXEC(SIGNOFF)
.
.
.
```

Figure 91. CL Program Example

```
.....CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComment
.
.
.
     C                   EXMFT SCREEN1               99
     C           *IN99   IFEQ '1'
     C                   MOVEL'SIGNOFF' SGNOFF  7
     C                   Z-ADD7         SGNLEN 155
     C                   CALL 'QCMDEXC'
     C                   PARM SGNOFF
     C                   PARM SGNLEN
     C                   END
.
.
.
```

Figure 92. RPG Program Example

```
ENVIRONMENT DIVISION.
.
.
.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT DISPLAY-FILE
             ASSIGN TO WORKSTATION
             ORGANIZATION IS TRANSACTION
             FILE STATUS IS WS-FILE-STATUS.
.
.
.
DATA DIVISION.
.
.
.
WORKING-STORAGE SECTION.
01  WS-FILE-STATUS            PIC X(2).
01  WS-SIGNOFF-VARIABLES.
    05 WS-SIGNOFF-CMD          PIC X(7) VALUE "SIGNOFF".
    05 WS-SIGNOFF-LEN          PIC 9(10)V9(5) COMP-3 VALUE 7.
.
.
.
PROCEDURE DIVISION.
.
.
.
        WRITE DISP-RECORD FORMAT IS SCREEN1.
        READ DISPLAY-FILE.
        IF WS-FILE-STATUS IS NOT EQUAL "00" THEN
           CALL "QCMDEXC" USING WS-SIGNOFF-CMD WS-SIGNOFF-LEN,
           STOP RUN.
.
.
.
```

Figure 93. COBOL Program Example

These examples show the logic that you may want to include in your application to ensure the closeout of the application and completion of the job. Obviously,

other steps that may be required to complete or close a transaction in your application are not shown here. The contents of the QDEVRCYACN system value also determines how the application program is informed of the session termination and may inhibit the return of error indications to the program.

5.7.2 Existing Applications

Webulator/400 allows you to choose if the AS/400 display data should be sent as HTML preformatted text or as an HTML table.

5.7.2.1 HTML Preformatted Text

HTML preformatted text is the default method. Webulator/400 sends preformatted text when the Tables Enabled configuration value is set to NO.

Preformatted text has the following characteristics:

- It instructs the browser to use a fixed width font. This means that all characters (including spaces) on the display are the same width. This results in the characters going across a line have the same spacing as in a traditional emulation program.
- It produces the smallest size HTML file.
- It is fast for the browser to render on the display.
- The width of input fields and submit buttons are larger than the number of characters that they hold. This means that the columns of any line that have input fields or submit buttons do not properly line up with columns on a line that have only output fields. The effects of this vary depending on the composition of the display. It may range anywhere from a small nuisance to having columns lining up with the wrong headings.

5.7.2.2 HTML Tables

Webulator/400 allows you to specify that all 5250 screen data should be included inside of an HTML table. Webulator/400 sends HTML tables when the Tables Enabled configuration value is set to YES.

HTML Tables have the following characteristics:

- Tables are able to keep track on the largest number of pixels that is needed by any column. This means that columns that have either input fields or submit buttons have more pixels allocated to them than columns that only have output fields. The result is that tables guarantees that all fields that start in the same column are properly aligned but there may be extra space between fields to compensate for the different widths of the various HTML elements used.
- Tables allow for the specification of a background color different from the body of the form. This allows for reverse image attributes to be honored.
- Tables by default use non-fixed width fonts. This makes the 5250 screen look quite a bit different than when viewed using a traditional emulation program. You can change the font that tables use by setting the Table Font Name configuration entry. You may need to experiment with the font name to ensure that all of your target browsers support it. You should be safe setting this value to COURIER.
- It produces much larger HTML files.
- It is slower for the browser to render on the display.

- It requires a browser that supports HTML tables.

Recommendations

You are better off using preformatted text if your 5250 screens do not use tabular data. It produces smaller files that are quicker to display and is supported by more browsers than tables. If your programs use tabular data and you are not satisfied with the way preformatted text aligns your columns, experiment with tables.

5.7.3 DDS HTML Support

This section describes the DDS HTML support considerations when using Webulator/400.

5.7.3.1 OS/400 Version Considerations

This section describes usability considerations applied for specific version of OS/400.

V3R1/V3R6 Considerations: Browsers are context sensitive to the data stream that they receive. In particular, they assume that all less than signs (<) are a start of an HTML keyword and do not display any characters until it reaches the next greater than sign (>). This causes display problems for any AS/400 data stream that contains a < that is not an HTML keyword. Because of this, Webulator/400 recognizes a subset of HTML keywords and escapes all < characters (change the < to a different set of characters) that are not part of a recognized HTML keyword. This has two sets of ramifications:

1. All < characters that are not a part of a recognized HTML keyword are displayed properly on Webulator/400 browser displays.
2. It is possible to embed certain HTML keywords into the 5250 data stream and have them interpreted by the browser. This means that you can include such things as images or links to other URLs inside of your displays. Please note that the number of supported HTML keywords is limited.

The following HTML keywords are passed to the browser intact:

```
<A ... > - Start of hyperlink anchor
</A> - End of hyperlink anchor
<IMG ... > - Image hyperlink
```

Note

Care should be taken before embedding HTML fields in your 5250 screens. You should take into account the following constraints when designing displays that will contain embedded HTML keywords:

- You may want to make output fields that contain HTML keywords to have a display attribute of Nondisplay (DSPATR(ND)). If you do not do this, users viewing the display with an application other than Webulator/400 see the HTML keywords as text and not as embedded images or links.
- HTML keywords usually require a large number of characters. You may run into problems fitting them on your display.
- You may run into some spacing problems. There is sometimes little to no correlation between the amount of space needed for the keyword and the amount of space needed for the resulting image or link.

V3R2 Considerations: With the release of V3R2, IBM has introduced the HTML DDS keyword that allows AS/400 applications to embed any HTML tag into the 5250 data steam. This method of embedding HTML offers the following advantages:

- HTML keywords do not occupy any display space. You can add HTML anywhere on the display no matter how crowded it is.
- All HTML tags are supported.
- HTML keywords are not passed to normal ″green screens″ or emulation programs.

Note

One limitation imposed is that HTML keywords cannot be included in subfile records. Please refer to the following publications for more information on the HTML DDS keyword:

- *AS/400 DDS Reference*, SC41-3712-01
- *AS/400 Application Display Programming*, SC41-3715-01

Webulator/400 supports the use of the DDS HTML keyword in the same way as Workstation Gateway. Further details for the use of the DDS HTML keyword can be found in Section 5.4.3, “DDS HTML Support” on page 187. However, Webulator/400 has some additional configuration items that can further enhance your application.

5.7.3.2 Emulating Color and Monochrome Displays

The Terminal Color configuration element lets you emulate either a color or monochrome display. If you emulate a color display, Webulator/400 uses the colors defined in the 5250 data description. If you emulate a monochrome display, Webulator/400 displays all text in a single color and displays high intensity characters as bold. The monochrome display configuration requires an HTML 2.0 compliant browser to properly view Webulator/400 displays. You can optionally define the text color by configuring the Monochrome Text Color.

Note: If you do this, you need a browser that supports extensions to HTML 2.0.

If you emulate a color display, Webulator/400 uses the colors defined in the 5250 Data Description Specifications (DDS). While this is more visually appealing, it does require a browser that supports extensions to HTML 3.0. You can optionally choose to define one or all of the colors by configuring the Screen Text Colors.

5.7.3.3 HTML Headers

- The standard Webulator/400 header contains the following HTML entries:

```
<HTML>
<HEAD>
<TITLE> (Text From First Output Field) </TITLE>
</HEAD>
<BODY TEXT=Monochrome Text Color
      BACKGROUND=Background Image
      BGCOLOR=Background Color
      ONLOAD=″SetFocus()″>
```

- Webulator/400 allows you to replace the standard header with either an HTML or plain text custom header. This feature allows you to control what

appears above the 5250 terminal data in the HTML form. For example, you can include a link to your company's home page or an address for sending e-mail. If you choose to create a custom HTML header file, it must include the following tags:

```
<HTML>
<HEAD>
<TITLE> (Title Text) </TITLE>
</HEAD>
<BODY (Optional BODY Attributes)>
(Optional BODY elements)
```

Note: All embedded file references must be qualified to the Web Server/400 document root by including a leading slash (/) in the file name. Failure to do this may result in broken links.

Note: Webulator/400 may insert some optional BODY attributes into your custom header based on your configuration settings. For example, if the session has a Background Color Value configured, but the custom header does not have a BGCOLOR attribute set, Webulator/400 inserts the proper attribute to honor the configuration value. If the custom header has a BGCOLOR attribute, Webulator/400 uses its value instead.

Example

The following example header places a link to a home page and a mail address for the local WebMaster.

```
<HTML>
<HEAD>
<TITLE>Webulator/400 Demo</TITLE>
</HEAD>
<BODY>
<A HREF="/home.htm">Home</A>
<A HREF="mailto:webmaster@xyz.com">WebMaster</A>
<P>
```

- If Send JavaScript is enabled, Webulator/400 automatically inserts ONLOAD="SetFocus()" into the form BODY tag. If your custom HTML header also includes a JavaScript function that must be called on the OnLoad event, you must include your function name in the BODY tag and a call to the SetFocus() function at the end of your function.

Example

Suppose that you have a JavaScript function called InitForm() and you want it to be called during the OnLoad event. Your custom HTML header should look similar to this:

```
<HTML>
<HEAD>
<TITLE>Webulator/400 Demo</TITLE>
</HEAD>
<BODY OnLoad="InitForm()">
<SCRIPT LANGUAGE = "JAVASCRIPT">
<!---
 function InitForm() {
     (some JavaScript statements)
     SetFocus();
     return;
 }
 // --->
 </SCRIPT>
```

- If you choose to create a custom plain text header file, Webulator/400 uses the standard Webulator/400 header followed immediately by the plain text header in a "Preformatted" text section.

5.7.3.4 HTML Footers

- The standard Webulator/400 header contains the </BODY> and </HTML> entries.
- Webulator/400 allows you to replace the standard footer with either an HTML or plain text custom footer. This feature allows you to control what appears below the 5250 terminal data in the HTML forms. For example, you can include a link to your company's home page or an address for sending e-mail. If you choose to create a custom HTML footer file, it must include the following tags:

  ```
  (Optional BODY elements)
  </BODY>
  </HTML>
  ```

  Note: All embedded file references must be qualified to the Web Server/400 document root by including a leading slash (/) in the file name. Failure to do this may result in broken links.

  Example

  The following example footer places a link to a home page and a mail address for the local WebMaster.

  ```
  <P>
  <A HREF="/home.htm">Home</A>
  <A HREF="mailto:webmaster@xyz.com">WebMaster</A>
  </BODY>
  </HTML>
  ```

- If you choose to create a custom plain text footer file, Webulator/400 inserts the plain text footer in a "Preformatted" text section followed immediately by the standard Webulator/400 footer.

5.7.4 Additional Customization of Webulator/400

The *Webulator/400 User Manual* has a more detailed discussion of the following customization/configuration items:

- AS/400 terminal size
- Screen background
- Virtual keyboard buttons
- Input field characteristics
- Output characteristics
- Screen text colors
- Graphical menus
- Converting keywords to buttons (identifying screen keywords)
- Embedding HTML in the 5250 data stream
- Default configuration values
- Sample directory based configuration files
- Webulator/400 commands

5.7.4.1 Reconfiguring Webulator/400

You may dynamically reconfigure Webulator/400 to use the latest configuration values at any time the Web Server/400 is active by running the Set WWW Configuration Values (SETWWWCFG) command or by setting the Update executing RPs (UPDATE) parameter on any of the configuration commands to *IMMED.

It is important to note that when you reconfigure the Webulator/400 settings, the new settings take effect for all future sessions and do not change any currently active Webulator/400 sessions. This is done as a way to prevent the Webulator/400′s appearance and functionality from changing from one display to another.

Note: The user profile of the person starting or reconfiguring the server must have read access to all configuration files. The user profile of the server does not need read access to any configuration files.

If the configuration commands are used to change the configuration, the user that runs them must have write access to the configuration files, as well as the temporary (*.tmp) and backup (*.bak) that are created by those commands.

5.8 Hints for Using Webulator/400

The following list describes some of the known limitations and offers possible ways to work around them:

1. Nothing happens when the Enter key is pressed. Some browsers recognize the Enter key and return the form data only if there is one, and only one, input field available on the form. If there are none or multiple input fields, the user must press the Enter submit button instead.
2. Nothing happens when a Function key is pressed. Since the browser is handling the user input functions, it traps the Function keys for its own use and not for Webulator/400. This means that when you press one of the function keys (for example, F1), it either ignores you or performs a browser function. Neither one of these is what you want. You must press the browser submit button that corresponds to the desired Function key.
3. Columns do not line up. By default, Webulator/400 generates preformatted text display data, which ensures that output fields line up properly.

Unfortunately, column data does not line up if there is either an input field or a submit button on the same line. This happens because the width of input fields and buttons are not the same as the width of an output field. If the columns are not lining up because of a Submit button that Webulator/400 created, you may be able to modify your keyword parsing configuration to prevent Webulator/400 from creating the Submit button.

Note: You can have Webulator/400 generate HTML tables instead of preformatted text. Tables guarantee that columns line up by inserting extra space between fields. Please see Section 5.7.2.1, “HTML Preformatted Text” on page 207 and Section 5.7.2.2, “HTML Tables” on page 207 for more information. Please note that you must have a browser capable of displaying tables for the display to appear.

4. No input fields have focus. Webulator/400 can generate a JavaScript routine that automatically positions the cursor in the correct field. Please see the JavaScript example on page 210 for more information.

   Note: You must have a JavaScript capable browser for the cursor to be positioned correctly. If you do not have Send JavaScript enabled or your browser does not support JavaScript, you are dependent on the browser to select an input field. Some browsers automatically select the first input field, while several do not select any fields but instead require the user to select a field before being able to enter data.

5. Field level prompting does not work. Webulator/400 can generate a JavaScript routine that automatically returns to the AS/400 system the location of the last input field that had focus. Please note that you must have a JavaScript capable browser for the cursor position to be reported to the AS/400 system correctly. If you do not have Send JavaScript enabled or your browser does not support JavaScript, you must take an extra step to get field level prompting. You can inform Webulator/400 of the field position you want to return to the AS/400 system by typing a configurable string (the system default is a ?) in the field and pressing a non-enter Submit button. When Webulator/400 recognizes the Field Level Prompting string, it strips it out of the 5250 data stream and returns the cursor location of the current field.

6. Browser does not properly display Webulator/400 displays. Various browsers support different levels of the HTML specifications. Browsers should ignore HTML tags that they do not understand. Unfortunately, some browsers incorrectly interpret these unknown tags. The results can vary from incorrect colors to improper formatting. Webulator/400 requires an HTML 2.0 compliant browser to run most configurations and a browser that supports HTML extensions to use some of the more advanced configuration options.

Browser Requirements

The following configuration elements require a browser that supports extensions to HTML 2.0:

- Body background color
- Body background image
- Monochrome terminal text color

The following configuration elements require a browser that supports extensions to HTML 3.0:

- Color terminal
- Color terminal text color

The following configuration elements require a browser that supports HTML tables:

- Tables enabled
- Table font name
- Table width

The following configuration element requires a browser that supports JavaScript:

- Send JavaScript

You may have also included some HTML 3.0 or HTML extensions in your custom header or footer file that may cause browser problems. Make sure you understand the ramifications that the HTML tags may have on older browsers before you include them in a custom header or footer file.

7. Unable to view all AS/400 output messages. Since the browser is only able to receive data from the AS/400 system after a user submits the HTML form, it is not possible for Webulator/400 to continually send display updates to the browser. Webulator/400 attempts to buffer all output messages and send them with the next display but it is not always able to do so. Webulator/400 clears all display data (seen and unseen) whenever it encounters a Clear Unit command in the 5250 data stream. A Clear Unit command is inserted into the 5250 data stream by various display I/O commands such as the EXFMT command in RPG.

   Note: Webulator/400 is also unable to support AS/400 break messages. AS/400 break messages are messages that interrupt the current display and require some sort of response (at a minimum, an Enter key) to return to the previous display. Break messages cause Webulator/400 to be out of synchronization with the AS/400 system and cause unpredictable results.

8. Extra displays are occasionally sent to the browser. Some host programs send a write/read screen I/O request followed immediately by a cancel read operation. The end result of these operations are the same as a write request. These requests are handled seamlessly by traditional emulators because of their constant communication link with the AS/400 system. Webulator/400 has a little more difficulty handling these I/O requests. Webulator/400 verifies that no Cancel requests have been generated by the AS/400 system before sending a display to the browser. Unfortunately, there is no guarantee that the Cancel request is received before the display is sent to the browser. This means that certain write/read/cancel requests get past Webulator/400's checking and are sent to the browser. If this happens, all you have to do is press a Virtual Keyboard button and the next display is sent to the browser.

9. AS/400 Cursor Control Keywords are ignored. Since Webulator/400 does not have control over how the Browser handles the user interface, the following DDS keywords have no effect:

   - CSRINPONLY - Cursor Movement to Input
   - Capable Positions Only
   - FLDCSRPRG - Cursor Progression Field
   - HOME - Home
   - MOUBTN - Mouse Buttons
   - MSGALARM - Message Alarm
   - SFLCSRPRG - Subfile Cursor Progression
   - SFLCSRRRN - Subfile Cursor Relative Record
   - Number
   - WRDWRAP - Word Wrap

10. Unable to perform Text Assist functions. Webulator/400 is unable to fully support programs that use the Text Assist functions (for example, OfficeVision/400™). Text Assist programs require more interaction with the AS/400 system than is possible using Webulator/400.

11. The Attention button does not always work. Care should be taken before allowing this button to be made available through Webulator/400. You must ensure that the user profile that is signed on for a session has an Attention program assigned to it. As long as it does, everything works fine. You can run into problems if a Webulator/400 user presses the Attention button and their user profile does not have an attention program defined for it. 5250 emulation programs are able to notify the AS/400 system that the Attention key was pressed and retain control of the current display until they are interrupted by the new attention program.

    Note: If there is no attention program to run, the 5250 emulation program continues to process the current display. Since the Web browser returns control to the AS/400 system when any submit button is pressed (including the Attention submit button), it must wait for a new display to arrive from the AS/400 system before allowing the user to interact with the display. If there is no attention program to generate a new display, the browser times out waiting for the new display and does not allow the user to continue with the display where the Attention button was pressed.

5.9 Other Implementation Tips

1. If you are going to make use of the HTML DDS keyword, most AS/400 programmers have never seen or used HTML. Rather than spend the hours working out the correct syntax, it is faster to use a graphical HTML editor. Such editors generate all of the HTML code, which you can simply cut and paste into your AS/400 application DDS file.

2. When debugging, set the number of clients sessions such that you only have a parent and child job. This makes finding the child job you want to debug easier.

5.10 HTML Gateway Comparison

The following brief list details some of the features/limitations you might want to consider when deciding whether to use Workstation Gateway or Webulator/400:

- The Webulator/400 has many more configuration items than Workstation Gateway, making it easier to use for the client. A quick look at the online documentation is available at the following URL:

  `http://www.inetmi.com/pubs/webulate/usrgdtoc.htm`

  This shows more than 20 configurable items relating to such things as:

  - Device capabilities
  - Screen appearance
  - Session limits
  - Access methods

- When using Webulator/400 with the I/NET Commerce Server/400, secure transactions can be made.

- The the Netscape plugin for Webulator/400 provides correct keyboard mappings for 5250 sessions viewed from the browser. For instance, the F12 key now provides the Cancel function (where appropriate). This enhanced functionality is not available with Workstation Gateway.

5.11 Further Information

This section presents you with additional information about using Workstation Gateway from IBM and/or Webulator/400 from I/NET.

5.11.1 Workstation Gateway

Workstation Gateway additional information.

5.11.1.1 Support Pages on the Internet

The support pages and FAQ for IBM′s Workstation Gateway can be found at the following URL:

`http://www.as400.ibm.com/ncc/webconn/htmlgate.htm`

This is a good site especially for late breaking news and PTF information about the Workstation Gateway, and is supported by the AS/400 team.

Another avenue for retrieving up-to-date news is Information APARs. At the time this document was printed, 5763-TC1 Information APAR II09450 contains the latest information. APARs can be found at the following URL:

`http://as400service.rochester.ibm.com/`

5.11.1.2 Code Snippets

As described elsewhere in this redbook, many business can use and are using the Internet and Intranet capabilities of the AS/400 system. In this redbook, we have tried to provide code snippets or examples that can be used to cut through the tangle of the Web and enhance your applications.

The following URL offers other examples and samples of code that have made implementing solutions simpler. These tools and techniques for untangling some of the ″mysteries″ of the Web were created by various groups within IBM′s AS/400 community.

http://as400.rochester.ibm.com/workshop/snippets/snippets.htm

It is anticipated that this site will be updated with new snippets as they become available:

5.11.1.3 Additional Publications

Further IBM material that might be of interest:

- *Tips and Tools for Securing Your AS/400*, SC41-3300-00
- *A Guide to the Internet Connection Servers*, SG24-4805-00

5.11.2 Webulator/400

Webulator/400 additional information.

5.11.2.1 Support Pages on the Internet

The Webulator/400 Frequently Asked Questions can be found at the following URL:

http://www.inetmi.com/products/webulate/wblfaq.htm

Other support for the Webulator/400 product can be found at the following URL:

http://www.inetmi.com/products/webulate/webulate.sht

Webulator/400 users also have the opportunity to join an electronic mailing list dedicated to discussions on Web Server/400 and Webulator/400. I/NET hosts a customer mailing list (List Server) that allows all participants to discuss their experiences with these products through e-mail. All messages sent to the mailing list are automatically forwarded to the e-mail addresses of the subscribers to the list. The sign-up page can be found at the following URL:

http://www.inetmi.com/cgi-bin/listreg

5.11.3 Additional Publications on the Web

URLs:

http://as400.rochester.ibm.com/
The AS/400 Home Page, a useful starting point
http://as400bks.rochester.ibm.com/
Contains the AS/400 Softcopy Library CD-ROMs online
http://www.developer.ibm.com/
The IBM Solution Developer Support Home Page
http://www.as400.ibm.com/nstation/infopage.htm
IBM Network Station: AS/400 Information
http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimerAll.html
A Beginner's Guide to HTML
http://www.netscape.com/comprod/mirror/client_download.html
Download Site For Netscape Navigator Products
http://www.microsoft.com/ie/download/
Download Site for Microsoft Internet Explorer 3.0
http://www.ncsa.uiuc.edu/demoweb/url-primer.html
A Beginner's Guide to URLs

5.12 HTML Gateway versus Other Methods

- Advantages
  - Until recently, AS/400 software developers often had to learn new languages, tools, and procedures in order to support the Web.
  - An HTML gateway allows existing AS/400 applications (whatever the language) to be run over the Web with *no code changes or conversion programs*.
  - DDS HTML keyword added allowing some display enhancements.
  - No retraining of staff who use functionally similar displays.
  - Workstation Gateway does not require HTTP server unless serving graphics, whereas Webulator/400 has to run on top of Webserver/400.
  - Hardware can be either “green screens” or Web browsers.
- Disadvantages
  - Useability
  - Performance versus CGI-bin and Net.Data
  - Security of data being transmitted
  - Synchronization of client session page with the browser display

5.13 Future Developments

This section describes how to better follow the enhancements for the Workstation Gateway and Webulator/400 in the future.

5.13.1 Workstation Gateway

Since, the Workstation Gateway server is not really an HTTP server, many of the foreseeable AS/400 developments will not impact the Workstation Gateway. Secure transactions using Workstation Gateway will not be available at the same time as secure HTTP. The best place to follow Workstation Gateway news is at the official Web site:

`http://www.as400.ibm.com/ncc/webconn/htmlgate.htm`

5.13.2 Webulator/400

To keep abreast of developments of the Webulator/400 product, it is recommended that you visit the Web site for the latest news:

`http://www.inetmi.com/products/webulate/webulate.sht`

5.14 Conclusions

To summarize, using a 5250-HTML gateway can give you the following benefits:

- Web access to thousands of existing AS/400 applications without re-coding.
- Existing displays can be modified to include graphics, text, and links using the new DDS keyword ″HTML″ (displayed on Web client only).
- Utilize existing tools and skills to develop new Web applications.
- Clients are no longer restricted to a particular emulator or operating system.

- Complex Web applications that need to save state through a series of steps can easily be done in DDS and 5250.
- Ability to run AS/400 applications using Web browsers running on multiple platforms.

Chapter 6. Further Enhancing Your AS/400 HTML Pages

This chapter describes the use of Java Applets and JavaScripts to further enhance your Web applications for better usability.

6.1 Java

Java is an object oriented programming language developed by Sun Microsystems. It was originally designed for programming consumer electronics. The language was designed to be “architecture neutral” so that it could run on the different computer chips used in various consumer electronic devices. But Sun soon realized that the language had the potential to become much more.

The architecture-neutral aspect of Java makes it ideal for programming on the Internet. It allows a user to receive software from a remote system and execute it on a local system, regardless of the underlying hardware or operating system. This is possible because of the interpreted nature of the language and the Java Virtual Machine.

Traditionally, the source code of a program is written in the programming language of choice and compiled into the machine code understood by a particular set of computer hardware. The Java compiler, however, does not generate machine code. Instead, it generates intermediate code called Java bytecodes. These bytecodes are interpreted by the Java interpreter, which executes the instructions on the particular hardware platform. The Java interpreter and run-time system are collectively called the Java Virtual Machine or JVM.

The bytecodes are what make Java programs portable. A program written in Java is compiled into bytecodes. These bytecodes can be transported across a network and executed on any system that implements a Java Virtual Machine.

6.1.1 Java Applets

An applet is a small Java program designed to be included in an HTML Web document. The HTML document contains tags that specify the name of the Java applet and its Uniform Resource Locator (URL). The URL is the location at which the applet bytecodes reside on the Internet. When an HTML document containing a Java applet tag is displayed, a Java-enabled Web browser downloads the Java bytecodes from the Internet and uses the JVM to execute the code from within the Web document. These Java applets are what enable Web pages to contain animated graphics or interactive content.

Because Java applets can be downloaded from any system, security mechanisms exist within the JVM to protect against malicious applets. The Java run-time system verifies the bytecodes as they are downloaded from the network to ensure they are valid bytecodes and that the code does not violate any of the restrictions placed on Java applets by the JVM. Java applets are restricted in what operations they can perform, how they access memory, and how they use the JVM. The restrictions are in place to prevent a Java applet from gaining access to underlying operating system or data on the system.

6.1.2 Java Applications

This section describes how to implement and serve Java applications.

6.1.3 How Do I Serve Java Applets from the AS/400 System

Once you have found or written an appropriate applet, you can use it on your site. There are three types of files available for downloading:

.java. files These are text files that contain the source code. These are not required if you have the `.class` files and do not want to modify the code.

.class files These are binary files that contain the compiled Java source code (bytecodes). Each applet may require several `.class` files.

Other files These are the various assorted files required to support the applet. If you have obtained your applet from the Web, the documentation should list these files. Make sure you download all of them.

To serve the Java applet from your AS/400 system, you can use either FTP or shared folders to place all of the `.class` and supporting files into an AS/400 directory on the IFS (Integrated File System). You need to use a directory in a file system that supports long file names, such as the Root on the IFS. No special directory permissions are required as long as that directory is already configured to allow access from the Internet. The HTTP server treats the applets the same as any other HTTP object such as `.gif` or `.html` files.

Note: You may also want to use the `QOpenSys` part of the IFS; this gives you the ability to use case-sensitive names for your Java applets.

Remember

To access the IFS using FTP, you need to change to naming format ″1″. To do this, issue the following command from your FTP client:

```
quote site namefmt 1
```

This works for Windows 95, but some clients might only need `site namefmt 1`. You can now use the IFS naming convention to access the directories.

Note: For more help on how to configure and use the FTP clients and servers on the AS/400 system, refer to *TCP/IP Configuration and Reference*, SC41-3420-04.

6.2 JavaScript

This section describes how to implement and serve JavaScripts.

JavaScript can provide logic within static HTML and also field-level validation. Be aware that if the browser does not support JavaScript or if the feature is disabled (the application CGI-bin), Net.Data should perform the validation.

6.2.1 What is JavaScript?

JavaScript is an HTML scripting language that allows HTML pages to interact with a browser. It was originally developed by Netscape as a way to extend the functionality of HTML. It is supported by Netscape Navigator 2.0 or later and Microsoft Internet Explorer 3.0. Various other browsers either support it or will support it in the near future. Browsers that do not support JavaScript should ignore the embedded JavaScript commands.

6.2.2 Webulator/400 and JavaScript

Webulator/400 can insert a small amount of JavaScript code into every HTML page that it generates. You can enable this support by setting the Send JavaScript configuration value.

Webulator/400 uses JavaScript to add two important usability enhancements. The first enhancement allows the browser to insert the cursor at the start of the correct input field on the display. It does this by calling a function during the onLoad event. This gives the user the ability to type into a field without first having to select it with the mouse. It also indicates to the user the first input field that may contain an error if the row and column are properly set by the AS/400 application. The second enhancement allows the browser to return the row and column position of the last input field that had focus. It does this by calling a function during the various onFocus events. This feature allows the user to have field-level prompting without typing a question mark (?) into the field.

6.2.3 Why Disable JavaScript?

There are a couple of reasons why you may want to disable the JavaScript support.

1. JavaScript is an interpreted scripting language that means the browser must perform extra work to follow the JavaScript instructions. Testing did not indicate that performance is noticeably slower when including the JavaScript functions. Depending on the equipment and browsers that you use, you may see different results and determine that the added functionality is not worth the price in performance.
2. JavaScript is relatively new and not all browsers support it. A browser should ignore any HTML tags (including JavaScript tags) that they do not understand. This is really why you would not want to use JavaScript in your Web applications. Of course, there is no guarantee that all browsers act appropriately. If your target audience uses ill-behaved browsers, you may want to disable these features.

6.3 Further Java/JavaScript Information

URLs:.

http://ncc.hursley.ibm.com/javainfo/wp.htm
Java for AS/400 - A White Paper

http://www.ibm.com/java/
IBM Networking Computing - Java Web Site

http://ncc.hursley.ibm.com/javainfo/hurindex.html
IBM Centre for Java Technology Development

http://www.javasoft.com/
Sun's Java Home page

http://www.dannyg.com/javascript/index.html
Danny Goodman's JavaScript Pages - provides many application excerpts

Recommended Reading:

JavaScript Bible, 2nd Edition published by IDG Books, ISBN 0-7645-3022-4.

Chapter 7. AS/400 Internet Technology Preview

This chapter gives you a preview for the future of the AS/400 system in the Internet world. This preview discusses some of the new implementations that will apply to the AS/400 system in future releases.

- AS/400 Firewall Technology
- Internet Connection Services/400
- SSL Support for ICCS/400
- AS/400 HTTP Server Enhancements
- Net.Commerce for AS/400
- Java on the AS/400 system

7.1 AS/400 Firewall Technology

Many companies are thinking of connecting their internal corporate networks to the Internet (and for good reason). There are potential rewards due to the increased visibility and the possibility for new types of applications.

However, many are rightly concerned about the security of their systems. For that reason, IBM is developing unique firewall technology that makes it easier than ever to safely attach a network containing AS/400 systems to the Internet.

A firewall acts as a ″chokepoint″ through which all traffic to and from the Internet flows. It prevents unwanted Internet traffic from entering your secure network, while selectively allowing users access to the Internet. Internal users can use browsers to access Web servers on the Internet and exchange mail with Internet users through the firewall, while TCP/IP access from the Internet can be selectively or entirely blocked.

7.1.1 AS/400 Firewall Benefits

- Concentrates security administration, enforcing I/T security policy and minimizing the opportunity for security configuration errors.
- Provides privacy by preventing internal network information form being accessed through the Internet.
- Logs traffic to and from the Internet, allowing network use and misuse to be monitored.
- Provides flexible firewall configuration, enabling support for various security policies. The administrator decides which services should be permitted and which should be blocked.

7.1.2 The AS/400 Advantage

Most security experts agree that it is best to run firewall functions on a separate processor from other system functions. In the past, that meant a separate system had to be introduced into the network. The unique AS/400 firewall technology is based upon an Integrated PC Server, which provides processor separation without requiring the introduction of a new system into the network.

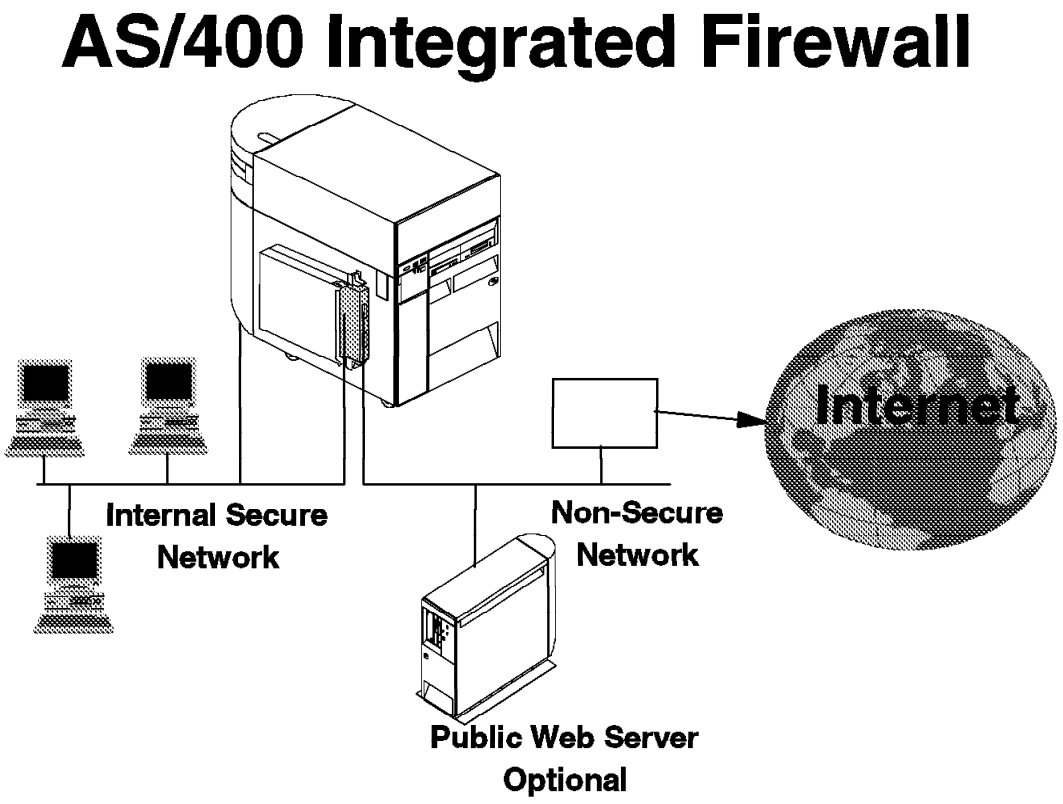


Figure 94. AS/400 Firewall Technology Preview

7.1.3 AS/400 Technology Advantages

Application programs and security programs are run on separate processor, which eliminates the possibility of processors interfering with each other. Compromised security programs running on the firewall cannot directly affect the main processor.

The software and configuration information used by the firewall processor resides on a read only disk. This eliminates the possibility of virus introduction or modification of programs performing the communications security functions.

The main processor and firewall communicate over an internal system bus that is not subject to ″sniffing programs″ or LANs.

The main processor can disable the firewall when tampering is detected, regardless of the state of the firewall.

The firewall software is installed in the same manner as any other AS/400 software product. There is no need to install and configure a separate machine and operating system as is typical with other firewalls.

Administration of the firewalls is performed by a Web browser on the internal (secure) network. Secure sockets can be used to protect the administration session. Authentication of the administrator is performed using the OS/400 security support so separate USERIDs or passwords are not required.

7.1.4 AS/400 Firewall Technology Components

- IP packet filtering for TCP, UDP, and ICMP
- SOCKS server
- Proxy server for HTTP, HTTPS, FTP, and GOPHER for Web browsers
- Telnet proxy
- Mail relay
- Split Domain Name Server (DNS)
- Logging
- Real-time monitoring

More AS/400 Firewall Information...

For more information on AS/400 firewall technology:

http://www.as400.ibm.com/firewall

7.2 Internet Connection Secure Server/400 (ICSS/400)

With the IBM Internet Connection Secure Server, business transactions can be secure. The secure server is enabled when 5769-NC1 or 5769-NCE is installed.

Internet Connection Secure Server provides HTTP secure (HTTPS) transactions with the Secure Sockets Layer (SSL) protocol. SSL is a security protocol that was developed by NetScape Communications Corporation, along with RSA Data Security, Inc. This protocol ensures that data transferred between a client and a server remains private. It also allows the client to authenticate the identity of the server.

Once your server has a digital certificate, SSL-enabled browsers such as the NetScape Navigator can communicate securely with a server using SSL. With SSL, a security-enabled Web site can easily be established on the Internet or on a corporate TCP/IP network.

SSL uses a security handshake to secure the 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 client authenticates the server. After that, SSL is used to encrypt and decrypt the information in both the https request and the server response, including:

- The URL the client is requesting
- The contents of any form being submitted
- Access authorization information such as user names and passwords
- All data sent between the client and the server

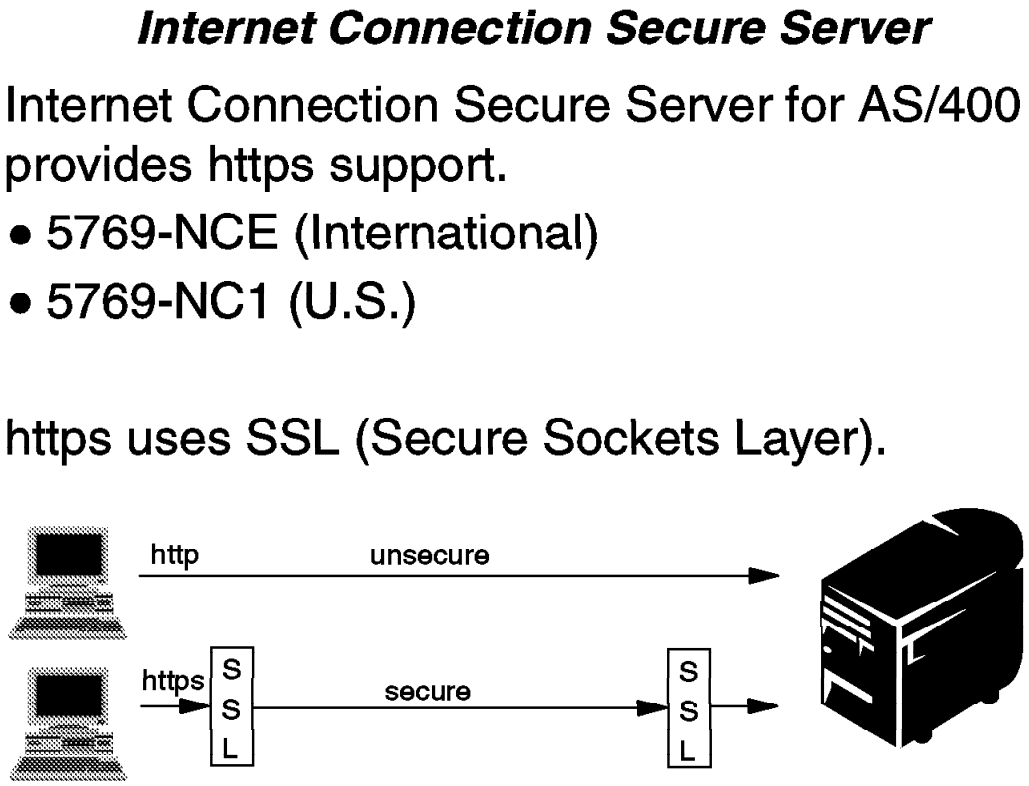


Figure 95. Internet Connection Secure Server Preview

7.3 New Versions of HTTP

The current version of the protocol today is HTTP 1.0, but it seems that HTTP 1.1 will introduce persistent connections. A significant difference between HTTP 1.1 and earlier versions of HTTP is that persistent connections are the default behavior of any HTTP connection. That is, unless otherwise indicated, the client may assume that the server will maintain a persistent connection. HTTP 1.1 is currently in the state of an Internet draft.

7.4 IBM Electronic Commerce - Net.Commerce

Net.Commerce provides a fully customizable virtual store front that allows secure commerce to take place over the Web. Net.Commerce comes with all of the software you need to build your first electronic store including a merchant server that manages the storefront interface to the customer and allows customers to browse through your catalog and place orders. The store manager allows store personnel to keep prices and product information up to date. The Secure Payment switch allows secure three-way credit card authorizations.

7.4.1 What Does Net.Commerce Do?

If you have a commercial business, Net.Commerce has practical solutions for getting you into the cyber market. Here is what Net.Commerce can do for your business.

7.4.2 Construct a Site for Your Business

Whether your business is a small shop or a large department store, whether you offer products or services, wholesale or retail, you can put your business on the Web with Net.Commerce. And Net.Commerce opens opportunities for commercial Web site developers to offer complete Web site packages to merchants.

You can build one store or a mall that showcases several stores. Choose a simple method of store creation and use the processes and Web pages that come supplied with Net.Commerce, or customize your store and incorporate legacy systems.

7.4.3 Create a Dynamic Shopping Experience

Because information about your merchandise is dynamically extracted from the database each time a shopper views your online store, you can quickly and easily tailor your store to changing markets and profit from emerging trends. You can also construct shopper profiles with information you collect from your shoppers electronically and develop and implement consumer-centered direct marketing strategies in your online store.

A handy design tool is provided to create visually appealing displays or pages to showcase your store and its products. You can include special effects on these pages such as 3D graphics, animation, sound, and Java applets.

7.4.4 Manage the Shopping Process End-to-End

Net.Commerce contains task macros and application program interface (API) functions that manage shopping tasks automatically. Shoppers register, select products, order, and submit payment online, letting you focus on managing your business strategy. Web pages for a shopping cart, registration form, and order forms are supplied for your shoppers to use, and can be customized to create a unique look and feel for your store. The supplied APIs suit many business applications but they can also be tailored to your needs. You can even write your own APIs for additional or unique requirements. And you can use them to incorporate such external business applications as tax calculation and inventory management systems into the online store you build with Net.Commerce.

Net.Commerce helps shoppers browse also. With the catalog building function, you define the categories or departments through which shoppers navigate to reach products, and you can lead shoppers to the same product through several navigation routes to maximize its exposure.

Products can have different prices for promotions, sales, or shopper groups. You define the conditions whereby these prices are valid and they are applied automatically.

Net.Commerce is a market leader in online shipping. You decide how your products are shipped and how to offer valued choices to consumers. You can

use as many shipping carriers as you want, and calculate shipping charges in ways that suit your product line such as by weight, dollar value, or quantity.

7.4.5 Help Manage Your Store

The Net.Commerce Store Manager tools make managing your store easy. You can quickly update such information about your online store as prices or product attributes using simple, fill-in-the-blank, on-screen forms. The information is stored in the database and the pages your shoppers see are immediately updated.

7.4.6 Protect Your Information and Your Shoppers

You can lock your database from unauthorized tampering and provide a password to managers who you choose to have access. Shoppers protect their information by providing and using a logon ID and password when they register.

User data such as credit card information is protected through Secure Sockets Layer (SSL) encryption. IBM Net.Commerce payment for merchants provides consumers, merchants, and financial institutions access to comprehensive, secure credit card usage over the Internet through the Secure Electronic Transaction (SET) protocol.

7.4.7 The IBM Net.Commerce Administrator

To build and manage an electronic store or mall, merchants use the IBM Net.Commerce Administrator, the merchant interface component of Net.Commerce. Merchants can easily enter store and product information and tailor product displays to suit their merchandising requirements. Changes they make automatically appear in the Web site of their store. And the multiple pricing capability, sophisticated shipping features, and market identification and targeting tools offer power and flexibility to profit from the changing marketplace and stay ahead of the competition.

The Net.Commerce Administrator contains two data management applications, the Site Manager and the Store Manager, and a Web page design tool called the Template Designer.

7.4.8 Site Manager and Store Manager

Merchants use the Site Manager to create and manage the infrastructure for their commercial Web site and the Store Manager to develop an online catalog and manage such other information as shipping options, shopper groups, and customer numbers. Using simple displays that look the same as fill-in-the-blank forms, merchants enter information about the store or mall and merchandise into the Net.Commerce database. Updating the information is quick and easy. Here are some examples of the information that merchants can store in the database and display on the Web pages:

- The store or mall name, logo, location, contact person, contact information, mission statement, policies, types of services and products offered, and currency used.
- The merchandise offered, including descriptions, product number or SKU (stock keeping unit), images, prices, availability dates, dimensions or weight, attributes such as size and color, and shipping methods for each product.
- The product categories (shipping companies and services that shoppers can select).

- Shopper groups that the merchant can define for the store.

Other information can be stored in the database for the merchant to view and use such as:

- Customer contact information, demographic data, shipping addresses, and a customer number.
- Information about the managers who have access to the store′s database.
- Administrative information required to manage shipping, prices, and tasks enabled through APIs.

7.4.9 Template Designer

Merchants can use the Template Designer to design Web pages. Its graphical look, drag-and-drop capabilities, and quick testing function help to create and test the pages. The design is laid out on a template that merchants can use to produce as many Web pages as they want. They can create different templates for different Web pages such as one for regular priced products and another for products on sale. This way, related elements can be visually consistent and unique elements visually distinct.

Merchants can create the following types of Web pages with the Template Designer:

- A home page for a mall or store with links to the inside.
- A page listing product categories from which shoppers select a category to browse.
- A page displaying the contents of a product category from which shoppers select a product to view in detail.
- A page showing a product with a description, price, image, and attributes such as color or size that shoppers can select, and a button that shoppers click to add the product to their shopping cart.
- Unique category and product pages for members of shopper groups.

7.5 Java for the AS/400 - A White Paper

Java is a hot, new programming language designed for today′s networked world. This white paper discusses what Java is and why it is a key piece of the application development strategy for the AS/400 system. Read on to find out about Java on the AS/400 system today and where it is going in the future.

7.5.1 Java Overview

Java is an object-oriented programming language developed by Sun Microsystems. It was originally designed for programming consumer electronics. The language was designed to be ″architecture neutral″ so that it could run on the different computer chips used in the various consumer electronic devices. But Sun soon realized that the language had potential to do much more.

The architecture-neutral aspect of Java makes it ideal for programming on the Internet. It allows a user to receive software from a remote system and execute it on a local system, regardless of the underlying hardware or operating system.

This is possible because of the interpreted nature of the language and the Java Virtual Machine.

Traditionally, the source code of a program is written in the programming language of choice and compiled into the machine code understood by a particular set of computer hardware. The Java compiler, however, does not generate machine code. Instead, it generates intermediate code called Java bytecodes. These bytecodes are interpreted by the Java interpreter, which executes the instructions on the particular hardware platform. The Java interpreter and run-time system are collectively called the Java Virtual Machine or JVM.

The bytecodes are what make Java programs portable. A program written in Java is compiled into bytecodes. These bytecodes can be transported across a network and executed on any system that implements a Java Virtual Machine.

An applet is a small Java program designed to be included in an HTML Web document. The HTML document contains tags that specify the name of the Java applet and its Uniform Resource Locator (URL). The URL is the location at which the applet bytecodes reside on the Internet. When an HTML document containing a Java applet tag is displayed, a Java-enabled Web browser downloads the Java bytecodes from the Internet and uses the JVM to execute the code from within the Web document. These Java applets are what enable Web pages to contain animated graphics or interactive content.

Because Java applets can be downloaded from any system, security mechanisms exist within the JVM to protect against malicious applets. The Java run-time system verifies the bytecodes as they are downloaded from the network to ensure they are valid bytecodes and that the code does not violate any of the restrictions placed on Java applets by the JVM. Java applets are restricted in which operations they can perform, how they access memory, and how they use the JVM. The restrictions are in place to prevent a Java applet from gaining access to the underlying operating system or data on the system.

But Java can be used for more than programs running within a browser. Java is a full-function programming language that can be used to write stand-alone applications that run outside of a Web browser.

Java is an object-oriented language, which means the focus is on the data and functions or methods that operate on the data. The data and the methods comprise a class that defines the state and behavior of an object. With the exception of a few primitive data types such as integers or floating point numbers, everything in Java is an object. In addition to the programming language constructs necessary for object-oriented program development, Java includes a rich set of predefined classes that are grouped together in what are called packages.

The Java Development Kit (JDK) version most widely available at this time is JDK 1.0.2. This version of Java contains the following packages:

- The java.lang, which contains the base classes
- The java.util, which contains classes for commonly-used data structures
- The java.io, which contains classes for input and output to files and streams
- The java.net, which contains classes for sockets programming and other networking

- The java.applet, which contains the classes needed for applet programming
- The java.awt, which contains the classes needed for GUI development

The most recent release of Java is JDK 1.1, which contains the following additional packages:

- The java.text, which contains classes for internationalization that allow applets and applications to be localized to different national languages and conventions.
- The java.security, which contains additional security features such as digital signatures that allow an applet to be signed by originator and cryptography so that information can be encrypted before traveling over the network.
- The java.rmi, which allows objects to be distributed across the network and called using remote method invocations.
- The java.beans, which is a cross-platform component model used to build applications using pluggable pieces.
- The java.sql, which allows database access using a standard SQL interface called JDBC (Java Database Connectivity).

Applications written using the JDK are portable. ″Write once, run anywhere″ has become the rallying cry of Java application programmers. Java applications developed using the integrated development environment on one system can be deployed on a different system without having to change or recompile the code. And Java is becoming widespread, with Java Virtual Machines available or planned for every major hardware platform and operating system.

One of the down sides of the portable, interpreted nature of Java is performance. While the performance of interpreted Java code is better than scripting languages and fast enough for interactive applications, it is slower than traditional languages whose source code is compiled directly into the machine code for a particular machine. To improve performance, Just-In-Time compilers (JITs) have been developed on many systems. A JIT compiler runs concurrently with the JVM and determines which methods within the Java code are called most often. These methods are compiled into machine code on-the-fly so that they do not need to be interpreted each time they are encountered within a program. Static compilers are also being developed that compile the Java source code into machine code that is executed without interpretation. This compilation is not done on-the-fly but is a separate step in the program development process. And it is important to note that the machine code generated is not portable and does not execute on other platforms.

7.5.2 Why Java for the AS/400 System?

There is a great deal of synergy between Java and the architecture of the AS/400 system. Java is an object-oriented programming language. The AS/400 system and its predecessors have had an object-based architecture from the very beginning. The AS/400 system knows objects!

The Java Virtual Machine and the platform independence it provides are also well-known concepts in the AS/400 world. The AS/400 system is the only computer system today that allows applications to move from 48-bit to 64-bit architecture without any porting or migration effort. The AS/400 system′s technology independent machine interface (TIMI) has shielded AS/400 customers from hardware technology changes for nearly 20 years.

Until now, the majority of the focus has been on client applets written in Java. But Java is a full-function programming language that can be used to write server applications. Java gives AS/400 developers the opportunity to move to object-oriented programming and modernize their applications without the complexity inherent in other object-oriented languages such as C++. Java is, by design, simpler than C or C++. For example, Java does not implement pointers. A Java program has a reference to an object, but that reference is not a pointer. Java also implements automatic garbage collection, which eliminates the memory leaks that cause ″out-of-memory″ errors in C and C++ code today. Other complex features of C++ that make the language both difficult to learn and to debug do not exist in Java. These include multiple inheritance, templates, and operator overloading. This makes Java a much simpler language and shortens both the learning curve and the development cycle when compared to other object-oriented languages.

For the AS/400 customer who does not want to write programs in Java, the bytecode portability ensures that ″100% Pure Java″ code will run without recompilation on the AS/400 system. This means that any application development environment can be used to develop the code and create the Java bytecodes. The application can be deployed on any platform that includes a JVM that conforms to the Sun specifications.

For large IT departments, this means that a ″one-size-fits-all″ approach can be used for application deployment. Only one version of a Java program is needed, regardless of how many different computer platforms are supported within the organization.

Java is an open, cross-platform, de facto industry standard language. It solves Web programming problems that are difficult to solve with more traditional languages. It is the ideal language for network computing (thin clients) and the World Wide Web. The AS/400 system is on its way to becoming the premiere server for network computing. Java is an important piece of the overall AS/400 Internet/intranet strategy.

7.5.3 Java on the AS/400 System Today

A technology preview of Java on the AS/400 system is available today. This preview is a port of Sun′s JDK 1.0.2 to the AS/400 system. The java.awt windowing classes are not currently included in this preview. The JDK 1.0.2 currently runs on top of OS/400 and is not integrated into the operating system. Because it is not optimized for the AS/400 system and does not include any JIT or other compiler technology, the performance characteristics are similar to the initial Java offering from Sun when they first released the technology. For many applications, the performance of the technology preview is slower than what is seen on comparable platforms using JIT or other optimizing technology.

The technology preview is not intended for application deployment. But because Java can be written on any platform and deployed on any other platform, the preview can be used to run programs written on other platforms without modification or recompilation. The true portability of Java is highlighted as the code moves from any platform and executes without change on the AS/400 system.

Java applications can also be written on client workstations today that access AS/400 programs and data. There are a number of different options available to do this. Client applets or applications can use:

- JDBC to access AS/400 database
- The java.net sockets to access server applications on the AS/400 system
- Native methods to call out to other client/server access services

At this time, many of these options entail writing middleware to connect to the AS/400 system. Java client extensions will be available in the future that can be used to access AS/400 programs and data from a Java applet or application on the client.

For information on other object-oriented or network computing options available for the AS/400 system today, visit the AS/400 Partners In Development Web page.

7.5.3.1 Strategy and Direction for Java on the AS/400 System

Java is a key application development language for the AS/400 system. As the Java technology evolves from Sun, the AS/400 system will take advantage of the new functions and features of the language.

There is an exciting future for Java on the AS/400 system. This section describes the current plans for Java technology, but is not a commitment to the function or performance of future products on the AS/400 system. This document will be updated as these plans change in the future.

Currently an effort is underway to integrate Java technology with OS/400. A Java Virtual Machine that resides below the TIMI is planned to enable fast interpretation and execution of Java code on the AS/400 system. In addition, a direct execution static compiler is being developed that can generate RISC machine code as well as the portable Java bytecodes. This will allow Java code to directly execute on the AS/400 system without the overhead of interpretation. High-performance garbage collection is also being developed for the AS/400 system to improve both the performance and the scalability of Java. The high level of integration and tuning of Java on the AS/400 system will meet the objective of competitive performance while preserving cross-platform portability.

Other technology is being developed that will allow GUI applications to run on the AS/400 system without modification. Support is planned for Java on the AS/400 system that intercepts GUI requests coming from a Java program and reroutes the requests to an attached workstation running its own JVM. The workstation can interpret and display the java.awt graphical components. This allows AS/400 programs to have graphical interfaces using the standard Java language on the AS/400 system.

In addition, plans include integration of other Java packages with OS/400 components for both ease-of-use and improved performance. All of the integration and performance work is being done ″under-the-covers″ of OS/400 without compromising portability of the Java code.

Some AS/400-specific classes are also planned that are not part of the JDK itself, but are supplied as a convenience to programmers who want to access more traditional AS/400 application environments from Java. These classes provide access to unique AS/400 features such as data queues, record-level database I/O, and traditional programming languages such as CL and RPG.

The unique single-level-store architecture is also being exploited to give Java objects on the AS/400 system an advantage not available on any other platform.

Java objects on the AS/400 system will be full-fledged system objects allowing them to be persistent, shared, secure, backed up, and restored. This allows the AS/400 system to offer persistent Java objects with performance and support that is unparalleled in the industry. The AS/400 single-level-store technology permits Java objects to be stored in their object form without the performance and maintenance overhead of two-level-store operating systems.

The AS/400 system is uniquely positioned to leverage Java as it evolves from its current Web focus to a full commercial application environment. The strengths of the AS/400 system are combined with Java′s object-oriented, network computing technology to provide solutions for the new millennium.

7.5.4 Summary

Java is the language of choice for programming in today′s network computing environment. It allows true portability of applications between platforms without modification or recompilation. It is an open, cross-platform, industry standard that is being supported by all of the major players in the computer industry today.

Java is available **now** for the AS/400 system. And the best is yet to come...

Chapter 8. Internet Application Performance

8.1 Internet Application Performance Overview

This chapter gives you an overview of what is important to select the right system and model for the applications serving the Internet. Also, it gives some configuration recommendations to optimize and tune the system for optimal Performance.

8.1.1 How Fast Can a Web Site Go?

You want to know how to design your Web site to get maximum speed. And you are probably thinking in terms of how many hits per day your Web server can handle. For example, ″Our www.askas400.com site needs to service 1 million hits per day.″

But wait! If you base your choices only on how many hits your Web server can serve per day, you may end up buying less equipment than you need.

8.1.2 How Many Connections per Second is Enough?

A hit and a connection are the same thing. Throughout our information, we use the maximum connections per second as a gauge, but how many seconds are in a day?

```
24 hours * 60 minutes per hour * 60 seconds per minute = 86,400 seconds
1 million hits per day = 1 000 000 / 86 400=11.6 connections per second
```

But are the requests to your site evenly distributed over 24 hours every day? In reality, most Web sites have peaks and valleys in the rate of requests. When doing capacity planning, you need to concentrate on planning for the peaks. Otherwise, you over estimate the maximum capacity of your Web site.

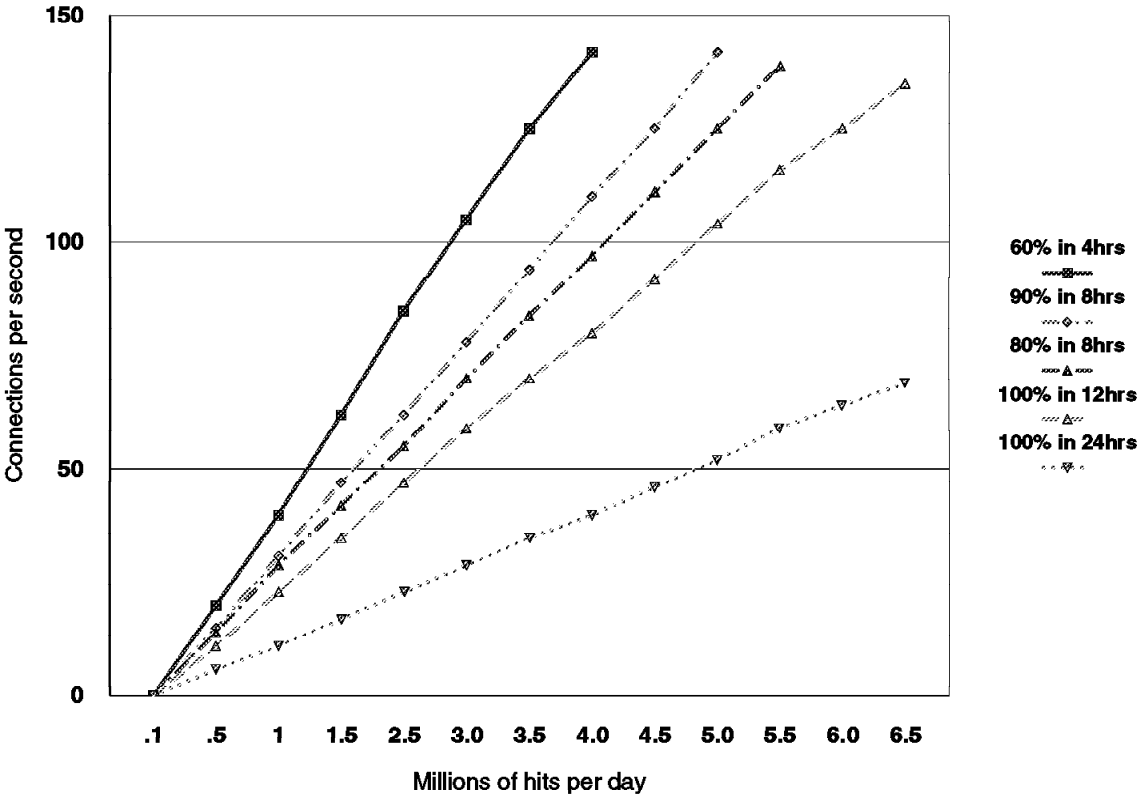


Figure 96. Distribution of Connections

This graph shows the importance of understanding the peaks in your Web request rate. To read the chart, first find the line that most closely matches your Web site:

Do 60% of your requests occur in only four hours of a day, for example, from 6PM to 10PM Eastern Standard Time? Do 90% of your requests occur in eight hours of a day, for example, 8AM to noon and 1PM to 5PM?

Do virtually all of your requests occur in 12 hours of a day, for example, 8AM Eastern Standard Time to 5PM Pacific Standard Time? Or does the worldwide appeal of your site result in a request load that is balanced around the clock?

Once you have found the line that most closely matches your Web site request load, find on the X-axis the number of millions of hits per day for your Web site. Finally, find the corresponding value on the Y-axis, which tells you the number of connections per second your Web site needs to support in order to achieve the corresponding number of hits per day.

For example, let′s assume that www.askas400.com needs to service 3.5 million hits per day. If you assume that the request rate is balanced throughout 24 hours a day, www.askas400.com needs to be configured to support 40 connections per second. However, the staff at www.askas400.com knows that 80% of their requests arrive in the same eight-hour period every day. Based on the graph, you can see that to service this peak request rate, www.askas400.com

needs to be configured to support nearly 100 connections per second, resulting in dramatically different hardware requirements!

You also need to take file size into consideration when thinking about connection rates. A hit can mean anything ranging from a quick sub-second browse request, all the way to the minutes (and computer resource) required for streaming audio or video. As a result, you should use Web benchmark numbers for relative sizing only. Even the popular WebStone benchmarking only covers GET transactions, not POST transactions, which are used with forms and CGI programs. This next graph shows why you need to consider both file size and the peak hit rate when planning your Web site requirements.

We took the maximum connection rate for a sample network design and created a chart showing the maximum hits per day expected based on file size and the peak hit rate, or hit distribution. In this example, when the average file size is 5KB, if the hit distribution means 60% of the hits occur in four hours, this configuration will support 2.1 million hits per day. However, if the connections are evenly distributed over 12 hours, the same configuration supports 3.7 million per day. Similarly, if the file size were 10KB and the hit distribution means 60% of the hits occur in four hours, this configuration supports 1.8 million hits per day; but if 80% of the connections occur in eight hours, 2.7 million hits per day can be supported.

So, which is it? 1.8 million, 2.1 million, 2.7 million, or 3.7 million per day? If you do not know the average file size or the peak hit rate, you cannot understand the maximum capacity of your Web site.

If you want to see a comparison of how different file sizes affect the response time, see Figure 100 on page 246.

8.2 Internet Connection Performance for AS/400 System

Performance information for the Internet Connection for AS/400 (IC/400) product is included in this section. There are many factors that can impact overall performance (for example, end-user response time and throughput) in the complex Internet environment, some of which are included in the following list:

- Web Browser
  - Processing speed of the client system
  - Performance characteristics of the Web browser
  - Client application performance characteristics
- Communications network
  - Speed of the communications links
  - Capacity of any proxy servers
  - Congestion of network resources
- AS/400 WWW (Web) server
  - AS/400 processor speed
  - Utilization of key AS/400 resources (CPU, IOP, memory, disk)
  - Web server performance characteristics
  - Application performance characteristics

The primary focus of this section is to discuss the performance characteristics of the AS/400 system as a server in a Web serving environment, providing capacity planning information and recommendations for best performance.

Data accesses across the Internet differ distinctly from accesses across ″traditional″ communications networks. The additional resources to support Internet transactions by the CPU, IOP, and line are significant and must be considered in capacity planning. Typically, in a traditional network:

- Request and response (between client and server).
- Connections/sessions are maintained between transactions.
- Networks are tuned to use large frames.

For Internet transactions, there are a dozen or more line transmissions (including acknowledgements) per transaction:

- A connection is established between client and server.
- Requests and responses (between client and server).
- The connection is closed.
- Networks typically have small frame (MTU) sizes.
- HTML displays to browser contain more bytes than traditional.
- One user transaction may contain several separate Internet transactions.

The information that follows is based on performance measurements and analysis done in the AS/400 division laboratory. The highlights, general conclusions, and recommendations are included. Results listed here do not represent any particular customer environment. Actual performance may vary significantly from what is provided here.

8.3 AS/400 Commercial Processing Workload (CPW)

The Commercial Processing Workload is a modified version of the TPC-C (Transaction Processing Performance Council) workload, which is an industry standard for measuring performance. CPW results represent the performance of many sophisticated commercial environments. By using CPW, IBM can more accurately represent our customers′ more robust commercial workloads.

Note

CPW results do not only represent the performance of IBM (or another vendor′s models), they represent the performance of many sophisticated commercial environments. By using CPW, you can more accurately benchmark the performance of the AS/400 system against that of other similar systems.

8.3.1 AS/400 Advanced Server Models V3R7

The following chart shows the relative Performance Characteristics of all AS/400 Advanced Server Models running with the current version of OS/400 V3R7.

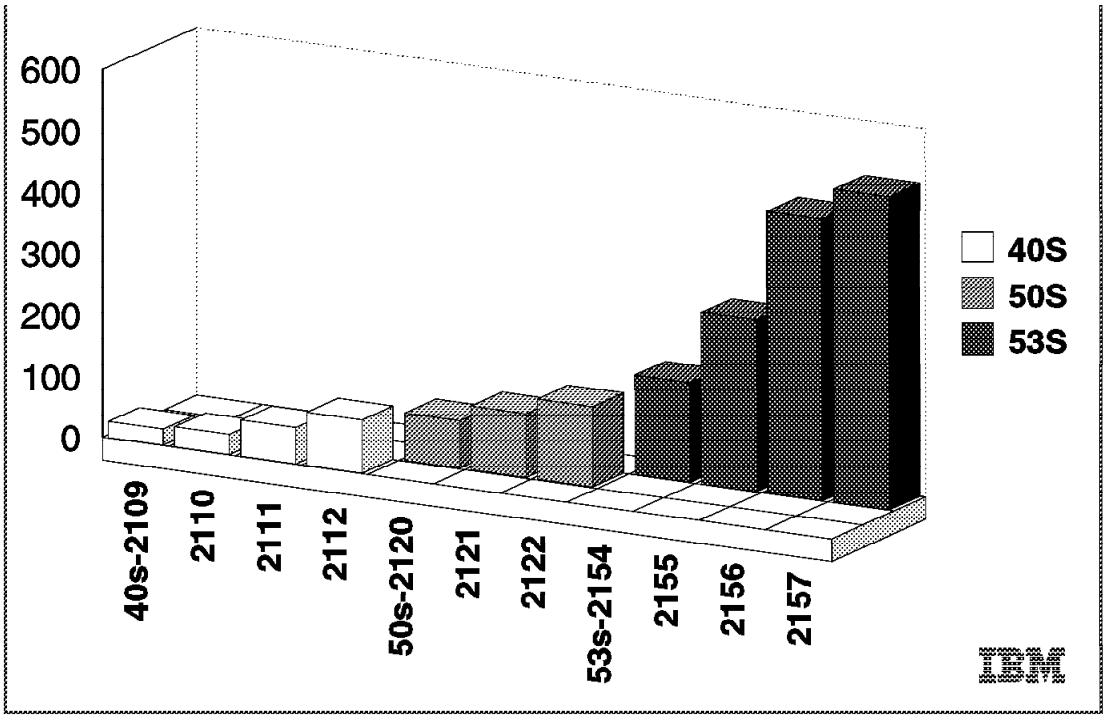


Figure 97. AS/400 Relative Performance - CPW for Advanced Server Models V3R7

As you can see on the preceding chart, the Performance Capability of the server models go over a wide range. It starts at about 30 CPW for the smallest of the 40S Models, over 80 CPW of the 50S, to the fastest model 53S showing more than 500 CPW.

The CPW Value is useful in determining the system performance growth provided by different AS/400 models in the Internet environment. As a first approximation, CPW values can be used when a quick estimate is required, followed by a more detailed analysis using BEST/1 for OS/400.

For example, a model 40S-2109 has a CPW value of 27 and the fastest 40S-2112 has a CPW value of 87. So based on the previous assumption, we can say that a 40S-2112 may provide about three times the throughput of a 40S-2109.

The ability of a system to process a unit of work is made up of many factors. The most important factors are CPU speed and DASD access time. It is important to note that changing the CPU only affects the CPU component of the workload. If you know the CPU component for a particular workload, you can estimate an improvement in the CPU component by using the Relative Performance Rating.

Note

The preceding method does not take into account wait time for queues in the system for CPU, DASD, or other system components that are a function of utilization, and cannot easily be modeled by hand. Capacity planning should be done with a capacity planning tool such as BEST/1 for OS/400.

8.4 Web Serving with the HTTP Server

The Hypertext Transfer Protocol (HTTP server) allows AS/400 systems attached to a TCP/IP network to provide objects to any Web browser. At a high level, the connection is made, the request is received and processed, a file system is accessed (or a CGI program is accessed), the data is sent to the browser, and the connection is ended. The HTTP server jobs and the communications router tasks are the primary jobs/tasks involved (there is not a separate user job for each attached user).

8.4.1 Web Serving Performance Measurements

The following charts provide a summary of the measured performance data. Results listed here do not represent any particular customer environment. Actual performance may vary significantly from what is provided here.

8.4.2 AS/400 HTTP Server Performance

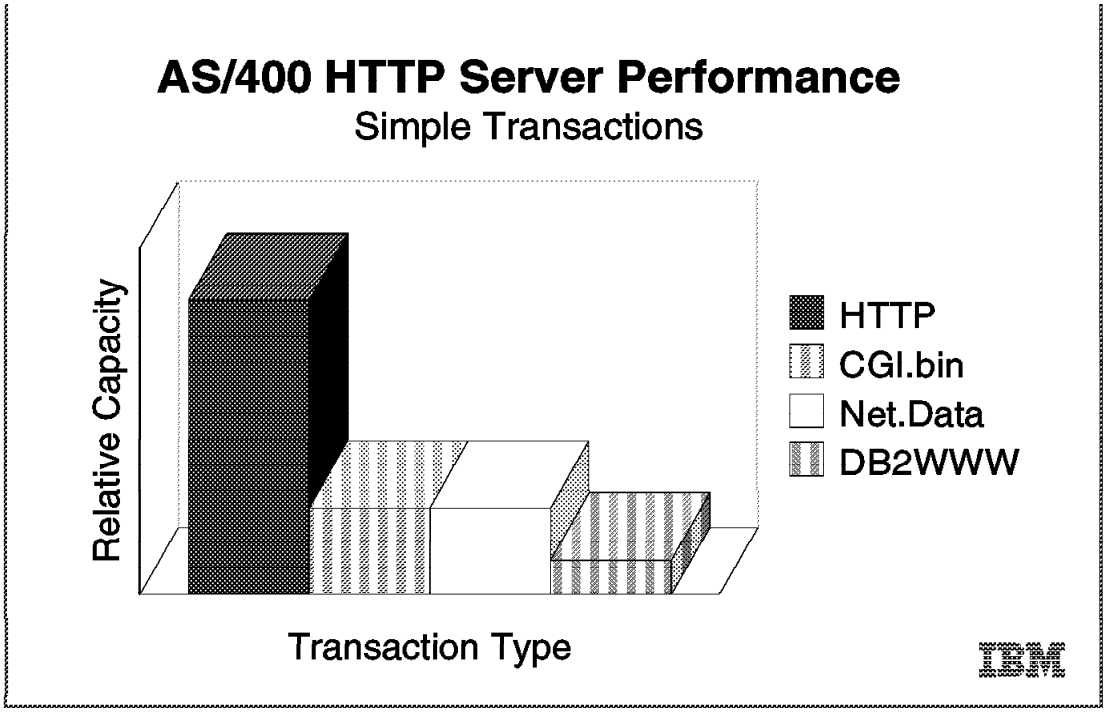


Figure 98. AS/400 HTTP versus CGI-Bin versus Net.Data

8.5 Web Serving Performance Recommendations

Throughput for Web serving is typically discussed in terms of the number of hits/second (connections/second and transactions/second mean the same). Typically, the CPU is the resource that limits capacity. If a large AS/400 model is used with a single IOP, the IOP may the limiting factor.

Note

For more detailed information regarding specific AS/400 model Web serving capacity, see your IBM Representative.

8.5.1 CISC versus RISC

The HTTP serving environment on RISC AS/400 models provides better performance (per CPW) than on CISC AS/400 models. The hits/sec/CPW factor for RISC models is almost twice that of the CISC models. This is due to the better RISC technology for C-language environments and to recent performance improvements to TCP/IP and IC/400.

Note

Note that CPWs (relative system performance metrics) are derived from a commercial workload′s capacity. Therefore, relative capacity ratios for Web serving may not track accordingly. However, they should be adequate for a high-level capacity planning exercise.

8.5.2 Response Time (General)

User response time is made up of Web browser (client work station) time, network time, and server time. The response times in measurements taken on the AS/400 system do not include Web browser time; they typically contribute 0.5 seconds to 2 seconds to response time. A problem in any one of these areas may cause a significant performance problem for an end user. To an end user, it may seem apparent that any performance problem is attributed to the server, even though the problem is elsewhere.

It is common for pages that are being served to have imbedded images (gifs). Each of these separate Internet transactions adds to the response time as they are serially retrieved from various servers (some browsers can concurrently retrieve two URLs).

8.5.3 HTTP and TCP/IP Configuration Tips

1. The **number of HTTP server jobs** (CHGHTTPA command) controls the minimum and maximum number of HTTP server jobs handling HTTP requests. The reason for having multiple server jobs is that when one server is waiting for a disk or communications I/O to complete, a different server job can process another user′s request. Also, for N-way systems, each CPU may simultaneously process server jobs. The system adjusts the number of servers that are needed automatically (within the bounds of the minimum and maximum parameters).

   The smallest minimum allowed is two (one parent and one child server). Typically three to five servers are adequate for smaller systems (50 CPWs or less). For larger systems (more than 50 CPWs) dedicated to HTTP serving, increasing the number of servers to 10 or more may provide better performance. Also, if CGI or Net.Data is used (which are more disk I/O intensive), configuring more servers may provide better performance. Try not to have more than what is needed as this may cause unnecessary system activity.

2. The **maximum frame size parameter** (MAXFRAME on LIND) can be increased from 1994 bytes for TRLAN (or other values for other protocols) to its maximum of 16 393 to allow for larger transmissions. Typically documents are larger than 1994 bytes.

3. The **maximum transmission unit (MTU) size** parameter (CFGTCP command) for both the route and interface affect the actual size of the line flows. Increasing these values from 576 bytes to a larger size (up to 16 388) probably reduces the overall number of transmissions, and therefore, increases the potential capacity of the CPU and the IOP.

   Similar parameters also exist on the Web browser. The negotiated value is the minimum of the server and browser (and perhaps any bridges/routers) so increase them all.

4. Increasing the **TCP/IP buffer size** (TCPRCVBUF and TCPSNDBUF on the CHGTCPA command) from 8K bytes to 64K bytes may increase the performance when sending larger amounts of data.

5. **Error and Access Logging:** Having logging turned on causes a small amount of system overhead (CPU time, extra I/O). Turn logging off for best capacity. Use the WRKHTTPCFG command to make these changes.

6. **Name Server Accesses:** For each Internet transaction, the server accesses the name server for information (IP address and name translations). These accesses cause significant overhead (CPU time, comm I/O) and greatly reduce system capacity. These accesses can be eliminated by using the WRKHTTPCFG command and adding the line ″DNSLookUp Off″. For V3R7, PTF SF35478 is a prerequisite for this change.

8.5.4 HTTP Server Memory Requirements

Follow the faulting threshold guidelines suggested in the *Work Management Guide* by observing/adjusting the memory in both the machine pool and the pool that the HTTP servers run in.

From one benchmark measured in the lab, (simple HTTP page serving with 1K byte files), the memory requirement for the HTTP server pool is:

1000 K bytes + 400 K bytes for each HTTP server

Factors that may significantly increase the memory requirements include using larger document sizes, using CGI-Bin programs, and using Net.Data.

8.5.5 AS/400 Model Selection

Use the information provided in this section along with the characterization of your HTTP workload environment in a capacity planning exercise (perhaps with BEST/1) to choose the appropriate AS/400 model. All the tasks associated with HTTP serving are ″non-interactive″, so AS/400 server models probably provide the best price/performance.

8.5.6 File System Considerations

Web serving performance varies significantly based on which file system is used. Each file system has different overhead and performance characteristics. Note in Figure 99 on page 245 that serving from the ROOT or QOPENSYS directories provide the best system capacity. If Web page development is done from another directory, consider copying the data to a higher-performing file system for production use.

8.5.6.1 AS/400 HTTP Server Performance, Simple Page Serving

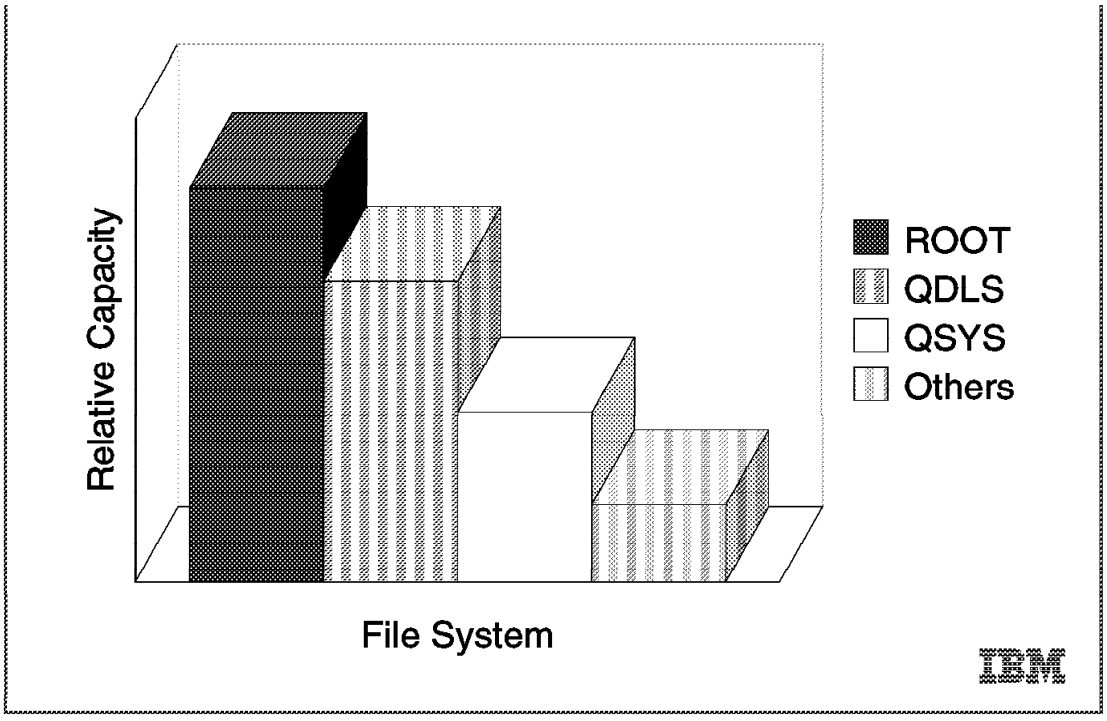


Figure 99. AS/400 HTTP Server Performance, Simple Page Serving

8.5.7 File Size Considerations

Note in Figure 100 on page 246 that serving larger files uses more system resources. The connect and disconnect costs are similar regardless of size, but cost for the transmission of the data with TCP/IP and the IFS access vary with size. As file size increases, the IOP is more efficient by being able to achieve a higher aggregate data rate. However, being larger, the files require more data frames, thus causing the hits/sec capacity for the IOP to go down accordingly.

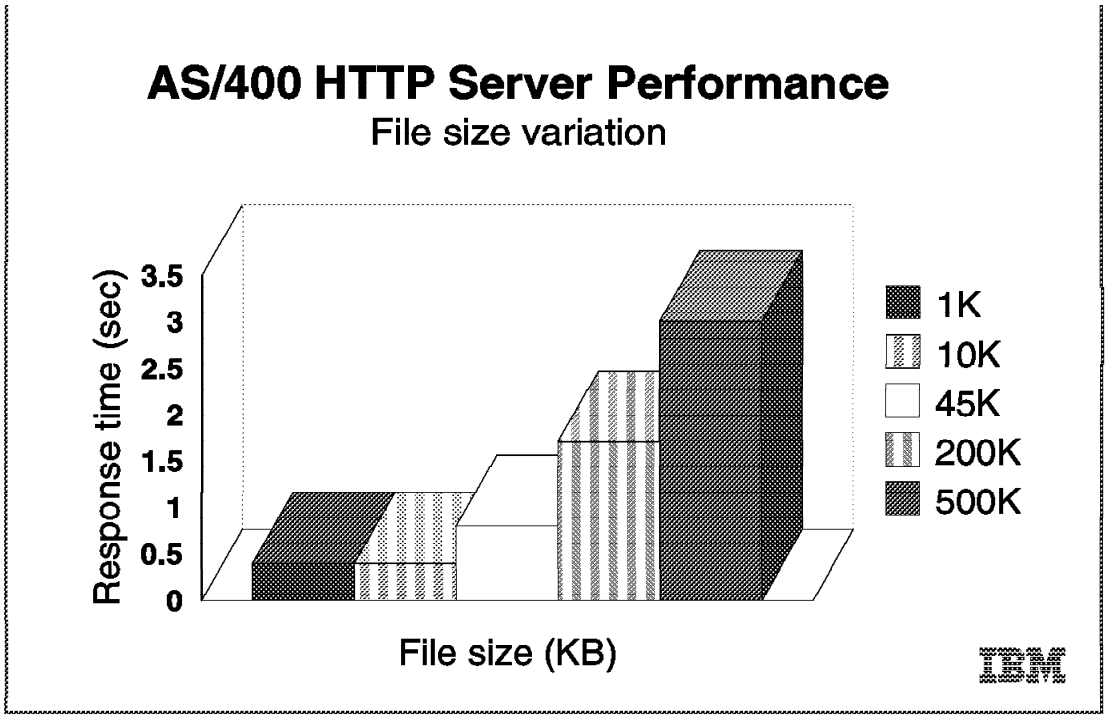


Figure 100. AS/400 File Size Variation

8.5.8 Communications/LAN IOPs

Since there are a dozen or more line flows per transaction, the Web serving environment utilizes the IOP more than other communications environments. Use the performance monitor (STRPFRMON) and the component report (PRTCPTRPT) to measure IOP utilization. Attempt to keep the average IOP utilization at 60% or less for best performance.

Use the data in Figure 100 to estimate 2619 TRLAN IOP capacity (2617 Ethernet IOP capacities are similar). IOP capacity depends on file size and MTU size (make sure you increase the maximum MTU size parameter).

On larger AS/400 models, the comm/LAN IOP may become the bottleneck before the CPU does. If additional HTTP capacity is needed, multiple IOPs (with unique IP addresses) can be configured. The overall workload must be ″manually″ balanced by Web browsers requesting documents from a set of interfaces. The load can also be balanced across multiple IP addresses by using a distributed name server.

8.6 Net.Data and DB2WWW

- Since Net.Data is the follow-on to DB2WWW version 1, ensure that you have the right PTFs applied to get the new version. See http://www.as400.ibm.com/netdata for the latest code levels and more product information.
- Net.Data performs significantly better than DB2WWW version 1. Note from the data in Figure 99 on page 245 that the relative capacity improved by more than a factor of two.

- Net.Data is more disk I/O intensive than typical HTTP transactions. Therefore, more HTTP server jobs may be needed to provide the optimal level of system throughput.
- A Net.Data SQL macro is slightly slower than an SQL CGI-Bin. This is because the SQL macro does some formatting while the SQL CGI-Bin does not do any formatting. There are functional advantages in using an SQL macro.
  - Direct reuse of existing SQL statements (no programming required).
  - Provides the built-in ability to format SQL results.
  - Provides the ability to store SQL results in a table and pass. the results to a different language environment (for example, REXX).
- A Net.Data HTML macro is slightly better than an HTML CGI-Bin because of the implementation of activation groups.
- If macro performance is critical, turn off the reporting function (uses more CPU time).
- For REXX macros, calling an external REXX program performs better than inline REXX statements.

8.7 5250/HTML Workstation Gateway

5250/HTML Workstation Gateway enables all Web browsers to be clients to existing 5250 applications without making changes to the application. At a high level, the connection is made, the request is received and processed, the user job processes the application, the workstation I/O is converted from 5250 to HTML, the HTML is sent to the browser, and the connection is ended. The tasks involved are the Workstation Gateway server jobs, the user job (one per client), the communications router tasks, the virtual terminal task, and the Telnet task.

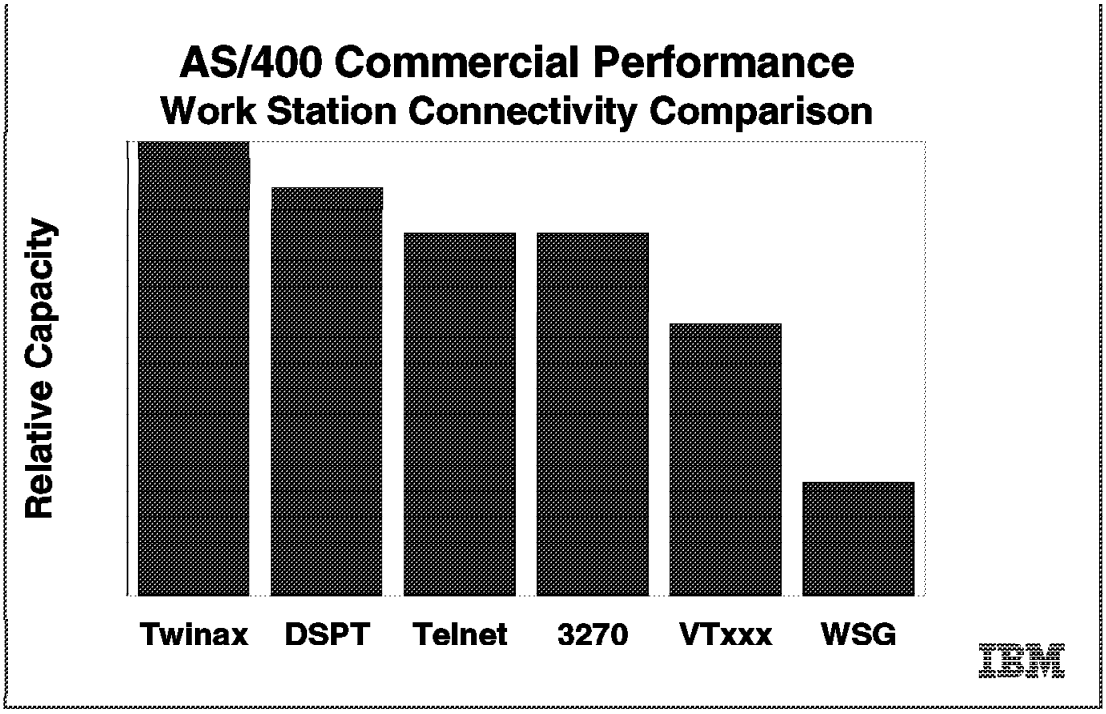


Figure 101. 5250/HTML Workstation Gateway Comparison

8.7.1 Workstation Gateway Performance Recommendations

This section describes specific WSG performance recommendations.

8.7.1.1 Workstation Gateway CPU Time and Capacity

Workstation Gateway involves a significant amount of CPU processing compared with other workstation connectivity types (for example, Telnet). Typical commercial transactions with Workstation Gateway use three times the CPU time of Telnet running the same application (on the target system). Therefore, it is recommended to use Workstation Gateway casually or for non-performance critical transactions. It is not a good price/performance decision to convert all of the workstations in an existing production environment from twinax or communications to Workstation Gateway. (Note that Workstation Gateway uses six times the CPU time of Telnet in CISC models, but only three times more CPU time in RISC models.)

8.7.1.2 Workstation Gateway Response Time

Response time is made up of browser time, network time, and server time. Browsers typically contribute 0.5 seconds to 2 seconds to response time, network time can vary widely, and AS/400 server time typically ranges from 0.2 seconds to 5 seconds depending on the model, load, and application.

A problem in any one of these areas can cause a significant performance problem for an end user. To an end user, it may seem apparent that any performance problem is attributed to the server even though the problem is elsewhere.

Better response time may be achieved by using the STYLE button on the browser (Workstation Gateway window) to remove the F-keys on the bottom. This reduces the browser time to process and show the display.

8.7.1.3 Workstation Gateway Configuration

Default settings for the Workstation Gateway related parameters typically work well. Changing some of the following parameters may improve performance:

- The number of clients per Workstation Gateway server job (CHGWSGA command): This parameter controls the maximum number of users that can be attached to a given Workstation Gateway server job. When the Workstation Gateway server is started, several server jobs start. As clients sign on, they are assigned to these server jobs (one server is assigned to its maximum before users are assigned to the next server).

  It is unlikely that this parameter needs to be changed. Increasing this parameter may be recommended if there are hundreds of casual users. Decreasing this parameter may be recommended if there are just several heavy users.

- MAXFRAME (on LIND): The maximum frame size parameter can be increased from 1994 bytes for TRLAN (or other values for other protocols) to its maximum of 16 393 to allow for larger transmissions.

- MTU size (CFGTCP command): The maximum transmission unit size parameter for both the route and interface effect the actual size of the line flows. Changing these values from 576 bytes to a larger size (perhaps the maximum of 16 388) may reduce the overall number of transmissions. This needs to be considered on both the server and the Web browser. Typically, even simple displays have several thousand bytes of HTML data.

8.7.1.4 Workstation Gateway Memory Requirements

Follow the faulting threshold guidelines suggested in the *Work Management Guide* by adjusting the memory in the machine pool (the pool that the Workstation Gateway servers run in) and the pool that the user jobs run in (use the WRKSYSSTS command). For the Workstation Gateway server pool, the actual memory required per server (working set size) varies significantly by the workstation I/O characteristics of the workload. From one benchmark measured in the lab, the memory requirement for the Workstation Gateway server pool is:

600 K bytes + 700 K bytes for each WSG server

8.7.1.5 Communications Lines and IOPs

Workstation Gateway/HTML sends significantly more bytes to the client workstation than Telnet does for the same application. Therefore, plan for additional communications line capacity to handle this extra load. For example, a Telnet transaction may consist of 1K bytes, while a Workstation Gateway transaction may be 6K bytes for the same transaction.

8.7.1.6 AS/400 Model Selection

Use the information provided here along with the characterization of your Workstation Gateway workload environment in a capacity planning exercise (perhaps with BEST/1).

Server models may not be the best choice for Workstation Gateway environments. The user job is labeled as an ″interactive″ job, while the other tasks involved (Workstation Gateway servers and communications tasks) all are tagged ″non-interactive″. For a typical commercial transaction with Workstation Gateway, the ″interactive″ portion of the transaction is about 20% of the total transaction′s CPU time. To avoid decreasing the effectiveness of your server model when using Workstation Gateway, it is recommended that you keep the ″interactive″ portion of your overall CPU utilization below the threshold at where the efficiency drops off.

- Do not do heavy applications through Workstation Gateway.
- Limit Workstation Gateway to casual use (low number of Workstation Gateway transactions).

Otherwise, use traditional (non-server) AS/400 models.

8.8 Net.Data Performance, Hints, and Tips?

Reference

For more detailed performance information and tips and techniques to improve performance, see Chapter 4, “Net.Data Implementation” on page 111.

8.9 CGI-Bin Performance, Hints, and Tips

Reference

For more detailed performance information and tips and techniques to improve performance, see Chapter 3, “Common Gateway Interface (CGI-BIN) Implementation” on page 57.

8.10 Workstation Gateway, Hints, and Tips

Reference

For more detailed performance information and tips and techniques to improve performance, see Chapter 5, “HTML Gateway Implementation” on page 169.

Appendix A. ILE RPG and ILE COBOL Sample CGI Programs

Disclamer!

The source code for the Cobol and RPG program are included for reference.

A.1 ILE RPG Sample CGI Program CUSTINFO

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----|
     *****************************************************************
     *
     *  Program:      Prompt user for mailing info and response
     *
     *  Language:     ILE RPG
     *
     *  Description:  This program is used to process an HTML Form
     *                as input and add a record to a database
     *                It is a SAMPLE program
     *
     *  APIs Used:    QtmhRdStin  - Read from Standard Input. This
     *                              API is used to read the input
     *                              Data from the user.
     *
     *                QtmhCvtDb   - Convert to DB. This API takes
     *                              the Stdin input and converts it
     *                              into a group data structure that
     *                              can be processed by an RPG pgm.
     *                              The field names in the DDS must
     *                              match the names in the HTML Form
     *
     *                QtmhWrStout - Write to Standard Output. This
     *                              API writes the output to the user.
     *                              The Output Must be in the correct
     *                              HTML format and it MUST have a
     *                              MIME Header.
     *
     *                QCMDEXC     - Execute a CL Command from within a
     *                              High Level Langauge Program.
     *                              Used to Override the output
     *                              database file.
     *
     * 1. COMPILE THIS SOURCE MEMBER AS MODULE CUSTINQRP (PDM OPTION=15)
     *
     * 2. CREATE PROGRAM CUSTINQRP FROM MODULE CUSTINQRP (PDM OPTION=26)
     *    WITH PROMPT (PF4) AND BNDSRVPGM(QTCP/QTMHCGI)
     *
     * Refer to the TCP/IP Configuration manual for API Parms
     *
     *****************************************************************
     * DEFINE YOUR FILES HERE
     *****************************************************************
    FCUSTPF    UF A E             DISK    USROPN
     *****************************************************************
     * DS contains the name of the file and library used by the CVTDB API *
     *****************************************************************
    dDBFILE           DS
    d                                10    inz('CUSTPF')
    d                                10    inz('WEBDB')
     *****************************************************************
     * Buffer used to convert the string data into fields          *
     *****************************************************************
    DDBBuff         E DS                  EXTNAME(CUSTPF)
```

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----|
     ******************************************************************
     * Standalone Fields Used for PARMS and work fields in the Program *
     ******************************************************************
    DINBuff           S           2048
    DINBuffLn         S              9B 0
    DINActLn          S              9B 0
    DDBRecLn          S              9B 0
    DDBCvtLn          S              9B 0
    DRspCode          S              9B 0
    DOutLn            S              9B 0
    DCmdLen           S             15P 5
    DEMAILM           C                   'We Will Contact you via E-Mail'
    DSMAILM           C                   'We Will Contact you via Surface +
    d                                     Mail'
    DPMAILM           C                   'We Will Contact you via Phone'
     ******************************************************************
     * Buffer used for standard API Error handling                    *
     * Copied from QSYSINC/QRPGLESRC - QUSEC                          *
     ******************************************************************
    DQUSEC            DS
    D*                                            Qus EC
    D QUSBPRV                 1      4B 0
    D*                                            Bytes Provided
    D QUSBAVL                 5      8B 0 inz(16)
    D*                                            Bytes Available
    D QUSEI                   9     15
    D*                                            Exception Id
    D QUSERVED               16     16
     ******************************************************************
     * This DS contains a Standard startup needed for HTML            *
     *   This is used as output to the user. You should change        *
     *   the text between the <title> text </title> tags.             *
     *                                                                *
     *===>      Hex 15 (x'15') is the newline character and is        *
     *===>      REQUIRED between lines of output                      *
     *===>   A Line of output may not exceed 120 Characters          *
     ******************************************************************
    dStdHdr           DS
    d                                23    inz('Content-type: text/html')
    d                                 2    inz(x'1515')
    d                                 6    inz('<HTML>')
    d                                 1    inz(x'15')
    d                                 6    inz('<head>')
    d                                 1    inz(x'15')
    d                                 7    inz('<title>')
    d                                 1    inz(x'15')
    d                                24    inz('Thank you for your input')
    d                                 8    inz('</title>')
    d                                 1    inz(x'15')
    d                                 6    inz('<body>')
    d                                 1    inz(x'15')
    d                                 4    inz('<hr>')
    d                                 1    inz(x'15')
     ******************************************************************
     * This DS contains a Standard trailer needed for HTML            *
     *   This is used as output to the user.                          *
     ******************************************************************
    dStdTrl           DS
    d                                 7    inz('</body>')
    d                                 1    inz(x'15')
    d                                 7    inz('</html>')
    d                                 1    inz(x'15')
```

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----|
     ****************************************************************
     * This DS contains the user output data. It should change     *
     * from program to program.                                    *
     ****************************************************************
     dOutInf           DS
     d                                6    inz('Name: ')
     dnameo                                like(Name)
     d                                4    inz('<BR>')
     d                                1    inz(x'15')
     d                                7    inz('Addr1: ')
     daddr1o                               like(addr1)
     d                                4    inz('<BR>')
     d                                1    inz(x'15')
     d                                7    inz('Addr2: ')
     daddr2o                               like(addr2)
     d                                4    inz('<BR>')
     d                                1    inz(x'15')
     d                                6    inz('City: ')
     dcityo                                like(city)
     d                                1    inz(' ')
     dstateo                               like(state)
     d                                1    inz(' ')
     dzipo                                 like(zip)
     d                                1    inz('-')
     dzipp4o                               like(zipp4)
     d                                4    inz('<BR>')
     d                                1    inz(x'15')
     d                                8    inz('Phone: (')
     dphaco                                like(phac)
     d                                2    inz(') ')
     dphpreo                               like(phpre)
     d                                1    inz('-')
     dphnumo                               like(phnum)
     d                                5    inz(' Ext ')
     dphexto                               like(phext)
     d                                4    inz('<BR>')
     d                                1    inz(x'15')
     d                                7    inz('EMail: ')
     deaddro                               like(eaddr)
     d                                4    inz('<BR>')
     d                                1    inz(x'15')
      ****************************************************************
      * This DS contains the user output message. It should change  *
      * from program to program.                                    *
      ****************************************************************
     dOutMsg           DS
     d                                4    inz('<hr>')
     domsg                           40
     d                                4    inz('<BR>')
     d                                1    inz(x'15')
     dOVRCMD           DS
     d                               20    inz('OVRDBF FILE(CUSTPF) ')
     d                               21    inz('TOFILE(WEBDB/CUSTPF) ')
     d                               12    inz('MBR(*FIRST) ')
     d                               14    inz('OVRSCOPE(*JOB)')
     dDLTOVR           DS
     d                               20    inz('DLTOVR FILE(CUSTPF) ')
     d                                9    inz('LVL(*JOB)')
```

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----|
     ****************************************************************
     *                                                              *
     * Set length of input buffer. Call the API to get input       *
     * from Standard Input. The string will have the field names   *
     * and values.  This is used for the POST Method. The number   *
     * of bytes received is set in the field WSR-LEN-RECEIVED.     *
     *                                                              *
     * Refer to the TCP/IP Configuration manual for API Parms      *
     *                                                              *
     ****************************************************************
    c                   MOVE      *blanks       INBuff
    c                   EVAL      INBuffLn = %SIZE(INBuff)
    c                   CALLB     'QtmhRdStin'
    c                   Parm                    INBuff
    c                   Parm                    INBuffLn
    c                   Parm                    INActLn
    c                   Parm                    QUSEC
     ****************************************************************
     *                                                              *
     * Set length of Database Buffer then call the convert to      *
     * Database Function. This takes the string of variable names  *
     * and values and maps them to field names like a normal read  *
     * from a screen.                                               *
     *                                                              *
     * Refer to the TCP/IP Configuration manual for API Parms      *
     *                                                              *
     ****************************************************************
    c                   EVAL      DBRecLn = %SIZE(DBBuff)
    c                   CALLB     'QtmhCvtDb'
    c                   Parm                    DBFILE
    c                   Parm                    INBuff
    c                   Parm                    INActLn
    c                   Parm                    DBBuff
    c                   Parm                    DBRecLn
    c                   Parm                    DBCvtLn
    c                   Parm                    RspCode
    c                   Parm                    QUSEC
     ****************************************************************
     * Setup the output message based on the mail selection.       *
     * MOVE the input data to the output buffer to format it.      *
     ****************************************************************
    c                   SELECT
    c                   when      MAIL = 'E'
    c                   MOVEL (P) EMAILM        omsg
    c                   when      MAIL = 'S'
    c                   MOVEL (P) SMAILM        omsg
    c                   when      MAIL = 'P'
    c                   MOVEL (P) PMAILM        omsg
    c                   ENDSL
```

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----|
     c                   MOVEL     NAME          NAMEo
     c                   MOVEL     ADDR1         ADDR1o
     c                   MOVEL     ADDR2         ADDR2o
     c                   MOVEL     CITY          CITYo
     c                   MOVEL     STATE         STATEo
     c                   MOVEL     ZIP           ZIPo
     c                   MOVEL     ZIPP4         ZIPP4o
     c                   MOVEL     PHAC          PHACo
     c                   MOVEL     PHPRE         PHPREo
     c                   MOVEL     PHNUM         PHNUMo
     c                   MOVEL     PHEXT         PHEXTo
     c                   MOVEL     EADDR         EADDRo
      ****************************************************************
      * Write the data to the user. Write the standard header,       *
      *       the user data, and the standard trailer.               *
      ****************************************************************
     c                   EVAL      OutLn = %SIZE(StdHdr)
     c                   CALLB     'QtmhWrStout'
     c                   Parm                    StdHdr
     c                   Parm                    OutLn
     c                   Parm                    QUSEC
     c                   EVAL      OutLn = %SIZE(OutInf)
     c                   CALLB     'QtmhWrStout'
     c                   Parm                    OutInf
     c                   Parm                    OutLn
     c                   Parm                    QUSEC
     c                   EVAL      OutLn = %SIZE(OutMsg)
     c                   CALLB     'QtmhWrStout'
     c                   Parm                    OutMsg
     c                   Parm                    OutLn
     c                   Parm                    QUSEC
     c                   EVAL      OutLn = %SIZE(StdTrl)
     c                   CALLB     'QtmhWrStout'
     c                   Parm                    StdTrl
     c                   Parm                    OutLn
     c                   Parm                    QUSEC
      ****************************************************************
      *                                                              *
      * Point to the Database file in the WEBDB Library.             *
      * Open the database file                                       *
      * Write the data that the user entered to the Database File    *
      * Close the database file                                      *
      * Delete the Override                                          *
      *                                                              *
      ****************************************************************
     c                   EVAL      CmdLen = %SIZE(OVRCMD)
     c                   CALL      'QCMDEXC'
     c                   Parm                    OVRCMD
     c                   Parm                    CmdLen
     c                   OPEN      CUSTPF
     c                   WRITE     custrec
     c                   CLOSE     CUSTPF
     c                   EVAL      CmdLen = %SIZE(DLTOVR)
     c                   CALL      'QCMDEXC'
     c                   Parm                    DLTOVR
     c                   Parm                    CmdLen
     c                   seton                                        lr
```

A.2 ILE Cobol Sample CGI Program CUSTINFO

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----|
       PROCESS APOST NOMONOPRC
       IDENTIFICATION DIVISION.
      *
      * NOMONOPRC is required so that the called procedure names are
      * not changed to UPPERCASE.
      *
      *****************************************************************
      *
      *  Program:      Prompt user for mailing info and response
      *
      *  Language:     ILE Cobol
      *
      *  Description:  This program is used to process an HTML Form
      *                as input and add a record to a database
      *                It is a SAMPLE program
      *
      *  APIs Used:    QtmhRdStin  - Read from Standard Input. This
      *                              API is used to read the input
      *                              Data from the user.
      *
      *                QtmhCvtDb   - Convert to DB. This API takes
      *                              the Stdin input and converts it
      *                              into a group data structure that
      *                              can be processed by a cobol pgm.
      *                              The field names in the DDS must
      *                              match the names in the HTML Form
      *
      *                QtmhWrStout - Write to Standard Output. This
      *                              API writes the output to the user.
      *                              The Output Must be in the correct
      *                              HTML format and it MUST have a
      *                              MIME Header.
      *
      *                QCMDEXC     - Execute a CL Command from within a
      *                              High Level Langauge Program.
      *                              Used to Override the output
      *                              database file.
      *
      *      After the MODULE is created a CRTPGM command is required
      *      to build the program. You must include the Service
      *      Program QTMHCGI from the library QTCP
      *
      *****************************************************************

       PROGRAM-ID. CUSTINFOCB.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
         SOURCE-COMPUTER. IBM-AS400.
         OBJECT-COMPUTER. IBM-AS400.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT CUSTINF ASSIGN TO DATABASE-CUSTPF
                          ORGANIZATION IS sequential
                          ACCESS IS sequential
                          FILE STATUS IS custinf-stat.
       DATA DIVISION.
       FILE SECTION.
       FD  custinf.
       01  custinf-record.
           COPY DDS-custrec OF custpf.
```

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----|
      *
       WORKING-STORAGE SECTION.
       77  WS-CMD-LEN         PIC S9(10)v9(5) comp.
       77  CUSTINF-STAT       pic xx.
       77  WSR-STDIN-DATA     PIC X(32768).
       77  WSR-LEN-REQ        PIC S9(9) BINARY.
       77  WSR-LEN-CVTD       PIC S9(9) BINARY.
       77  WSR-RSP-CODE       PIC S9(9) BINARY.
       77  WSR-LEN-RECEIVED   PIC S9(9) BINARY.
       77  WSR-LEN-OUT        PIC S9(9) BINARY.
       77  WSR-LEN-BUFF       PIC S9(9) BINARY.
       01  ws-ovrdbf-cmd.
           05     pic x(53) value
             'OVRDBF FILE(CUSTPF) TOFILE(WEBDB/CUSTPF) MBR(*FIRST)'.
           05     pic x(15) value
             'OVRSCOPE(*JOB)'.
       01  ws-dltovr-cmd.
           05     pic x(53) value
             'DLTOVR FILE(CUSTPF) LVL(*JOB)'.
       01  WSR-DB-NAME.
           05  ws-db-name     pic x(10) value 'CUSTPF'.
           05  ws-lib-name    pic x(10) value 'WEBDB'.
      ****************************************************************
      * Buffer used to convert the string data into fields           *
      ****************************************************************
       01  CUST-BUFF.
           COPY DDS-CUSTREC of custpf.
       COPY QUSEC OF QSYSINC-QLBLSRC.
      ****************************************************************
      * This group contains a Standard startup needed for HTML       *
      *   This is used as output to the user. You should change      *
      *   the text between the <title> text </title> tags.           *
      *                                                              *
      *===>      Hex 15 (x'15') is the newline character and is      *
      *===>      REQUIRED between lines of output                    *
      ****************************************************************
       01  WS-STANDARD-HEADER.
           05      PIC X(23) VALUE 'Content-type: text/html'.
           05      PIC XX    VALUE X'1515'.
           05      pic x(6)  value '<HTML>'.
           05      pic x     value x'15'.
           05      pic x(6)  value '<HEAD>'.
           05      pic x     value x'15'.
           05      pic x(7)  value '<Title>'.
           05      pic x(24) value 'Thank you for your input'.
           05      pic x(8)  value '</Title>'.
           05      pic x     value x'15'.
           05      pic x(7)  value '</HEAD>'.
           05      pic x     value x'15'.
           05      pic x(6)  value '<Body>'.
           05      pic x     value x'15'.
           05      pic x(4)  value '<hr>'.
           05      pic x     value x'15'.
```

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----|
      ****************************************************************
      * This group contains a Standard trailer needed for HTML       *
      *   This is used as output to the user.                        *
      ****************************************************************
       01  WS-STANDARD-TRAILER.
           05      pic x(7)  value '</Body>'.
           05      pic x     value x'15'.
           05      pic x(7)  value '</html>'.
           05      pic x     value x'15'.
      ****************************************************************
      * This group contains the user output data. It should change   *
      * from program to program.                                     *
      ****************************************************************
       01  ws-output-info.
           05            pic x(6) value 'Name: '.
           05  name      like name of cust-buff.
           05            pic x(4) value '<br>'.
           05            pic x value x'15'.
           05            pic x(7) value 'addr1: '.
           05  addr1     like addr1 of cust-buff.
           05            pic x(4) value '<br>'.
           05            pic x value x'15'.
           05            pic x(7) value 'addr2: '.
           05  addr2     like addr2 of cust-buff.
           05            pic x(4) value '<br>'.
           05            pic x value x'15'.
           05            pic x(6) value 'City: '.
           05  city      like city of cust-buff.
           05            pic x value space.
           05  State     like state of cust-buff.
           05            pic x value space.
           05  zip       like zip  of cust-buff.
           05            pic x value space.
           05  zipp4     like zipp4 of cust-buff.
           05            pic x(4) value '<br>'.
           05            pic x value x'15'.
           05            pic x(7) value 'Phone: '.
           05            pic x value '('.
           05  phac      like phac of cust-buff.
           05            pic xx value ') '.
           05  phpre     like phpre of cust-buff.
           05            pic x value '-'.
           05  phnum     like phnum of cust-buff.
           05            pic x(6) value ' Ext '.
           05  phext     like phext of cust-buff.
           05            pic x(4) value '<br>'.
           05            pic x value x'15'.
           05            pic x(6) value 'Email '.
           05  eaddr     like eaddr of cust-buff.
           05            pic x(4) value '<br>'.
           05            pic x value x'15'.
       01  ws-output-message.
           05         pic x(4)  value '<hr>'.
           05  ws-message   pic x(40) value spaces.
           05            pic x(4) value '<br>'.
           05            pic x value x'15'.
      *
       LINKAGE SECTION.
```

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----|
      *
      * Beginning of mainline
      *
       PROCEDURE DIVISION.
       MAIN-LINE.
           MOVE LENGTH OF QUS-EC TO BYTES-PROVIDED OF QUS-EC.
      ***************************************************************
      *                                                             *
      * Set length of input buffer. Call the API to get input       *
      * from Standard Input. The string will have the field names   *
      * and values.  This is used for the POST Method. The number   *
      * of bytes received is set in the field WSR-LEN-RECEIVED.     *
      *                                                             *
      * Refer to the TCP/IP Configuration manual for API Parms      *
      *                                                             *
      ***************************************************************
           MOVE length of WSR-STDIN-DATA to wsr-len-req.
           CALL procedure 'QtmhRdStin' USING WSR-STDIN-DATA
                                     WSR-LEN-REQ
                                     WSR-LEN-RECEIVED
                                     QUS-EC.
      ***************************************************************
      *                                                             *
      * Set length of Database Buffer then call the convert to      *
      * Database Function. This takes the string of variable names  *
      * and values and maps them to field names like a normal read  *
      * from a screen.                                              *
      *                                                             *
      * Refer to the TCP/IP Configuration manual for API Parms      *
      *                                                             *
      ***************************************************************
           MOVE length of CUST-BUFF to WSR-LEN-BUFF.
           Call procedure 'QtmhCvtDb' using WSR-DB-NAME
                                      WSR-STDIN-DATA
                                      WSR-LEN-RECEIVED
                                      CUST-BUFF
                                      WSR-LEN-BUFF
                                      WSR-LEN-CVTD
                                      WSR-RSP-CODE
                                      QUS-EC.
      ***************************************************************
      * Write the data to the user. Write the standard header,      *
      *       the user data, and the standard trailer.              *
      ***************************************************************
           MOVE LENGTH OF WS-STANDARD-HEADER TO WSR-LEN-OUT.
           CALL procedure 'QtmhWrStout' USING WS-STANDARD-HEADER
                                     WSR-LEN-OUT
                                     QUS-EC.
           MOVE corr custrec of cust-buff to ws-output-info.
           MOVE LENGTH OF WS-output-info     TO WSR-LEN-OUT.
           CALL procedure 'QtmhWrStout' USING WS-output-info
                                     WSR-LEN-OUT
                                     QUS-EC.
```

```
|---+----1----+----2----+----3----+----4----+----5----+----6----+----7----|
          Evaluate mail of cust-buff
              when 'E'
                move 'We will contact you via E-mail' to ws-message
              when 'S'
                move 'We will contact you via Surface Mail' to
                     ws-message
              when 'P'
                move 'We will contact you via phone' to ws-message
          end-evaluate.
          MOVE LENGTH OF WS-output-message  TO WSR-LEN-OUT.
          CALL procedure 'QtmhWrStout' USING WS-output-message
                                   WSR-LEN-OUT
                                   QUS-EC.
          MOVE LENGTH OF WS-STANDARD-TRAILER TO WSR-LEN-OUT.
          CALL procedure 'QtmhWrStout' USING WS-STANDARD-TRAILER
                                   WSR-LEN-OUT
                                   QUS-EC.
     ****************************************************************
     *                                                              *
     * Point to the Database file in the WEBDB Library.             *
     * Open the database file                                       *
     * Write the data that the user entered to the Database File    *
     * Close the database file                                      *
     * Delete the Override                                          *
     *                                                              *
     ****************************************************************
          MOVE LENGTH OF ws-ovrdbf-cmd to ws-cmd-len.
          CALL 'QCMDEXC' using ws-ovrdbf-cmd ws-cmd-len.
          OPEN extend custinf.
          MOVE cust-buff to custinf-record.
          write custinf-record.
          Close custinf.
          MOVE LENGTH OF ws-dltovr-cmd to ws-cmd-len.
          call 'QCMDEXC' using ws-dltovr-cmd ws-cmd-len.
      EXIT-DONE.
          GOBACK.
```

A.3 HTML Input Form for CUSTINFO

```
<HTML>
<HEAD>
<title>Get Customer Feedback</title>
</head>
<body>
<FORM METHOD="post"
action="/cgi-bin/custinfo.pgm">
<h1>Customer Information/Satisfaction Survey</h1><br>
<hr>
Name  <INPUT type="text" name="name" SIZE="30"><br>
Address1  <INPUT type="text" name="addr1" SIZE="30"><br>
Address2  <INPUT type="text" name="addr2" SIZE="30"><br>
City  <INPUT type="text" name="city" SIZE="25"><br>
State  <INPUT type="text" name="state" SIZE="2">
Zip  <Input type="text" name="zip" size="5">
 -  <Input type="text" name="zipp4" size="4"><br>
Phone ( <input type="text" name="phac" size="3">)
<input type="text" name="phpre" size="3"> -
<input type="text" name="phnum" size="4">
Ext <input type="text" name="phext" size="4"><br>
E-Mail Address: <input type="text" name="eaddr" size="45><br>
I would like to correspond via
·<input type="radio" name="mail" Value="E" Checked>E-Mail“
·<input type="radio" name="mail" Value="S">Surface Mail“
·<input type="radio" name="mail" Value="P" >Phone Call“
<hr>
If you have used our service in the past please complete the following form.<br>
How satisfied are you with our service to date?<br>
<sl>
<li><input type="radio" name="sat" value="1"> Very Satisfied
<li><input type="radio" name="sat" value="2"> Satisfied
<li><input type="radio" name="sat" value="3"> Neither Satisfied nor Dissa
<li><input type="radio" name="sat" value="4"> Dissatisfied
<li><input type="radio" name="sat" value="5"> Very Dissatisfied
<li><input type="radio" name="sat" value="6" Checked> I have not used you
</sl>
<br><input type="submit" value="Enter"><br>
</FORM>
</body>
</html>
```

A.4 DDS for the CUSTPF file

```
-----A----------T-Name++++++RLen++TDpB------Functions++++++++++++++++++
                R CUSTREC
                  NAME          30
                  ADDR1         30
                  ADDR2         30
                  CITY          25
                  STATE          2
                  ZIP            5
                  ZIPP4          4
                  PHAC           3
                  PHPRE          3
                  PHNUM          4
                  PHEXT          4
                  EADDR         45
                  MAIL           1
                  SAT            1
```

Appendix B. HTML Gateway Code Examples

This appendix discusses sample codes you might want to try and use when developing AS/400 Internet applications using the HTML gateway.

B.1 HTML Field Overlap Examples

The following examples show where the HTML appears when you use the following coding:

Example 1

The HTML field has a starting column 2 before an output field. The HTML appears before the field:

```
A  01        FLD1A         20   0 15  7DFTVAL('Output Field')
A  01                                15  5HTML('<p>HTML code')
```

and results in:

```
HTML code
Output Field
```

Example 2

The HTML field has a starting column 1 before an output field, meaning that the HTML starts at the attribute byte of the output field. The HTML appears before the field:

```
A  01        FLD1A         20   0 15  6DFTVAL('Output Field')
A  01                                15  5HTML('<p>HTML code')
```

and results in:

```
HTML code
Output Field
```

Example 3

The HTML field has a starting column equal to an output field. The HTML appears before the first character of the field:

```
A  01        FLD1A         20   0 15  6DFTVAL('Output Field')
A  01                                15  6HTML('<p>HTML code')
```

and results in:

```
HTML code Output Field
```

Example 4

The HTML field has a starting column 1 past the starting column of an output field. The HTML appears after the first character of the output field:

```
A  01        FLD1A         20   0 15  6DFTVAL('Output Field')
A  01                                15  7HTML('<p>HTML code')
```

and results in:

```
OHTML codeutput Field
```

Example 5

The HTML field has a starting column 1 past the ending column of an output field, meaning that it overlaps the ending attribute. The HTML appears after the last character of the output field:

```
A  01        FLD1A         20   0 15  6DFTVAL('Output Field')
A  01                                15 27HTML('<p>HTML code')
```

and results in:

```
                        Output FieldHTML code
```

Example 6

The HTML field has a starting column 2 past the ending column of an output field. The HTML appears after the output field:

```
A  01        FLD1A         20   0 15  6DFTVAL('Output Field')
A  01                                15 28HTML('<p>HTML code')
```

and results in:

```
                        Output Field
                        HTML code
```

Notes:

1. Merging HTML and DDS fields does not occur for input fields. Merging occurs only for output fields.
2. HTML tags are inserted into the data stream if the device query indicates that the device is an AS/400 5250 Workstation Gateway virtual terminal. Otherwise, for normal displays, the HTML tags are ignored.

B.2 Logon Exit Code Examples for Workstation Gateway

This section shows some sample codes for Logon Exit function implementation.

```
/* Module Description ************************************************/
/*                                                                   */
/*                                                                   */
/* Source File Name: exitpgm.c                                       */
/*                                                                   */
/* Module Name: Workstation Gateway Server logon exit program.       */
/*                                                                   */
/* Service Program Name:  n/a                                        */
/*                                                                   */
/* Source File Description:                                          */
/*                                                                   */
/*   This module contains functions to allow a client browser to     */
/*   bypass an AS/400 sign-on panel and invoke an application.       */
/*                                                                   */
/* Reference:                                                        */
/*   TCP/IP Configuration and Reference SC41-3420 Appendix-I.        */
/*                                                                   */
/* End Module Description ********************************************/


#define _EXITPGM_C


/*********************************************************************/
/* All file scoped includes go here                                  */
/*********************************************************************/


#ifndef __stdio_h
#include <stdio.h>
#endif

#ifndef __string_h
#include <string.h>
#endif

#ifndef __stdlib_h
#include <stdlib.h>
#endif


/*********************************************************************/
/* All file scoped Constants go here                                 */
/*********************************************************************/


#define  SIZE    10
#define  FNAME   21              /* Qualified database file name size */
#define  FWIDTH  240             /* Width of one database file record */
#define  BLANK   ' '


/*********************************************************************/
/* All file scoped type declarations go here                         */
/*********************************************************************/
/* Structure for data passed to Server Logon exit program.           */
/*********************************************************************/
```

Figure 102 (Part 1 of 7). Sample Exit Program for HTML Gateway

```
typedef struct
{
  char *OperSpecInfo_p;         /* Operation Specific Info    (Input) */
  int  Lgth_OperSpecInfo;       /* Operation Spec Info length (Input) */
  char ClientIPaddr[15];        /* Client IP Addr.            (Input) */
  int  CCSID;                   /* CCSID of operation info    (Input) */
  char AllowOper[1];            /* Allow Operation '0'=N,'1'=Y(Output)*/
  char UserProfile[SIZE];       /* User Profile.             (Output) */
  char Password[SIZE];          /* Password.                 (Output) */
  char ProgramLib[SIZE];        /* Library of program to run.(Output) */
  char ProgramName[SIZE];       /* Program to invoke.        (Output) */
  char InitialMenu[SIZE];       /* Initial menu to invoke.   (Output) */
} QAPP0100_I_t;


/*********************************************************************/
/* All file scoped Macro invocations go here                         */
/*********************************************************************/

/*********************************************************************/
/*                                                                   */
/* Macro name: TRACE                                                 */
/*                                                                   */
/* Log test result entry to an output file for workstation gateway.  */
/*                                                                   */
/*   Arguments:            x - Text string of application-specific   */
/*                             trace data from the calling program.  */
/*                         y - Test results output file pointer.     */
/*                                                                   */
/*********************************************************************/


#define TRACE(x, y)                                                   \
 {                                                                    \
 memset(file_buff, BLANK, sizeof(file_buff));                         \
 sprintf(file_buff, "%s%c", x, '\0');                                 \
 fwrite(file_buff, FWIDTH, 1, y);                                     \
 }


/*********************************************************************/
/* All internal function prototypes go here                          */
/*********************************************************************/


static void  exitpgm
  (char *,int,char *,int,char *,char *,char *,char *,char *,char *);


static void WriteParms
  (char *,int,char *,int,char *,char *,char *,char *,char *,char *);


/*********************************************************************/
/* All file scoped variable declarations go here                     */
/*********************************************************************/


char file_name[FNAME];        /* Output results database file name    */
FILE *test_file;              /* Output results database file pointer */
char file_buff[FWIDTH];       /* Output results file buffer           */
```

Figure 102 (Part 2 of 7). Sample Exit Program for HTML Gateway

```
/* Function Specification ********************************************/
/*                                                                   */
/* Function Name: Main                                               */
/*                                                                   */
/* Descriptive Name: Application Logon exit program sample program.  */
/*                                                                   */
/*   This test exit program provides control over signon panels via  */
/*   the WSG server in the V3R2 release.                             */
/*                                                                   */
/* Notes: For V3R2 the "argv[]" parameters are "char *" by definition.*/
/*         Reference integers as "*(int(argv[1]))", for example.     */
/*         Consider the method for passing them back to the caller.  */
/*                                                                   */
/*   Dependencies:                                                   */
/*      WSG Application Logon exit point QIBM_QTMT_WSG format QAPPO100*/
/*       was registered during WSG V3R2 installation.                */
/*                                                                   */
/*   Restrictions:                                                   */
/*                                                                   */
/*       None                                                        */
/*                                                                   */
/*   Messages:                                                       */
/*                                                                   */
/*       None                                                        */
/*                                                                   */
/*   Side Effects:                                                   */
/*                                                                   */
/*       None                                                        */
/*                                                                   */
/*   Functions/Macros called:                                        */
/*                                                                   */
/*       TRACE - Write one data record to test results file.         */
/*                                                                   */
/* Input:                                                            */
/*   chat * argv[1]        - Operation specific information          */
/*   int    argv[2]        - Length of operation specific information */
/*   char * argv[3]        - IP address of the remote host system.   */
/*   int    argv[4]        - CCSID of the operation specific info    */
/*   char * argv[5]        - Allow operation '0'=No, '1'=Yes(output) */
/*   char * argv[6]        - User profile to be used (output)        */
/*   char * argv[7]        - Password to be used (output)            */
/*   char * argv[8]        - Program library to be used (output)     */
/*   char * argv[9]        - Program name to be used (output)        */
/*   char * argv[10]       - Menu panel to be used (output)          */
/*                                                                   */
/* Exit Normal: Return AllowOper value to server application.        */
/*                                                                   */
/* Exit Error: None                                                  */
/*                                                                   */
/* End Function Specification ****************************************/


void main(int argc, char *argv[])
{
     exitpgm(argv[1],
             *((int *)(argv[2])),
             argv[3],
             *((int *)(argv[4])),
             argv[5],
             argv[6],
             argv[7],
             argv[8],
             argv[9],
             argv[10]);
  return;
}            /* End main */
```

Figure 102 (Part 3 of 7). Sample Exit Program for HTML Gateway

```
/* Function Specification ********************************************/
/*                                                                   */
/* Function Name: exitpgm                                            */
/*                                                                   */
/* Descriptive Name: Workstation Gateway Server (WSG) Logon exit.    */
/*                                                                   */
/*   This test exit program provides control over user authentication */
/*   to a workstation gateway in the V3R2 release.                   */
/*                                                                   */
/* Notes:                                                            */
/*                                                                   */
/*   Dependencies:                                                   */
/*                                                                   */
/*       Workstation Gateway Logon exit point QIBM_QTMT_WSG was      */
/*        registered during WEB V3R2 installation.                   */
/*                                                                   */
/*   Restrictions:                                                   */
/*                                                                   */
/*       None                                                        */
/*                                                                   */
/*   Messages:                                                       */
/*                                                                   */
/*       None                                                        */
/*                                                                   */
/*   Side Effects:                                                   */
/*                                                                   */
/*       None                                                        */
/*                                                                   */
/*   Functions/Macros called:                                        */
/*                                                                   */
/*       None                                                        */
/*                                                                   */
/* Input:                                                            */
/*  char * OperSpecInfo_p     - Operation Specific Information.      */
/*  int    Lgth_OperSpecInfo  - Length (in bytes) of Operation       */
/*                              Specific Information.                */
/*  char   ClientIPaddr       - Client Internet Protocol Address.    */
/*  int    CCSID              - CCSID of operation info              */
/*                                                                   */
/* Output:                                                           */
/*  char * AllowOper          - Allow Operation ('0' = Reject),      */
/*                                              ('1' = Accept).      */
/*  char * UserProfile        - User Profile to be used for sign on. */
/*  char * Password           - Password to be used for sign on.     */
/*  char * ProgramLib         - Library of program to invoke.        */
/*  char * ProgramName        - Name of program to invoke.           */
/*  char * InitialMenu        - Initial menu to invoke.              */
/*                                                                   */
/* Exit Normal: (See OUTPUT)                                         */
/*                                                                   */
/* Exit Error:  None                                                 */
/*                                                                   */
/* End Function Specification ****************************************/


static void exitpgm(char *OperSpecInfo_p,         /* Entry point */
                    int Lgth_OperSpecInfo,
                    char ClientIPaddr[15],
                    int  CCSID,
                    char AllowOper[1],
                    char UserProfile[SIZE],
                    char Password[SIZE],
                    char ProgramLib[SIZE],
                    char ProgramName[SIZE],
                    char InitialMenu[SIZE])

{       /* exitpgm start from here */
```

Figure 102 (Part 4 of 7). Sample Exit Program for HTML Gateway

```
  /*********************************************************************/
  /*                                                                   */
  /* You can design your own logical flow here to check if user is     */
  /* authorized to bypass sign on screen. Following are some examples.*/
  /*                                                                   */
  /* (1) Validate Client IP address.                                   */
  /* (2) Parse the "OperSpecInfo_p" input string.                      */
  /*                                                                   */
  /* Then                                                              */
  /* Return AllowOper of '0' - Reject this clients request.            */
  /* Return AllowOper of '1' - Accept this clients request.            */
  /* Set related return values for client browser.                     */
  /*                                                                   */
  /*********************************************************************/
  /*                                                                   */
  /******  Be sure to modify below for your environment  **************/
  /*                                                                   */
  /* Following is an example by checking client IP address.            */
  /*                                                                   */
  /* Accept the following IP address for AS/400 sign-on bypass.        */
  /*                                                                   */
  /*    9.5.100.110 /9.5.100.111 / 9.5.100.112                         */
  /*                                                                   */
  /* Then assign the user id, password and program name, program       */
  /* library.                                                          */
  /*                                                                   */
  /*********************************************************************/


   /* change here for your logical */

   char Accept_IP[] = "9.5.100.110    9.5.100.111    9.5.100.112    ";

   if (strstr(Accept_IP, ClientIPaddr)) {
       memcpy(UserProfile, "JOENORMAL ", SIZE);
       memcpy(Password,    "JOENORMAL ", SIZE);
       memcpy(ProgramLib,  "ITSOIC400 ", SIZE);
       memcpy(ProgramName, "SHTMLR    ", SIZE);
       memcpy(InitialMenu, "          ", SIZE);
       memcpy(AllowOper,   "1",          1);
    }  else {
           memcpy(AllowOper,   "0",          1);
    }


  /*********************************************************************/
  /* Call function to write input parameters as received to file.      */
  /*********************************************************************/


  WriteParms(OperSpecInfo_p,
             Lgth_OperSpecInfo,
             ClientIPaddr,
             CCSID,
             AllowOper,
             UserProfile,
             Password,
             ProgramLib,
             ProgramName,
             InitialMenu);

  return;                  /* End program exitpgm.c               */

}    /* end of exitpgm */
```

Figure 102 (Part 5 of 7). Sample Exit Program for HTML Gateway

```
/* Function Specification ********************************************/
/*                                                                   */
/* Function Name: WriteParms                                         */
/*                                                                   */
/* Descriptive Name: Write the input parameters as received to file.  */
/*                                                                   */
/*                                                                   */
/* Notes:                                                            */
/*                                                                   */
/*   Dependencies:                                                   */
/*      None                                                         */
/*                                                                   */
/*   Restrictions:                                                   */
/*       None                                                        */
/*                                                                   */
/*   Messages:                                                       */
/*       None                                                        */
/*                                                                   */
/*   Side Effects:                                                   */
/*       None                                                        */
/*                                                                   */
/*   Functions/Macros called:                                        */
/*                                                                   */
/*       system                                                      */
/*       sprintf                                                     */
/*       TRACE - Write one data record to test results file.         */
/*                                                                   */
/* Input:                                                            */
/*  char * OperSpecInfo_p     - Operation Specific Information.      */
/*  int    Lgth_OperSpecInfo  - Length (in bytes) of Operation       */
/*                              Specific Information.                */
/*  char * ClientIPaddr       - Client Internet Protocol Address.    */
/*  int    CCSID              - CCSID of operation information       */
/*  char * AllowOper          - Allow Operation ('0' = Reject),      */
/*                                              ('1' = Accept).      */
/*  char * UserProfile        - User Profile to be used for sign on. */
/*  char * Password           - Password to be used for sign on.     */
/*  char * ProgramLib         - Library of program to invoke.        */
/*  char * ProgramName        - Name of program to invoke.           */
/*  char * InitialMenu        - Initial menu to invoke.              */
/*                                                                   */
/* Output:                                                           */
/*   A single, serial data record written to the test results file.  */
/*                                                                   */
/* Exit Normal: (See OUTPUT)                                         */
/*                                                                   */
/* Exit Error:  None.                                                */
/*                                                                   */
/* End Function Specification ****************************************/


static void WriteParms(char *OperSpecInfo_p,         /* Entry point */
                       int Lgth_OperSpecInfo,
                       char ClientIPaddr[15],
                       int  CCSID,
                       char AllowOper[1],
                       char UserProfile[SIZE],
                       char Password[SIZE],
                       char ProgramLib[SIZE],
                       char ProgramName[SIZE],
                       char InitialMenu[SIZE])

{
```

Figure 102 (Part 6 of 7). Sample Exit Program for HTML Gateway

```
  /*********************************************************************/
  /* Local Variables                                                   */
  /*********************************************************************/

  char loc_buff[FWIDTH];                        /* Local TRACE buffer */

  /*********************************************************************/
  /* Create a database file to write the logon attempt results to.     */
  /*********************************************************************/

  system("QSYS/CRTPF FILE(QUSRSYS/EXITPGM) RCDLEN(240) SIZE(*NOMAX)");

  /*********************************************************************/
  /* Build up the qualified database file name.                        */
  /*********************************************************************/

  sprintf(file_name, "%s", "QUSRSYS/EXITPGM");

  /*********************************************************************/
  /* Open database file for permanent log of test results.             */
  /*  mode = a(ppend) b(inary) 128-byte fixed-length records           */
  /*********************************************************************/

  test_file = fopen(file_name,"ab, lrecl=240, type=record, recfm=f");

  /*********************************************************************/
  /* Build up the input record for the test results database file      */
  /* Write parameters that were passed in to the output file:          */
  /*********************************************************************/

  sprintf(loc_buff,
          "IP: %s CCSID: %d Allow: '%c' Profile: %s Password: %s "
          "Library: %s Program: %s Menu: %s  OperLen: %d Oper: >%s<",
          ClientIPaddr,
          CCSID,
          *AllowOper,
          UserProfile,
          Password,
          ProgramLib,
          ProgramName,
          InitialMenu,
          Lgth_OperSpecInfo,
          OperSpecInfo_p);
  TRACE(loc_buff, test_file);
  fclose(test_file);
  return;
}

#undef _EXITPGM_C
```

Figure 102 (Part 7 of 7). Sample Exit Program for HTML Gateway

A live version of the following code can be seen in action on the AS/400 Web Technology page by going to:

http:/www.as400.ibm.com/internetdemo

Then go to the 5250 section, Ship Status (the third icon down in the lefthand frame).

```
*****************************************************************
* PROGRAM NAME        : ORDSTAT
* PROGRAM DESCRIPTION : This program will request an order number
*                       and if it is in file, will then display
 *                       order status.  If it is not, will
 *                       display an error message.
 *
 * COMPILATION         : To compile this program, use the
 *                       standard CRTRPGPGM command.
 *
 *
 * PROGRAMMER          :
 * DATE                : 07/15/96
 *****************************************************************
FCUSMSTP IF  E           K        DISK
FCUSFMT  CF  E                    WORKSTN
C           *IN15     DOWEQ'0'
 *
 * Overlay order not found (CUSPMT) if error (IN99)
C           *IN99     CASEQ'0'       HEADNG
C                     END
C                     EXFMTCUSPMT
 *
 * If not end, and valid order number (444), display status
C           *IN15     IFEQ '0'
C           ORD       CHAINCUSREC               99
C           *IN99     IFEQ '0'
C                     EXFMTCUSFLDS
C                     ENDIF
C                     ENDIF
C                     ENDDO
C                     MOVE '1'       *INLR
 *  Subroutine to display title and PF key line
C           HEADNG    BEGSR
C                     WRITECUSFTG
C                     WRITECUSHDG
C                     ENDSR
```

```
A*************
A* FILE NAME          : CUSMSTP
A* FILE DESCRIPTION   : Customer Master File
A* DATE               : 07/15/96
A*********
A*
A          R CUSREC
A*                                ORD must be unique
A            ORD            5          TEXT('Order number')
A                                       COLHDG('Order' 'Number')
A            NAME          20          TEXT('Customer Name')
A                                       COLHDG('Customer' 'Name')
A            STAT          10          TEXT('Status ')
A                                       COLHDG('Status')
A          K ORD
```

```
A*%%TS  SD  19960716  130224  MARIAN      REL-V3R1M0  5763-PW1
A*************************************************************
A*  FILE NAME: CUSFMT
A*  DESCRIPTION: DISPLAY FILE FOR ENTERING ORDER STATUS
A*************************************************************
A*%%EC
A                                     DSPSIZ(24 80 *DS3)
A                                     CHGINPDFT(CS)
A                                     CA03(15 'End')
A                                     PRINT
A                                     INDARA
A          R CUSHDG
A                                     OVERLAY
A                                   1 27'Check Order Status        '
A                                     DSPATR(HI)
A          R CUSFTG
A                                  23  2'F3=Exit'
A                                     COLOR(BLU)
A          R CUSPMT
A                                     OVERLAY
A                                   3  2'Type selection, press Enter.'
A                                     COLOR(BLU)
A            ORD            5A  I  5 39DSPATR(PC)
A                                     DSPATR(HI)
A  99                                 ERRMSG('No orders found,try 0-9' 99)
A                                     COLOR(WHT)
A                                   5  2'Order number . . . . . . . . . . .'
A          R CUSFLDS
A                                     OVERLAY
A                                   3  2'Customer Name. . . . . . . . . . :'
A            NAME          20A  O  3 37
A                                   5  2'Order Number . . . . . . . . . . :'
A            ORD            5A  O  5 37
A                                   7  2'Status . . . . . . . . . . . . . :'
A            STAT          10A  O  7 37
A                                   7 48DATE EDTCDE(Y)
A                                  21  2'Press Enter to continue.'
A                                     COLOR(BLU)
A
A*************************************************************
A*
A*  You will have to edit the next statement to your system name.
A*  (F11 to scroll right)
A*  Make sure you right justify it (leave no blanks in URL).
A*                For example, change --->"http://s1046401.sysland.ibm.com-
A*                                 to this ---->          "http://9.5.10.126-
A*
A*************************************************************
A                                   23  2HTML('<IMG SRC=
A                                         "http://s1046401.sysland.ibm.com-
A                                        /QOpenSys/ORDER/IMG/ordstat.gif"   -
A                                         align="right"    > ')
A
```

Appendix C. Special Notices

This publication is intended to help those people who are responsible for the task of recommending and implementing an AS/400 network computing solution. The information in this publication is not intended as the specification of any programming interfaces that are provided by the products mentioned in this book. See the PUBLICATIONS section of the IBM Programming Announcement for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 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 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.

The following document contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples contain the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

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:

AnyNet
APPN
C/400
COBOL/400
IBM
OS/400
RS/6000
VisualAge
WebExplorer
Application System/400
AS/400
Client Access/400
DB2/400
Integrated Language Environment
RPG/400
SQL/400
WebConnection

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 to Get ITSO Redbooks” on page 279.

- *Using the Information Super Highway*, SG24-2499
- *Cool Title About the AS/400 and Internet*, SG24-4815

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
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
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
Personal Systems Redbooks Collection	SBOF-7250	SK2T-8042

D.3 Other Publications

These publications are also relevant as further information sources:

- *TCP/IP Configuration & Reference V3R7*, (SC41-3420)
- *TCP/IP Fastpath Setup Version 3*, (SC41-3430)
- *AS/400 Sockets Programming V3R7*, (SC41-4422)
- *AS/400 System API Reference V3R7*, (SC41-4801)
- *AS/400 Integrated File System Introduction V3R7*, (SC41-4711)
- *AS/400 NLS Planning Guide*, (GC41-9877)
- *ILE C/400 Programming Guide V3R7*, (SC09-2069)
- *ILE C/400 Programming Reference V3R7*, (SC09-2070)
- *ILE COBOL/400 Programming Guide V3R7*, (SC09-2072)
- *ILE COBOL/400 Programming Reference V3R7*, (SC09-2073)
- *ILE RPG/400 Programming Guide V3R7*, (SC09-2074)
- *ILE RPG/400 Programming Reference V3R7*, (SC09-2077)
- *AS/400 REXX/400 Programmers Guide*, (SC24-5553)
- *AS/400 REXX/400 Reference*, (SC24-5552)

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 URL 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:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type GOPHER.WTSCPOK.ITSO.IBM.COM
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  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:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Home Page on the World Wide Web**

  http://w3.itso.ibm.com/redbooks

- **IBM Direct Publications Catalog on the World Wide Web**

  http://www.elink.ibmlink.ibm.com/pbl/pbl

  IBM employees may obtain LIST3820s of redbooks from this page.

- **REDBOOKS category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

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** (Do not send credit card information over the Internet) — 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	1-403-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1)001-408-256-5422 (Outside USA)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** - send note to softwareshop@vnet.ibm.com
- **On the World Wide Web**

Redbooks Home Page	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).

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.

DO NOT SEND CREDIT CARD INFORMATION OVER THE INTERNET.

Index

Special Characters

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

ITSO Redbook Evaluation

Unleashing AS/400 Applications on the Internet
SG24-4935-00

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.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redeval@vnet.ibm.com

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!)**

_____

_____

_____

_____

_____

Printed in U.S.A.