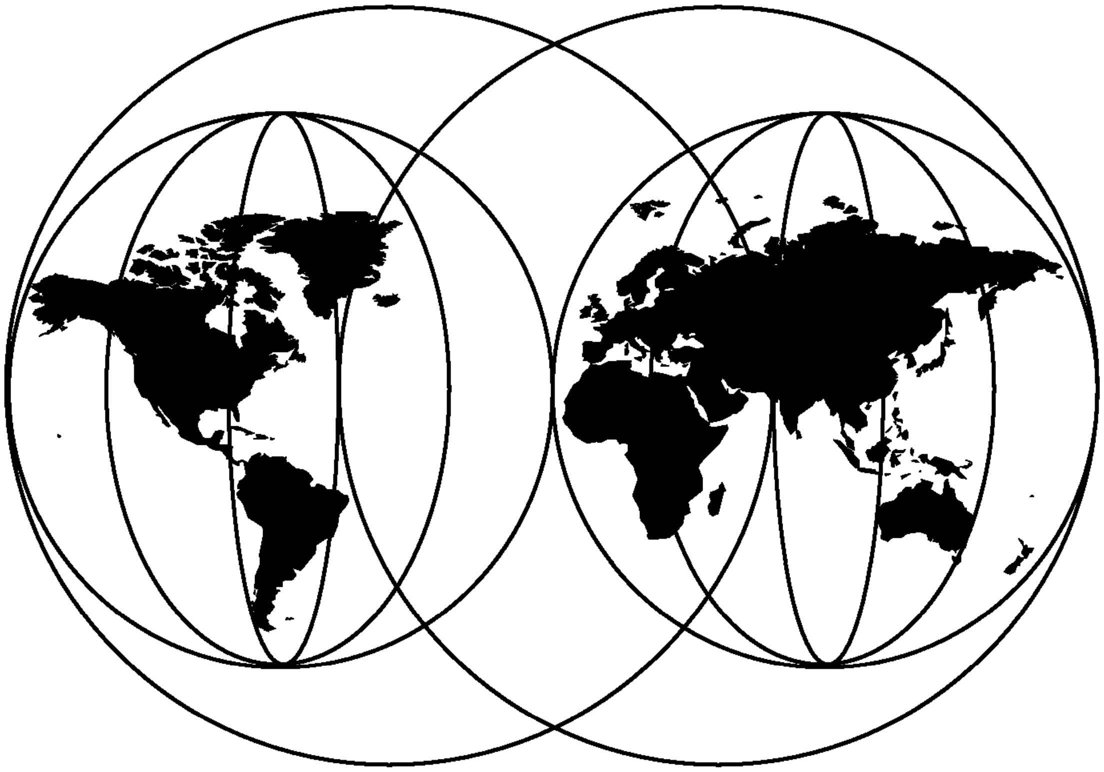
Web-Enabling VM Resources

Erich Amrehn, Stephane Faure, Nicholas J. Gimbrone
Bruce J. Hayden, Stephen Record, Paul Sienicki



International Technical Support Organization

http://www.redbooks.ibm.com

This book was printed at 240 dpi (dots per inch). The final production redbook with the RED cover will be printed at 1200 dpi and will provide superior graphics resolution. Please see "How to Get ITSO Redbooks" at the back of this book for ordering instructions.

SG24-5347-00

International Technical Support Organization

SG24-5347-00

Web-Enabling VM Resources

February 1999

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix D, “Special Notices” on page 241.

(February 1999)

This edition applies to VM/ESA Version 2 Release 3.0, EnterpriseWeb/VM, EnterpriseWeb Secure/VM, VM:Webgateway Release 2.2 and Webshare.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HYJ Mail Station P099
522 South Road
Poughkeepsie, New York 12601-5400

Contents

Figures

Tables

Preface

This redbook describes programming techniques that can be used with VM Web servers on VM/ESA. Samples included in this book and on the accompanying CD demonstrate how to connect Web clients (browsers) to data and applications on a VM platform.

This redbook was written for the systems programming and development staff engaged in the design and implementation of applications on the VM platform.

Familiarity with Web terminology, VM/CMS, CMS Pipelines and REXX is assumed. A Web server operating on VM is assumed to be available. For Web terminology, static page development, and the Web server install instructions, consult the companion redbook *Web Server Solutions for VM/ESA*, SG24-4874.

The following VM Web servers are discussed:

- Webshare
- EnterpriseWeb Secure/VM
- EnterpriseWeb/VM
- VM:Webgateway Release 2.2

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 Poughkeepsie Center.

The project was planned and managed by:

Erich Amrehn, IBM International Technical Support Organization, Poughkeepsie Center. He is a certified Senior IT Specialist at the International Technical Support Organization, Poughkeepsie Center. Before joining the ITSO, he worked as technical consultant to the IBM System/390 division for e-commerce on S/390 in Europe, the Middle East and Africa. He also has 13 years of VM experience in various technical positions in Germany and other areas in Europe.

The authors of this document are:

Stephane Faure is an Advisory IT Specialist in Systems Management with IBM Global Services in France. He has nine years of experience in the computing field. He is part of the VM IBM French Support Center, where he is the webmaster of a VM intranet site. He holds a degree in Computing from Gustave Eiffel College, Bordeaux. His areas of expertise include VM/ESA, TCP/IP, AFP, REXX programming, CMS Pipelines and DB2.

Nicholas J. Gimbrone is a Senior Software Engineer for Sterling Software, Inc. in the United States, where he has worked for six years. He has 21 years of experience in the computing field. He was the lead engineer in the creation of VM:Webgateway Release 1.0 and has contributed to each of its seven releases to date, including serving as the architectural lead for VM:Webgateway Release 2.2. Prior to working for Sterling Software, Inc. he worked for Cornell University for 15 years after receiving his degree there. He is active with SHARE, where he has served as member of the VM Cluster′s VM Technical Steering Committee for the past eight years. His areas of expertise include VM/ESA, TCP/IP, application performance, and REXX programming. He has written numerous papers and articles on topics such as VM and the POSIX environment, VM and TCP/IP performance, and running various flavors of UNIX under VM.

Bruce J. Hayden is an IT Specialist, Senior System Management Integrator with IBM Global Services in the United States. He has 16 years of experience in the computing field, all with IBM. He holds a bachelor′s degree in Computer Science from the University of Missouri. He is currently part of the World Wide VM Platform organization that supplies and supports the VM operating systems and products used on IBM systems. He is part of a worldwide board that defined the standard architecture and procedures used by IBM to install, maintain, and use VM products and applications. His areas of expertise include VM/ESA, REXX and CMS Pipelines programming, and Web programming.

Paul K. Sienicki is a Senior Infrastructure Specialist for Electronic Data Systems (EDS) in the United States. He has 21 years of experience in the computing field. He has 16 years of experience in software implementation. He holds a bachelor′s degree in Computer Science from Allentown College of St. Francis DeSales. His areas of expertise include VM/ESA, TCP/IP, REXX.

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

Alan Altmark
IBM Endicott

Pam Bradford
Sterling Software, Inc.

Doug Breneman
IBM Endicott

Richard Lewis
IBM Gaithersburg

Mick Lickmann
IBM Hursley

Michael Ludé
Beyond Software Inc.

Ross Patterson
Sterling Software, Inc.

Stephen Record
IBM Böblingen/Endicott

Gary Richtmeyer
IBM Global Services

Romney White
IBM Endicott

Brian Wade
IBM Endicott

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

Chapter 1. Introduction

What does Web-Enabling VM Resources mean? It means developing programs that interface and access information on VM or that are accessed through the VM system. The purpose of these programs is to create output in a format viewable by a Web browser.

What is a VM resource? It could be any application or data located on a VM minidisk, Shared File System (SFS) or Byte File System (BFS). This could include databases like DB2/VM or other database products. Since VM supports guest operating systems, a VM resource could include guest VSE or OS/390 (MVS) systems as well as associated products like CICS and IMS.

The development and debugging process on VM is quick and easy. The primary tools for Web development are the REXX interpreter and CMS Pipelines. CMS Pipelines provides a powerful data transformation and manipulation capability. Pipelines is a great performer, too. This powerful data transformation environment, when combined with VM′s excellent connectivity to guest systems, other systems, and platforms, makes VM an excellent choice for Web-enabling resources from many environments.

Therefore, VM′s strengths allow a short development cycle for gathering data from many sources, the massaging of that data into information, and subsequent serving of that information to be displayed on a Web browser.

1.1 How Difficult Is Web Programming?

The skill level needed to code Web programs on VM is low. If you know how to code a REXX or CMS Pipelines (PIPE) program, you know the languages used, and you have the basic skill level necessary to code Web programs on VM. Web programs are known as Common Gateway Interface (CGI) programs. The shift from VM development to VM CGI programming is easy.

Also, if you have worked with Script, General Markup Language (GML) or BookMaster, the concept of a markup language (like HTML) should be familiar to you.

1.2 Web Programming on VM

One of VM′s strengths is the interactive environment. REXX is an interpreter that is easy to learn, use and debug. CMS Pipelines is a powerful data manipulation tool that has good execution performance.

Another of VM′s strengths is and always has been the fact that the Service Virtual Machine (SVM) is its own entity and can be tailored and replicated without effect on other VM users or SVMs. Establishing several development environments is relatively easy. In our test we had three different Web server products handling different ports in one TCP/IP environment. The same approach could be taken with one Web server product to provide different development environments on one or more TCP/IP environments. The multiple Web server environment capability is especially useful, as CGI program development bugs can disrupt Web server operation. With VM/ESA there is no need to have multiple hardware platforms only to provide test and scalability facilities.

The stability of VM/ESA is unmatched, as shown by supporting material covered later in this chapter. The need to recycle or an unexpected outage is much less than for an average server farm.

In addition to the reasons outlined in the "VM/ESA As a Web Server" section of *Web Server Solutions for VM/ESA*, SG24-4874, the strengths of VM and the S/390 platform marry well with the emerging needs of the Web and e-business.

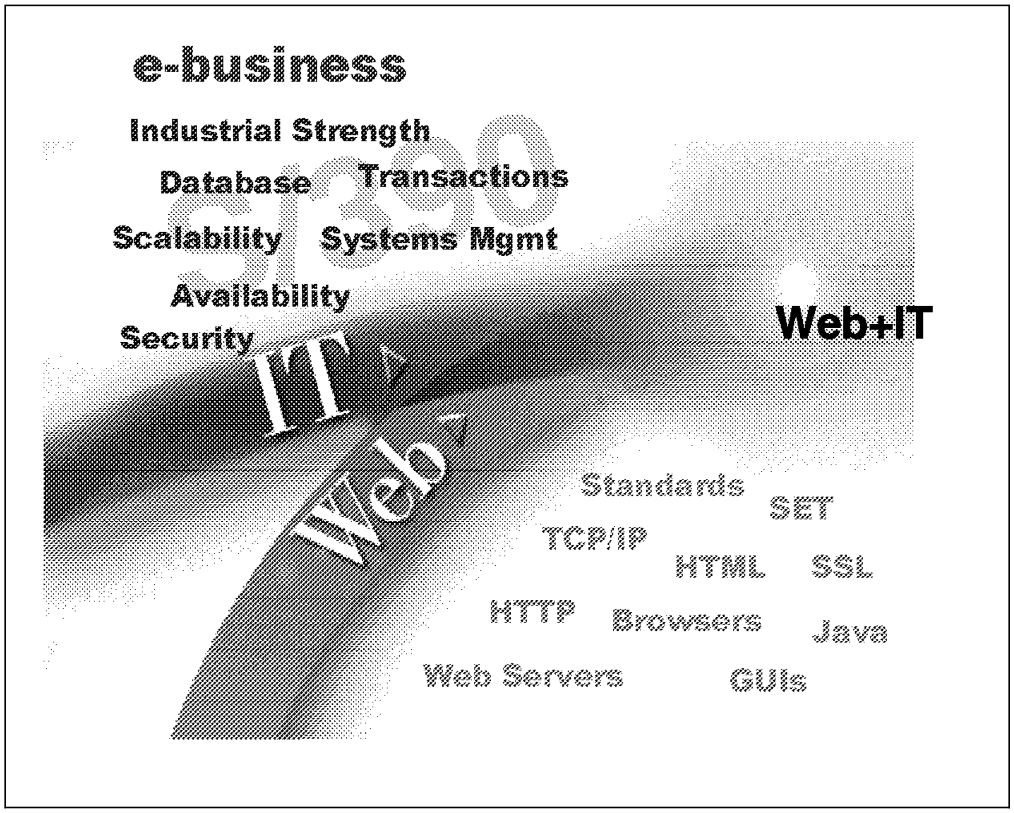


Figure 1. The Web and IT

S/390 technology boasts the industry′s highest availability possible today: 99.99 percent, which translates to less than 10 minutes of unplanned downtime per year. Since e-business is a 7x24 proposition, companies must ensure that their systems are up and running at all times.

As noted by Mike Kahn, Chairman, Clipper Group, "The S/390 is better than UNIX or Windows NT when it comes to running 24 hours by 7 days with very little downtime and with the ability to scale."

By using existing S/390 servers, businesses can avoid the added expense of a separate platform to take advantage of the growing electronic business marketplace. They also avoid expensive replication of storage.

According to Cost of Scalability, a 1996 comparative study by International Technology Group, per user costs for S/390 were between 1.2 and 3.7 times lower, depending on industry and application, than for UNIX servers. The report points out that in S/390 installations, per user costs declined in direct proportion to system size. In contrast, "Economies of scale were consistently weaker in

UNIX server installations, due to duplication of servers, software and support staff.”

That is why International Technology Group reports, in a study called *Strategies for Scalability*, that “If S/390 is not on your information systems agenda for electronic commerce, it should be placed there now. Your business will, quite literally, soon be on the line.”

1.3 VM for e-business

The Internet makes it possible to conduct business in a whole new way: e-business. e-business is a vehicle for transformation, in which companies take what they have and transform their business processes, providing value to their customers and partners in new ways. See Figure 2.

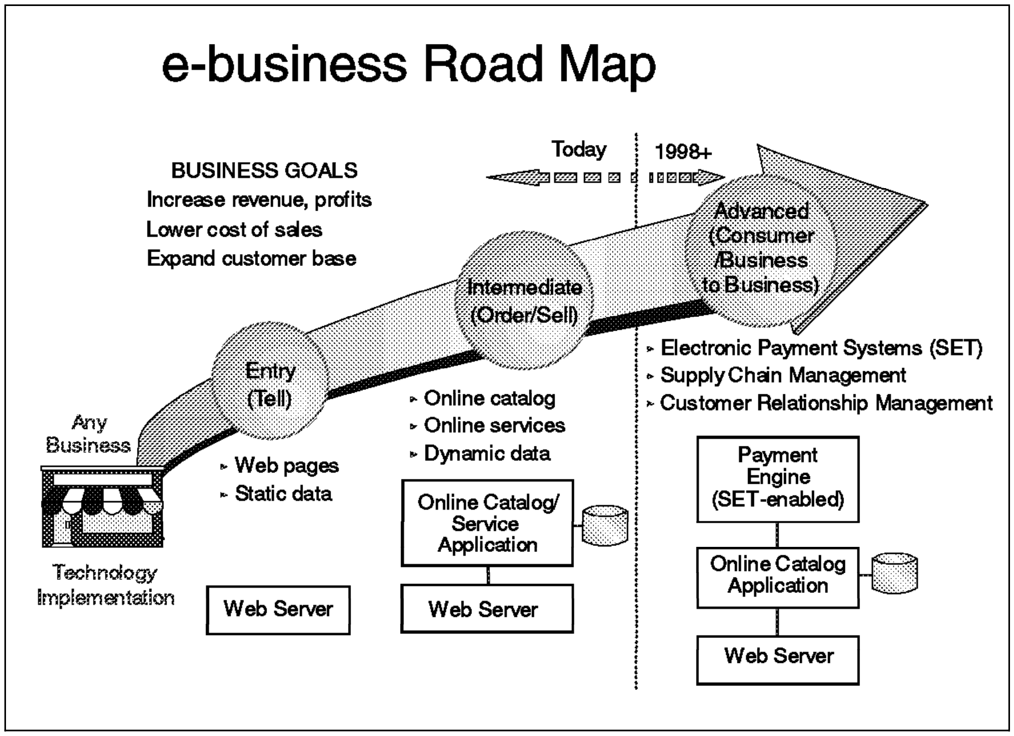

e-business Road Map
BUSINESS GOALS
Increase revenue, profits
Lower cost of sales
Expand customer base
Today
1998+
Any Business
Technology Implementation
Entry (Tell)
Intermediate (Order/Sell)
Advanced (Consumer /Business to Business)
Web pages
Static data
Online catalog
Online services
Dynamic data
Electronic Payment Systems (SET)
Supply Chain Management
Customer Relationship Management
Web Server
Online Catalog/ Service Application
Web Server
Payment Engine (SET-enabled)
Online Catalog Application
Web Server

Figure 2. e-business Road Map

1.4 Levels

Most IT organizations pass through several phases in transforming themselves to support e-business. The first important phase is often to give employees access to the Internet or to static HTML pages on the intranet and or Internet. On the intranet side, documents such as a company handbook are often selected. With a home page on the World Wide Web, employees can find distributors, addresses and telephone numbers, and send e-mail requests or feedback.

The second or intermediate phase in becoming an e-business involves connecting browsers to applications. CGI programs provide a method for linking a Web server to existing data and applications, that reside on S/390.

Programmers can write CGI programs to access to virtually any kind of data or program.

By obtaining access through a Web server to dynamic data and applications, the Web-enabled data or applications have a much higher business value, not only to the company, but also to the end user.

A company reaches the third or advanced phase of e-business when a very large part of its daily business is conducted using Web-based and related technologies. In this case, there will often be a requirement for confidential data to be transmitted securely over public networks, intranets, or extranets. Electronic business transactions may be conducted with both clients and suppliers.

1.5 The Network Computing Environment

Network computing evolved from the client/server computing model, which combines the ease-of-use features, such as a graphical user interface (GUI) and the personal productivity applications of the personal computer, with critical server-based business applications and data. The goals of client/server were greater productivity at a reduced cost of computing. However, over time, application development efforts began to target more application functions on client workstations. Lack of standards, in turn, complicated the task of managing and connecting those workstations. As a result, as consultant studies have shown, client/server computing can be very expensive.

Network computing, by building on open Internet standards, overcomes the mix of different, often incompatible client operating systems and GUIs by relying on the universal client, the Web browser. Building on existing applications and data cuts costs and time to market. Once these critical business assets become available over the Web, security becomes essential. Three elements are involved here: network security, system security, and transaction security.

Network security can be provided in a number of ways: packet filters, a proxy server, a SOCKS server, Domain Name Services (DNS), and encrypted IP tunnels. A VM server can connect to the public Web through one of these methods. Another method to connect VM to the Internet and intranet is through two independent IP stacks. Security is covered both in Chapter 1 of *Web Server Solutions for VM/ESA*, SG24-4874 and in this publication in Chapter 5, “Security Issues” on page 131.

System security starts with the integrity of the operating system. Since 1973, there have been Statements of System Integrity for VM. Transaction security is available on VM Web Servers supporting SSL.

1.6 How the World Wide Web Works

A Web browser is mainly a markup language interpreter but also does things like handle Java and Javascript. The markup language is Hyper Text Markup Language (HTML). The Web browser reads the HTML codes and formats the information according to the code instructions. The Hypertext Transport Protocol (HTTP) is how the Web browser communicates with the Web server.

The HTTP protocol is sent from the system image containing the Web browser to the system image containing the Web server over a network supporting the Transmission Control Protocol (TCP).

As shown in Figure 3, the Web browser also sends data to the server in HTTP. The server, depending on the data sent, can respond with HTML or launch a program to process the request or data. The Web server communicates with the program via the Common Gateway Interface (CGI). The CGI program is launched and processes the information. The CGI program may acquire information from other resources on the system or on the network. It may also perform manipulation or transformation on the data. Then the CGI program sends the information to the Web server and terminates. The output contains headers and HTML. The Web server passes the HTML to the browser.

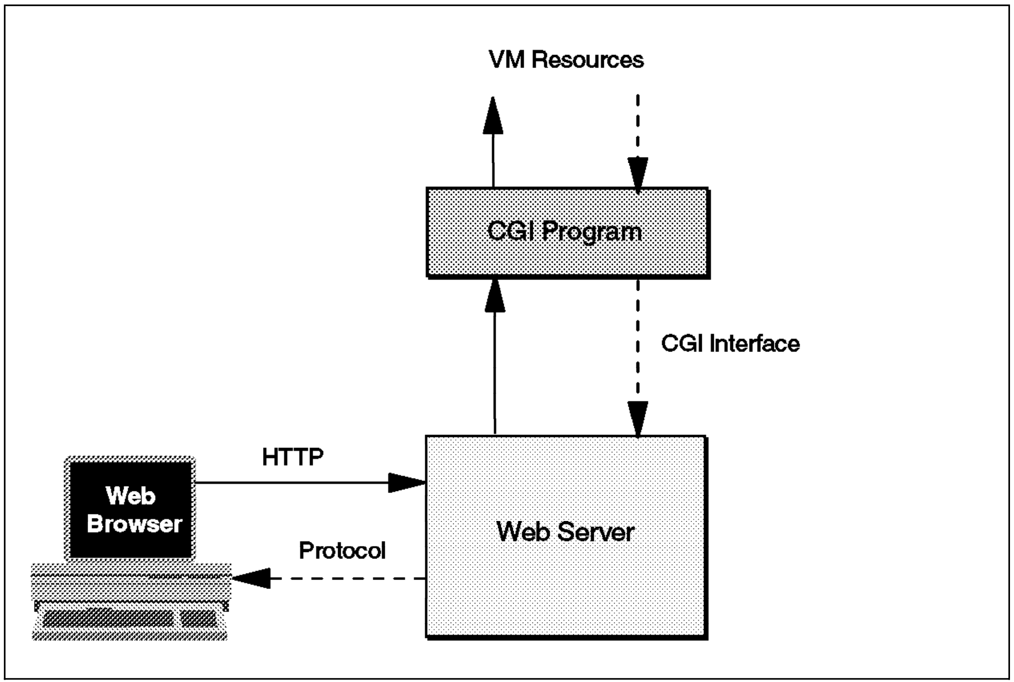


Figure 3. Web-Enabling VM Resources

The Web CGI program extends the functionality that the Web server provides.

1.7 Assumptions

This book is not intended to cover the installation or configuration of the needed VM/ESA features and products, although examples of the installation or configuration of a particular VM/ESA feature or product may be given in this book. This information is solely for the purpose of explaining the environment we used in Poughkeepsie.

It is assumed that all the features of VM/ESA 2.3.0, including TCP/IP function level 310, as well as a Web server, are installed, configured, and available.

For information on installing a Web server, see the companion book *Web Server Solutions for VM/ESA*, SG24-4874.

1.8 Environment

The software products and levels that were used for our environment follow. Additional PTSs, APARs or product levels may have been necessary for a particular function covered in this book. The function level dependencies will be addressed in the appropriate area of the book.

VM/ESA Version 2 Release 3.0, service level 9802

CMS Level 14, Service Level 802

UM29016 VM61794 BFS synchronous I/O problem

VM TCP/IP Level 310, RSU 9804

UQ20816 PQ14734 FTP incorrectly translated binary files
UQ20817 PQ18558 FTP to BFS resulted in 0 length files

VM:Webgateway Release 2.2

VM:Webgateway Web Server Release 2.2 plus fixes:
W220940 - Set 8-character WEBSHARE CGIUSERS CGI var correctly
W220941 - Set WORKERWEBSHARE CGI environment vars correctly
W220945 - Access MDISK/SFS domain in WORKERWEBSHARE CGIs
W220929 - Prevent CGI003/CGJ003 abends
VM:Webgateway OfficeVision Interface Release 1.4
VM:Webgateway CGI Extension Release 1.3 plus fixes:
G130203 - VIGLMS0033E on VIG USER CONNECT
G130204 - RDTERM macro returning incorrect information
VM:Operator 2.4

EnterpriseWeb/VM Release 1.4 and EnterpriseWeb Secure/VM Release 1.2

PTF LVL WEB00431 - Changing HTM definition in Media Map. This PTF applies to EWLOAD,EWMENU and EWMSG.

Webshare Release 1.2.4

1.9 Standards

The standards established for the Internet are called Requests For Comment (RFCs). An RFC passes through many stages before being designated (if indead it is) an accepted standard. Some standards and RFCs for the protocols discussed follow. For RFCs, the information listed is the RFC number, title, authors, date issued, file format and status.

HTML *RFC1866 Hypertext Markup Language - 2.0.* T. Berners-Lee & D. Connolly. November 1995. (Format: TXT=146904 bytes) (Status: PROPOSED STANDARD)

HTTP *RFC1945 Hypertext Transfer Protocol -- HTTP/1.0.* T. Berners-Lee, R. Fielding & H. Frystyk. May 1996. (Format: TXT=137582 bytes) (Status: INFORMATIONAL)

CGI The Common Gateway Interface is an Internet Draft. The draft specification can be found at The National Center for Supercomputing Applications. Web URLs are:

```
http://hoohoo.ncsa.uiuc.edu/cgi/
http://luna.bearnet.com/ietf-drafts/draft-robinson-www-interface-00.html
```

1.10 Summary

In summary, the VM/ESA interactive environment makes Web CGI programming a quick and easy task. REXX is the tool used to develop and debug CGI programs. CMS Pipelines′ strength is high-performance data manipulation and transformation. The excellent connectivity of VM/ESA facilitates data gathering from many sources.

The VM/ESA platform provides a highly tailorable environment with the capability of supporting several different servers. The multiple-server environment can provide security through server isolation. VM/ESA has a track record of proven stability. The scalabilty of VM/ESA presents a cost-effective growth curve.

Chapter 2. Introduction to Common Gateway Interface Programming

One of the main strengths of a VM system is that programs can be developed and debugged quickly using REXX. CMS Pipelines adds a powerful data manipulation and transformation capability to REXX programs. This makes VM an excellent platform to work with data for presentation or for further processing. VM′s excellent connectivity to other platforms makes it possible to gather data from a variety of sources and present unique views of that data. To use all of this power in the modern network connected world, a VM system must move beyond the familiar 3270 interface for presenting its data. CGI programs that use familiar VM concepts and run on VM systems let VM-gathered data be made available to the wide audience of the Internet. And it is very easy, as you will learn.

Before we show you how to create CGI programs on VM, let us introduce the basic concepts and standards used on the World Wide Web that allow a Web browser and a Web server to communicate with each other. Then we will introduce how a Web server and a CGI program interface with each other, and finally how you can use VM to do all of this.

2.1 The Common Gateway Interface

The CGI is a standard interface supported by a Web server that defines how information is exchanged between a Web server and an external program (CGI program). CGI programs can be written in any language supported by the operating system on which the server is run. The language can be a compiled programming language, like C, or it can be a scripting language, such as REXX. The CGI program can even be written in a combination of languages such as a REXX program calling a COBOL program to obtain data or a CMS Pipelines program that calls REXX language stages for data processing.

The CGI standard applies to any platform or Web server, not just VM-based Web servers. The implementation of the standard varies across platforms, as you might expect, and indeed varies between Web servers. But the common elements that make up the standard must be present in all implementations. This chapter introduces the CGI standard and gives you an overview of how CGI programs interact with Web servers and browsers.

2.2 HTTP

Let us first describe how a Web server and browser work together to display a simple HTML document. You do not need to do any programming to make this work, it is a function built into every Web server. But what really happens when you enter the URL[1] of this file on your Web browser screen? As you might suspect, the Web browser and the server communicate with each other in a standard, platform independent way. We will create an example to illustrate this interaction.

[1] For a description of a URL, see the "Internet Concepts" chapter of *Web Server Solutions for VM/ESA*, SG24-4874.

2.2.1 Request Header

Here is a very simple example of an HTML document:

```
<HTML><BODY><H1>Hello world!</H1></BODY></HTML>
```

We name this file hello.html and place it in the "root" directory of our Web server.

Note: The file names that you are allowed to specify may be dependent on the file system used by the Web server. Assume for this example that hello.html is an allowable name.

The TCP/IP name of our system is wtscvmt.itso.ibm.com. So, to display this file from a Web browser (our client), we enter the URL `http://wtscvmt.itso.ibm.com/hello.html`. This specifies the protocol to be used, the name of the system with the server, and the file name. The protocol used is HTTP. This is the protocol that Web servers use to communicate with Web browsers. It is much easier to understand HTTP if we just show you what will be exchanged between the Web server and the client when the hello.html file is requested.

The client sends the following text (called a *header*) to the server:

```
GET /hello.html HTTP/1.0
Accept: text/plain
Accept: text/html
Accept: text/*
User-Agent: Charlotte/2.1.0 VM_ESA/2.3.0 CMS/14
Host: wtscvmt.itso.ibm.com
   -- A blank (null) line --
```

Notice that the lines that are sent are readable ASCII text. This set of lines is called a *request header* and its format must comply with the RFC 1945 standard. A blank (null) line in this context is not a sequence of space characters, but the characters CR - LF - CR - LF. All headers must contain a blank line at the end.

This is the same format header used to send mail using TCP/IP. The first line of the message is called the request line and contains the *request method*. This GET method asks for the file `/hello.html` and also announces that the client is using HTTP version 1.0 to communicate.

2.2.1.1 Request Header Fields

The series of lines following the request up to the blank line are called *request header fields*. These provide information to the server about the client. Typically they may specify what format of data the client will accept, who it is from, encoding of data, authentication, and so forth. In the example, the Multipurpose Internet Mail Extension (MIME) types that this client will accept are `text/plain`, `text/html`, and any other text type (`text/*`). The client used is Charlotte 2.1.0 running on VM/ESA 2.3.0. The "host" field specifies where the request is sent.

There are other header lines that may be sent, depending on which client is used and which level of HTTP is implemented. See Table 1 on page 11 for a list of commonly used request header fields.

Table 1. Common Request Header Fields

Accept	Contains a list of MIME content types that the client will accept. Many browsers will accept any type, so they list "*/*" as an acceptable type. Text only browsers may only request text types, such as `text/*`.
Accept-Charset	Lists the character sets preferred by the browser, in order of preference. A character set of * means that any character set is accepted. An example is `iso-8859-1,*`. More information is found in section 3.4.6, "National Language Considerations" on page 50.
Accept-Language	Lists the languages preferred by the browser, in order of preference. For example French, with an alternative of English, would be sent as "fr, en."
Authorization	If the server indicated that authorization is required to access the resource, then authentication information is sent in this header field. **Note:** This field's name is a misnomer, as the HTTP Authorization field contains authentication information, not authorization information. It is normally sent by the browser in response to a response from the server with a status code of 401 (unauthorized) and containing an HTTP WWW-Authenticate field.
Content-Length	Gives the length in bytes of the object sent to the server. If no object follows the request header, this field is normally omitted.
Content-Type	Gives the MIME content type of the object sent. The most common type for a POST request is `application/x-www-form-urlencoded`. If no data is sent, this field is not sent.
Cookie	This is a nonstandard field implemented by Netscape to allow the server to store a small amount of data on the client. If the client has "cookie" information available to send with the request, this field is included with that information. See 3.4.5, "Using Cookies" on page 47 for more information.
Date	The time and date the request was made, always expressed in Greenwich Mean Time (GMT). See 3.4.3.1, "HTTP Time Stamp Formats" on page 45 for a list of the allowed formats.
Host	The domain name and port number of where the request is being sent.
If-Modified-Since	If this is sent, it makes the request for an object conditional. The server can determine if the object needs to be resent or if the object that the client has stored in its cache can be reused. See 3.4.3.1, "HTTP Time Stamp Formats" on page 45 for a list of the allowed formats.
User-Agent	Provides information about the client software making the request.

If the client has data to send to the server, it creates a slightly different request:

```
POST /cgi-bin/hello HTTP/1.0
Accept: text/plain
Accept: text/html
Accept: text/*
User-Agent: Charlotte/2.1.0 VM_ESA/2.3.0 CMS/14
Host: wtscvmt.itso.ibm.com
Content-Length: 11
Content-Type: application/x-www-form-urlencoded

happy=yes
```

This example shows that the data sent by the client (in this example, `happy=yes`, followed by a CR and LF for a total of 11 characters) is placed following a blank line after the header.

2.2.2 Response Header

Once the server receives the request, it responds back to the client with a *response header* and some data. The format of the header is very similar to the request header. Here is the actual response to our first example GET request:

```
HTTP/1.0 200 Ok
Content-Type: text/html
Content-Length: 49
Last-Modified: Thu, 22 Oct 1998 14:53:37 GMT
Server: VM:Webserver/02.2
Date: Thu, 22 Oct 1998 19:39:56 GMT

<HTML><BODY><H1>Hello world!</H1></BODY></HTML>
```

The first line is known as the *status line*. It shows that the server agrees to use HTTP version 1.0 for communication and sends a response status code of 200, which means that the server successfully processed the request. The string "Ok" is just a readable text string that means the same thing as the response status code.

Status codes of 200-299 indicate a successful transaction. Codes of 300-399 indicate that the requested object is not located at the address requested. It may have been moved or can be found in the browser's cache. Codes of 400-599 indicate an error of some type, with 400-499 indicating a client error (such as bad syntax), and 500-599 indicating a server error. Some of the more common ones are listed in Table 2.

Table 2. List of Common HTTP Response Status Codes

200	The request completed successfully.
202	The request was accepted, but processing was not completed or the results of the processing are unknown.
203	The request was accepted, but only partial information was returned.
204	The request was accepted, but there is no new information to return. The browser should continue to display the same document.
301	The requested document has been permanently moved to a new location. The new location should also be sent as a response header field. Most browsers will automatically request the document at this new location.
302	The requested document can be found in a new location. This response is very similar to the 301 response, but it does not indicate that the redirection is permanent. Most browsers will automatically request the document at the new location.
304	The requested document has not been modified. This can only be returned if the browser has a cached version of the document and sends to the server the "last modification date" of the document that it has saved. If the server determines that the document in the browser's cache is current, it tells the browser to use the cached version.
400	The syntax of the request was incorrect.
401	The request requires the client to be authorized. This response along with authorization and authentication fields allow the client and server to negotiate data encryption and user authentication schemes.
403	The request is for something that is forbidden. No explanation needs to be provided and no authorization can be negotiated.
404	The request is for an object that cannot be found.
500	The server encountered an internal error and cannot process the request.
501	The server does not support the specified method of the request.

2.2.2.1 Response Headers

The series of lines following the response up to the blank line are called *response header fields*. They specify information about the object that follows the header. In the example, the client is told that the object after the header is MIME type `text/html`, is 49 bytes long (including its trailing CR and LF characters), the date it was last changed, the server that is sending it, and the date and time of the response. Other fields may be included for other types of requests. Note that most fields are optional. See Table 3 for a list of commonly seen response header fields.

Table 3. Common Response Header Fields

Date	The date and time the response is sent, always expressed in Greenwich Mean Time (GMT). See 3.4.3.1, "HTTP Time Stamp Formats" on page 45 for a list of the allowed formats.
Location	Contains a URL that the client should request instead. This is returned when the server knows that the document has moved to a new location. Most browsers will automatically request the document at the new location.
Server	Contains an information string about the software running on the server. The program and version are separated by a slash (/).
Set-Cookie	This is a nonstandard response implemented by Netscape to allow the server to store a small amount of data on the client. If the client supports this field, then it updates the previous information that it has stored with this information or creates a new entry.
Content-Language	Gives the language of the document sent in this response.
Content-Length	Gives the length in bytes of the document sent to the client. (including any trailing CR and LF characters)
Content-Type	Gives the MIME content type of the object sent. Common types are `text/html` and `image/gif`. Optionally, the character set associated with the object can be included. An example of this field is `text/html; charset=ISO-8859-4`.
Expires	Gives the date after which the object sent should be discarded if the client maintains a local cache. See 3.4.3.1, "HTTP Time Stamp Formats" on page 45 for a list of the allowed formats.
Last-Modified	Gives the date and time that the original object was last changed. The client can save this information along with the object in its cache so that it can issue a conditional GET (which would include an If-Modified-Since field) the next time it requests the object. See 3.4.3.1, "HTTP Time Stamp Formats" on page 45 for a list of the allowed formats.

The last line of the example is the actual HTML document. It will be "rendered" (interpreted) by the client and shown on the browser window.

Once the document is shown, the client-server transaction is finished and the connection is closed. The user may initiate another transaction by selecting a document link on the browser screen or entering a new URL, which would start the whole process over again. This interaction between the client and the server is *stateless*, meaning that no "state" or memory of the transaction is saved by the server.

2.3 CGI Standard

An HTTP URL may identify a file that contains a program to be run instead of an HTML document to be shown. The server knows that the object referenced is a program through server-specific configuration directives. Typically these directives associate a file extension or the location of the file with whether or not the object is executable, not the file name. If it is an executable program, the

Web server invokes it as a separate process and communicates with it via the CGI. The CGI standard specifies how the Web server passes data and variables to the gateway program and how it receives data from the program. Any data from the program is sent back to the client Web browser that made the request. Since the CGI is a standard that is implemented by most Web servers, the basic design of CGI programs is portable across platforms. Of course, how the interfaces between the Web server and a CGI program actually work and how variables are retrieved will likely be different across platforms. This section discusses only the CGI standard. VM-specific implementations of the standard are covered in Chapter 3, "CGI Programs on VM/ESA" on page 21.

2.3.1 CGI Environment Variables

Environment variables are used to pass data about the request from the client and the server to the CGI program. The CGI standard specifies a number of universal variables that all Web servers should implement. Web servers may provide additional environment variables outside of the standard set. For compatibility with potential future CGI standards, it is suggested that any additional variables provided start with the characters "X_." If your Web server provides these and you use them in a CGI program, be aware that other Web servers may not provide them. See Table 4 for a list of the standard environment variables and a short description of them.

Table 4 (Page 1 of 2). CGI Environment Variables

AUTH_TYPE	If the CGI program is access protected, this variable is set with the type of authentication performed. In order for the client to be authenticated, it must have returned an authorization response field. If the Web server authenticates the client, this variable will be set to `Basic`. Other authentication types may be implemented in the future.
CONTENT_LENGTH	The length, in bytes, of any data sent by the client using the POST method. If no data is sent (for example, the request was GET), the variable is undefined.
CONTENT_TYPE	The MIME content type of the data sent from the client using the POST method. If no data is sent (for example, the request was GET), the variable is undefined. Most of the time, if data is sent the value is `application/x-ww-form-urlencoded`.
GATEWAY_INTERFACE	The version of the CGI standard used by the server. The current standard is CGI 1.1, so this variable will be set to `CGI/1.1`.
HTTP_*name*	This set of variables with names that begin with "HTTP_" contain additional request header fields sent from the client. The name of each field is capitalized, all dashes in the name converted to underscores, and the string `HTTP_` is prefixed to this name to form the variable name. For example, the variable `HTTP_ACCEPT_LANGUAGE` contains the value `en` from the field `Accept-Language: en` sent from the client. **Note:** When the content of a request header field is already used to set another environment variable (such as the Content-Length field setting the CONTENT_LENGTH variable), it is permissible for the Web server to not set the corresponding "HTTP_" variable.
PATH_INFO	Extra path information that is present in the URL that invoked the CGI program. This additional path information is not interpreted by the Web server but is made available to the CGI program. It can identify a resource or a path to a resource to be accessed by the CGI program. For example, in a URL of `http://wtscvmt.itso.ibm.com/cgi-bin/where/data/street`, the `/data/street` part of the URL would be placed in the PATH_INFO variable that is available to the "where" CGI program. If no additional path information is provided in the URL, this variable is undefined.

Table 4 (Page 2 of 2). CGI Environment Variables

PATH_TRANSLATED	If the PATH_INFO variable contains data, the Web server will attempt to translate the resource referenced by the additional path data to a file system-specific path name. Note that this is not set to the location of the CGI program. This value is platform specific and might not be implemented by all Web servers.
QUERY_STRING	The query string part of the URL, encoded in the normal URL encoding method. This part of the URL is anything that follows the first question mark (?) in the string. For example, in the URL `http://wtscvmt.itso.ibm.com/cgi-bin/where?street=pine`, the QUERY_STRING is set to `street=pine` Note that a CGI program must decode this string before using it.
REMOTE_ADDR	The numeric IP address of the client. Note that if the user is accessing the Web server via an agent (for example, a firewall), this will be the address of the agent, not the user′s client.
REMOTE_HOST	The fully qualified domain name of the client. If this information is not available, this variable is undefined. This is the domain name of the client identified in the REMOTE_ADDR variable.
REMOTE_IDENT	The remote user name that is retrieved by the server from the client. It is not implemented by most Web servers.
REMOTE_USER	If the AUTH_TYPE variable is set to `Basic`, then this variable is set to the authorized user ID of the client. Otherwise it is undefined.
REQUEST_METHOD	The HTTP method sent in the request from the client. Normally it is GET or POST, but there are other methods such as PUT, HEAD, and DELETE that are not implemented by all Web servers or clients.
SCRIPT_NAME	The URL path and name that identifies the CGI program. The contents of this variable and the SERVER_NAME and SERVER_PORT variables and a source of infromation to map the port number to the protocol being served (sometimes available as an added CGI variable) and a source of information to map the port number to the protocol being served sometimes available as an added CGI variable can be used to reconstruct the URL that references the CGI program. See Figure 29 on page 71 for an example. For example, if the full URL referencing the CGI program is `http://wtscvmt.itso.ibm.com/cgi-bin/where`, the value of the SCRIPT_NAME variable is `/cgi-bin/where`.
SERVER_NAME	The fully qualified domain name of the server. If the fully qualified name is not known, then the variable is set to the IP address of the server.
SERVER_PORT	The port number of the server that received this request. If a specific port number was not specified in the request, the default port number for HTTP requests is 80 and for HTTPS requests it is 443.
SERVER_PROTOCOL	The protocol being used between the client and the server and its version number. This allows a CGI program to use new protocol features if the client supports it. The protocol in most common use is HTTP 1.0, so this variable would contain the value `HTTP/1.0`. Usage of the HTTPS protocol does not affect the contents of this variable.
SERVER_SOFTWARE	The name and version of the software running on the Web server, in the format *name/version*.

2.3.2 Sending Data to a CGI Program

If a Web browser client is requesting a simple HTML document from the server, no additional data needs to be sent from the client except the URL of the document (along with the normal header.) However, when the URL refers to a CGI program, the client will frequently need to send additional data with the request so that the CGI program can figure out what it needs to do. There are four different ways the client can send data to a Web server using the HTTP protocol. They can be used separately or in combination. The Web server receives this data and makes it available to the CGI program. The method the CGI program uses to access this data depends on how the data is sent. The methods are:

- The query string in the URL
- Extra path information in the URL
- Additional request header fields
- POST data sent following the request header

The query string and the extra path information are made available to the CGI program via the QUERY_STRING and PATH_INFO environment variables described in Table 4 on page 14. The query string can be explicitly coded in a URL specification, but it may also be automatically supplied by a Web browser from an ISINDEX HTML tag, from an ISMAP attribute on an IMG tag, or from an HTML FORM using the GET method. The ISINDEX tag and the GET method for forms are not used as often as the POST method to request data from a user.

Additional request header fields are made available as HTTP_*name* global variables, where *name* is the name of the field. This is also described in Table 4 on page 14. These fields could be standard ones such as authorization information or an `If-Modified-Since` field that identifies the date of the client's cached copy. Or they could be from a client with additional function, such as Netscape cookie fields. How to use Netscape cookies will be described in 3.4.5, "Using Cookies" on page 47.

Data sent using the POST method is made available to a CGI program by the Web server via "standard input." This form of input is meaningful in some operating system environments, but in other environments where it does not have a standard meaning it can be implemented in different ways. How Web servers on VM provide this data is covered in 3.3.3, "Using a FORM with POST" on page 35

2.3.3 Encoding

You should be aware that the input available to a CGI program may be *URL encoded* and must be decoded before it can be used. URL encoding means that all space characters are changed to plus signs (+) and any nonprintable or special characters are changed to their ASCII hexadecimal code (in the ISO Latin-1 character set) prefaced by a percent sign (%). Which characters are "nonprintable" and "special" depends on where the string appears or how it will be interpreted. This is defined by the HTTP standard. For safety, the client may encode more characters than are necessary. Percent characters (%), plus signs (+), and equals signs (=) in the original input string are always encoded. For example, the string `a?b` may be encoded as `a%3Fb` if a question mark might be interpreted incorrectly. Many Web servers provide utilities or functions that can decode variables or input strings as needed.

Any query string data available in the QUERY_STRING variable is always URL encoded. Extra path information in the PATH_INFO variable may be encoded. In fact, the Web server may provide additional information about the PATH_INFO value in the PATH_TRANSLATED variable. Data available in additional request header fields is not encoded. POST data is FORM encoded if the CONTENT_TYPE environment variable has a value of `application/x-www-form-urlencoded`. Some Web servers will automatically decode the POST data for you. There are also utilities available to assist with decoding.

2.3.4 Sending Data from a CGI Program to the Client

Once the CGI program has completed its processing, it must send the results back to the client. It does this by writing *server directives* and the actual data to "standard output." Just like standard input, this form of output is meaningful in some operating system environments, but in other environments it can be implemented in different ways. This is the only way a CGI program can pass its results back to the server. The server directives are used by the server to create the HTTP response header. The actual data is sent following the server directives. Usually, this data will be an HTML document that is displayed on the client. But it could be any type of data that the client is able to display. The Content-Type server directive written by the CGI program specifies the type of the data.

Server directives look a lot like normal HTTP response header fields, and in fact most of them are. But they are called server directives in the CGI standard because some of them are intercepted by the Web server and are used to create the actual HTTP response header sent to the client. The ones that have meaning to the Web server are listed in Table 5. Other HTTP response header fields such as those listed in Table 3 on page 13 may also be written as server directives as needed, and they will not be interpreted by the Web server. However, you must not write any server directives that duplicate any of the fields that the Web server will write automatically. For example, the header fields Server and Date are server specific and will typically be written by the server in any response.

Table 5. CGI Server Directives

Status	This indicates to the server what status code is sent as the response. It must be of the form `Status:` *`code string`*, where *`code`* is one of the response codes listed in Table 2 on page 12 and *`string`* is a description of the status. If a status directive is not written, a status of "200 OK" is assumed.
Content-Type	This indicates the MIME type of the data returned to the client. It must be a valid MIME type and in the form *`type/subtype`*.
Location	This indicates to the server that the client should be redirected to a different URL, which is specified on this field. A status directive of 301 or 302 must be returned with this field. (See Table 2 on page 12 for a description of these codes.) Some servers will automatically send a response code of "302 Redirection" if the CGI program sends this field without a status directive. Most Web browsers will automatically send a request to the new URL when they receive this field along with the 301 or 302 response code.

2.4 Characteristics of Web Transaction Processing

One of the most basic things to remember about HTTP is that it is a stateless protocol. What this means is that the Web server and CGI program normally do not retain any knowledge of previous transactions. A request comes to the Web server, the CGI program referenced in the request is located and started, it runs, and data is sent back to the user. When the CGI program ends, any variables that were set are gone. Of course, the CGI program could have written data to a file or used other mechanisms to remember something about the transaction. But remember that the user might not continue the transaction at all (for example, they power off their system) or they may continue the transaction hours later. They may even use the same application the next day, but would be surprised if the application did not start from the beginning.

Another possibility is that the next request from the user may not even be executed on the same Web server machine. For better performance, Web servers such as Webshare and EnterpriseWeb/VM can be configured so that more than one server machine can respond to requests. This allows requests to be processed in parallel. EnterpriseWeb/VM may respond using the same server machine but use a different execution thread. This means a CGI program could run in parallel with itself on the same Web server. And VM:Webgateway has the capability to define a collection of server worker machines that can run a CGI program on any of them. The point here is that a CGI program must be careful about any data that it stores between executions or data that it updates. This makes the design of a CGI program a bit more complicated than simply writing an EXEC running in a virtual machine.

2.4.1 Creating a Web Transaction

There are techniques that can be used to create a Web based transaction involving a series of HTML documents and CGI programs. All of these techniques send transaction identification data back and forth from the server to the client back to the server, and so on. One way to think of it is that the identification data sent to the client is "held" by the client until it begins another transaction. When the server receives the transaction, it also receives the identification data it sent in the previous transaction.

Note: This "identification data" is not necessarily authentication information (for instance, a user ID and password) that identifies the client through some verification scheme. It can be, but it can also be something as simple as a time stamp generated when the user started the transaction that identifies that user throughout the transaction. More information about authentication can be found in section 5.5.4, "Client and Server Authentication" on page 139.

2.4.1.1 Hidden Fields

One very common method of identification is hidden input fields on HTML forms. The input form sent to the client contains fields that are not displayed on the browser window but are sent back to the server when the form is submitted. The identification information is stored in these hidden fields. This technique is demonstrated in section 3.3.3, "Using a FORM with POST" on page 35.

2.4.1.2 Cookies

Another method is for the server to send a *Cookie* to the Web browser. A Cookie is really just some *name=value* data that the Web browser stores on a local storage device. It is implemented using a Set-Cookie response field (to create a Cookie) and a Cookie request field (to send the saved data back to the server.) But note that Cookies are an extension to the standard HTTP protocol, which is not implemented by all Web browsers. Clients can also be configured to refuse to accept Cookies. Despite these limitations, they are useful to save preferences or to remember which pages were visited. The format of this data and how to set and retrieve it is discussed in 3.4.5, "Using Cookies" on page 47.

2.4.1.3 Query String

If you remember from section 2.2.1, "Request Header" on page 10, the first thing a client sends to the server is the request. The first word is GET or POST (most of the time) and the second word of the request is the location of the resource. The query string is anything that follows the "?" in the second word. Since it is part of a URL, a CGI program that dynamically creates an HTML document containing hypertext links can place transaction identification information in the query string part of the URLs. When the user selects one of these URLs, the query string is sent to the server and to the CGI program that uses it to identify the transaction. The user may also save a URL in a bookmark file, which would also save the query string and thus the transaction identification data. A user could move a bookmark file to a different client and still be identified by the server.

2.4.2 Summary

This chapter has covered basic concepts and some of the standards used on the World Wide Web. More information about Internet standards can be found in the Web pages and publications listed in Appendix E, “Related Publications” on page 243. The World Wide Web is still evolving, and new standards continue to be proposed and adopted. We encourage you to continue to learn about recent standards and technologies so that you can apply them to your Web programming.

Chapter 3. CGI Programs on VM/ESA

Now that we have presented the HTTP protocol and the CGI standard, how does all this apply to VM? We have already explained that VM is a very good system for quickly and easily creating powerful programs. Now let us learn how to write CGI programs using REXX and how to use the Web server products that are available for VM.

3.1 A CGI Program Example

The previous chapter covered all kinds of things about what a CGI program can do. Let us begin with a very simple CGI program that only outputs an ordinary HTML document. As we move through this chapter we will learn how to create dynamic output, read and write headers, and so forth.

Our initial CGI program outputs a table of telephone numbers to a Web browser. We need a CGI program to do this because the table is a CMS file that does not have any HTML encoding in it. Our program reads the table and inserts HTML tags as needed to create a table in HTML format. Besides the table, we also need other HTML tags to create a complete HTML document. So we have put our HTML “prologue” and “footer” tags into separate files that are read by our CGI program. If you use these same files in other CGI programs, it makes it very easy to change the “look” of many pages of your Web site by just changing one or two files.

3.1.1 Sample HTML and Data Files

For example, let us create a CGI program that will display a table of telephone numbers and office data of a group of people. This data could reside on VM as a DB2/VM table or be part of an office application. For this example, the data will be in a simple file contained in SFS named PHONE DATA in directory SFSTEST:VMWEBCD.WEBSHARE.DATA.Figure 4 shows the sample file.

```
*
* File of Telephone data and office data for the ITSO Intranet Dept.
*
* Num  Name                 Phone number Loc  Office    Status
*----- -------------------- ------------ --- --------- ------
111111 John Q Public        888-555-1111 POK   8-2-C14 In
222222 Eric A Adams         888-555-2222 POK   8-2-C15 In
333333 Paul B Smith         888-555-3333 FRA  16-1-003 In
444444 Steven C Fritz       888-555-4444 FRA  16-1-007 In
555555 Nick D Grimes        888-555-5555 END 250-3-H09 In
666666 Bruce E Hardy        888-555-6666 END 250-3-H06 Out
```

Figure 4. Sample PHONE DATA File

Any line in the file that starts with an * will be ignored by the application. We also use a common HTML prologue and footer that are contained in separate files. Figure 5 on page 22 shows the prologue file and Figure 6 on page 22 shows the footer file.

```
<HTML><HEAD>
<TITLE>ITSO Telephone List</TITLE>
</HEAD>
<BODY>
<H1>ITSO Telephone List</H1>
<P>The ITSO Intranet department is listed below.</P>
```

Figure 5. Sample HTML Prologue File PROLOG1 HTMLPART

```
<HR>
<B>{
  <A HREF="home.html">Home Page</A> |
  <A HREF="http://www.ibm.com">IBM Home Page</A> |
  <A HREF="http://w3.ibm.com/">IBM Intranet</A>
}</B>
</BODY></HTML>
```

Figure 6. Sample HTML Footer File FOOTER HTMLPART

3.1.2 Sample CGI Program

Now that we have all the data needed, we write a CGI program that creates the HTML document for the user to see. For this example, we assumed our Web server is Webshare or any Web server that is compatible with the Webshare CGI environment. We use REXX and CMS Pipelines to create our program, PHONE1 CGI, which you can see in Figure 7 on page 23.

Programming Note

You must be aware of the REXX command environment of your CGI program. CGI programs run by a Webshare or EnterpriseWeb/VM server will always have a CMS Pipelines default command environment upon entry into the program. CGI programs run in Webshare compatibility mode by VM:Webgateway also use the same default environment. Any external EXECs, MODULEs, CMS, or CP commands executed by your CGI program should be proceeded by “Address Command” or use the CMS Pipelines COMMAND or CP stages.

```
/*  PHONE1 CGI program example */
PhoneFile = 'PHONE DATA'          /* File of telephone number data  */
DataDir   = 'SFSTEST:VMWEBCD.WEBSHARE.DATA' /* data file location   */

'callpipe < PROLOG1 HTMLPART | *:'/* Display the beginning HTML     */
                               1
'output <TABLE BORDER="1">'       /* Set up the HTML table          */
'output <TH>Employee Number</TH>' /* Put a title on each column     */
'output <TH>Name</TH>'
'output <TH>Phone Number</TH>'
'output <TH>Location</TH>'
'output <TH>Office</TH>'
'output <TH>Status</TH>'
                               2
/* Get the telephone data from the file. */
'callpipe <sfs' PhoneFile DataDir '| stem phone.'

Do i=1 to phone.0                 /* Format the file for display    */
   If left(phone.i,1) = '*' then  /* Ignore the comments            */
      iterate
   'output <TR>'                  /* Beginning of a table row       */
   'output <TD>' substr(phone.i, 1, 6) '</TD>'  /* Each             */
   'output <TD>' substr(phone.i, 8,20) '</TD>'  /*   table          */
   'output <TD>' substr(phone.i,29,12) '</TD>'  /*      cell        */
   'output <TD>' substr(phone.i,42, 3) '</TD>'
   'output <TD>' substr(phone.i,46, 9) '</TD>'
   'output <TD>' substr(phone.i,56, 5) '</TD>'
   'output </TR>'                 /* End of a table row             */
End
'output </TABLE>'
                               3
'callpipe < FOOTER HTMLPART | *:' /* The ending HTML                */

Exit RC
```

Figure 7. Sample CGI Program Named PHONE1 CGI

This program is not very complicated; it just uses simple REXX statements that are probably familiar to you. The program performs three main actions:

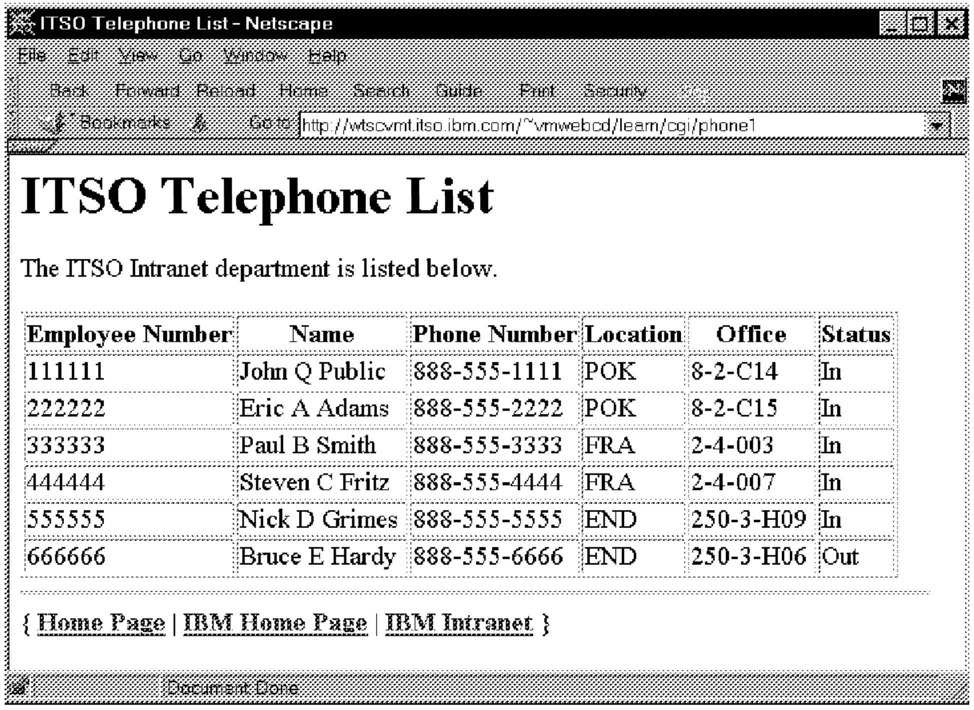
1 This part of the program just outputs the HTML prologue tags from the PROLOG1 HTMLPART file and then outputs the tags that define the HTML table. It uses callpipe and pipeline output commands to pass the HTML data to the Web server because that is the interface used by Webshare. No Status or other response fields are written by this program, which means that the Web server will automatically output a status of 200 OK and its default response fields.

2 This part of the program reads the PHONE DATA file and formats it with HTML tags. Each phone stem variable will contain one line of the PHONE DATA file. The statements in the Do loop produce a beginning and ending table data HTML tag (<TD></TD>) before every element in the PHONE DATA file. (However, also see the discussion in 6.3.2.1, "Fast Rendering by Avoiding HTML Tables" on page 161.)

3 The last part of the code outputs the HTML tag that defines the end of the table and then outputs the final HTML tags from the FOOTER HTMLPART file.

Readers who are experienced with CMS Pipelines know that this sample could be written to use more pipelines functions. But very little CMS Pipelines knowledge is needed to create a working CGI program as shown in Figure 7. Later examples in this chapter will use more pipe features in the examples because CMS Pipelines is a very powerful and useful feature of CMS REXX programming and we would like to encourage you to learn about it and use it.

To run the CGI program, we enter the URL `http://wtscvmt.itso.ibm.com/~ vmwebcd/learn/cgi/phone1` into our Web browser. The result of this CGI program is shown in Figure 8.


ITSO Telephone List - Netscape
File Edit View Go Window Help
Back Forward Reload Home Search Guide Print Security
Bookmarks Go to: http://wtscvmt.itso.ibm.com/~vmwebcd/learn/cgi/phone1
Document Done

ITSO Telephone List

The ITSO Intranet department is listed below.

Employee Number	Name	Phone Number	Location	Office	Status
111111	John Q Public	888-555-1111	POK	8-2-C14	In
222222	Eric A Adams	888-555-2222	POK	8-2-C15	In
333333	Paul B Smith	888-555-3333	FRA	2-4-003	In
444444	Steven C Fritz	888-555-4444	FRA	2-4-007	In
555555	Nick D Grimes	888-555-5555	END	250-3-H09	In
666666	Bruce E Hardy	888-555-6666	END	250-3-H06	Out

{ Home Page | IBM Home Page | IBM Intranet }

Figure 8. Result of Running the PHONE1 CGI Program

3.1.3 Sample CGI Program for VM:Webgateway

If our Web server is VM:Webgateway and we are using the native CGI program environment instead of the Webshare compatibility support, some changes need to be made to the sample.

Tip

We recommend that you use the native VM:Webgateway CGI program commands if you are using this server and are writing a CGI program that does not need to be portable among the VM Web server products. The native commands are more flexible and give you more control over the CGI environment than is available using the Webshare compatibility environment.

The most important difference between Webshare and VM:Webgateway native CGI programs is the default REXX command environment. Native VM:Webgateway CGI programs use CMS as the default environment just like a normal CMS exec. Any execution of an external EXEC, MODULE, CMS, or CP command should use the “Command” environment. To run a CMS Pipelines program in this environment, the PIPE command is used instead of CALLPIPE.

VM:Webgateway provides a CGI command that a CGI program uses to communicate with the Web server. To write all or portions of an HTML document to the Web browser, use:

```
CGI WRITE DOCUMENT (TRANSLATE USENGLISH CRLF STRING data
```

The command options instruct the Web server to translate the data provided in EBCDIC to ASCII and add “carriage return/line feed” characters (EBCDIC X′0D25′ before translation) to the end of the data. HTML documents do not require a CRLF at the end of each record, but this makes viewing the file on an ASCII-based system look similar to the record structure of CMS files. The CGI command also supports the options VAR *varname* and STEM *stemname* in case your data is contained within REXX variables instead of a STRING.

Going back to our previous example, in Figure 7 on page 23, after marker **1**, the line:

```
′output <TABLE BORDER=″1″>′
```

is changed to:

```
CGIwrite=′CGI WRITE DOCUMENT (TRANSLATE USENGLISH CRLF STRING′
CGIwrite ′<TABLE BORDER=″1″>′
```

Notice that the full CGI command is assigned to a variable so that it is easy to write.

The pipe just before marker **1** is changed to:[2]

```
′PIPE < PROLOG1 HTMLPART′,
     ′| change //CGI WRITE DOCUMENT (TRANSLATE USENGLISH CRLF STRING /′
     ′| command′
```

This will read the PROLOG1 HTMLPART file and transform every line to a CGI command by inserting the string `CGI WRITE...` at the beginning. The commands are executed by the command stage.

A complete sample program is available on the VM Web CD as PHONE1 SVMEXEC.

[2] Also see the note on page 28 for information on how to access the HTMLPART file.

Performance Tip

Every time the CGI command is executed, data is sent from the CGI program through the Web server to the client. This gives the client a fast response but causes additional overhead in the server. To reduce this overhead, we can issue fewer CGI commands with more data supplied each time. This is very easy to do with the CMS Pipelines JOIN stage. Our pipeline becomes:

```
'PIPE < PROLOG1 HTMLPART',
    '| join * x0D25 65535',
    '| change //CGI WRITE DOCUMENT (TRANSLATE USENGLISH CRLF STRING /'
    '| command'
```

The stage "join * x0D25 65535" takes all the records from PROLOG1 HTMLPART and joins them together into one record, inserting a CRLF (X'0D25') between each record. The number 65535 [3] means that join will not create a record larger than 65535 bytes; it will create another output record for the additional records. This is essentially a *blocking* stage that takes the small records from the file and makes a large "block" of data to write to the Web server. The number 65535 was chosen as a trade-off between storage use and Web server overhead. More information on overhead and performance is found in Chapter 6, "Performance Issues" on page 157. EnterpriseWeb/VM and Webshare automatically block data written to the output stream, so blocking by the CGI program is not needed.

3.2 Fetching CGI Global Variables

As we learned in section 2.3.1, "CGI Environment Variables" on page 14, the CGI standard specifies a set of environment variables that are made available by the Web server to a CGI program. Remember that they are used to pass data about the request from the client to the CGI program. Each different Web server implementation specifies how a CGI program gets access to these variables. Note that changing these variables will not have any effect on the Web server. They are only used to pass data from the Web server about the client request *to* the CGI program.

How to retrieve global variables for each VM Web server product is discussed in the following sections.

3.2.1 Webshare Global Variables

CGI global variables are available to a CGI program in the GLOBALV group HTTPD. For example, to retrieve the variables REQUEST_METHOD and HTTP_USER_AGENT into variables with the same name, you code:

```
Address command 'GLOBALV SELECT HTTPD GET REQUEST_METHOD HTTP_USER_AGENT'
```

Address command is necessary in the statement because the default REXX programming environment of a Webshare CGI program is CMS Pipelines.

The CGI standard specifies that all request header fields sent with the request are passed to the CGI program in global variables whose names begin with

[3] VM:Webgateway release 2.2 worker machine support has a bug where CGI commands longer than 4096 bytes can abend the Web server worker. This problem is expected to be fixed in release 3.0. If you are using VM:Webgateway release 2.2 and worker machines, use 4000 for a blocking number instead of 65535 to avoid the problem.

“HTTP_” (unless the data in the field is already available in another variable, such as CONTENT_LENGTH.) Webshare provides some of the fields as global variables in GLOBALV, but not all of them. All of the header fields are available in the pipe secondary input stream to the CGI program. So, variables containing the header fields can be retrieved and set with a short CMS Pipelines program. When creating variables not set by the Web server, begin the variable names with X_ so that you know they are an extension of the CGI standard. If the Web server provides the header field contents in a variable, a CGI program should use that variable. Here is the pipeline that sets the variables:

```
/* WEBSHARE/EWEB don′t create variables for all header fields so,   */
/* create ″extension variables″ of X_HTTP_name                       */
′callpipe *.input.1:′,           /* Get header records              */
      ′| xlate w1 upper - _′,    /* Make proper variable names      */
      ′| spec fs : f1 1 f2-* strip 21′,  /* Expand for JOIN         */
      ′| sort 1-20′,
      ′| join keylength 20 /, /′,        /* Combine matching hdrs  */
      ′| spec /=X_HTTP_/ 1 w1 next /=/ next 21-* next′, /* Vars    */
      ′| varload′
```

This creates X_HTTP_*name* variables out of each header field, and does the standard translation of any dashes in the field name to underscores in the variable name.

Note: Webshare sets more variables in the HTTPD GLOBALV group than just the standard CGI global variables. If you use these variables, remember that they may not be implemented in all Web server installations.

3.2.2 EnterpriseWeb/VM Global Variables

CGI global variables are available using the same interfaces that are used in Webshare. Since GLOBALV is a resource shared by all programs running under one user ID, the GLOBALV group name of some of the global variables changes depending on which EnterpriseWeb/VM *thread* the CGI program is using. The name of the thread is passed to the CGI program in pipe input stream 2 and is intended to be loaded to the variable eweb.thread using this statement:

```
′callpipe *.input.2: | varload′
```

The variables whose values do not change when a different thread is used are available in the HTTPD GLOBALV group. These variables are:

- GATEWAY_INTERFACE
- SERVER_NAME
- SERVER_PORT
- SERVER_SOFTWARE

All other global variables are found in a THREAD*n* group where *n* is the value of the eweb.thread variable. For example, to retrieve the variables REQUEST_METHOD and HTTP_USER_AGENT into variables with the same name, you code:

```
Address command,
′GLOBALV SELECT THREAD′||eweb.thread ′GET REQUEST_METHOD HTTP_USER_AGENT′
```

Address command is necessary in the statement because the default REXX programming environment of an EnterpriseWeb/VM CGI program is CMS Pipelines.

Just like Webshare, EnterpriseWeb/VM does not provide all of the HTTP request header fields as GLOBALV variables. However, just like Webshare, they are available in the pipe secondary input stream to the CGI program and can be retrieved using the same pipeline listed in section 3.2.1, "Webshare Global Variables" on page 26. EnterpriseWeb/VM also makes all the header fields available as GLOBALV variables named HEADER.*x*., where *x* counts from 1 to the number of header fields; set in the HEADER*n* group, where *n* is the value of the `eweb.thread` variable. Therefore, GLOBALV commands can be used instead of the pipeline to retrieve the header fields, if desired.

Note: EnterpriseWeb/VM sets many more variables in the HTTPD GLOBALV group than just the standard CGI global variables. If you use these variables, remember that they may not be implemented in all Web server installations.

3.2.3 VM:Webgateway Global Variables

All CGI global variables are retrieved using the CGI GETVAR command. For example, to retrieve the variables REQUEST_METHOD and HTTP_USER_AGENT into variables with the same name, you code:

```
'CGI GETVAR REQUEST_METHOD  (VAR REQUEST_METHOD'
'CGI GETVAR HTTP_USER_AGENT (VAR HTTP_USER_AGENT'
```

The name of the variable that is retrieved and the REXX variable receiving the data do not need to have the same name. You can also retrieve all CGI global variables with one call to CGI GETVAR and place all of them into a stem variable. The statement is:

```
'CGI GETVAR * (STEM GV.'
```

This sets the global variables REQUEST_METHOD and HTTP_USER_AGENT under the `gv.` stem prefix as `gv.request_method` and `gv.http_user_agent`. More information on performance issues related to CGI GETVAR can be found in "VM:Webgateway Specific Considerations" on page 167.

Tip

If you accidentally use a string as the tail of the stem name that is already set as a variable in your program, you will get unexpected results! For example, you may code:

```
'CGI GETVAR * (STEM GV.'
query_string = gv.query_string
```

If later in your program, you reference `gv.query_string`, you will not get the value you expect because the symbol "query_string" is now a variable. A safer method is to force the stem tail names to be variable names that you are not likely to use in your program. For example:

```
'CGI GETVAR * (STEM GV.?'
query_string = gv.?query_string
```

If you do this, one thing to note is that the CGI command sets the variable *stem*.0 with the names of all the variables retrieved. In this example, all the variable names are found in the `gv.?0` variable.

Note: VM:Webgateway makes more global variables available to a CGI program than just the standard CGI global variables. Any additional variable names begin with `X_` so that you know that they are not standard variables. If you use these variables, remember that they may not be implemented in all Web server installations.

Unlike Webshare and EnterpriseWeb/VM, VM:Webgateway does not automatically access the disk or directory that contains the CGI program that it is running. So you cannot assume that other files that need to be read by the CGI program are available. VM:Webgateway provides a global variable named X_SCRIPT_NAME_TRANSLATED that contains the location of the CGI program that is running. This variable, along with the CMS Pipelines "<" stage, can be used to gain access to other files a CGI program needs. See the "Script_Location" subroutine in any of the VM:Webgateway sample CGI programs supplied on the VM Web CD.

3.3 Receiving User Input

Now that we have shown how a CGI program on VM can create dynamic HTML documents and we know how to get the global variables, let us see how to receive user input. Section 2.3.2, "Sending Data to a CGI Program" on page 15 gave us a general overview of the ways a CGI program can receive input. All of these methods are implemented in different ways in the VM Web server products. We start with the query string.

3.3.1 Using the Query String

This is the simplest form of "user input" available to a CGI program. The query string is anything that follows the "?" in the second word of the request line in the request header.

A user could manually type in a URL that includes a query string, but it is usually automatically included by the Web browser. There are four reasons it would be included:

1. An HTML FORM with METHOD=GET is submitted.
2. An <ISINDEX> HTML tag is specified in the header of the HTML document currently displayed on the client and the user enters some data.
3. An ISMAP attribute is specified on an <IMG> HTML tag and the user selects some part of that image with the mouse.
4. It is part of a URL specified in an HTML hypertext reference tag or a saved bookmark entry that is selected.

It is important to remember that the query string data is always URL encoded by the browser before it is sent. (See 2.3.3, "Encoding" on page 16 for a description of encoding.) In addition, if the query string is generated by a FORM, it will be form encoded. It must be decoded before you can use it, either by the CGI program or automatically by the Web server.

The query string is passed to the CGI program as a global variable named QUERY_STRING. Webshare compatible servers also pass the QUERY_STRING to the CGI program as a normal REXX argument string. (But note that the argument string may have one or more blanks appended to it.) First, you have to determine what kind of data it contains. Data from a FORM with METHOD=GET will create a QUERY_STRING with a value such as this example:

```
name=Jeff&company=IBM&title=Web+Programmer
```

Data from an HTML document with ISINDEX (sometimes called an ISINDEX query) will create a QUERY_STRING with a value such as this example:

```
Jeff+%26+company%3DIBM
```

Note: The HTML <ISINDEX> tag allows a CGI program to receive data from a client without creating an HTML FORM. It is a simple and quick way to prompt the user for input and receive the data. An HTML FORM allows careful control of the input format and data structure and is usually better suited for production applications.

A CGI program can determine if the query string is from an ISINDEX query or from a FORM by examining the query string. An ISINDEX query never contains unencoded equals signs (=). In other words, any equal signs in the query are changed to the string %3D before they are sent. A query string from a FORM will have one or more name=value strings from the form. So, the type of query is easy to determine using REXX:

```
from_isindex = (pos('=',query_string) = 0)
```

(But first make sure that the query string is not empty!)

Once you know how the query string was created, it still has to be decoded. Webshare and EnterpriseWeb/VM make the decoded query string available as the normal input stream (pipe input stream 0) to the CGI program but only if the value of the REQUEST_METHOD variable is GET. If the REQUEST_METHOD is POST, your CGI program has to decode the query string. Fortunately, there are utilities available to do this. The following examples assume that the query string is the result of an ISINDEX query. If it is the result of a FORM, see section 3.3.3, “Using a FORM with POST” on page 35, which discusses how to retrieve FORM data.

3.3.1.1 Webshare

Webshare does not provide a utility to perform decoding. However, the CMS Pipelines level that is included with CMS level 14 has a URLDEBLOCK stage that will perform the decoding. The data must be translated from EBCDIC to ASCII before decoding and then back to EBCDIC after decoding.

```
Parse arg query_string .
from_isindex = (pos('=',query_string) = 0 & query_string <> '')
/* Is the query_string from an ISINDEX tag? */
If from_isindex then
   'callpipe var query_string',
          '| xlate from 1047 to 819',
          '| urldeblock',
          '| xlate from 819 to 1047',
          '| var isindex'
else
   isindex=''
```

The xlate (translate) stages translate the data from code page 1047 (EBCDIC Latin 1/Open Systems) to 819 (ASCII ISO 8859 Latin Character Set 1) and back again. This translation is the most widely recommended one to use for Internet EBCDIC to ASCII translations. More information on code pages can be found in section 3.4.6.1, “What is a Code Page?” on page 50.

Note

If CMS Pipelines on your system does not have the URLDEBLOCK stage or the mentioned codepages, you can get a recent version from the “runtime library distribution” World Wide Web site. The URL is http://pucc.princeton.edu/%7Epipeline[4]

3.3.1.2 EnterpriseWeb/VM

EnterpriseWeb/VM provides an EWGET CMS Pipelines stage with a DECODED option that will decode the query string. Here is an example of how to decode the data from an ISINDEX query:

```
Parse arg query_string .
from_isindex = (pos('=',query_string) = 0 & query_string <> '')
/* Is the query_string from an ISINDEX tag? */
If from_isindex then
   'callpipe var query_string',
          '| ewget decoded',
          '| var isindex'
else
   isindex=''
```

Note that the translation table used by EWGET DECODED cannot be specified, however Beyond Software Inc. states that "the built-in translation tables should work for all languages that use the Latin alphabet, such as English, French, Swedish, German, Italian, and so on."

3.3.1.3 VM:Webgateway

VM:Webgateway provides a CGI URLDECODE command to decode the query string or any other URL-encoded data. To decode the query string, use the MODE TRANSFORMED option. Here is an example of how to decode the data from an ISINDEX query:

```
'CGI GETVAR QUERY_STRING (VAR QUERY_STRING'
/* Is the query_string from an ISINDEX tag? */
from_isindex = (pos('=',query_string) = 0 & query_string <> '')
If from_isindex then
   'CGI URLDECODE (VAR QUERY_STRING MODE TRANSFORMED',
                 'INTO ISINDEX.',
                 'TRANSLATE USENGLISH'
else
   isindex.1 = ''
```

This will place the decoded query string data into the variable isindex.1. A translation table must be specified on this command so that any encoded national language characters are translated to the correct EBCDIC characters.

3.3.2 Enhancing Our Sample Program

Let us enhance our telephone number application and use the QUERY_STRING to search for records. Our CGI program will accept a QUERY_STRING that is either the result of an ISINDEX query or from a FORM with METHOD=GET. HTML forms are discussed in the next section, so we will not use one to send a GET request to our CGI program. We have manually created an equivalent URL in an HTML document that invokes our CGI program when selected. This is a common practice that allows several different selections or queries to be handled by one CGI program. The basic format of the query string when it is used this way is name=value&name=value&.... Remember that any values must be encoded. For example, if we previously ran our CGI program with a URL of

```
http://wtscvmt.itso.ibm.com/~ vmwebcd/learn/cgi/phone1
```

4 The %7E that appears in this URL is an example of a URL encoded tilde character (~). The tilde character is not correctly displayed in all character sets, thus it is an example of an unsafe character which must be encoded for reliable transmission across the Web.

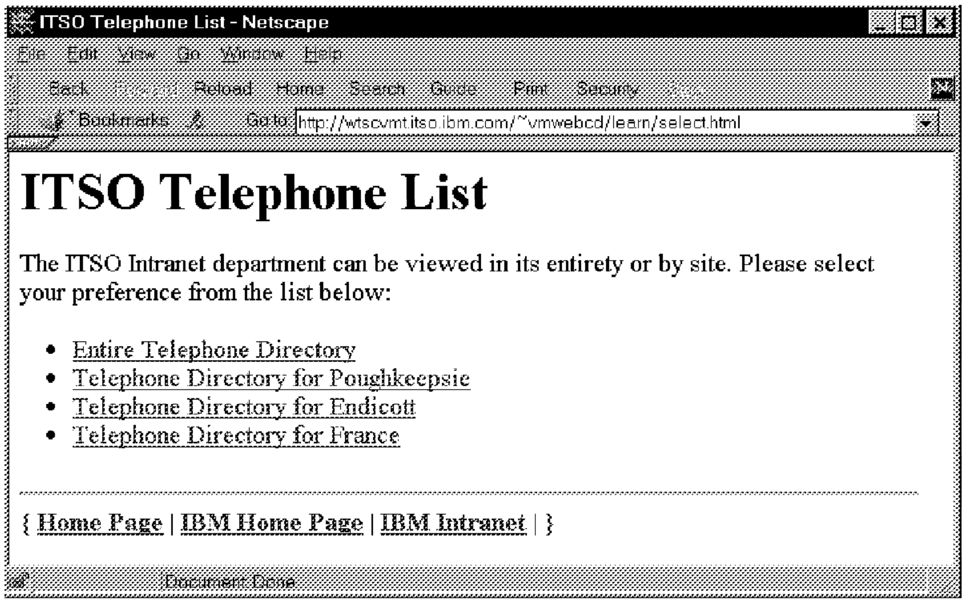
we will now run a new program named PHONE2 CGI and add a simple query string to the end:

```
http://wtscvmt.itso.ibm.com/~ vmwebcd/learn/cgi/phone2?location=POK
```

To make this easier for the user to select, we created a file named SELECT HTML that has URL references to the three locations in our PHONE DATA file. Figure 9 shows this file and Figure 10 shows what it looks like on a Web browser.

```
<HTML><HEAD>
<TITLE>ITSO Telephone List</TITLE>
</HEAD><BODY>
<H1>ITSO Telephone List</H1>
<P>The ITSO Intranet department can be viewed in its entirety or by
site.  Please select your preference from the list below:</P>
<UL>
<LI><A HREF="cgi/phone2">
Entire Telephone Directory</A>
<LI><A HREF="cgi/phone2?location=pok">
Telephone Directory for Poughkeepsie</A>
<LI><A HREF="cgi/phone2?location=end">
Telephone Directory for Endicott</A>
<LI><A HREF="cgi/phone2?location=fra">
Telephone Directory for France</A>
</UL>
<HR>
<B>{
  <A HREF="home.html">Home Page</A> |
  <A HREF="http://www.ibm.com">IBM Home Page</A> |
  <A HREF="http://w3.ibm.com/">IBM Intranet</A> |
}</B>
</BODY></HTML>
```

Figure 9. Sample SELECT HTML Document


ITSO Telephone List - Netscape
File Edit View Go Window Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Go to: http://wtscvmt.itso.ibm.com/~vmwebcd/learn/select.html
Document: Done

ITSO Telephone List

The ITSO Intranet department can be viewed in its entirety or by site. Please select your preference from the list below:

- Entire Telephone Directory
- Telephone Directory for Poughkeepsie
- Telephone Directory for Endicott
- Telephone Directory for France

{ Home Page | IBM Home Page | IBM Intranet | }

Figure 10. Display of Sample SELECT HTML Document

Now let us change to our PHONE CGI program to read the QUERY_STRING and take action based on its content. If no query string is sent, the entire table is displayed as before. If an ISINDEX query is sent, only records that contain the specified string are displayed. Otherwise, we display only records for a specified location. To enable an ISINDEX query, the PROLOG1 HTMLPART file is renamed to PROLOG2 HTMLPART and an <ISINDEX> tag is added before the </HEAD> tag. The modified program is shown in Figure 11.

```
/*  PHONE2 CGI program example */
PhoneFile = 'PHONE DATA'          /* File of telephone number data  */
DataDir   = 'SFSTEST:VMWEBCD.WEBSHARE.DATA' /* data file location   */

Parse arg query_string .          1
parm.=''                          /* Initialize default values      */
from_isindex = (pos('=',query_string) = 0)
Select
  When query_string = ''  then nop /* Nothing to process            */
  When from_isindex Then
    'callpipe var query_string',
           '| xlate from 1047 to 819',  /* EBCDIC to ASCII          */
           '| urldeblock',
           '| xlate from 819 to 1047',  /* ..and back               */
           '| var isindex'
  Otherwise
    /* Webshare will make query string data available on stream 0   */
    'callpipe *:',                2
           '| xlate 1-* 05 40',   /* In case of any tab chars       */
           '| strip',             /* Ignore any empty ones          */
           '| locate 1',
           '| xlate fieldsep = f1 upper',  /* Upper case field name */
           '| specs "=PARM.?" 1 1-* next', /* Put it in varload fmt */
           '| varload'
End
Search=''                         /* Which entries to show.         */
SearchHTML=''                     /* Any additional text to show.   */
Select                            3
  When query_string = ''  then nop /* Nothing to search for         */
  When from_isindex Then do
    Search='| locate anycase 1-*  "'isindex'"'
    SearchHTML='<P>The results of your search request are shown.</P>'
  End
  When parm.?location <> ''  Then do
    SearchArg = translate(left(parm.?location,3)) /* Search string  */
    Search='| locate 42-44 "'SearchArg'"'
    SearchHTML='<P>Only telephone numbers in location' SearchArg,
               'are shown.</P>'
  End
  Otherwise
End
```

Figure 11 (Part 1 of 3). Sample PHONE2 CGI Program

```
'callpipe < PROLOG2 HTMLPART | *:'/* Display the beginning HTML     */
                                 4
If SearchHTML <> ''  then        /* Extra message to display?      */
   'output' SearchHTML

'callpipe <sfs' PhoneFile DataDir, 5
       '| nlocate 1-1 "*"',      /* Ignore "comments" in col. 1    */
          Search,                /* Maybe search for a location    */
       '| stem phone.'           /* Save result                    */
```

Figure 11 (Part 2 of 3). Sample PHONE2 CGI Program

```
'output <TABLE BORDER="1">'      /* Set up the HTML table          */
'output <TH>Employee Number</TH>' /* Put a title on each column    */
'output <TH>Name</TH>'
'output <TH>Phone Number</TH>'
'output <TH>Location</TH>'
'output <TH>Office</TH>'
'output <TH>Status</TH>'          6

Do i=1 to phone.0                /* Format the file for display    */
   'output <TR>'                 /* Beginning of a table row       */
   'output <TD>' substr(phone.i, 1, 6) '</TD>'  /* Each            */
   'output <TD>' substr(phone.i, 8,20) '</TD>'  /*   table         */
   'output <TD>' substr(phone.i,29,12) '</TD>'  /*      cell       */
   'output <TD>' substr(phone.i,42, 3) '</TD>'
   'output <TD>' substr(phone.i,46, 9) '</TD>'
   'output <TD>' substr(phone.i,56, 5) '</TD>'
   'output </TR>'                /* End of a table row             */
End
'output </TABLE>'
'callpipe < FOOTER HTMLPART | *:' /* The ending HTML               */
Exit RC
```

Figure 11 (Part 3 of 3). Sample PHONE2 CGI Program

These notes refer to Figure 11 on page 33.

1 The QUERY_STRING is passed to the CGI program as an argument. The Web server may include some trailing blanks (which should not be there) so we only get the first word. If it is empty, do nothing. If it is from an ISINDEX query, decode it using URLDEBLOCK. (For more information about URLDEBLOCK, see 3.3.1.1, "Webshare" on page 30.)

2 If the query string is the result of an HTML form, read standard input, transform the data and set variables. This will set a series of variables named `PARM.?`*name*`=value`, where *name* is the name of the field from the form and `value` is its value. In our example, if the query string contains the string `location=pok`, then the VARLOAD stage will set `PARM.?LOCATION=pok`. See the Tip on page 28 for an explanation of why we are setting variables named `PARM.?`*name* instead of just naming them `PARM.`*name*.

Note: For historical reasons, Webshare makes the decoded query string available as the normal input stream (pipe input stream 0). EnterpriseWeb/VM and VM:Webgateway in Webshare compatibility mode also do this. This is why our CGI program can read the decoded query string from pipe input stream 0 instead of decoding the query_string variable.

3 This select statement sets up the search based on how the query string was decoded. If no query string was passed, the entire file is shown by default. If the query string is from an ISINDEX search, then search the entire telephone record for the string (case does not matter.) If the query string is from a location variable sent to the program, then search for that location.

4 These two pipeline commands output the PROLOG2 HTMLPART file to the client and output the "search" informational message if a search request was made.

5 This pipeline is based on the pipeline after marker **2** in Figure 7 on page 23 with additional stages in the middle. Instead of using a REXX "If" statement to ignore comment lines, we use a pipelines NLOCATE stage. Also, if a search string was specified, the stage placed in the `Search` variable will be executed. CMS Pipelines has a rich set of stages that can be used for selecting and manipulating data, so using pipeline stages for selecting records is preferred.

6 This part of the program is the same as PHONE1 CGI (see Figure 7 on page 23) except that the "If" statement that ignored comment lines has been removed.

3.3.2.1 VM:Webgateway Changes

The same changes and considerations explained in section 3.1.3, "Sample CGI Program for VM:Webgateway" on page 24 still apply to our latest example. We must also change how global variables are fetched and how the query string is decoded. How to do this has been documented in section 3.2.3, "VM:Webgateway Global Variables" on page 28. However, we have not mentioned how to decode FORM data, but it is very similar to decoding the data from an ISINDEX query. This statement will set values in the `parm.?`*name* stem in the same manner as the pipeline at marker **2** in Figure 11 on page 33:

```
'CGI URLDECODE (VAR QUERY_STRING MODE TRANSFORMED',
              'INTO PARM.?',
              'TRANSLATE USENGLISH'
```

A complete sample program is available on the VM Web CD as PHONE2 SVMEXEC.

3.3.3 Using a FORM with POST

Our sample CGI program now has search capabilities but does not have a way for users to update their own telephone number entries. We can add that capability by sending the user an HTML FORM that has fields for the updated data. It also specifies a CGI program to run when the data is submitted. We will create a new HTML document that contains the update form and a new CGI program that will process the update.

First, one small change is made to the existing PHONE2 CGI program. It displays a hypertext link to the new update form on the telephone number display window after the table. For Webshare and EnterpriseWeb/VM we write:

```
'output </TABLE>'
'output <P><A HREF="../update.html">Update your telephone entry</A></P>'
```

Note: You may notice that the HREF refers to ″../update.html″ instead of just ″update.html″. This is called a relative path, because the path to the referenced object is *relative* to the location of the current object. In this case, the current URL location is the CGI subdirectory and the update.html document is in the "base" directory. Relative path names are used in all the examples in this chapter so that all references are portable from server to server. More information can be found in the section "Relative URL Addressing Inside an Application" in *Web Server Solutions for VM/ESA*, SG24-4874.

VM:Webgateway and EnterpriseWeb/VM can store both CGI programs and other objects in the same directory, so a separate CGI directory is not necessary. Both of these products determine the content type of a file from its file type instead of using a FILELIST file.

3.3.3.1 HTML Form

First let us look at the HTML input form in the file UPDATE HTML. Besides the usual prologue, heading, and footer HTML tags, it contains an HTML form that starts with a <FORM> tag and ends with </FORM>. A better description of forms and form elements is found in the section "Creating Forms" in *Web Server Solutions for VM/ESA*, SG24-4874. See Figure 12, which shows the HTML document, and the notes following it.

```
<HTML><HEAD>
<TITLE>ITSO Telephone Number Update</TITLE>
</HEAD><BODY>
<H1>ITSO Telephone Number Update</H1>
<FORM METHOD=POST ACTION=″cgi/update″>                        1
<P>Please enter your new or updated Telephone information
and press the <B>Submit</B> button.</P>
<P>Employee Number:
  <INPUT TYPE=″text″ NAME=″empnum″ SIZE=6 MAXLENGTH=8></P>    2
<P>Name:
  <INPUT TYPE=″text″ NAME=″name″ SIZE=20 MAXLENGTH=20></P>
<P>Telephone number:
  <INPUT TYPE=″text″ NAME=″phone″ SIZE=12 MAXLENGTH=12></P>
<P>Location (select one)                                      3
  <INPUT TYPE=″radio″ NAME=″location″ VALUE=″POK″> Poughkeepsie
  <INPUT TYPE=″radio″ NAME=″location″ VALUE=″END″> Endicott
  <INPUT TYPE=″radio″ NAME=″location″ VALUE=″FRA″> France</P>
<P>Office location:
  <INPUT TYPE=″text″ NAME=″office″ SIZE=9 MAXLENGTH=9></P>
<P>Status (select one):
  <INPUT TYPE=″radio″ NAME=″status″ VALUE=″In″ CHECKED> In    4
  <INPUT TYPE=″radio″ NAME=″status″ VALUE=″Out″> Out</P>
  <INPUT TYPE=″hidden″ NAME=″version″ VALUE=″1.2″>            5
<P>Reset fields:
  <INPUT TYPE=″reset″>                                        6
  (Pressing this will clear all fields.)</P>
<P>Select to submit your responses:
  <INPUT TYPE=″submit″ VALUE=″Submit″></P>                    7
</FORM></BODY></HTML>
```

Figure 12. Sample HTML Document Named UPDATE HTML with Input Fields

The following notes refer to Figure 12.

1 This is the beginning of the HTML form. Notice that we have defined the *method* that the Web browser will use to return data to the server (in this case the POST instead of the GET.) The other thing you see in the FORM tag is the *action* that defines what object in the Web server will receive the data. In this case, it is our UPDATE CGI program.

2 This is the first input definition in the form. It defines a *text* type input field that should be wide enough on the screen to enter 8 characters (`SIZE=8`) and not allow any more than 8 characters to be entered (`MAXLENGTH=8`). This field also has a NAME of *empnum*, which is used by our CGI program to identify the value entered into this field. You may also notice that between the input fields in the form we can put any other text or HTML tags that are needed to make the form readable and usable.

3 This is another type of input definition. A *radio* type will be displayed on the browser screen as a set of buttons that the user can "push" using the mouse. Notice that the set of buttons all have the same name but different values. Only one of the values (depending on which button was pushed) will be sent to our CGI program.

4 This is a radio button definition just like the last one, but notice that a keyword of `CHECKED` has been added. This instructs the browser to initially select this button instead of initializing the buttons without any of them selected. If the user does not select another button of this group, this default selection will be sent to the CGI program.

5 This is a *hidden* input type. As the name implies, it is not shown on the browser window but the name and its associated value are returned to the CGI program. In this example, the "version" of the HTML form is sent to the CGI program so that it can check to see if an old form was used. See the discussion of cache validation on page 158 for background on the importance of such tests for statically built forms.

6 A *reset* input field defines a button that, when pressed, will reset all the input fields to their initial values. This function is performed by the browser, but only if a reset type field is present in the form. Whether or not you provide this function on your forms depends on your application.

7 The *submit* button is the button "pressed" by the users once they have filled out the form and want to send it to the Web server. The *Value* field defines what text is used to label the button; it is not sent to the CGI program.

3.3.3.2 POSTTEST CGI Utility

Before we present the CGI program that will process this form, let us mention a utility CGI program that will help you test and debug HTML forms and CGI programs. The Webshare Web server comes with a sample program named POSTTEST CGI. When POSTTEST is referenced as the ACTION of an HTML form, it returns an HTML document that displays all of the HTML FORM input fields and values sent from the client. It also displays all the environmental information available to a CGI program, such as all of the header fields sent from the client and all CGI global variables. It is very useful for debugging your HTML forms before you write your CGI program.

Note: If you do not have the POSTTEST CGI program available on your Web server, you can use the "Sample Universal CGI for Use with All VM Web Servers" CGI program documented in an appendix of *Web Server Solutions for VM/ESA*, SG24-4874 and available on the VM Web CD as TEST CGI. This CGI program can be run on any VM Web server and will display HTML FORM INPUT fields as well as CGI global variables and environment information.

In order to test our UPDATE HTML document before we create the UPDATE CGI program, simply change the ACTION reference of the form (see **1** in Figure 12 on page 36) to invoke POSTTEST CGI. This line of UPDATE HTML now reads:

```
<FORM METHOD=POST ACTION="/cgi-bin/posttest">
```

A portion of the output from POSTTEST CGI looks like this example, showing all input fields by name on the form and the value filled in by the user:

```
Results: (what you posted (if anything))
empnum=111111
name=John Q Public
phone=888-555-1111
location=POK
office=8-2-C14
status=In
version=1.2
```

3.3.3.3 Processing the Form

Now that we have written and tested the form that the user can fill out, we present the UPDATE CGI program that actually updates our PHONE DATA file. The following figures show the program with explanations following each figure. VM:Webgateway changes are also described in the explanations. A complete sample program is available on the VM Web CD as UPDATE SVMEXEC.

```
/*  UPDATE CGI program example */
PhoneFile = 'PHONE DATA'          /* File of phone number data      */
DataDir   = 'SFSTEST:VMWEBCD.WEBSHARE.DATA' /* data file location   */
parm.=''                          /* Initialize default values      */

'callpipe (end %)',
       '*:',                      /* Read input from the form       */
      '|  xlate 1-* 05 40',       /* Convert any tab chars to space */
      '|  strip',                 /* Ignore any empty ones          */
      '|  locate 1',
      '|  xlate fieldsep = f1 upper',  /* Upper case field name     */
      '|f:fanout',
      '|  specs "=PARM.?" 1 1-* next', /* Put it in varload format  */
      '|  varload',
      '%f:',
      '|  chop before string "="',     /* Save just the field name  */
      '|  join * " "',                 /* Make 1 string of names    */
      '|  var parm.?0'
```

Figure 13. UPDATE CGI Program, Part 1 of 5

Figure 13 shows a CMS Pipelines program that reads the form data sent using the POST method from the form. The data sent from the Web browser is actually URL form encoded See 2.3.3, "Encoding" on page 16 for more information on encoding in ASCII, but the Web server decodes it and converts it to EBCDIC for us. However, some characters may still cause problems, so this pipeline converts any tab (EBCDIC X'05') characters to a space and removes any leading or trailing spaces and any empty lines. It then loads the parm.? stem variable with our input data and the variable parm.?0 with the names of all the input fields.

For a VM:Webgateway native CGI program, the CGI READ command is used to read form data sent from the client and the CGI URLDECODE command to decode it:

```
'CGI READ 1 (VAR INPUT TRANSLATE USENGLISH'
'CGI URLDECODE (VAR INPUT MODE TRANSFORMED INTO PARM.?',
   'TRANSLATE USENGLISH'
```

The command `CGI READ 1` actually reads all of the input data that is available, not just one POST variable. The World Wide Web is an ASCII-based system and “records” that we are used to seeing in CMS do not have the same meaning. All the encoded data is loaded into one variable. `TRANSLATE USENGLISH` means that we want the ASCII characters translated to EBCDIC using a built-in translation table provided with VM:Webgateway. If `TRANSLATE NONE` is specified instead, the input data is stored in the variables in ASCII.

To decode the input data, we once again use the CGI URLDECODE command and place the results in a stem. CGI URLDECODE places data in the stem in essentially the same way the CMS Pipelines program in Figure 13 on page 38 does.

Note

A Web browser could post data to your CGI program with duplicate, unreadable or missing field names. Also, the values returned with the fields could be longer than you expected. This may happen when a new form is being developed or because someone is being malicious. For example, if the HTML form had more than one input field named "phone" or a field with NAME="++.&*", these names *will be* sent to your CGI program! Make sure your program will not abnormally terminate if unknown fields are sent. This is also why the examples in this book read form variables into REXX stems instead of directly setting variables. It prevents a malicious user from manipulating critical variables in your CGI program by creating form variables that match your variables.

In our simple example, duplicate or invalid field names are ignored. If your CGI program needs to be able to handle duplicate names or fields with special names, most Web servers have utilities that help process the input. EnterpriseWeb/VM contains an EWGET utility to assign FORM POST input to a set of stem variables in your CGI program. The CGI URLDECODE command of VM:Webgateway has a predefined algorithm for how it assigns duplicate field names to variables and how it translates “obscure” field names into valid REXX variable names. Consult the programming documentation of each respective Web server product for more details on these utilities.

```
FormVars  = parm.?0
FormError = 0
Do while FormVars <> ''           /* Go through all form variables  */
   Parse var FormVars onevar FormVars
   onevar = '?'||onevar
   If parm.onevar = ''  Then Do    /* Are any of them blank?         */
      FormError = 1               /* If so - issue an error message */
      Leave
   End
End
```

Figure 14. UPDATE CGI Program, Part 2 of 5

Either the pipeline program in Figure 13 on page 38 or CGI URLDECODE has placed the names of all the input fields into the variable parm.?0 and all the values into parm.?*name*, where *name* is the name of each input field. In Figure 14, a simple loop checks to make sure no input field on the form is empty. If it finds an empty field, it sets a flag so that the user is informed of the error.

```
Select
/* Any field missing? */
  When FormError Then Do                 1
    Update = '<B>not successful</B>.'
    Reason = 'The failure was due to a blank or missing input field. '
    Action = 'Please <A HREF="../update.html">return</A> to the',
             'update screen and resubmit your update.'
  End
  When parm.?version /== '1.2' Then Do   2
    Update = '<B>not successful</B>.'
    Reason = 'You have an old or incorrect level of the input form.'
    Action = 'Please <A HREF="../update.html">return</A> to the',
             'update screen and reload the form.'
  End
  Otherwise                              3
    RC = UpdateFile()

    If RC = 0 Then Do
       Update = 'successful.'
       Reason = 'Thank you for keeping your records up to date!'
       Action = ''
    End
    Else Do
       Update = '<B>not successful</B>.'
       Reason = 'The failure was due to a program error.'
       Action = 'Please notify system support.'
    End
End
```

Figure 15. UPDATE CGI Program, Part 3 of 5

Next we process the form. In Figure 15, a Select statement is used to check for some conditions:

1 If any field is empty, we set some variables with friendly messages to the user that are sent back in our response. Notice that we can include any HTML tags that we like, even references back to the form that the user has completed.

2 Here is where the hidden input field is examined. If the user used an old version of the form and submitted it (due to the user′s Web browser using an old copy of the form from its cache), this program does not accept their input. See the discussion of cache validation on page 158 for background on the importance of such tests for statically built forms.

3 If there are no errors, a subroutine is called to update the file and messages are created to tell the user if it worked. The subroutine is shown in Figure 17 on page 42.

```
/* HTML document to output to tell the user if their update  */
/* worked or failed.  Also, give them a link to go to.       */
out.1 =′<HTML><HEAD>′
out.2 =′<TITLE>ITSO Telephone Update</TITLE>′     4
out.3 =′</HEAD><BODY>′
out.4 =′<H1>ITSO Telephone Update</H1>′
out.5 =′<P>Your update was′ Update ′</P>′
out.6 =′<P>′ Reason Action ′</P>′
out.7 =′<P><A HREF=″phone2″>Return to the telephone′,
       ′number table.</A></P>′
out.8 =′<HR>′
out.9 =′<B>{′
out.10=′<A HREF=″home.html″>Home Page</A> |′
out.11=′<A HREF=″http://www.ibm.com″>IBM Home Page</A> |′
out.12=′<A HREF=″http://w3.ibm.com/″>IBM Intranet</A>′
out.13=′}</B>′
out.14=′</BODY></HTML>′
out.0=14

′callpipe stem out. | *:′              5

Exit RC
```

Figure 16. UPDATE CGI Program, Part 4 of 5

Figure 16 shows how a response is sent back to the user.

4 This is a skeleton HTML document, assigned to a series of stem variables, that is used to send a response to the user. Notice how the variables set in Figure 15 on page 40 are used in the response. This is a very simple example of a CGI program creating a dynamic HTML document.

5 In a VM:Webgateway native CGI program, the output statement is:

`′CGI WRITE DOCUMENT (TRANSLATE USENGLISH CRLF STEM OUT.′`

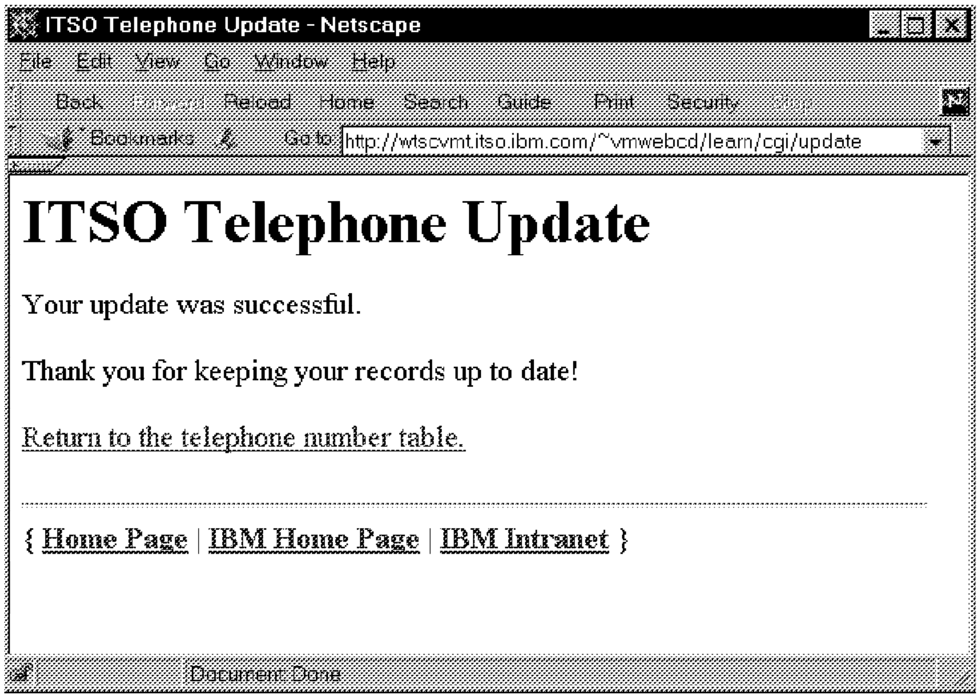
```
UpdateFile:
/* Update the file                                               */
/* Create a record with the data entered onto the form           */
UpdateLine = left(parm.?empnum,6) left(parm.?name,20),
             left(parm.?phone,12) left(parm.?location,3),
             left(parm.?office,9) left(parm.?status,5)

'callpipe (end ?)',                /* Update the file             */
         '<sfs' PhoneFile DataDir,
      '|c:strfind "*"',            /* Put all comments at the top */
      '|f:fanin',
      '|  >sfs' PhoneFile Datadir,
      '?c:',
      '|  preface var UpdateLine',  /* Add in the update           */
      '|  sort unique 1.6',         /* Keep only 1 employee number */
      '|f:'
Return RC
```

Figure 17. UPDATE CGI Program, Part 5 of 5

If all the fields are filled in, the subroutine in Figure 17 updates the PHONE DATA file with the new data. This is a fairly simple CMS Pipelines program that adds the new record to the file and then uses SORT UNIQUE to eliminate any duplicate entries.

3.3.3.4 Successful Execution

If the user fills in the form correctly and submits the update, a screen like the one shown in Figure 18 is shown.


ITSO Telephone Update - Netscape
File Edit View Go Window Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Go to: http://wtscvmt.itso.ibm.com/~vmwebcd/learn/cgi/update
Document: Done

ITSO Telephone Update

Your update was successful.

Thank you for keeping your records up to date!

Return to the telephone number table.

{ Home Page | IBM Home Page | IBM Intranet }

Figure 18. Successful Telephone Number Update

3.4 Server Directives and Headers

Back in 2.2.1, “Request Header” on page 10, we showed how the client sends request header fields with its request and in 2.2.2, “Response Header” on page 12, how the server sends response header fields with any response. The request header fields are available to a CGI program so that the program can change its response or discover other information about the client. Also, the CGI program can send additional response header fields to the client to communicate additional information about the transaction, such as the type of the object being sent, a time stamp of the object being sent, or a cookie (see 3.4.5, “Using Cookies” on page 47 for more on cookies.)

3.4.1 Sending Header Fields

In section 2.3.2, “Sending Data to a CGI Program” on page 15, we said that a CGI program can send *server directives* to the Web server, which creates the status line and response header fields to send to the client. CGI programs are not required to send any server directives because the VM Web server automatically writes a response and some normal header fields if none are sent by the program. The first directive that is sent is `Status` with one of the response codes listed in Table 2 on page 12. Usually this is `Status: 200 OK`. Your CGI program should also write out a Content-Type directive. For Webshare or EnterpriseWeb/VM, instead of sending a Status directive, you output the actual HTTP response line, like this:

```
'output HTTP/1.0 200 OK'
'output Content-Type: text/html'
'output'
```

Since the HTTP protocol specifies that a blank (null) line must be sent between the header and the actual object being sent, a Webshare CGI program must send a blank (null) line at the end of the header.

VM:Webgateway has a command just for writing server directives: CGI WRITE HEADER. Just like CGI WRITE DOCUMENT, it has options for STRING, VAR, and STEM. VM:Webgateway does not send the response headers to the client until a CGI WRITE DOCUMENT command is executed by the CGI program and then it automatically inserts the required blank (null) line. To write the status and content type, use the following code:

```
'CGI WRITE HEADER (STRING Status: 200 OK'
'CGI WRITE HEADER (STRING Content-Type: text/html'
```

Tip

The `Content-Length` header field tells the client the length (in bytes) of the data portion of the response. If it is not sent, the client reads data until the connection is closed. If you know the length of the data to be sent, it is a good idea to send this header field. If sent, the `Content-Length` header must be exact to the byte. The client will know how much data to expect, be able to display the percentage that has been loaded as it is received, and may be able to keep the server connection open for receiving another data object.

If the data written by your CGI program is not an HTML document, then the Content-Type header field you write must agree with the data you are writing. For instance, if your CGI program sends an image you would use the following:

```
'CGI WRITE HEADER (STRING Status: 200 OK'
'CGI WRITE HEADER (STRING Content-Type: image/gif'
'PIPE <' image_file '| stem image. | count bytes | var length'
'CGI WRITE HEADER (STRING Content-Length:' length
'CGI WRITE DOCUMENT (TRANSLATE NONE STEM IMAGE.'
```

Notice that TRANSLATE NONE is specified so that the Web server does not translate the data from EBCDIC to ASCII.

Tip

You may have a CGI program that takes longer than average to gather data in order to send a response (for instance, it has to access a remote system.) Instead of not sending anything to the user until the data is gathered, the CGI program could send the response header and the initial HTML "prologue" tags so that the client gets an immediate response. For example:

```
'CGI WRITE HEADER (STRING Status: 200 OK'
'CGI WRITE HEADER (STRING Content-Type: text/html'
'CGI WRITE DOCUMENT (TRANSLATE USENGLISH CRLF STRING',
   '<HTML><HEAD><TITLE>Query Results</TITLE></HEAD>'
/* Now begin processing the transaction */
```

However, do not send this response if your CGI program needs to sometimes send a different response code, such as 404 Not Found when the request cannot be processed. Only one status code can be sent in a response, and once 200 Ok is sent, a different status code, such as 404, cannot be sent later.

3.4.2 Reading Header Fields

All header fields are made available to your CGI program as environment variables that are named HTTP_*name*, where *name* is the name of each header field (capitalized) and with all dashes in the name translated to underscores. See section 3.2, "Fetching CGI Global Variables" on page 26 for information on how to read these variables.

Note: Remember back in section 3.2.1, "Webshare Global Variables" on page 26 we stated that Webshare and EnterpriseWeb/VM do not create environment variables of all response header fields. Use the CMS Pipelines program example shown in that section to read the header fields that are available on pipe input stream 1 and set the proper variables.

3.4.3 Making Efficient Use of the Browser's Cache

Many Web browsers maintain a *cache* of objects that they have received. This is done so that frequently visited Web sites can appear to load much faster because the data does not need to be sent over the network again. This is especially important for graphic images, which can be large and numerous. However, the Web is a dynamic media, and data *does* change, so the Web browser's cache must be dynamic also. It is done by the Web server sending a time stamp with each object which is stored in the cache, and the Web browser sending back that time stamp whenever it requests an object. Web servers send this information automatically for static HTML documents and graphic images that are sent. A CGI program can also send this information with the HTML document it is sending, if a "last modified date" can be determined.

3.4.3.1 HTTP Time Stamp Formats

The HTTP protocol specifies formats for sending and receiving time stamps. There are three allowed formats, presented here as examples in order of preference. Times are always expressed as UTC (Coordinated Universal Time) but the time zone designation is always given as GMT (Greenwich Mean Time.)

1. `Mon, 09 Nov 1998 09:10:18 GMT`
   This format is specified in RFC 1123 and is the required format when sending a date. The day of the week and the month are always specified as English abbreviations.

2. `Monday, 09-Nov-98 09:10:18 GMT`
   This is a commonly used format defined in RFC 850. Notice that the year is only 2 digits.

3. `Mon Nov  9 09:10:18 1998`
   This format is defined by the ANSI C language `asctime()` format. A time zone is not specified; it is assumed to be GMT.

Our CGI program could receive time stamps in any of these formats. However, we should only send out time stamps in the first format.

3.4.3.2 Converting Time Stamps in REXX

REXX does not provide a built-in function to change time stamps from the local time zone to UTC and back. The CSL function DateTimeSubtract can do this conversion, along with some REXX code to create the proper format. We created REXX function programs that convert file time stamps in ISODATE format (`YYYY-MM-DD HH:MM:SS`) to the standard HTTP format and back. We named these functions ISOHTTP and HTTPISO that are available on the VM Web CD. Here is an example of their use from the command line. (They can also be called as functions.)

```
ISOHTTP 1998-11-09  4:10:18
Input = 1998-11-09  4:10:18
Output= Mon, 09 Nov 1998 09:10:18 GMT

HTTPISO Mon, 09 Nov 1998 09:10:18 GMT
Input = Mon, 09 Nov 1998 09:10:18 GMT
Output= 1998-11-09  4:10:18

HTTPISO Thursday, 9-Nov-00 09:10:18 GMT
Input = Thu, 9-Nov-00 09:10:18 GMT
Output= 2000-11-09  4:10:18
```

The best way to get a time stamp of a file is to use the CMS Pipelines STATE stage, because it can get the time stamp of a file in SFS without requiring the directory to be accessed. Here is a function we added to the program:

```
GetISOdate: procedure
/* Use PIPE STATE to get a date/time stamp on a file. */
Parse arg file
'callpipe var file',
      '| state isodate',         /* Get the date of the data file  */
      '| spec 57-75 1',          /* (in ISODATE format.)           */
      '| var filedate'
Return RC filedate
```

3.4.4 Examples of Reading and Writing Headers

We will change our PHONE CGI program so that it outputs the file time stamp on the PHONE DATA file as a Last-Modified header field. Also, if the client sends an If-Modified-Since header field, we will compare that time stamp with the file time stamp. If they are the same, we only have to send a response of 304 Not Modified instead of the entire HTML document.

3.4.4.1 The Last-Modified Header Field

Using the date and time conversion functions discussed in 3.4.3.2, "Converting Time Stamps in REXX" on page 45, additional code is added to the PHONE2 CGI program to create a Last-Modified header field. This code is shown in Figure 19. The complete header must be written to the server before any of the HTML document is sent, so this code fragment is inserted before the HTML "prologue" is written. The updated program is named PHONET CGI (or PHONET SVMEXEC for VM:Webgateway) on the VM Web CD.

Note: A Web browser saves in its cache only objects it receives from an HTTP GET request. This means that an object sent as a result of an HTTP POST from an HTML FORM will not be saved in the browser's cache. The example given here is invoked with an HTTP GET request, with additional parameters specified in the query string of the request.

```
'CGI WRITE HEADER (STRING Status: 200 OK' /* Response and headers   */
'CGI WRITE HEADER (STRING Content-Type: text/html'

/*-----------------------------------------------------------------*
 * Get the date stamp on our PHONE file and send it in
 * the correct HTTP format as the last modified date.
 *-----------------------------------------------------------------*/
Parse value GetISOdate(PhoneFile DataDir) with RC filedate
If RC=0 then
   Parse value ISOHTTP(filedate) with RC httpdate
If RC=0
   then 'CGI WRITE HEADER (STRING Last-Modified:' httpdate
   else 'CGI EMSG (MSGFILE STRING 0690E RC='rc 'from ISOHTTP('filedate')'
```

Figure 19. Example of Sending a Last-Modified Header Field

3.4.4.2 The Expires Header Field

If you know that the data sent to the client is only valid for a limited time, you can tell the client with the Expires header field when it should delete it from its cache. Its format is the same as the Last-Modified header field. This field is fairly easy to create in REXX if the data expires in days instead of hours. For example, if our data expires in two days, this example shows how to send an Expires field that is approximately correct:

```
expires = date('B') + 2   /* Data expires in 2 days or so. */
expires_header = left(date('W',expires,'B'),3)',',
                 date('N',expires,'B') '23:59:59 GMT'
'CGI WRITE HEADER (STRING Expires:' expires_header
```

Note: If the level of REXX on your system does not allow date conversions using the built-in DATE function, you may be able to use the DATECONVERT CMS Pipelines stage if you have a recent level of pipes. See the note on page 30 for information on how to obtain a more recent CMS Pipelines run time library.

3.4.4.3 Inspecting If-Modified-Since

If the Web browser has a cached copy of the document that it is requesting, it will send a request header field named If-Modified-Since with the time stamp of the file in its cache. Our CGI program may optionally inspect this field and compare the time stamp with the time stamp of the file requested. If they match, the Web server does not need to send a new file to the client. This code fragment is inserted near the beginning of the program. (The example shows VM:Webgateway as the Web server.)

```
/*-----------------------------------------------------------------*
 * Before we do any processing, see if we can skip all
 * of it because the browser has a good cached copy.
 *-----------------------------------------------------------------*/
'CGI GETVAR HTTP_IF_MODIFIED_SINCE (VAR CLIENT_DATE'
If client_date <> '' then
   Call Check_Modified PhoneFile DataDir, client_date
```

It calls the Check_Modified subroutine shown in Figure 20.

```
Check_Modified:
/*-----------------------------------------------------------------*
 * Convert the date from the If-Modified-Since header to ISODATE
 * format and compare with the date stamp on the file.
 * If they match, this routine never returns to the main program.
 *-----------------------------------------------------------------*/
Parse arg file, client_date
Parse value GetISOdate(file) with RC filedate
If RC <> 0 then Return
Parse value HTTPISO(client_date) with RC cachedate
If RC <> 0 then Return
If filedate == cachedate then do
   /* They match!  Just write our status and exit */
   'CGI WRITE HEADER (STRING Status: 304 Not Modified'
   Exit
end
Return
```

Figure 20. Example of Checking the If-Modified-Since Header

3.4.5 Using Cookies

In 2.4.1, "Creating a Web Transaction" on page 18, we discussed different ways to save "state" information across Web transactions. One way mentioned was to use *Cookies* (sometimes called *Netscape cookies*.) Netscape introduced this mechanism so that a Web browser can store on a local storage device specific state information sent by the server. The Web server sends a Set-Cookie header field with a normal HTTP response. The cookie data in this field is stored by the Web browser. Then, whenever the cookie domain is a tail match for the document and the cookie path is a prefix match for the document path, the cookie data is automatically sent back as Cookie request header fields. For example, if the cookie had been sent with DOMAIN=acme.com and PATH=/coyote, and if the browser were requesting the document http://westcoast.acme.com/coyote/orderhistory.html, the browser would return the cookie.

Note

Cookies are not universal because not all Web browsers implement them. Receiving cookies can sometimes be turned off in a browser that does support them. Also, if too many cookies are received, the Web browser is allowed to arbitrarily delete some. For these reasons, you should not absolutely rely on cookies always working.

The data sent in a cookie header field is `name=`*`value`* data similar to data received from an HTML FORM. Program techniques similar to those used to read FORM data are used to read cookie data.

3.4.5.1 Cookie Example

As an example, we will use cookie data to preload some data on our telephone data change form. If the user has filled out this form in the past, a cookie is sent that saves the employee number and name. Then, when the form is presented again, they do not have to fill in the employee number and name fields again.

Reading Cookies To request the form, the server runs a small CGI program instead, shown in Figure 21.

```
/* UPDATEFM CGI program example */
/* Retrieve cookie data and display the UPDATEFM HTMLPART document  */

/* WEBSHARE/EWEB don′t create variables for all headers, so         */
/* create ″extension variables″ of X_HTTP_name                      */
′callpipe *.input.1:′,            /* Get header records             */  1
      ′| xlate w1 upper - _′,     /* Make proper variable names     */
      ′| spec fs : f1 1 f2-* strip 21′,  /* Expand for JOIN         */
      ′| sort 1-20′,
      ′| join keylength 20 /, /′,        /* Combine matching hdrs  */
      ′| spec /=X_HTTP_/ 1 w1 next /=/ next 21-* next′, /* Vars    */
      ′| varload′

cookie.=′′
If symbol(′X_HTTP_COOKIE′) = ′VAR′ then
   ′callpipe var x_http_cookie′,    /* Get values from cookies      */  2
         ′| split ″;″′,
         ′| strip′,
         ′| xlate fs = f1 upper′,
         ′| xlate fs = f2-* || blank ″ blank′,
         ′| change //=COOKIE.?/′,
         ′| varload′

′callpipe < UPDATEFM HTMLPART′,                     /* Read the form   */
      ′| change ″&&EMPNUM&&″′cookie.?empnum′″′, /* Put in data     */  3
      ′| change ″&&NAME&&″′translate(cookie.?empname,′ ′,′+′)′″′,
      ′| *:′
Exit
```

Figure 21. Example of Retrieving Cookie Header Fields

Some notes about this example:

1 This pipe retrieves all of the header fields sent from the client and sets the proper variables. See 3.2.1, “Webshare Global Variables” on page 26 for more information. EnterpriseWeb/VM also places the cookie data in

GLOBALV variables that can be used instead of reading the header fields. See the product documentation for more information on how to use these variables. For VM:Webgateway, all you need to do is retrieve the header variable using the command `CGI GETVAR HTTP_COOKIE (VAR HTTP_COOKIE`.

2 If a cookie is sent, it will be in the form `name=value; name=value`. This CMS Pipelines program sets `cookie.?`*name* variables for each cookie sent. The cookie name is translated to upper case and any special pipeline delimiter characters in the cookie′s value are translated to blanks. In the example, the delimiter characters are the vertical bar (|) and the double quote (″). More information on special handling of data from Web browsers can be found in section 5.6.2, "Do Not Trust Incoming Data Validity" on page 142.

3 In our HTML form, we changed the INPUT lines for the employee number and the name to be:

```
<INPUT TYPE=″text″ NAME=″empnum″ VALUE=″&&EMPNUM&&″ SIZE=6 MAXLENGTH=8>

<INPUT TYPE=″text″ NAME=″name″ VALUE=″&&NAME&&″ SIZE=20 MAXLENGTH=20>
```

This pipe fills in the VALUE field in the form with any cookie data that was sent. If no data was sent, no initial value is specified. This is a simple example of how useful a CGI program is for creating dynamic HTML documents.

Writing Cookies To set a cookie, you send a Set-Cookie response header field. This field has several rules and optional parameters that are not described in this document. For a complete description, see: `http://home.netscape.com/newsref/std/cookie_spec.html`. The fields we are concerned with are *name*=, `path=`, and `expires=`.

name=
This is the name of the value to save, and its value. The value should not contain space, semicolons, or comma characters.

`path=`
Specifies the URL path under the current domain for which the cookie is valid. This keeps cookies that you sent from being read by other servers in other domains.

`expires=`
Defines the date at which the cookie expires and should be deleted. The Netscape standard defines the format of this time stamp as `Day, dd-Mon-yyyy hh:mm:ss GMT`. Notice that this is a slightly different format from any of the formats given in 3.4.3.1, "HTTP Time Stamp Formats" on page 45.

Figure 22 on page 50 shows the code added to the phone number update CGI program (UPDATE CGI), which is named UPDATECO CGI and UPDATECO SVMEXEC on the VM Web CD. This example shows part of UPDATECO SVMEXEC.

```
/*-----------------------------------------------------------------*
 * Send cookie data to the browser.
 * First, write the status header and then any Set-Cookie headers.
 * The cookie will expire in 14 days at midnight GMT.
 *-----------------------------------------------------------------*/
'CGI WRITE HEADER (STRING Status: 200 OK'
'CGI WRITE HEADER (STRING Content-Type: text/html'
If Success = 1 then do
   /* Set the path variable with the same path as this CGI program  */
   /* (that is, everything in script_name up to the last "/")       */
   'CGI GETVAR SCRIPT_NAME (VAR SCRIPT_NAME'
   Parse value reverse(script_name) with '/'  path
   path = reverse(path)
   expires = date('B')+14        /* This cookie expires in 14 days */
   /* Form the rest of the required parts of the cookie header      */
   /* Time stamp must be in the standard format with dashes "-"     */
   c_rest = 'path='path';',
            'expires='left(date('W',expires,'B'),3)',',
            translate(date('N',expires,'B'),'-',' ') '23:59:59 GMT;'
   'CGI WRITE HEADER (STRING',
         'Set-Cookie: EMPNUM='parm.?empnum';'  c_rest
   'CGI WRITE HEADER (STRING',
         'Set-Cookie: EMPNAME='translate(parm.?name,'+',' ')';'  c_rest
end
```

Figure 22. Example of Sending Set-Cookie Response Header Fields

Notice that the cookie expires 14 days from today. We also read the SCRIPT_NAME variable to find out the URL path to our program and set the cookie path value to match it.

3.4.6 National Language Considerations

The Internet is a world-wide network (hence the name World Wide Web) and different languages and sets of characters (symbols) are used in different places. Most computer systems use an 8-bit byte to store characters and symbols, but there are far more than 256 characters and symbols. A *code page* defines the translation of 8-bit binary values to displayable characters and symbols. The HTTP protocol uses the term *character set* instead of code page, and we will use both terms in this section.

3.4.6.1 What is a Code Page?

In 8-bit ASCII and EBCDIC, there are only 256 possible values to represent a very large number of symbols. Groups of symbols with common attributes are called *character sets*. Each symbol in the set must be assigned a unique numeric value (called a *code point*) within the 256 values that are available to create a code page. However, a character set may have more than 256 symbols, so many code pages are required to create useful groupings for different applications. For example, the set of symbols commonly used in the United States is different from the set used in Sweden. Both countries may use the same “character set” because there are common characters (Latin alphabet, numerals, and some “special characters”). But each country would use a different code page because there are differences in the symbols that are most commonly used. Sometimes the same symbol is used, but is assigned a different code point, perhaps because different languages sort the character differently in their alphabet.

Code pages and character sets are important because to correctly display a symbol on a client′s browser window, a Web server and a CGI program must know which character set is used by the browser and the character set of the data stored on the server. The character sets acceptable to the browser are listed on the `Accept-Charset` header field. If this field is sent, the Web server sets the HTTP_ACCEPT_CHARSET global variable with the field′s value. The CGI program can examine this field and perform the necessary translation to correctly display the data on the client. If the client provided multiple acceptable character sets, the CGI program must inform the browser which character set it is using in the Content-Type header field, like this example:

```
Content-Type: text/html; charset=ISO-8859-1
```

IBM systems use code page numbers to define character sets. Most character sets have an equivalent code page number, but we did not find a complete reference source for this information. Many of the ISO-8859 standard character sets are shown in Table 6.

Table 6. Common IBM Code Pages and Character Sets (ASCII Encoding)

819	ISO-8859-1	Latin 1 (or "8-bit ASCII")
912	ISO-8859-2	Latin 2
913	ISO-8859-3	Latin 3
914	ISO-8859-4	Latin 4
915	ISO-8859-5	Latin/Cyrillic
1089	ISO-8859-6	Latin/Arabic
813	ISO-8859-7	Latin/Greek
916	ISO-8859-8	Latin/Hebrew
920	ISO-8859-9	Latin 5 Turkey and western Europe
919	ISO-8859-10	Latin 6 (Baltic/Scandinavian)
874	ISO-8859-11	Latin/Thai
923	ISO-8859-15	Latin 9 (Similar to Latin 1 but includes the Euro symbol)

3.4.6.2 Translations

When you enter data from a 3270-equivalent terminal, the code page you are using depends on how your terminal session is customized. Unfortunately, CMS does not retain the code page used to create a file in the file directory, so the code page number associated with each file is unknown. This is not a problem unless the files need to be correctly displayed on devices that use a different code page or need to be translated to a different encoding method. Then, a correct translation requires that both the source and target code pages are known to create a correct translation table.

We could not find a standard source for these translation tables. TCP/IP for VM includes several translation tables for various languages as TCPXLATE files. The CONVXLAT utility supplied with TCP/IP converts these files to a binary form that FTP and VM:Webgateway can use. The name of the translation table is specified as the TRANSLATE option on the CGI command in VM:Webgateway. It is also used in the TRANSLATE option of the CONFIG FILETYPE command to associate a file type to a set of characteristics.

For EnterpriseWeb/VM, use the *transport filter* specification in your media map file to associate a static file type to a translation table. CMS Pipelines also provides support for code page translations with the xlate stage. Consult the online pipelines help information (via PIPE HELP XLATE) for information on what conversions are supported on your level of CMS Pipelines.

3.4.6.3 Using Languages

Some Web browsers allow the user to configure a language preference of responses from a server. This preference is sent to the Web server in the Accept-Language header field. Language codes in this field are listed in order of preference. For example, a header of `Accept-Language: fr, en` says that the client prefers a response in French but will also accept English. The actual language of the object sent is given in the Content-Language response header field.

For example, if you have HTML “prologue” files available in English, French, and Danish, you can display the correct one with a routine like this:

```
...
prologue.  ='ENGLISH'            /* This is the default            */
prologue.EN='ENGLISH'            /* Define some other languages    */
prologue.FR='FRENCH'
prologue.DA='DANISH'
languages = 'en fr da'           /* List of ones we have           */
/* Earlier we fetched the CGI variable HTTP_ACCEPT_LANGUAGE...      */
response_lang = HTTP_ACCEPT_LANGUAGE||', en'  /* English if unknown */
Do While response_lang <> ''
   Parse Var response_lang lang ',' response_lang
   Parse Var lang lang '-'       /* Remove any additional data     */
   lang = Strip(lang)
   If Wordpos(lang,languages) > 0 Then Leave /* Do we have this one?*/
End /* While response_lang <> '' */
/* Webshare example of output */
'output HTTP/1.0 200 OK'
'output Content-Type: text/html'
'output Content-Language:' lang  /* Tell which language we used    */
'output'
...  optionally handle the character set issues here
'callpipe <' prologue.lang 'HTMLPART | *:' /* Output the correct one*/
...  rest of our program
```

3.4.7 Serving HTML Directives as Data

By now you have undoubtedly noticed that HTML has several reserved characters which have special meaning. As such, these characters can not be directly displayed by including them in a data stream. For instance, if a < character is sent to the browser in an HTML data stream, then the next character will be interpreted as the start of an HTML tag name. Neither it nor the < will be rendered as data characters. In fact, no data until the > is likely to be rendered as literal data, all of it being interpreted as part of the HTML tag.

In order to display a <, >, & or ″ you must provide the browser with a special order, a symbolic name for the character to display, rather than these raw characters. In particular, you must make the following changes:

< into &lt;
> into &gt;
& into &amp;

″ into &quot;

While there are many other characters that you can display by using a symbolic name for them, the above must be displayed in this manner. Refer to an HTML reference source (such as the RFC) for a full list of these symbolic names.

The VM:Webgateway CGI Extension routine VIGRTNS HTMLSAFE (described in 4.2.4, “Programmer Productivity Tools” on page 75) provides a tool to accomplish this transformation on a data string containing these characters. We could not find any documented EnterpriseWeb/VM facilities to address this need, but the problem is quite solvable using standard CMS Pipelines coding techniques.

3.4.8 Server Push

A Web server can send more than one object to the client in one transaction using a *server push* technique. It is more complex than sending a single object because response header fields have to be sent between each object. Also, it is not supported by all Web browsers. It is useful if your CGI program is presenting a series of documents or images to the user. The Web server keeps the connection to the client open and sends the next document after a period of time that is immediately displayed. In order for this to work, the data sent to the Web server from your CGI program has to be sent immediately to the client without first buffering it in the server. Consider the simple CGI program in Figure 23 that does server push:

```
/* SRVPUSH SVMEXEC (environment SVMEXEC) */
'CGI WRITE HEADER (STRING Status: 200 OK'
'CGI WRITE HEADER (STRING',
  'Content-type: multipart/x-mixed-replace;boundary=ECURB'

cmd_doc = 'CGI WRITE DOCUMENT (TRANSLATE USENGLISH CRLF STRING'

cmd_doc '--ECURB'
cmd_doc 'Content-type: text/plain'
cmd_doc ''
cmd_doc 'Found 1 peanut'
cmd_doc '--ECURB'
'CP SLEEP 3 SEC'
cmd_doc 'Content-type: text/html'
cmd_doc ''
cmd_doc '<p>Found 2 peanuts</p>'
cmd_doc '--ECURB'
'CP SLEEP 3 SEC'
cmd_doc 'Content-type: text/html'
cmd_doc ''
cmd_doc '<h1>Found 3 peanuts</h1>'
cmd_doc '--ECURB--'
exit
```

Figure 23. Example of a Server Push CGI Program

Notice that there is only one space in the `Content-type` line, and that there must be a null line between the `Content-type` line and each part of the document. Also notice that each boundary must be preceded by exactly two hyphens, and the last boundary is followed by two hyphens. The string used as the boundary string is arbitrary. When this CGI program is run, it displays the message `Found`

1 peanut for three seconds before the next message is displayed. It ends and closes the connection when the last boundary string is written.

Note: This example uses the CP SLEEP command to simply illustrate some process, which takes time. Running this example in a VM:Webgateway SVM would cause the entire server to wait. If your CGI program has to actually wait on an external event or a period of time, use standard CMS multitasking facilities to perform the wait. Then, other threads of execution in the SVM could proceed as this one waits. If commands such as CP SLEEP are to be used, we recommend that all such processing actually be performed in a VM:Webgateway worker environment, such as WORKEREXEC. For further discussion of related issues refer to 6.3.6, "Serializable Server Resource Access" on page 170, 5.6.4, "Denial of Service" on page 143 and 5.7.2.4, "Monopolization of Web Server SVM Resources" on page 150. For further information on CMS's multitasking facilities, refer to *Exploiting Recent CMS Function: A User's Guide to CMS Application Multitasking*, SG24-5164.

3.4.8.1 Using Server Push in Applications

Server push can be useful in Web applications that are long running or require server access to remote resources. For example, queries that must reference multiple databases or systems to create responses will take much longer than a simple access of local data. Users would not see any response from the server until the entire transaction completes. They may think the server is not working and press "stop" on their browser or request another document. For these kinds of applications, using server push to immediately return a response (saying "Please wait while I perform the search") and then returning the results many seconds later lets the users know that the server is working on their request. But keep in mind that not all Web browsers support receiving multiple objects in the same transaction.

We have included on the VM Web CD a simple example of a long-running transaction. TIMEQRY WRKEXEC is a VM:Webgateway CGI program that asks for a node name as a search input string. When the string is received (as the query string), the program uses RSCS to send a command to a remote system and waits for the response. While the remote system is responding, a "Please Wait" screen is displayed. When the remote system responds, the response is shown to the user. This program must be run in a VM:Webgateway worker machine. It will not work on the server because the CGI program uses IUCV and the server must have exclusive use of IUCV.

3.5 Filter Programming

Filter is a term normally associated with CMS Pipelines programming (or even real plumbing). Just as in real plumbing, the input data "stream" is changed in some way by the filter and then sent to the output data "stream." Filter implementations in VM Web servers follow the same model. They are designed to let users with a Web browser request unformatted data that is then "filtered" or transformed into viewable data for display.

Do not confuse filter programs with CGI programs that run in the CMS Pipelines programming environment. Both types of programs use pipelines for all input and output and have pipelines as the default command environment. Both also transmit any data written to the primary output stream to the client for display. The important difference is the source of data for the primary input stream. A

CGI program is provided data from the client (usually as a result of an HTML form) as its primary input stream. A filter program is provided the object identified by the URL path as its input. For example, a URL of `http://wtscvmt.itso.ibm.com/sg245347.script` provides the SG245347 SCRIPT file as input to the filter.

How does the server know that a file should be filtered? VM Web servers use the file type of a file specified in a URL to determine how to process a request. To use filters, you must configure one or more file types that identify a filter that is used to return data to the client. Then, when a file with that file type is referenced in a URL, the filter is called with that file available to the input stream of the filter and any output is sent to the client.

3.5.1 A Filter Example

Our PHONE1 CGI program, which we saw in Figure 7 on page 23, can be changed into a filter program. All this CGI program does is “transform” the PHONE DATA into a form that is viewable on a Web browser. If we change our URL that references `http://wtscvmt.itso.ibm.com/~ vmwebcd/learn/cgi/phone1` to instead reference `http://wtscvmt.itso.ibm.com/~ vmwebcd/data/phone.data`, we can “filter” the file instead. For the VM:Webgateway Web server, we configured it to call the PHONE1 filter for DATA type files with this command:

```
SMSG VMWEBSRV CONFIG FILETYPE ADD DATA FILE TRANSLATE USENGLISH
     CONTENT-TYPE text/html SSI NO FILTER WORKERPIPE /PHONE1/
```

Notice that this command specifies that objects with a file type of DATA are FILES (instead of CGI programs), what translation and content type to use and the name of the filter. The name of the filter, PHONE1, is enclosed within forward slash (“/”) characters because the syntax of the command requires a delimiter character before and after the filter name.

Figure 24 on page 56 shows the sample filter program PHONE1 REXX. This filter must be placed on a disk or directory that is part of the server′s CMS search order. Notice that the name of the data file does not appear anywhere in this example. This is because the Web server has already located the file and provided it as the pipeline′s primary input stream.

```
/*  PHONE1 REXX filter program example */
/* This filter must be accessible by the server                     */

'callpipe < PROLOG1 HTMLPART | *:'/* Display the beginning HTML     */

'output <TABLE BORDER="1">'       /* Set up the HTML table          */
'output <TH>Employee Number</TH>' /* Put a title on each column     */
'output <TH>Name</TH>'
'output <TH>Phone Number</TH>'
'output <TH>Location</TH>'
'output <TH>Office</TH>'
'output <TH>Status</TH>'

/* Get the telephone data from the file and display */
'callpipe *:',
       '| nlocate 1-1 "*"',
       '| spec "<TR>" next',
             '"<TD>" next  1.6  next "</TD>" next',
             '"<TD>" next  8.20 next "</TD>" next',
             '"<TD>" next 29.12 next "</TD>" next',
             '"<TD>" next 42.3  next "</TD>" next',
             '"<TD>" next 46.9  next "</TD>" next',
             '"<TD>" next 56.5  next "</TD>" next',
         '"</TR>" next',
       '| *:'

'output </TABLE>'

'callpipe < FOOTER HTMLPART | *:' /* The ending HTML                */
Exit RC
```

Figure 24. Example Filter Program PHONE1 REXX

Filter programs in VM:Webgateway can utilize additional CMS Pipelines input and output streams defined by the server. Input stream 1 (the secondary input stream) contains the proposed HTTP header fields that the server will send to the client. The filter program can decide to modify, delete, or send additional header fields to the client. It does this by writing them to output stream 1. Input stream 2 contains filter variables. These have the same name as CGI variables (see Table 4 on page 14 for a list of the standard ones) but the values are based on the file being filtered. Output stream 2 accepts server log records and output stream 3 accepts any messages to be written on the server's console. More information on VM:Webgateway filters is found in the product's documentation.

We have also provided, on the VM Web CD, a sample filter named TEST REXX. This filter creates an HTML document that displays the contents of all the filter input streams. It should be useful in developing and debugging filter programs that use data from the additional input streams.

3.5.2 Processing SCRIPT Files

The Document Composition Facility (more commonly known as SCRIPT) and BookMaster are very commonly used on S/390 systems to create documents, manuals, memos, books, and so forth. Many installations have large numbers of BookMaster documents on their VM systems. Since BookMaster is a markup language like HTML, transforming BookMaster documents to HTML documents is desirable because it makes the documents available to a broader audience.

A tool has been written by Gary Richtmeyer of IBM Global Services named B2H EXEC (BookMaster to HTML converter) that does this conversion. It is available from the VM download library (http://www.ibm.com/s390/vm/download) and is also available on the VM Web CD.

Note: You may want to check if a newer level is available from the download library than the level available on the VM Web CD.

B2H is ideal for filtering SCRIPT files for display on a Web browser. B2H EXEC can be called as a CMS Pipelines stage, so a complete B2H filter can be written as:

```
/* B2HQUICK REXX */
'callpipe *.input.0: | rexx (B2H EXEC) | *.output.0:'
```

From this very simple example, we can add additional features that utilize B2H options, filter variables, and HTTP headers. On the VM Web CD,[5] you will find a complete program named B2HFILTR REXX. This program demonstrates how to serve a pre-translated HTML file instead of the SCRIPT file (if one is found), how to output a Last-Modified header field and how to examine the incoming If-Modified-Since header field. It also supports multiple option files for B2H so that the same filter can be used on multiple types of input files.

3.6 Debugging Your CGI Programs

We all try to write perfect programs, but we usually have a few bugs to find and eliminate. REXX has excellent tracing capabilities, but debugging a CGI program that is running on a Web server is a bit different from debugging on your own user ID. Ideally, you would turn tracing on in your program, log onto the Web server console and watch your program run. But you do not want to interrupt your production Web server with your debugging work. Also, you may have several Web server machines running in parallel - which one do you log on to?

Clearly the ideal solution is one or more additional Web server IDs dedicated to development and debugging. We recommend that CGI developers be given their own dedicated single test Web server to use in program development. By only using one SVM user ID and dedicating that test environment to one CGI developer, it becomes easy for a developer to log onto these user IDs and use normal CMS Pipelines and REXX development techniques to develop programs. Customize these servers to connect to a different TCP/IP port (other than port 80 or any other well known port.) Then, specify the URL of the CGI program but with the port number of your debug server. For instance, if you start a Web server on port 8080, specify the URL as http://wtscvmt.itso.ibm.com:8080/~vmwebcd/cgi/program.cgi. As long as you are careful to either not hard code absolute URLs in your HTML and CGIs, or to parameterize the host name and port portion, you will find it easy to develop applications on a different port or even a different VM system from the one that will be used in production. If you are running VM:Webgateway and use worker machines, your test Web server also needs to have its own (single) worker machine.

If your Web server program allows it, you can simply run the Web server code in your own virtual machine and execute and debug your CGI programs. You will

5 See B.1.7.1, "BookMaster to HTML Conversion Tools" on page 232 for more information.

still need to configure it to use an unallocated TCP/IP port and have access to the Web server code. We did not use this method to debug CGI programs, but it should not be difficult to do with Webshare or EnterpriseWeb/VM. VM:Webgateway requires additional resources that may make this method of debugging more difficult to set up.

To test your CGI programs:

- Use POSTTEST CGI or TEST CGI (included on the VM Web CD) to debug your HTML forms before you begin debugging your CGI program. See 3.3.3.2, "POSTTEST CGI Utility" on page 37 for more information.
- Use a REXX compiler program to debug the syntax of your program before running it on the Web server. For instance, to use IBM′s REXX compiler for this purpose, enter `REXXC PROGRAM CGI A (NOCOMPILE SOURCE SLINE TRACE`. (See 6.3.3.1, "Use a REXX Compiler" on page 163 for additional motivations for the use of a REXX compiler for program development.)
- Run your program using several Web browsers and check its behavior on each. If you also intend to support VM/CMS users (for example, the Charlotte browser), be sure to test using that browser also.
- For repeatable regression testing, consider creating one or more HTML documents that contain many hypertext link anchors referencing your CGI program with various query string values that exercise different paths through your program. Or create a series of forms with hidden input fields for each variable and only display a series of "submit" buttons to test each one. You may also be able to use a Web crawler type application in conjunction with these pages to automatically regression test your application.

And when you are looking for a bug:

- Use the "View HTML source" function of your browser to see what is really being sent to the client.
- Insert `Call diag 8,′MSG userid′ var` statements into your CGI program to see intermediate values as the program executes.
- Write intermediate output files (usually within CMS Pipelines programs) as the program executes.

Other very handy utilities are TCPSNIFF EXEC and WEBSNIFF EXEC, included on the VM Web CD. TCPSNIFF EXEC was written by John Hartmann, the author of CMS Pipelines, as a demonstration of how easy it is to write a TCP/IP server as a pipeline. WEBSNIFF EXEC is an improved version of TCPSNIFF that is better suited for debugging Web transactions. It defaults to ASCII to EBCDIC translations for its log file and also deblocks the lines into records at the CRLF breaks. The log file shows all data sent between the client and server and is especially useful when you need to look at headers and responses.

To use WEBSNIFF to trace the transactions between a browser and a Web server running on `wtscvmt.itso.ibm.com` port 82, enter:

```
WEBSNIFF 7654 wtscvmt.itso.ibm.com 82
```

The number 7654 is the port that your user ID has connected to TCP/IP, and any connections on this port will automatically connect to the domain and port specified. The URL that is specified to the Web browser is `http://wtscvmt.itso.ibm.com:7654/`*path*. (Assuming you are logged on to `wtscvmt.itso.ibm.com`.) To stop WEBSNIFF, enter STOP on the console. Then examine the WEB TRACE file created on your A disk. Lines from the client (the Web browser) are identified with a `C` in the first column and lines from the server are identified with an `S`.

Chapter 4. Web Access to Applications and Business Data

Web-enabling applications sounds complicated, but it does not have to be. There are some basic rules that help keep it simple. This chapter describes these rules and offers some examples.

This chapter describes:

- Some Web application design concepts
- Alternatives that you will be faced with
- What to consider before porting applications to the Web
- How to access VM and non-VM resources
- Sample procedures that show you how to use VM product resources

4.1 Useful Concepts

Client/server migrations in the past almost always required you to develop brand new applications. Now, Web enabling gives you many more alternatives in porting applications and application interfaces to the Web. However, many of the same concepts apply. There is still a client requesting services and a server expected to provide these services. The client is the Web browser and the server is the Web server. You should think about how your application fits into the following three layers:

- The *presentation layer*, which is the logic of the application that interacts with the user. It includes functions for managing the interaction between the user and the application, such as screen formatting, and keyboard and mouse handling. In the Web environment, these functions are dedicated to browser, JavaScript code to improve browser abilities, or Java applets.
- The *function* or *business layer*, which includes such activities as performing calculations, data enrichment and any rules defined by the user that determine what particular processing should take place. It is not concerned with end-user interactions or any database activity. CGIs fit into this category, but Java applets may also be involved.
- The *data management layer*, which consists of both the code that requests a database change, such as SQL, and the database processing itself. It also includes CGIs performing data access and Web "enablers" (also known as connectors).

Web design is less flexible than client/server architecture because browser capabilities do not permit any data management processing. As a consequence, Web applications fit into three styles:

- Distributed Presentation

  Some portions of the user interface are placed on the user node and others on the server node. For example, "screen scraping" tools transform 3270 host presentations to HTML document directives: At the same time, two different representations exist for the same application screen.

- Remote Presentation

  All presentation tasks are managed by the user node.

- Distributed Function

Some part of the application logic resides on the user side and must also communicate with server side code. Applets may work in this style, processing user input data before sending it back to the server where a CGI will accomplish what can only be done on the Web server host.

4.1.1 Designing Applications for the Web

The following sections discuss considerations to keep in mind when designing Web applications.

4.1.1.1 Application Interface

Designing Web applications will make you change your thinking about how applications are presented to the user. You should keep in mind that 3270 "green" screens have a limited amount of space available to display data compared to most Web browser windows. The HTML documents displayed on the Web browser window may also contain "mouse actions," such as push buttons, selection boxes, and so forth. As a result, you will need fewer application screens to navigate through the entire application. Do not try to transplant 3270 screen hierarchies onto the Web browser. The need to make the user click and click again through several screens can easily be avoided on Web pages.

To shorten the navigation through your Web application, you can use:

- Long HTML pages with HTML anchors
- HTML frames

  They permit you to manage several independent logical windows on the same physical browser window. One frame can be used as a menu that can control the other frame contents. This feature is not supported by all Web browsers, and some users feel that frames are awkward to use and would prefer not to interact with them. (Also refer to the information in 5.5.2, "Can You Trust the Displayed Information in a Frame" on page 138.) If you do make use of them, consider presenting a path through your application that does not use them (as found on many existing Internet pages).
- A variety of input methods that are available in HTML forms

  Besides the keyboard, a user can use a mouse to select radio and push buttons, select check boxes, and select items from a list.
- JavaScript procedures that allow you to perform some checking on the input data before the data is sent to the server

  User interaction with the browser for simple field checking is much faster than sending the data to the server for verification. However, you may find that many browsers will have disabled JavaScript due to the considerations outlined in 5.5.1, "JavaScript" on page 137. In addition, if you do make use of JavaScript, remember that there is no formal definition of the language, and that each browser may implement a different and incompatible variation of the language. You must be careful to make sure that you use just the subset of all variations of all browsers that your user community′s browsers implement, which is a non-trivial problem that leads to many broken pages in the Internet.
- Java applets, which can be very powerful

  They allow you to provide in one piece of code all the interface design and the validations of the input data. If you intend to use a lot of Java applets as front end interfaces, you may want to use Java development tools such as VisualAge for Java. Moreover, there are plenty of Java applet sources on the Internet that help in designing useful interfaces.

You must also think about your potential users′ habits and environments:

- What browsers do they run (vendor, version)?
- What plug-ins do they already have or can obtain trivially?
- Do they support JavaScript?
- Do they support Java?
- What size (resolution) is their screen?
- Does it support graphics and color?
- Are they connected with high-speed LANs or low-speed dial connections?
- Do they run high-speed processors or old PCs?

A valuable rule is to never expect an HTML document to appear the same way it does on your own browser.

When you create a CGI program, the CGI environment variable HTTP_USER_AGENT contains the identification of the browser that referenced the CGI. You can use it to check for browser types and levels that are unsupported by your application and then warn the users about potential problems. A good tip is also to create a help HTML document on your server that indicates the optimal browser configuration to use for your site. Even better is to make sure that your HTML pages will run on all the popular browsers (vendor, version, helpers) used by your entire user community.

4.1.1.2 Application Logic

There are new things you should keep in mind while developing for the Web. A Web server does not just act like a new network interface. All data goes through the Web server, and a CGI program may or may not run in its own private virtual machine. Your Web site may also be a group of Web servers sharing system resources. You should do the following:

- Only run well tested CGI programs in production. If the CGI program abends it can also stop the entire Web server (even if a simple REXX error is not a problem), because a CGI program may run on the Web server machine (instead of a separate worker machine),

- Make sure the server is always available for incoming HTTP requests. Long-running CGI programs can keep the Web server machine busy so that it cannot process incoming requests. Refer to 5.6.4, “Denial of Service” on page 143 for additional considerations.

- Be aware of VM resource sharing. In a regular VM system, resources are typically shared between VM user IDs. With a Web server, you may have Web applications that may run in one or several VM Web servers or VM:Webgateway workers:

  – If your Web application runs in a multitasking server virtual machine, look for possible conflicts between CGIs running in different threads at the same time that may try to use the same resources. You should also check if the code used in the CGI program really supports a multithreading environment. Some programs appear to run OK in a thread, but they keep the thread active until it ends. This effectively disables the CPU sharing mechanism of multitasking applications.

  - If your Web application runs in several server virtual machines, make sure that the rules for sharing resources between user IDs on a VM system are followed. See also 5.6.6, "Reentrant and Serially Reusable Resources and CGIs" on page 144 for more information.

  Refer to 6.3.6, "Serializable Server Resource Access" on page 170 and 5.7.2.4, "Monopolization of Web Server SVM Resources" on page 150 for additional considerations.

- Address new security considerations as they develop. See 5.10, "References" on page 154 for additional sources of information.

4.1.2 Moving an Existing Application to the Web

There are several things you should remember when considering moving an existing application to the Web:

- Existing application interface design is based on 3270 limitations:
  - Screen size
  - Limited character formats (fonts, colors)
  - No "point and click" interfaces
  - No graphics
- Existing applications run in separate CMS user IDs:
  - Authentication was checked by VM/ESA when the user logged on and does not need to be verified again.
  - Program execution environment is persistent and long running.
  - Usually only one application runs at a time.
- Existing application screen navigation may also be influenced by 3270 limitations.
- Programming styles differ. For example, many full screen VM applications give the users some kind of "command line" so that they can enter any command while the application is running.
- Operating system architecture also impacts migration. Normally, a VM user must use his own virtual machine to run an application. A Web user running an application may have to share Web server services with other Web users.
- The entire network environment may be different. The Web flow uses the IP network. Response time may be different from SNA terminals or local screens. You may need to consider retuning your network.

4.1.3 Applications and Command Lines with Line Mode Output

Applications that can display "line mode" output in response to some input are probably the easiest applications to move. Included in this set are applications that have programming APIs that can be used in CGI programs to create a Web version of the application.

- Simple commands

  Simple commands are applications that accept all their input from the command line and output their results before exiting. In other words, they behave like simple CMS commands, such as LISTFILE. These are pretty easy to invoke using a CGI program. All the CGI program needs to do is:

  - Pass the command and parameters to VM/ESA.
  - Receive the command output in some buffer.
  - Format this output into an HTML document.

  - Send the response back to the server, which will send it to the browser.

- Multiple response applications

  These are commands or applications that interact with the user. For instance, they may prompt the user for more information before performing their work. They can be somewhat more difficult to invoke from a CGI program, but this is usually not impossible. Here are some approaches you can take:

  - Modify the application in such a way that it delivers all the output from one single call. You may need to add additional command parameters that used to be typed at program prompts.
  - Use a "screen scraping" tool as described in section 4.2, "Screen Scripting with VM:Webgateway CGI Extension" on page 66.

Applications that need the user′s virtual machine environment may also be easier to migrate to the Web using screen scraping tools. They usually run applications in the original user′s environment.

4.1.4 3270 Full Screen

Since it is not easy to capture and manipulate a 3270 screen in a program, Web enablement of 3270 full screen applications requires more work to migrate the applications. Fortunately, some tools exist that can help you move these more complex applications to the Web more easily and more quickly.

4.1.4.1 Writing a New Web-Oriented Interface

If the application you intend to port into the Web is too rooted in 3270 design and characteristics, it could be easier to develop a completely new application that will just access the same data and perform the same logic.

4.1.4.2 Modify 3270 Applications to Help Web Enablement

Writing a brand new application may not be worth the effort, though. You would rather reuse as much of the existing application as you can. If the way your old application accessed its data is not too "session oriented," you could do some restructuring of the application to make it fit the Web programming model better. Think about splitting the 3270 application into two or more of the following modules:

- Modules extracting the data
- Modules dealing with the application logic
- Modules responsible for displaying the data

Depending on the complexity of your application, you will be able to share modules between the 3270 and Web versions of the application.

4.1.4.3 Use Web Terminals

TN3270 (Telnet 3270) emulations within Web browsers are delivered by several companies. The advantage of these implementations is that there is nothing more that must be installed to put your 3270 application "within" a Web browser. However:

- The 3270 operations may be confusing for a Web user.
- It does not take advantage of Web interface languages.
- There is no need of any Internet browser to get TN3270 emulation.
- What is the added value?

4.1.4.4 Use a Screen Scraping Tool

Instead of displaying the 3270 screen directly on the browser, it is better to use screen scraping tools that act as a gateway between the HTML interface and the 3270 screen.

The VM:Webgateway CGI Extension function, which is a standard part of the VM:Webgateway server, can do this. We will show how this product interacts with a 3270 application and helps porting 3720 screens to the Web in 4.2, "Screen Scripting with VM:Webgateway CGI Extension."

4.2 Screen Scripting with VM:Webgateway CGI Extension

Most business applications use a 3270 full screen interface or need a specific user environment. Porting them to the Web without any help often means writing a new application and coping with new issues, such as:

- User authentication has to be adapted to Web applications.
- Data may be accessed from a different place than the user′s virtual machine.
- Internal application logic and checking must be reconsidered.

For these and other reasons, 3270 screen scraping tools provide quick and easy ways to enable applications for Web users. These products manage the CGI program interface to the 3270 application screens. To *screen scrape* an application means to:

1. Interpret full screen prompts
2. Emulate user keystrokes
3. Interpret, reformat, combine multiple screen sources, and send back to the Web server the application data from the 3270 screens

The application that runs is the same you have used previously:

- User authentication is still verified by the same VM security routines, including both the VM directory and any other existing VM-based authentication tests.
- User authorizations are still based upon VM security facilities, such as RACF, SFS access control lists, and the VM directory.
- Data validations are still made by the old application.
- Data integrity is checked by the same code.
- The application environment is unchanged: it is the user′s virtual machine.

The only concern is to reformat the existing input and output from 3270 screen styles to the Web′s GUI style. This interface may be just a slightly new design of a 3270 application, or you can enhance the Web interface to completely change the navigation and add new functions that did not exist in the 3270 application.

4.2.1 VM:Webgateway CGI Extension Description

VM:Webgateway CGI Extension[6] is the VM:Webgateway component that extends the CGI programs that run on this server to facilitate Web enhancement of existing VM applications.

VM:Webgateway CGI Extension is a set of commands and utilities called from VM:Webgateway CGI programs. They help CGI programmers with the following:

[6] The former product VM:Webserver Gateway.

- Keeping a 3270 session active between multiple HTTP exchanges
- Acting as a session manager for the 3270 session
- Extracting data from a 3270 session

VM:Webgateway CGI Extension can run two modes of operation:

- *CMS line mode* is when the target application has a line mode interface but needs to run on a particular user′s virtual machine, or when the line mode application requires several responses from the user.
- *3270 full screen mode* is when the target application has a full-screen interface or when the application is not located on the local VM system. This mode of operation requires VM:Operator to be installed and running the 3270 sessions. With this mode, you can Web-enable all 3270 full screen-based applications, including CMS, VSE, OS/390, and CICS applications.

In both of these modes, CGI programs must run on the server virtual machine in the SVMEXEC environment; they cannot run on worker machines.

VM:Webgateway CGI Extension is described in *VM:Webgateway Tutorial*, with an example of a Web-enhanced full screen application.

You can also find a presentation on the Sterling Software, Inc. Web site showing how VM:Webgateway compares to NT and UNIX Webenablers. To view it, you need to use the Microsoft PowerPoint Animation Player that you can download from the following URL:

```
http://officeupdate.microsoft.com/index.htm#PowerPointdownloads
```

The VM:Webgateway CGI Extension presentation is at:

```
http://www.vm.sterling.com/multimedia/gateway.htm
```

4.2.2 VM:Webgateway CGI Extension Line Mode Support

When a line mode application depends on the VM user ID environment, regular CGI execution in a Web server or worker machine cannot totally emulate this environment. Therefore, it is safer to run the application in the original user ID virtual machine. The VIG USER commands are routines that help in creating and controlling such an execution.

The VM:Webgateway CGI Extension can also be used to run line mode procedures requiring multiple inputs from the user.

4.2.2.1 VIG USER Commands

This virtual machine operation interface is the set of these VIG USER commands:

VIG USER ABEND	Terminates the command that is currently running on the user′s virtual machine and reinitializes the user ID.
VIG USER CONNECT	Connects to a user ID and creates an IUCV connection between VM:Webgateway and the user ID.
VIG USER ENTER	Invokes application programs on a user ID or responds to console prompts. The application response can be retrieved in REXX stem VIG_USER_OUTPUT.
VIG USER LEVEL	Returns a return code that indicates the functional level of VIGUSER MODULE.

VIG USER RELEASE	Releases exclusive control of a user's virtual machine.
VIG USER RESERVE	Obtains exclusive control of a user's virtual machine.
VIG USER WAITINIT	Makes the CGI program wait until the user ID has finished logging on.

The Web interface communicates with the VIGUSER MODULE running in the user virtual machine thru an IUCV connection. If the user virtual machine is to be autologged, the initialization of the VIGUSER MODULE has to be done in PROFILE EXEC or in SYSPROF EXEC. This module simulates keystrokes and traps command outputs in order to send them to the VM:Webgateway CGI Extension interface.

To initialize this module in REXX, call it as follows:

```
VIGUSER INIT VMWEBSRV
```

where VMWEBSRV is the name of your VM:Webgateway server virtual machine. If your user ID is to be used from more than one server, you may list multiple server user IDs on this command line. See the VM:Webgateway CGI Extension online documentation for more information. Figure 25 shows the full implementation of the line mode interface.

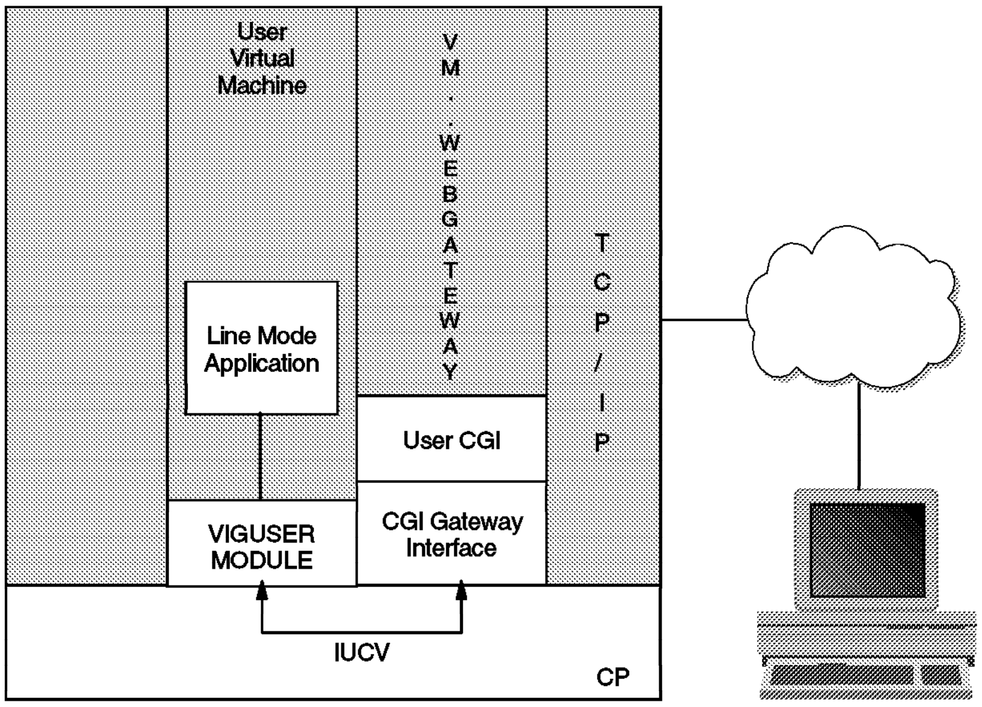


Figure 25. VM:Webgateway CGI Extension Line Mode Implementation

4.2.2.2 Example of Line Mode Scripting

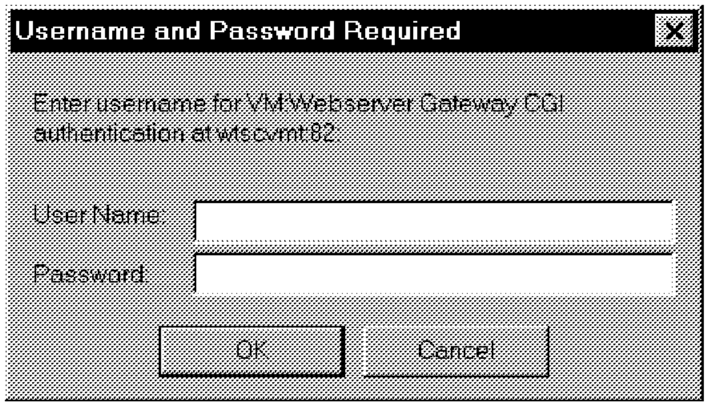
The sample presented in Figure 29 on page 71 is a Web interface to line mode commands that the Web user will type in a form entry field.

When a user submits a command using the HTML form, we have asked VM:Webgateway to check that they entered their VM user ID and password previously in order to know in which VM virtual machine the user has to log on. So, you may want to ensure that the LMCMD VMGW procedure is protected by VM:Webgateway security mechanisms. We accomplish this using a VMWEBSRV DIRMAP file placed in the same directory as the CGI and containing:

```
* Start our basic security tests for serving URLs in this directory.
SELECT
   WHEN FILE * VMGW
      * We have a VM:Webgateway CGI Extension based CGI filetype.
      SELECT
         * Base our client authentication upon the VM directory.
         REALM VM:Webserver Gateway CGI authentication
         PASSWORD VMDIR
         * Authenticate the client's knowledge of a VM user ID
         WHEN USER *
            * They got it right, allow it.
            ALLOW
         OTHERWISE
            * They got it wrong, disallow it, no overrides!
            DENY
            SKIP ALL
      END SELECT
END SELECT
```

Note: See the discussion in 6.3.3.2, "Reduce CPU Cost of URL Resolution and Basic Security" on page 164 for performance hints on constructing DIRMAPs.

This file insures that access of LMCMD VMGW will set off a popup window asking for user ID and password to be checked with VM directory. At CGI call it will pop up a window onto the browser screen resembling the one shown in Figure 26.


Username and Password Required

Enter username for VM:Webserver Gateway CGI authentication at wtscvmt:82:

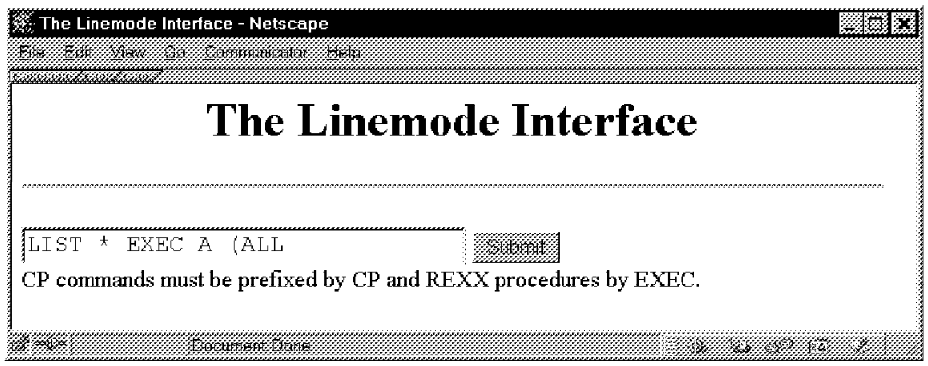
User Name:

Password:

Figure 26. VM:Webgateway CGI Extension Authentication

Moreover, the user ID virtual machine should have initialized the interfacing module, VIGUSER, as described in section 4.2.2.1, "VIG USER Commands" on page 67.

After the authentication process, the user is presented with the first page (Figure 27).


The Linemode Interface - Netscape
File Edit View Go Communicator Help
Submit
Document: Done

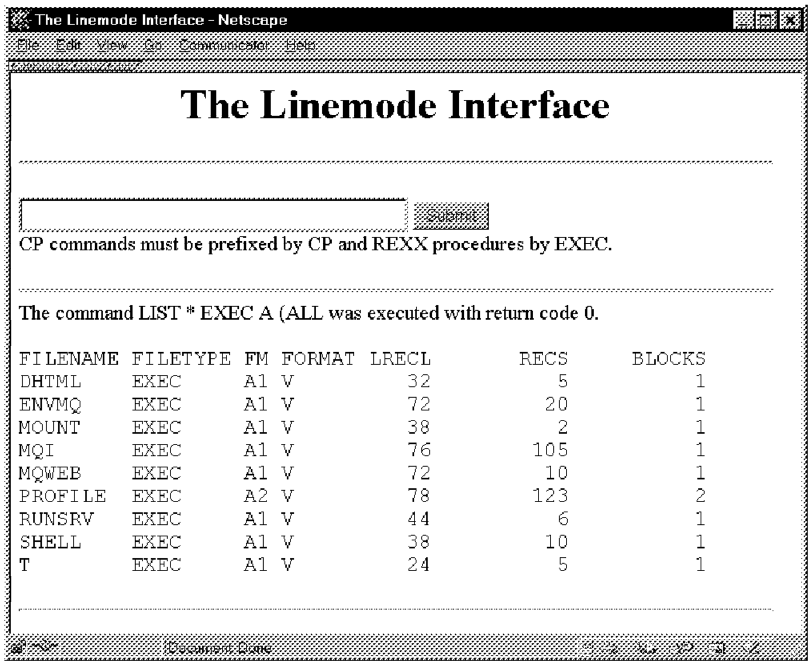
The Linemode Interface

```
LIST * EXEC A (ALL
```

CP commands must be prefixed by CP and REXX procedures by EXEC.

Figure 27. Sample Gateway CGI First Page

When the user enters a command on the form shown in Figure 27, the CGI program executes the command in the user virtual machine and sends back the response to the browser, as shown in Figure 28.


The Linemode Interface - Netscape
File Edit View Go Communicator Help
Submit
Document: Done

The Linemode Interface

CP commands must be prefixed by CP and REXX procedures by EXEC.

The command LIST * EXEC A (ALL was executed with return code 0.

```
FILENAME FILETYPE FM FORMAT LRECL       RECS     BLOCKS
DHTML    EXEC     A1 V          32         5          1
ENVMQ    EXEC     A1 V          72        20          1
MOUNT    EXEC     A1 V          38         2          1
MQI      EXEC     A1 V          76       105          1
MQWEB    EXEC     A1 V          72        10          1
PROFILE  EXEC     A2 V          78       123          2
RUNSRV   EXEC     A1 V          44         6          1
SHELL    EXEC     A1 V          38        10          1
T        EXEC     A1 V          24         5          1
```

Figure 28. Sample Gateway CGI Response

You can find the CGI code in Figure 29 on page 71.

```
/* LMCMD VMGW : line mode connection to a user ID                    */

       /*  FORM method is POST                                        */
'CGI READ 1 (TRANSLATE USENGLISH VAR ARGSTRING'
'CGI URLDECODE (VAR ARGSTRING INTO PARMS.'

       /* X_SCRIPT_NAME_TRANSLATED gives the place from where CGI was */
       /* loaded to find HTML file                                    */
'CGI GETVAR X_SCRIPT_NAME_TRANSLATED (VAR WHEREAMI'  1
MYDIR=word(whereami,4)

'CGI GETVAR X_SERVER_SCHEME ( VAR X_SERVER_SCHEME'   2
'CGI GETVAR SERVER_NAME     ( VAR SERVER_NAME'
'CGI GETVAR SERVER_PORT     ( VAR SERVER_PORT'
'CGI GETVAR SCRIPT_NAME     ( VAR SCRIPT_NAME'
MYURL=X_SERVER_SCHEME'://'SERVER_NAME':'SERVER_PORT||SCRIPT_NAME

'CGI WRITE HEADER (STRING Status: 200 OK'
'CGI WRITE HEADER (STRING Content-type: text/html'

address command 'PIPE < LMCMDH HTML 'mydir,          3
                    '| change \%MYURL%\'MYURL'\ ',
                    '| stem htmlfile.'
'CGI WRITE DOCUMENT (TRANSLATE USENGLISH CRLF STEM HTMLFILE.'

      /* parametered call or initial call ?                           */
If SYMBOL('PARMS.CMD')='VAR' then do
      /*              it is not the first call       4              */
       Cmd = translate(PARMS.Cmd)

      /* This requires that the CGI has AUTHHEADERPASS authorization. */
      'CGI GETVAR HTTP_AUTHORIZATION (VAR AUTH_HDR'  5
      If auth_hdr = '' Then
            Call Error 'Can''t run.  Not protected by authentication'
      Else Do
         Call VIGRTNS 'USERPASS'                     6
         If Result = 0 Then Parse Var vigrtns_result . userid ':' pswd
      End

      /*   Connect to user ID                        7              */
      'VIG USER CONNECT 'userid
      If Result <> 0 Then Call Error 'Can not Connect 'result

      /*   Pass the entered command                  8              */
      'VIG USER ENTER 'Cmd
      If Result <> 0 Then Call Error 'Can not ENTER  'result

      /*   Displays the result                       9              */
      CmdRc="<HR>The command "Cmd" was executed with return code ",
             VIG_USER_RETCODE"."
      'CGI WRITE DOCUMENT (TRANSLATE USENGLISH CRLF VAR CMDRC'
      'CGI WRITE DOCUMENT (TRANSLATE USENGLISH CRLF STRING <PRE>'
      'CGI WRITE DOCUMENT (TRANSLATE USENGLISH CRLF STEM VIG_USER_OUTPUT.'
      'CGI WRITE DOCUMENT (TRANSLATE USENGLISH CRLF STRING </PRE><HR>'
 End /* SYMBOL('PARMS.CMD')='VAR' */
```

Figure 29 (Part 1 of 2). LMCMD VMGW Code

```
     /*   Ends HTML document                                    */
'CGI WRITE DOCUMENT (TRANSLATE USENGLISH CRLF STRING </BODY></HTML>'
exit 0

Error:
parse arg etext
     /*   write trace in server A disk                          */
address command "PIPE var etext | >> GATECGI "userid" A"
Return
```

Figure 29 (Part 2 of 2). LMCMD VMGW Code

The following notes should help you to understand the code in Figure 29 on page 71:

1 The CGI is intended to be served from an SFS directory. It has to read an HTML file (LMCMDH VHTML) placed in the same directory as the CGI program. Using X_SCRIPT_NAME_TRANSLATED avoids the need to code the directory name; the program retrieves it at execution time. Moreover, the procedure will run wherever it is. The only requisite is that the HTML file must be there also. In a production level application you should also handle the case of a non-SFS domain being returned. See the VM:Webgateway online documentation for more information.

2 The HTML file will be modified to provide the CGI URL, so the CGI reads environment variables to rebuild its own URL.

3 Here is the substitution of the variable %MYURL% in the HTML code:

```
<HTML>
<HEAD>
 <TITLE>The Linemode Interface</TITLE>
</HEAD>
<BODY>
 <H1><Center>The Linemode Interface</Center></H1>
 <HR>
<FORM action="%MYURL%" method="post" name="QRDR">
   <INPUT type="text"   size="30"  name="Cmd">
   <INPUT type="submit" name="Sub" value="Submit">
  <BR>CP commands must be prefixed by CP and REXX procedures by EXEC.
</FORM>
```

It can also be done with the VIG SUBSTITUTE command described in 4.2.4, "Programmer Productivity Tools" on page 75.

4 The program tests the CMD variable to determine if this is the first call of the procedure. This variable is sent only when the user fills the HTML form. If the variable is found, the procedure has to execute this command and send back the result.

5 The CGI must ensure that the user was requested to provide a user ID and password. We will need this information to log on to the virtual machine.

Important

Although *VM:Webgateway Tutorial* suggests using the HTTP_AUTHORIZATION header for this validation, we do not believe that this is the correct CGI variable to use because the existence of this request header does not prove that the data in the header has been authenticated. We believe that you should verify that the CGI variable X_AUTH_VERIFIED has been set to 1 (REXX′s true setting), in order to validate that the CGI is protected by VM:Webgateway′s security profiling.

While this does not prove that the AUTHORIZATION header was verified (since the authorization verification could have been based upon other criteria), it does verify that some criteria was checked, rather than simply attesting to the existence of the AUTHORIZATION header. Since VIGRTNS USERPASS is documented as also testing that the AUTHORIZATION header is present, it does not make sense to duplicate that effort in your CGI rather than simply relying upon VIGRTNS′s processing.

6 Read the authentication data. In a production level application you should also handle the case of a return code other than 0 from the VIGRTNS USERPASS call. See the VM:Webgateway CGI Extension online documentation for more information.

7 Connect to the VIGUSER interface running in the user virtual machine. If the virtual machine is logged off, VM:Webgateway CGI Extension will autolog it.

8 Execute the command.

9 Retrieve command results from VIG USER variables.

VM:Webgateway provides sample line mode applications that demonstrate how to implement a Web interface to some common VM commands, such as QT VMGW giving VM time or RDR VMGW listing the user′s reader contents. A skeleton is also provided to help line mode interface programming: SKELUSER XVMGW. You will find all these files on the 194 minidisk of the VM:Webgateway service virtual machine.

4.2.3 VM:Webgateway CGI Extension Full Screen Support

VM:Webgateway Tutorial covers full-screen Web-enabling considerations extensively. It includes an example of migrating a 3270 application to the Web to demonstrate various techniques, such as the conversion of multiple 3270 screens into a single HTML document and the addition of functions that did not already exist in the original 3270 application.

In this redbook we do not go into extensive detail on the use of the VM:Webgateway CGI Extension interfaces for full screen support because the *VM:Webgateway Tutorial* has done such a good job of presenting this material. Instead, we are including the tutorial on the VM Web CD, as described in B.1.9, “VM:Webgateway Tutorial” on page 234, and strongly recommend that you examine it. It shows that Web enabling is not just putting on new clothes, but instead may be more like changing to a better tailor.

VM:Operator, which comes with VM:Webgateway, needs to be up and running to control logical devices, as shown in Figure 30 on page 74.

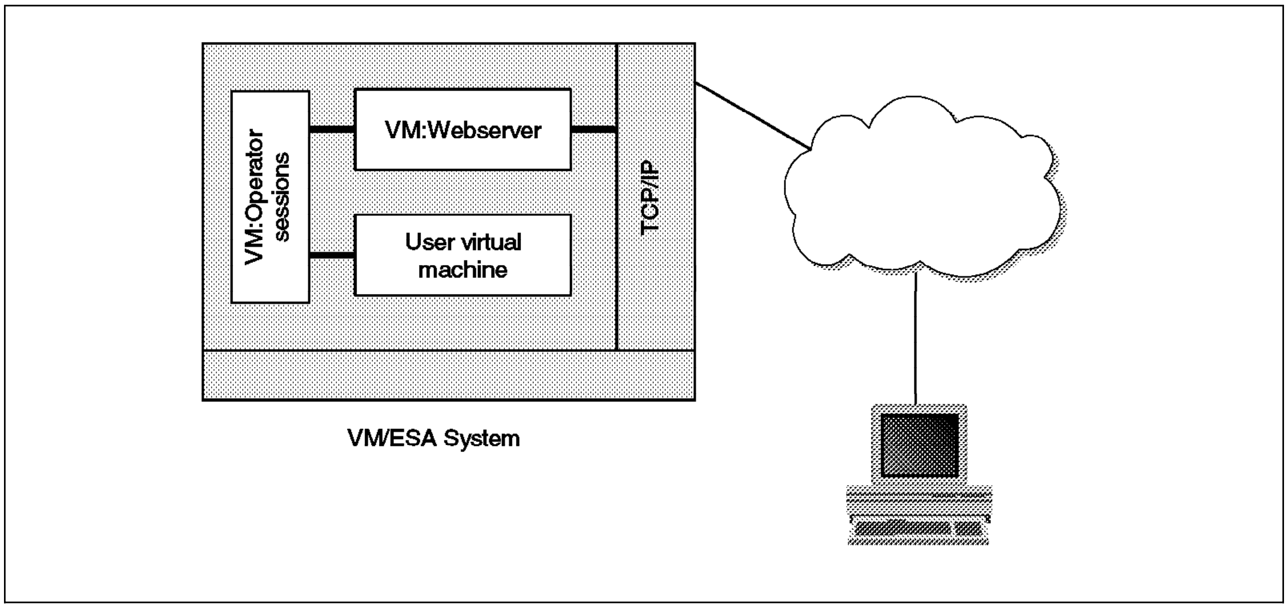


Figure 30. VM:Webgateway CGI Extension Full Screen Implementation

4.2.3.1 VIG SESSION Commands

There are far more VIG SESSION commands than VIG USER commands because the full screen controls need to be tighter. They need to control each character position on the screen and have to simulate more control keystrokes.

We will only describe the more frequently encountered controls here. Refer to the VM:Webgateway CGI Extension online documentation for a full description of all VIG SESSION subcommands.

VIG SESSION CREATE	Is used to ask VM:Operator to create a new logical device session.
VIG SESSION DESTROY	Tears down a session.
VIG SESSION FIELD	Puts data in a 3270 entry zone.
VIG SESSION PRESS	Simulates a keystroke.
VIG SESSION WAIT	Waits for the screen to be updated.

4.2.3.2 Typical Structure of a Full Screen Scripting CGI

A typical application will look like Figure 31 on page 75.

```
VIG SESSION CREATE VMOPER GATEWAY SESSION1 /* Create a session         */

VIG SESSION FIELD 1 userid                 /* Supply userid            */
VIG SESSION FIELD 2 password               /* and password             */
VIG SESSION PRESS ENTER                    /* Initiate logon sequence  */

VIG SESSION WAIT                           /* Wait for screen to update*/
     ...
     Invoke the application, step through interactions
     produce and display output for the browser
     ...

VIG SESSION PRESS PF3                     /* Exit application         */

VIG SESSION FIELD 1 LOGOFF                 /* Type LOGOFF on cmd line  */
VIG SESSION PRESS ENTER                    /*  and execute it          */

VIG SESSION DESTROY                        /* Tear down the session    */
```

Figure 31. VM:Webgateway CGI Extension Full Screen Typical Sequence of Commands

The application is as easy to code as the PC robots used for years to pilot 3270 screens. The same simple techniques apply and do not require any new learning. The big advantage is that the code is in your server in one single version that you maintain on the host. You do not have to deliver a new version of your program to thousands of PCs each time the 3270 application is modified.

VM:Webgateway includes sample full screen applications providing Web interfaces to some common VM commands. A skeleton is also provided to help full screen interface programming: SKELSESS XVMGW. You will find all of these files together with the line mode samples on the 194 minidisk of the VM:Webgateway service virtual machine.

Full screen scripting may also be used to access 3270 applications other than local VM ones. A good example provided as a sample in VM:Webgateway is the LOC VMGW CGI, which uses the CICS 3270 screens from the U.S. Library of Congress and creates a friendly front end to allow browsing of their catalog. This program logs on a local user ID and then calls the VM Telnet protocol to log on to the Library of Congress system. The same kind of thing is possible with SNA connected hosts: On the VM logo, use the DIAL VTAM command in the COMMAND field rather than the standard LOGON field.

4.2.4 Programmer Productivity Tools

The programmer productivity tools comprise a set of CGI helpers:

64ENCODE Converts data to base64 format.

This format survives translation from one character format to another (for example, from EBCDIC to ASCII). It may be used to protect the values of hidden variables from server translation and to guard against the coincidental presence of HTML reserved characters that may affect HTML interpretation. The only parameter of 64ENCODE is the string to be protected and the returned string should be found in VIGRTNS_RESULT.

64DECODE Decodes encoded base64 format data to its original format (same use as 64ENCODE).

DIALVTAM Accesses a VTAM application. The VTAM application can be running on a system other than the system that is running VM:Webgateway.

Before invoking the DIALVTAM routine, your CGI program must create the 3270 session using the VIG SESSION CREATE command, and the 3270 session must be displaying the VM logon banner screen.

The routine parameters are the VTAM user ID, the target application ID and the logmode to be used.

HTMLSAFE Converts HTML reserved characters in a string into a format that a Web browser can display.

These characters are reserved for HTML tags; if you need to include one of them, the routine transforms them into equivalent HTML entities:

< into &lt;
> into &gt;
& into &amp;
″ into &quot;

The string to process is the procedure parameter. Output is directed into the VIGRTNS_RESULT variable.

SUBSTITUTE Assigns the HTML in a specified CMS file to a REXX variable in your CGI program. The HTML file can contain VIG:VAR references that the SUBSTITUTE routine replaces with the corresponding values of REXX variables from your CGI program.

You have to code the references to your REXX variable in the HTML file formatted as

```
VIG:VAR(rexx_variable)
```

When SUBSTITUTE is called with the file name as parameter, it will try to resolve all VIG:VAR references. The resulting HTML code will be set to the VIGRTNS_RESULT variable.

This is a way to separate CGI development from HTML design.

USERPASS If the HTTP Authorization header is present, the USERPASS routine stores the user ID and password from the header into the VIGRTNS_RESULT REXX variable. If the HTTP Authorization header is not present, the USERPASS routine either ends (if a realm[7] is not specified) or has the Web browser issue a user ID/password challenge (if a realm is specified).

VMLOGON Logs on a VM user ID and invokes the specified full screen command from the CMS command line.

The routine can be run after a 3270 session is created by VIG SESSION COMMAND. It supports the following parameters :

- User ID

[7] A realm is simply a way of grouping user name and password information so that the browser need not prompt the user for a new name and password if subsequent requests access the same realm.

- Password
- LOGON command parameters
- Name system or address to IPL
- Command to enter after the user ID is logged on

The VMLOGON command handles the different virtual machine states (logged off, disconnected, logged on).

WEBENABL Web enables screens of a 3270 session. See section 4.2.5, “Using the WEBENABL Tool” for more information.

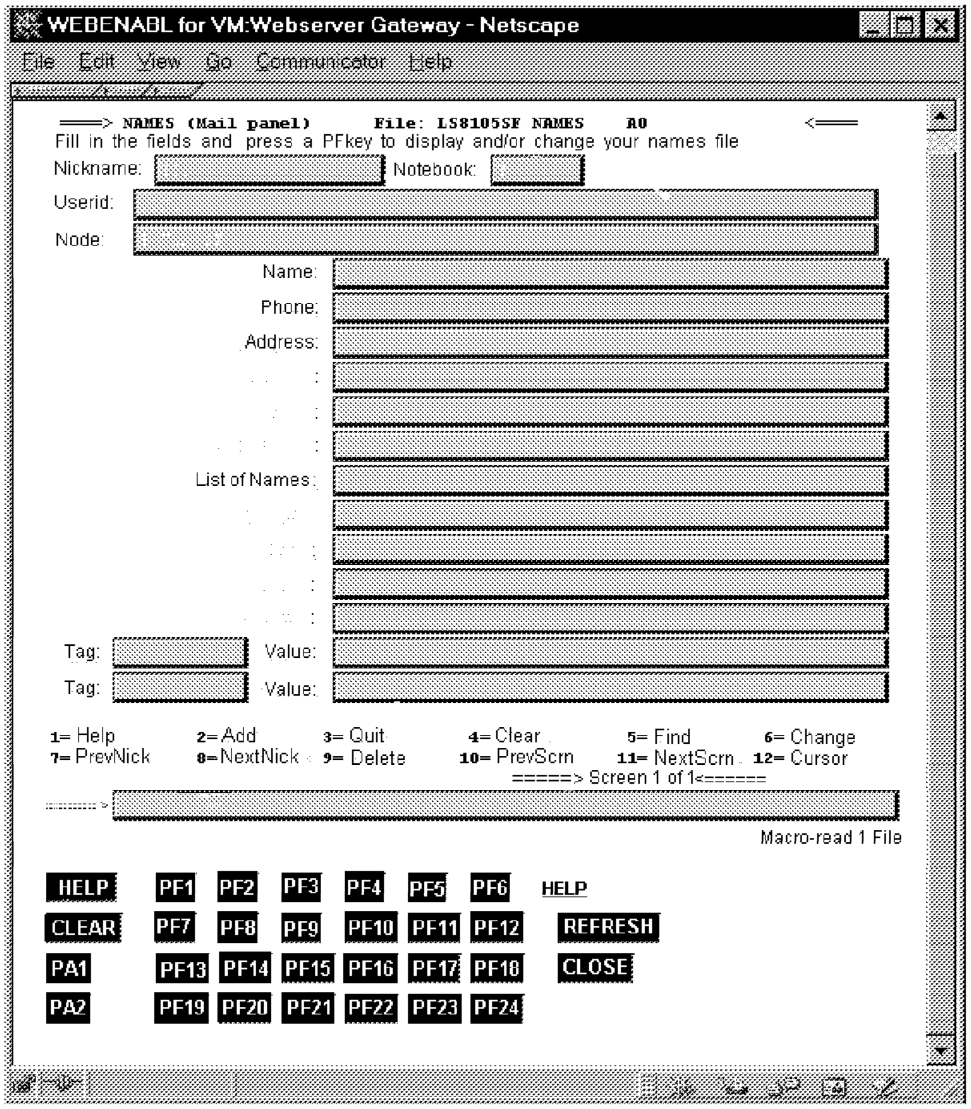
4.2.5 Using the WEBENABL Tool

WEBENABL is a powerful tool that can be used for two distinct but related purposes:

1. If you have a full-screen 3270 application that has many different screens, you may not have time to write code to enhance all the screens. Or you may have a full-screen 3270 application that contains some screens that are used infrequently, and you do not want to take the time to write code to Web enhance these screens.

   In these sorts of situations, the WEBENABL routine can be used to process and present the screens that your CGI program is not Web enhancing.

   Figure 32 on page 78 shows the result of using WEBENABL on the familiar 3270 panel constructed by the CMS NAMES command.


WEBENABL for VM:Webserver Gateway - Netscape
File Edit View Go Communicator Help

```
===> NAMES (Mail panel)      File: LS8105SF NAMES    A0          <===
Fill in the fields and press a PFkey to display and/or change your names file
Nickname:                 Notebook:
Userid:
Node:
                     Name:
                    Phone:
                  Address:
                         :
                         :
                         :
            List of Names:
                         :
                         :
                         :
                         :
Tag:                Value:
Tag:                Value:

1= Help       2= Add       3= Quit      4= Clear       5= Find       6= Change
7= PrevNick   8= NextNick  9= Delete   10= PrevScrn   11= NextScrn  12= Cursor
                                 =====> Screen 1 of 1<======
====>
                                                          Macro-read 1 File
```

HELP PF1 PF2 PF3 PF4 PF5 PF6 HELP
CLEAR PF7 PF8 PF9 PF10 PF11 PF12 REFRESH
PA1 PF13 PF14 PF15 PF16 PF17 PF18 CLOSE
PA2 PF19 PF20 PF21 PF22 PF23 PF24

Figure 32. NAMES Display by WEBENABL

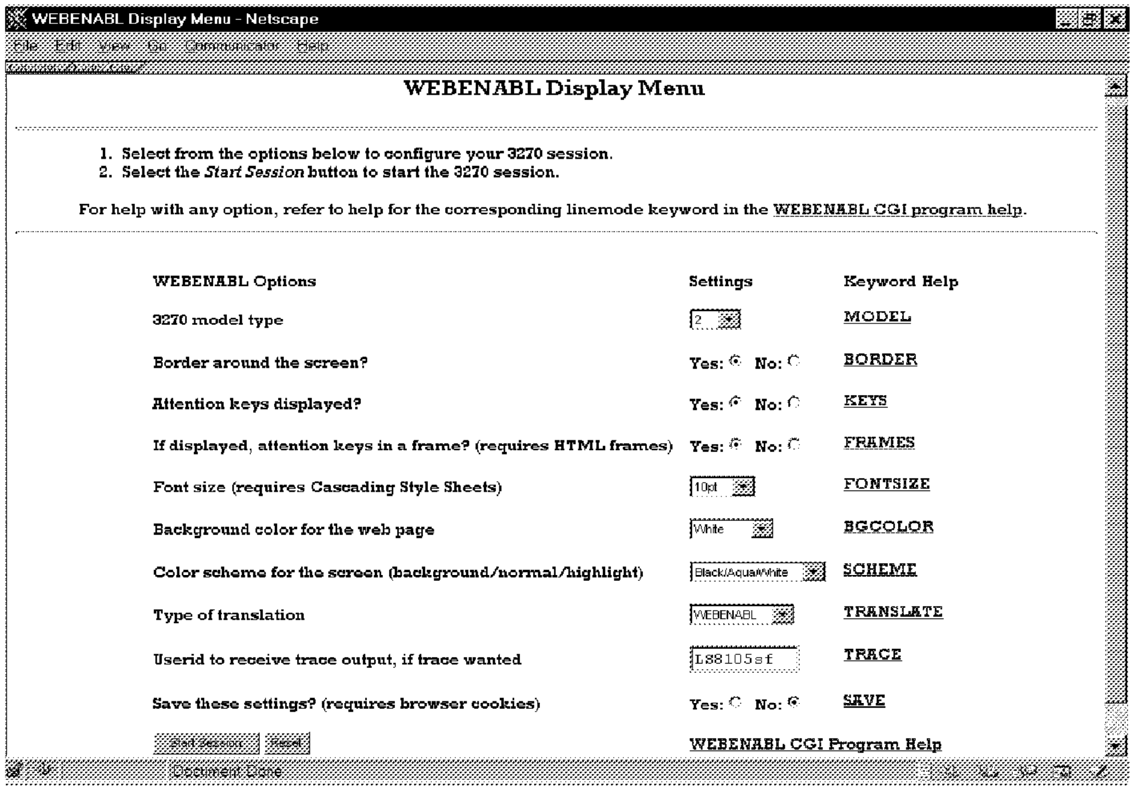
2. To trace the WEBENABL TN3270 inside the browser interface.

   Each user action and all 3270 screen field content is then kept in a trace file that can be used to help create your VIG SESSION scripts. This is fully described in *VM:Webgateway Tutorial*.

   To produce a trace, invoke the WEBENABL routine from your browser using the following sort of URL:

   ```
   http://yourserver/VM:Webserver/gateway/webenabl.vmgw
   ```

   The WEBENABL Display Menu will appear, asking you to set WEBENABL parameters, as illustrated in Figure 33 on page 79.



WEBENABL Display Menu - Netscape
File Edit View Go Communicator Help
Document: Done

WEBENABL Display Menu

1. Select from the options below to configure your 3270 session.
2. Select the *Start Session* button to start the 3270 session.

For help with any option, refer to help for the corresponding linemode keyword in the WEBENABL CGI program help.

WEBENABL Options	Settings	Keyword Help
3270 model type	2	MODEL
Border around the screen?	[✓ Yes:] No:	BORDER
Attention keys displayed?	[✓ Yes:] No:	KEYS
If displayed, attention keys in a frame? (requires HTML frames)	[✓ Yes:] No:	FRAMES
Font size (requires Cascading Style Sheets)	10pt	FONTSIZE
Background color for the web page	White	BGCOLOR
Color scheme for the screen (background/normal/highlight)	Black/Aqua/White	SCHEME
Type of translation	WEBENABL	TRANSLATE
Userid to receive trace output, if trace wanted	LS8105sf	TRACE
Save these settings? (requires browser cookies)	Yes: [✓ No:]	SAVE

Start Session Reset WEBENABL CGI Program Help

Figure 33. WEBENABL Menu

To activate the tracing facility, simply identify a user ID to receive the trace output. When the session is completed, the trace file is sent by the Web server to the reader of the specified user ID. This trace will contain a lot of information:

- All the 3270 screens (in a visual display)
- All the commands you entered, transformed into their VIG SESSION equivalents
- Additional comments on screen structures
- Descriptions of I/O flows

These traces can be used to help you to create your own gateway CGIs. Just run a trace session, and the information you need will be in the trace file.

Figure 34 on page 80 is a hand-annotated small part of a trace file from a session in which we logged onto a user virtual machine and edited a file. The complete trace file was 1400 lines long (420 lines without the imbedded 3270 screen captures).

```
16:56:24 VMYLGS357I VIG SESSION TRACE START LS8105SF
* Connection to logical device
16:56:34 VMYLGS357I VIG SESSION CONNECT B1459C7FF7B2F78200E31BF800000002
* Pressing the CLEAR button on logo screen
16:56:34 VMYLGS357I VIG SESSION PRESS CLEAR 20 17
* Receiving :
* Enter one of the following commands:
*
*    LOGON userid            (Example:  LOGON VMUSER1)
*    DIAL userid             (Example:  DIAL VMUSER2)
*    MSG userid message      (Example:  MSG VMUSER2 GOOD MORNING)
*    LOGOFF
*    UNDIAL
* login
16:56:34 VMYLGS357I VIG SESSION FIELDREF ALL
16:56:34 VMYLGS357I VIG SESSION TRACE COMMENT ScreenField 001: Length=0139 Location=23 001
                                              InputField=001
16:56:34 VMYLGS357I VIG SESSION TRACE COMMENT ScreenField 002: Length=0018 Location=24 061
                                              Protected
16:56:34 VMYLGS357I VIG SESSION TRACE COMMENT ScreenField 003: Length=1760 Location=24 080
                                              Protected
16:56:34 VMYLGS357I VIG SESSION FIELDREF INPUT
16:56:34 VMYLGS357I VIG SESSION DETACH
16:56:51 VMYLGS357I VIG SESSION CONNECT B1459C7FF7B2F78200E31BF800000005
16:56:51 VMYLGS357I VIG SESSION ACTION
16:56:51 VMYLGS357I VIG SESSION FIELDREF INPUT
16:56:51 VMYLGS357I VIG SESSION FIELD 1 logon myuserid
16:56:51 VMYLGS357I VIG SESSION PRESS ENTER 23 1
* password
16:57:07 VMYLGS357I VIG SESSION FIELD 1 mypassword
16:57:07 VMYLGS357I VIG SESSION PRESS ENTER 23 1
* RACF message causes HOLDING status so I press PA2
16:57:21 VMYLGS357I VIG SESSION PRESS PA2 23 1
*
16:57:49 VMYLGS357I VIG SESSION FIELD 1 filel
16:57:49 VMYLGS357I VIG SESSION PRESS ENTER 23 1
*
16:58:21 VMYLGS357I VIG SESSION PRESS PF8 10 1
* Press PF11 on eleventh line
16:58:54 VMYLGS357I VIG SESSION PRESS PF11 11 1
*
16:58:54 VMYLGS356I Keyboard is locked
16:58:54 VMYLGS361I Screen input operation: Read/Modified, 3 bytes
16:58:54 VMYLGS355I Screen output operation: Erase/Write, 35 bytes; keyboard is Unlocked
16:58:54 VMYLGS360I VIG SESSION PRESS completed
16:58:54 VMYLGS355I Screen output operation: Write, 52 bytes; keyboard is Unlocked
16:58:54 VMYLGS355I Screen output operation: Write, 54 bytes; keyboard is Unlocked
16:58:54 VMYLGS355I Screen output operation: Write, 37 bytes; keyboard is Unlocked
16:58:54 VMYLGS355I Screen output operation: Erase/Write, 1261 bytes; keyboard is Unlocked
* ending XEDIT
16:59:05 VMYLGS357I VIG SESSION PRESS PF3 2 7
* ending FILELIST
16:59:16 VMYLGS357I VIG SESSION PRESS PF3 11 1
* LOGOFF
16:59:24 VMYLGS357I VIG SESSION FIELD 1 logoff
16:59:25 VMYLGS357I VIG SESSION PRESS ENTER 23 1
```

Figure 34. WENENABL Trace Elements

4.2.5.1 Line Mode Application Demonstration

The LMCMD VMGW line mode interface can be run in any user ID, but if the user virtual machine is to be autologged, the initialization of the VIGUSER MODULE has to be done in the PROFILE EXEC or in SYSPROF EXEC.

You can also start it from the CMS command line and leave your user ID logged on:

```
LINK VMRMAINT 193 493 RR
ACCESS 493 fm
VIGUSER INIT VMWEBSRV
```

If you intend to use VM:Webgateway CGI Extension, you may copy the VIGUSER module on the Y disk and initialize it in SYSPROF EXEC.

To access the sample CGI from the browser, we used the following URL:

`http://wtscvmt.itso.ibm.com/~ vmwebcd/cgigateway/lmcmd.vmgw`

The VMGW type of the CGI indicates the CGI is a REXX procedure running in a VM:Webgateway virtual machine.

4.3 Byte File System

The Byte File System (BFS) emulates a UNIX-like file system on CMS. It can be an excellent place to store HTML documents that are served by your Web server because the application requires UNIX directory structures or file names. VM:Webgateway can be configured to serve HTML documents from BFS and also read and execute CGI programs that are stored in BFS. The changes needed to read data or run a VM:Webgateway CGI program from a BFS directory are not extensive.

The easiest way to read BFS files from a CGI program is to use CMS Pipelines. The <oe stage can read data directly from a BFS file and automatically deblock the data into records. The < stage will automatically call the <oe stage when the file specification is BFS. Here is an example of reading a BFS file:

```
PIPE <oe /../VMBFS:VMSYS:ROOT/home/cmsps/pages/phone.data
```

Attention

The normal PIPE help files do not document the <oe stage. Help is available using the command `PIPE HELP <oe` (or on some systems, `PIPE AHELP <oe`.) These commands view the *author's* help files instead of the normal CMS help files. Enter `PIPE HELP MENU` to see a menu of all help topics.

As this is written, only VM:Webgateway supports BFS directories for data and CGI programs. The CGI program support does not include Webshare compatibility programs when the program is executed from a BFS directory.[8]

A modified version of the sample CGI program PHONE1 SVMEXEC follows Figure 35 on page 82. See 3.1.2, "Sample CGI Program" on page 22 for the original data files. See also 3.1.3, "Sample CGI Program for VM:Webgateway"

[8] This limitation exists as a part of the definition of the Webshare compatibility mode. This defines that the source directory is accessed on the server, but CMS does not support a BFS directory as an argument to the CMS ACCESS command.

on page 24 for the native VM:Webgateway CGI program. The modified sample CGI program is named phonebfs.svmexec.

```
/* phonebfs.svmexec program example */
/* (type SVMEXEC is configured as ENVIRONMENT SVMEXEC or WORKEREXEC)*/
Address Command
PhoneFile = 'phone.data'        /*     file name changed for BFS */
DataDir   = '/../VMBFS:VMSYS:ROOT/home/cmsps/pages/'
/*                                  BFS location of the Phone file */
                               1
/* Define variables that make it easy to write the CGI output. */
CGIwrite='CGI WRITE DOCUMENT (TRANSLATE USENGLISH CRLF STRING'
CGIpipe='join * xOD25 65535',     /* Block lines for efficiency     */
      '| change //'CGIwrite '/',  /* Insert the WRITE command      */
      '| command'
?Access = 0                       /* Was a disk accessed?           */
CGIlocation = Script_Location()   /* Find location of this CGI      */
                               2
'PIPE <oe' CGIlocation||'/prolog1.htmlpart',  /* Write prolog tags  */
    '|' CGIpipe

CGIwrite '<TABLE BORDER="1">'       /* Set up the HTML table        */
CGIwrite '<TH>Employee Number</TH>' /* Put a header on each column  */
CGIwrite '<TH>Name</TH>'
CGIwrite '<TH>Phone Number</TH>'
CGIwrite '<TH>Location</TH>'
CGIwrite '<TH>Office</TH>'
CGIwrite '<TH>Status</TH>'

/* Get the telephone data from the file. */
'PIPE <oe' DataDir||PhoneFile '| stem phone.'

Do i=1 to phone.0                 /* Format the file for display    */
   If left(phone.i,1) = '*' then  /* Ignore the comments            */
      iterate
   CGIwrite '<TR>'                /* Beginning of a table row       */
   CGIwrite '<TD>' substr(phone.i, 1, 6) '</TD>' /* Each            */
   CGIwrite '<TD>' substr(phone.i, 8,20) '</TD>' /*   table         */
   CGIwrite '<TD>' substr(phone.i,29,12) '</TD>' /*      cell       */
   CGIwrite '<TD>' substr(phone.i,42, 3) '</TD>'
   CGIwrite '<TD>' substr(phone.i,46, 9) '</TD>'
   CGIwrite '<TD>' substr(phone.i,56, 5) '</TD>'
   CGIwrite '</TR>'               /* End of a table row             */
End
CGIwrite '</TABLE>'

'PIPE <oe' CGIlocation||'/footer.htmlpart',  /* Write ending tags   */
    '|' CGIpipe

If ?Access = 1 then               /* Did we access a disk?          */
   'RELEASE' CGIlocation          /* If so - clean up.              */

Exit RC
```

Figure 35 (Part 1 of 2). Sample CGI Program phonebfs.svmexec

```
Script_location:  Procedure expose ?Access
/* Find out where this CGI is located */
/* This will be used by PIPE "<" stage will work with "domains" */
/* of SFS, BFS, or CMS.  For MDISK, the disk is accessed.       */
'CGI GETVAR X_SCRIPT_NAME_TRANSLATED (VAR' xlocation
Parse var xlocation . . type location . vaddr .
If type = 'MDISK' then do
   call CSL 'DMSGETFM rc rs location'   /* Get a free filemode      */
   If rc <> 0 then
      'CGI EMSG (MSGFILE STRING 0699E RC='rc 'from DMSGETFM'
   Else do
      'ACCESS' vaddr location
      If rc <> 0 then
         'CGI EMSG (MSGFILE STRING 0698E RC='rc 'from ACCESS'
      ?Access = 1                /* Remember we accessed a disk.   */
   end
end
Return location
```

Figure 35 (Part 2 of 2). Sample CGI Program phonebfs.svmexec

The following notes describe the BFS access in Figure 35 on page 82.

1 The variable PhoneFile contains the file name of the file containing the data. The variable DataDir contains the directory that the PhoneFile is located in.

2 The CGIlocation variable has been set to the directory that the CGI was served from. The following information consists of the CGI line from the sample, followed by the resultant BFS file location:

```
'PIPE <oe' CGIlocation||'/prolog1.htmlpart' ,  /* Write prolog tags */
  /../VMBFS:VMSYS:ROOT/home/cmsps/pages/htbin/prolog1.htmlpart

'PIPE <oe' DataDir||PhoneFile ,
  /../VMBFS:VMSYS:ROOT/home/cmsps/pages/phone.data

'PIPE <oe' CGIlocation||'/footer.htmlpart' ,  /* Write ending tags */
  /../VMBFS:VMSYS:ROOT/home/cmsps/pages/htbin/footer.htmlpart
```

The VM:Webgateway setup for the user can be found in 7.6.1, "VM:Webgateway BFS Access Setup" on page 196. An overview of the POSIX (UNIX) Style terminology and the directory structure commonly found in the BFS on VM can be found in 7.4.1, "POSIX Terminology" on page 193. See Chapter 7, "Desktop Web Publishing to VM Web Servers" on page 185 for information on how to publish and serve Web pages developed with desktop Web publishing tools.

4.4 Shared File System

The Shared File System (SFS) is mentioned many times in this book for storing data and CGI programs. SFS has been available for CMS for quite awhile and is a very stable file system. It has advantages over CMS minidisks, especially for CGI programming. One big advantage of using SFS for Web serving and programming is the capability to truly create a hierarchical file structure. True subdirectories can free you from the need to make countless entries into a Webshare or EnterpriseWeb/VM FILELIST or VM:Webgateway DIRMAP files.

The CGI program examples in this book reside in an SFS and use SFS for data files. The sample VM Web CD that comes with this book (see Appendix B, “Contents of the Associated CDs” on page 227 for a list of the contents) is designed to be loaded into SFS directories.

An easy way to access SFS data from a CGI program is to use CMS Pipelines. For VM:Webgateway native CGI programs, use the PIPE command, and for Webshare and EnterpriseWeb/VM CGI programs, use the CALLPIPE subcommand. The < stage will call the <sfs stage if a directory is specified instead of a file mode. Here is an example of reading a file:

```
PIPE < filename filetype Filepool:userid.subdir
```

The > stage will write an SFS file if an SFS directory name is specified by calling the >sfs stage. It is a very good idea to use the <sfs and >sfs stages to read and write SFS files because this forces CMS Pipelines to use the newer CMS file interfaces. These interfaces understand the SFS file system and automatically preserve aliases and authorizations on files that are updated. A CMS access of the SFS directory is not necessary for PIPE or CALLPIPE to read or write files contained within the directory.

SFS Directory Control directories perform better than the default File Control directories, but with some loss of flexibility. You give up the file level control of access permissions and the ability to create aliases. They operate much like a CMS minidisk. For directories with frequent updates by different individuals, or updated by CGI programs in a multiple Web Server environment, file control directories are desired.

If you are going to read a file, obtain updates, and then update the file, you must remember that multiple copies of your CGI program could be running on different servers or workers at the same time. Therefore, you must make sure the file is locked for updating before you begin reading it. SFS will implicitly lock a file when it is opened for updating, as long as the file remains opened while your program performs the update. If your program is using CMS commands or CMS Pipelines programs that close the file after reading it, then you must first lock the file with either the CREATE LOCK or the DMSCRLOC CSL routine before you read any file that you intend to update. After an update, commit the change and then clear the lock.

Figure 36 on page 85 shows a code fragment that obtains a lock, performs an update, and then returns the lock. It contains a loop that waits for the lock to become available if it is used by another server. If the CGI program containing this code fragment is running in a Web server worker machine, CP SLEEP can be used to perform the wait. Otherwise, CMS multitasking routines should be used to perform the wait so that other threads of execution in the server are allowed to execute.

```
file = 'DATA FILE SFSTEST
:VMWEBCD.DATA'
/* File to be read and updated  */
lock = 'UPDATE SESSION'
Call CSL 'DMSGETWU rc rs wuid'    /* Get a work unit               */
/* Create an UPDATE SESSION lock on the file, within our work unit  */
Call CSL 'DMSCRLOC rc rs file' length(file) 'lock' length(lock) 'wuid'
/* If someone else has the lock, assume it will be available soon   */
Do 10 while RC = 8 & (rs >= 2700 & rs <= 2900)
   Address Command 'CP SLEEP 1 SEC' /* Wait just a bit             */
   Call CSL 'DMSCRLOC rc rs file' length(file) 'lock' length(lock) 'wuid'
End
If RC < 0 | RC > 4 then do        /* Look for failures             */
   If RC = 8 & (rs >= 2700 & rs <= 2900)
      then say 'Lock held by another user not released.'
      else say 'DMSCRLOC Return code=' rc 'Reason=' rs
   Signal Error
end
'PIPE <sfs' file 'WORKUNIT' wuid, /* Use our workunit to read file  */
   '| locate /'marker'/',         /* Is the record in the file?     */
   '| take 1',
   '| count lines',
   '| var found'
If found = 0 then do              /* Record is not in the file      */
   /* Insert code that creates a new record here.. */
   'PIPE literal new   RECORD',   /* Add the record to the file     */
      '| >>sfs' file 'WORKUNIT' wuid
end
Call CSL 'DMSCOMM rc rs wuid'     /* Commit our workunit            */
If RC >= 8 then Signal Error      /* Did it work?                   */
/* Remove the lock */
Call CSL 'DMSDELOC rc rs file' length(file) 'wuid'

Exit:
Call CSL 'DMSRETWU rrc rrs wuid'  /* Return the work unit           */
Exit RC

Error:   /* Something bad happened, roll back our update ..         */
Call CSL 'DMSROLLB rrc rrs wuid'
If rrc > 4 then say 'ERROR IN ROLLBACK, RC=' rrc 'Reason=' rrs
Signal Exit
```

Figure 36. Sample REXX Code Fragment Illustrating Locking

4.5 CMS Minidisks

CMS minidisks are the "traditional" place to store data on a VM system. They can also be utilized in a Web server environment, but they are not the best choice for frequently updated data. We recommend that SFS or BFS be used for this kind of environment. However, if the data is very infrequently updated or already exists on minidisks, it does not need to be moved.

If you must access data stored on CMS minidisks, it is preferable to have the information in the CMS search order of the Web server. However, if a CGI program must write on a minidisk, it must access and release the disk each time it performs an update. If a CGI program must read data from a minidisk that is frequently updated, it must perform a re-access of the disk anytime it reads a file. Otherwise, it may read old data or encounter read errors. (It may encounter read errors even after a re-access, so a retry mechanism should also be used.) Care must be taken in all CGI programs that all read/write accessed disks are released before program termination, even if the program abnormally terminates. Otherwise, another Web server or worker will not be able to access the disk.

If your Web server supports running multiple simultaneous CGI programs on a Web server, we recommend that you do not run CGI programs that update mini disks in this environment. Two or more CGI programs in one server may attempt to access the same read/write disk with different access modes and CMS does not allow multiple accesses of read/write disks. Therefore, one CGI program will alter the environment of another one. Either use worker machines or configure your Web server to only run one CGI program at a time in one server (if this is possible.) Even if a CGI program is only linking and reading data from a minidisk, it must be careful that the execution environment of another CGI program does not change. See 5.6.6, “Reentrant and Serially Reusable Resources and CGIs” on page 144 for more information on these resources.

4.6 DB2 Databases

Relational databases have been the most popular warehouses for business data for years. Part of this product family, DB2/VM has shown its performance and reliability strengths. Added to these qualities, the REXX SQL interface eases and accelerates DB2 developments, which is why accessing relational tables from VM is the more efficient way to put them on the Web. Many DB2/VM sites already run REXX SQL procedures. They may adapt their procedures to be called by CGI programs, or use the cross platform IBM DB2 Web connector, DB2 World Wide Web Connection Version 1 (DB2 WWW).

4.6.1 DB2 World Wide Web Connection Version 1

DB2 WWW is a CGI gateway allowing Web access to DB2 databases. Application developers write applications called “macro specification files.” These macro files contain sections written with HTML and Structured Query Language (SQL) that control the presentation and execution of the application. DB2 WWW′s runtime engine substitutes the HTML and SQL definitions in the application, allowing the user to submit SQL commands to a DB2 database and to view the data returned from the operation. The application developer who knows HTML and SQL can efficiently write an application and bring it into production.

This section describes the gateway and the samples we have included on the VM Web CD in more detail. We also describe the process of developing Web-enabled DB2 applications, including the development of SQL and DB2 WWW macro files.

Attention

If you prefer to install the DB2 WWW Demonstration and Samples before continuing, see 4.6.1.6, “VM Web CD DB2 Sample Applications” on page 97.

4.6.1.1 Control Flow for a Simple Application

Let us begin by reviewing the flow of an application request from the client (Web browser) to the server.

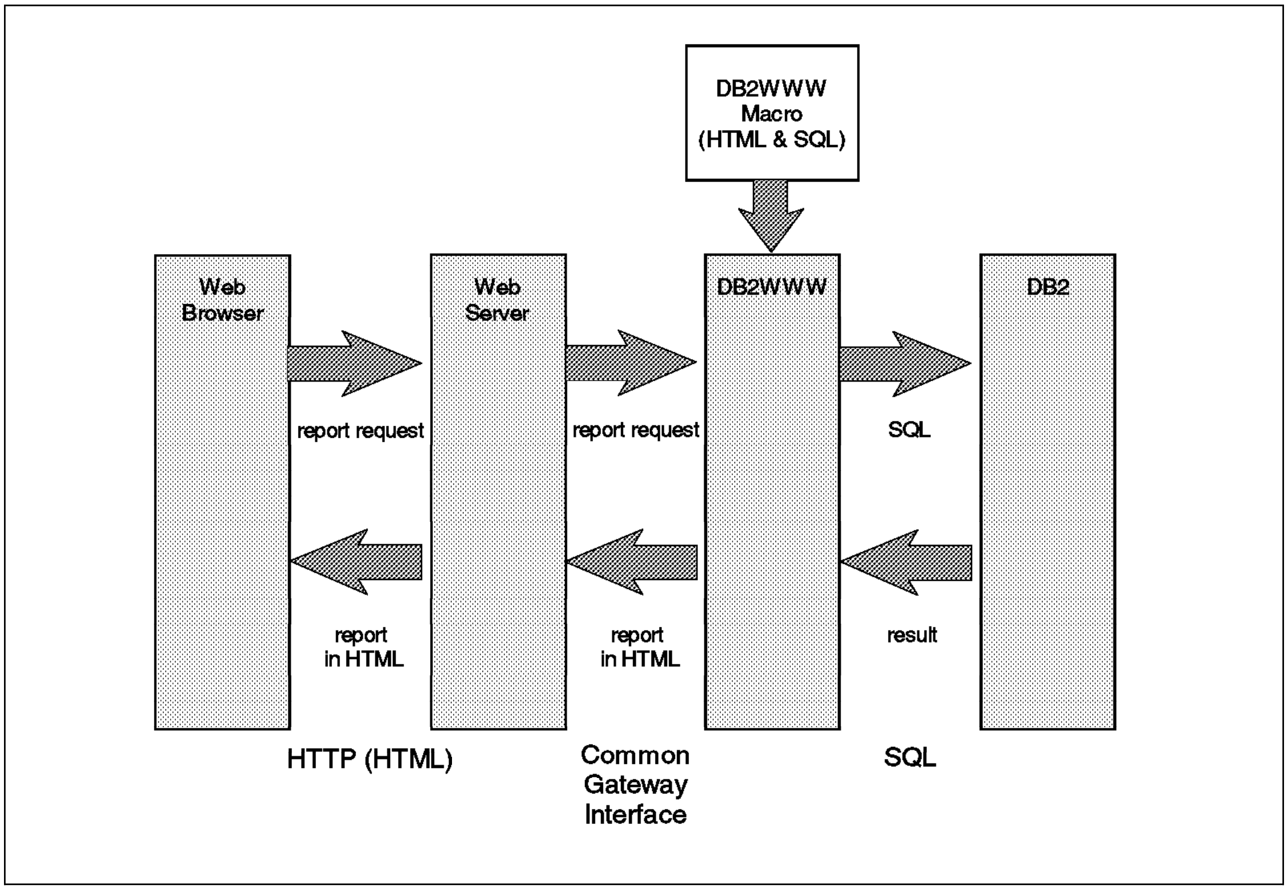


Figure 37. Application Flow for a Simple DB2 WWW Request

Figure 37 shows the application flow for a simple request, where the client initiates a request (providing no input) and the server returns the result of an SQL query. The flow shown in Figure 37 assumes that the browser is displaying an HTML page that has an option to start some DB2 processing. For example, the browser might be displaying a menu of options, and selecting one option will cause a DB2 request. The option on the screen would look like the following:

```
Click here to get a list of Employees
```

This would be encoded in HTML as:

```
<a href="/cgi-bin/db2www/empqry1.d2w/report">
Click here to get a list of Employees</a>
```

This “menu” was sent to the client browser previously, and is not considered in this control flow. When the user clicks on the appropriate option, the browser sends the action request (encoded in the HTML page) to the server.

The server receives a GET request for the CGI program (db2www), with parameters of the macro file name to be used (empqry1.d2w), and the section of the macro to be used (in this instance, the report section).

The server first examines control file directives to determine how to process the user request. The request for `/cgi-bin/db2www/empqry1.d2w/report` is parsed to the CGI environment as:

```
path        =   cgi-bin
script_name =   db2www
path_info   =   /empqry1.d2w/report
```

Moreover, the CGI_Path parameter in Webshare and EnterpriseWeb/VM servers needs to point to the directory to be interpreted as path information. Check DB2 World Wide Web Connection Version 1 requisites for minimum Web server versions that support this parameter (see the VM Web CD file /db2/db2www/install.htm).

The path information is only used to indicate the DB2 WWW macro file name and parameter.

The db2www CGI program is distributed as part of DB2 World Wide Web Connection Version 1 in object code form. Tailoring and configuration (for your own applications) are done using macro files and an “ini” file that describes where the macros are located:

```
* MACRO_PATH cms:*    <-- in the CMS search order
* MACRO_PATH cms:B    <-- on the B disk or dir.
* MACRO_PATH sf:dirid <-- on unaccessed SFS dirid
MACRO_PATH MYFP:WEB:WEBSHARE.D2WSAMP
```

A macro has four sections:

1. A DEFINE section to define variables used in a macro
2. An HTML input section to receive input from the client Web browser and to place it in the SQL query
3. An SQL section (could be one or more) to define the query and send it to the database
4. An HTML report section to invoke the SQL query and display the results to the client Web browser

An example of a macro file is shown in Figure 38 on page 89.

```
%define{ 1
  LOGIN    = "WWWTS"
  PASSWORD = "WWWTST"
  DATABASE = "SQLDBA"
  ow       = "SQLDBA"
  dbtbl    = "employee"
%}
%SQL{    2

  SELECT * FROM $(ow).$(dbtbl)

  %SQL_REPORT{     3
    <CENTER>
    <TABLE BORDER>
    <TR>
      <TD>$(N1)</TD>    4
      <TD>$(N2)</TD>
      <TD>$(N3)</TD>
      <TD>$(N4)</TD>
      <TD>$(N5)</TD>
      <TD>$(N6)</TD>
    </TR>
    %ROW{
      <TR>
        <TD>$(V1)</TD>   5
        <TD>$(V2)</TD>
        <TD>$(V3)</TD>
        <TD>$(V4)</TD>
        <TD>$(V5)</TD>
        <TD>$(V6)</TD>
      </TR>
    %}
    </TABLE>
    </CENTER>
    <pre>
  %}
%}
%HTML_REPORT{  6
  <TITLE>DB2 Query of Employee table</TITLE>
  <h1>Telephone Directory</h1>
  <p>
  <hr>
  %EXEC_SQL   7
  <p>
  <hr>
%}
```

Figure 38. EMPQRY1 D2W Code

Notes for Figure 38:

1 The DEFINE section allows you to define local variables for use with the macro. Here we are defining the SQL user, SQL password, table owner and table name that will be used in the SQL request. SQL user (LOGIN) and SQL password (PASSWORD) are DB2 WWW system variables, while table owner (ow) and table name (tc) are user variables. User variables provide an easy way to dynamically build SQL commands and to communicate from one macro to another.

2 The SQL request to be executed is defined here. Although it is possible to define and name many SQL requests, we have (in this case) one very simple SELECT request, which will return the entire contents of one table. The SQL in a section is executed when it is called by %EXEC_SQL in the HTML report section.

3 This SQL Report subsection defines how the result of the query should be sent to the user. You can use any HTML tags to format your output. Here we are using an HTML 3 table. If you have no SQL report subsection, a default table is displayed with column names at the top.

4 The DB2 column names are used as the table header names.

5 Each row of the resulting table generates a row of the HTML table.

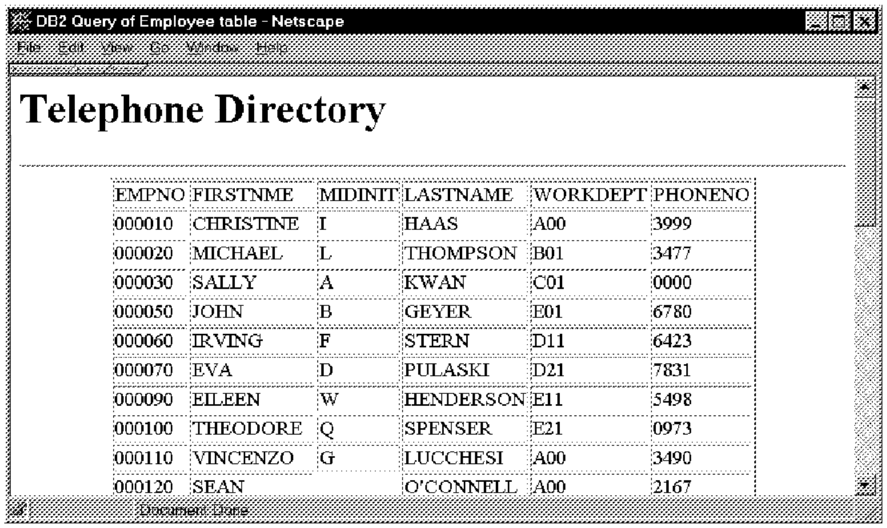
6 This is the section of the macro file that is requested by the incoming client request. You can place a title, HTML meta information, graphics, and any other information you consider relevant in this section.

7 Following the information to be placed at the beginning of the output HTML page, the SQL is executed and the results returned as defined in the SQL_REPORT section (line **3**). Following the output results, you can place additional "footer" information to complete the page.

So, the employee table looked like this under ISQL:

```
EMPNO   FIRSTNME      MIDINIT  LASTNAME         WORKDEPT  PHONENO
------  ------------  -------  ---------------  --------  -------
000010  CHRISTINE     I        HAAS             A00       3979
000020  MICHAEL       L        THOMPSON         B01       3477
000030  SALLY         A        KWAN             C01       0000
000050  JOHN          B        GEYER            E01       6780
000060  IRVING        F        STERN            D11       6423
000070  EVA           D        PULASKI          D21       7831
000090  EILEEN        W        HENDERSON        E11       5498
000100  THEODORE      Q        SPENSER          E21       0973
```

Now, however, it looks as shown in Figure 39.


DB2 Query of Employee table - Netscape
File Edit View Go Window Help
Document Done

Telephone Directory

EMPNO	FIRSTNME	MIDINIT	LASTNAME	WORKDEPT	PHONENO
000010	CHRISTINE	I	HAAS	A00	3999
000020	MICHAEL	L	THOMPSON	B01	3477
000030	SALLY	A	KWAN	C01	0000
000050	JOHN	B	GEYER	E01	6780
000060	IRVING	F	STERN	D11	6423
000070	EVA	D	PULASKI	D21	7831
000090	EILEEN	W	HENDERSON	E11	5498
000100	THEODORE	Q	SPENSER	E21	0973
000110	VINCENZO	G	LUCCHESI	A00	3490
000120	SEAN		O'CONNELL	A00	2167

Figure 39. EMPQRY1 D2W Output

This macro was called with the report parameter, so DB2 WWW searched for the %HTML_REPORT section, executed %SQL actions and sent formatted output to the Web browser.

Another type of DB2 WWW macro exists, which is the "input" macro. For such a call, DB2 WWW CGI searches for a %HTML_INPUT section and sends it to the browser. This section is supposed to contain form data that will be used to build %SQL actions occurring after validation.

One DB2 WWW can handle both parameters, as in Figure 40.

```
%define{
  LOGIN    = "WWWTS"
  PASSWORD = "WWWTST"
  DATABASE = "SQLDBA"
  ow       = "SQLDBA"
  tc       = "employee"
%}
```

Figure 40 (Part 1 of 4). EMPQRYC1 D2W Code

```
%SQL{ 1
  select EMPNO, FIRSTNME, MIDINIT, LASTNAME , WORKDEPT, PHONENO
          from $(ow ).$(tc)
          where EMPNO='$(employee)'

  %SQL_REPORT{  2
    %ROW{
      <FORM METHOD="POST"
            ACTION="/cgi-bin/db2www/empchg1.d2w/report">  3
        <INPUT TYPE="hidden" SIZE="30"        4
               NAME="empnum" VALUE="$(V1)">
        <INPUT TYPE="hidden" SIZE="30"
               NAME="ow" VALUE="$(ow)">
        <INPUT TYPE="hidden" SIZE="30"
               NAME="tc" VALUE="$(tc)">
        Please review and update the telephone number of<BR>
         $(V2) $(V3) $(V4), Employee number $(V1) of dept $(V5)<br>
        <PRE>
         Telephone Number:
         <INPUT TYPE="text" SIZE="30" NAME="phone" VALUE="$(V6)">
        </PRE>
        <INPUT TYPE="submit" VALUE="UPDATE">
      </FORM>
    %}
  %}
  %SQL_MESSAGE{  5
  100 : "<B>No employees for this number.<p>" : continue
  %}
%}
```

Figure 40 (Part 2 of 4). EMPQRYC1 D2W Code

```
%HTML_REPORT{    6
  <TITLE>Employee Informations</TITLE>
  <P>
  <H1>Employee Informations</H1>
  <P>
  %EXEC_SQL
  <P>
  <hr>
%}
```

Figure 40 (Part 3 of 4). EMPQRYC1 D2W Code

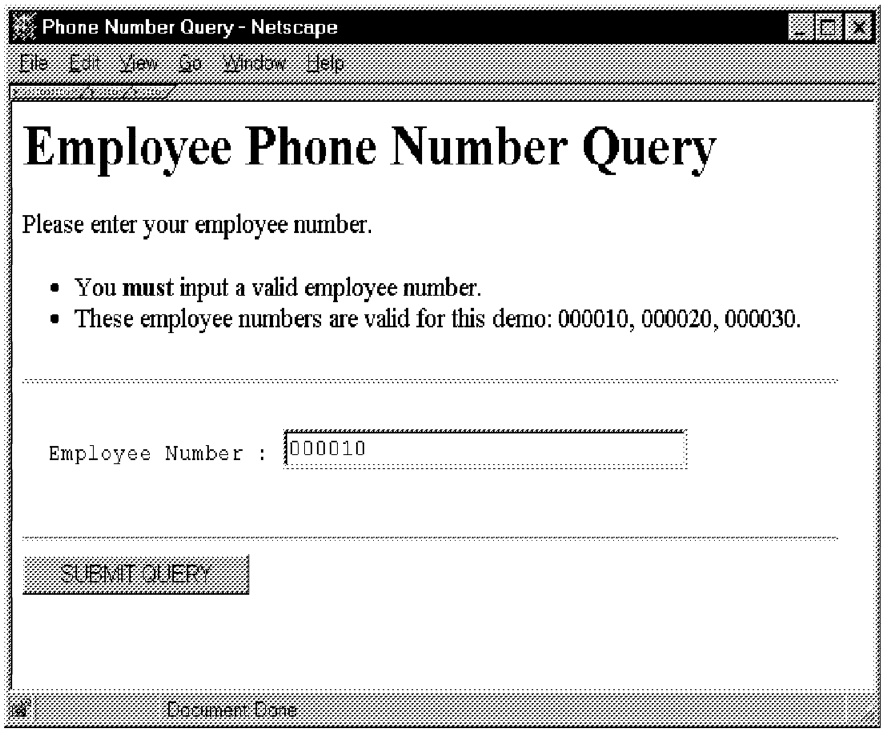
```
%HTML_INPUT{     7
  <TITLE>Phone Number Query</TITLE>
  <P>
  <H1>Phone Number Employee Information Query</H1>
  <p>
  Please enter your employee number.
  <P>
  <UL>
     <LI> You <strong>must</strong> input a valid employee number
     <LI>These employee numbers are valid for this demo:
      000010, 000020, 000030
  </UL>
  <FORM METHOD="POST"
     ACTION="/cgi-bin/db2www/empqryc .d2w/report">   8
     <hr>
     <PRE>
     Employee Number :
     <INPUT TYPE="text" NAME="employee" SIZE=30>      9
     </PRE>
     <hr>
     <INPUT TYPE="submit" VALUE="SUBMIT QUERY">
  </FORM>
  </b>
%}
```

Figure 40 (Part 4 of 4). EMPQRYC1 D2W Code

The URL to use this form would be encoded as:

```
<a href="/cgi-bin/db2www/empqryc1.d2w/input">
Click here to modify phone number assignments</a>
```

The input parameter informs DB2 WWW to display the %HTML_INPUT section (7) as in Figure 38 on page 89.



Employee Phone Number Query

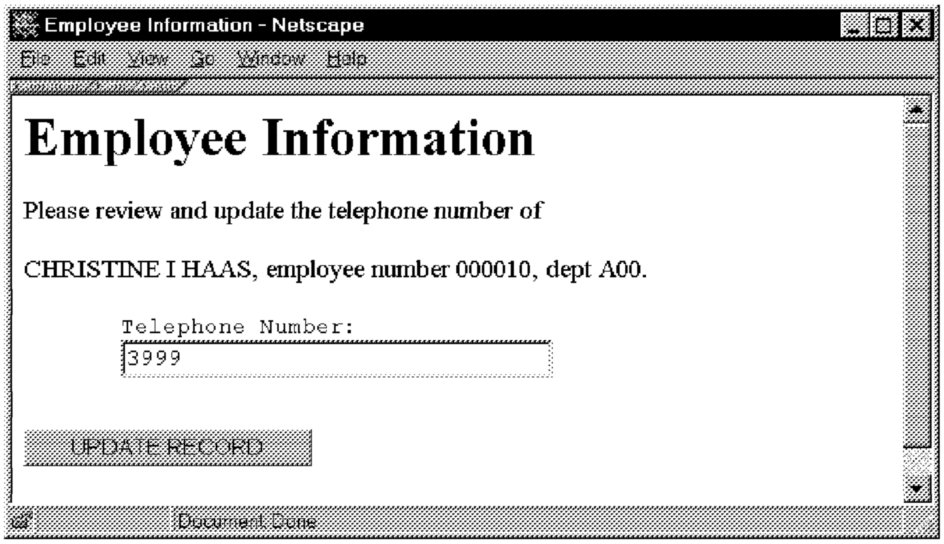
Please enter your employee number.

- You **must** input a valid employee number.
- These employee numbers are valid for this demo: 000010, 000020, 000030.

Employee Number : 000010

Figure 41. EMPQRYC1 D2W Input Request Output

Once the user clicks **SUBMIT**, the action tag calls the same DB2 WWW macro with the report parameter (**8**). This next call executes %SQL statements (**1**) according to employee values entered at the first invocation (**9**). Then, SQL results are merged with %HTML_REPORT (**6**):


Employee Information - Netscape
File Edit View Go Window Help

Employee Information

Please review and update the telephone number of

CHRISTINE I HAAS, employee number 000010, dept A00.

Telephone Number:
3999

UPDATE RECORD
Document Done

Figure 42. EMPQRYC1 D2W Report Request Output

This time, the telephone number extracted from the SQL database is displayed. The Web user is able to modify it and to submit the update, which is processed by another DB2 WWW (**3**) macro.

You may have also noticed hidden variables (**4**) to be passed to the updating macro and inline processing of SQL return codes (**5**).

The routine responsible for updating the employee table is shown in Figure 43.

```
%define{
  LOGIN    = "WWWAUT"
  PASSWORD = "WWWAUTP"
  DATABASE = "SQLDBA"
%}
%SQL{
  update $(ow).$(tc)
  set phoneno='$(phone)'  where empno='$(empnum)'
%}
%HTML_REPORT{
<TITLE>Employee Telephone Update</TITLE>
<P>
<H1>Employee Telephone Update Result</H1>
<P>
%EXEC_SQL
<P>
Your information has been updated.
To review your <strong>updated</strong>
information, simply run another query.
<P>
<hr>
%}
```

Figure 43. EMPCHG1 D2W Code

In Figure 43, ow (table owner), tc (table name) and empnum (employee number) are hidden variables on the form while phone variable was provided in a regular entry zone.

4.6.1.2 Summary of Flow Control for a Simple Application

1. The Web browser sends a request to the server, naming the CGI program to be executed, the DB2 macro file to be used, and the section of the macro file (input or report).
2. The Web server receives the request, finds the CGI program, and starts it. User environment data is passed to the CGI program.
3. The CGI program reads the macro file, opens a connection to the database, and executes SQL specified in the macro file. Results are prepared according to the format in the macro, and then written to stdout by the CGI program. The CGI program also performs standard processing, such as specifying the content type to be sent to the browser.
4. The Web server sends the result as an HTML document to the browser.
5. The browser formats and displays the results to the user, as shown in Figure 38 on page 89.

4.6.1.3 Application Development Overview

Having reviewed how DB2 WWW applications work, we are in a position to describe the application development process. Briefly, the steps involved in writing a DB2 WWW application are:

- Designing and building DB2 tables for the application if they do not already exist. There is nothing unique to DB2 WWW for this step, and you should design and build tables in the same way as you would for any other application using DB2 as the database.
- Setting up the appropriate access authority to the database, and binding the appropriate plans.
- Writing SQL to retrieve, insert, and update data in the database tables. This step is not significantly different than writing SQL for any other application.
- Designing the application presentation in HTML, for both user input and reporting.
- Combining the SQL and HTML into DB2 WWW macro files, making them available to the gateway program, and testing them.
- Making the application available by linking it to existing HTML pages.

The reference manual for DB2 World Wide Web Connection Version 1 is available on your copy of the VM Web CD: `Db2/Db2www/db2wdoc.htm`

4.6.1.4 DB2 Security and Access Control

One of the attractions of the Web, based on the Internet, is that it vastly extends the reach of your applications. If you decide to deploy an application on the Internet, anyone with access to the Internet and a Web browser could potentially use your application. Even if you restrict deployment to an intranet, the number of users is potentially much higher than for business applications using other interfaces (such as 3270).

You therefore need to think carefully about the security and integrity of your data, and design a protection scheme that meets your needs. A major advantage of using VM/ESA for your Web server is that it already has strong access controls that have proved extremely difficult to defeat.

You will probably want to make some of your data generally accessible for anyone who wants to see it. Examples would be “advertising data,” such as interest rates for different products of banks, or lists of products with prices and availability for distribution and retailing companies. For generally accessible data, you will need to make sure that the database tables are accessible to all, and especially to the user ID coded using the `LOGIN` DB2 WWW variable in the DB2 WWW macro configuration file. It needs at least to be explicitly granted CONNECT authority. We accomplished this by binding a plan to the appropriate database, and then issuing the following DB2 command:

```
GRANT CONNECT TO user ID IDENTIFIED BY password
```

Then, of course, you have to set authorities on tables you intend to expose.

For some databases, or some actions, you will want to restrict access with an online identification. For a further discussion of security see Chapter 5, “Security Issues” on page 131.

4.6.1.5 User Presentation

Like any other application, you will need to give some thought to how you present the application to the user and you must decide how that application interacts with the user. Some people have compared HTML conceptually to 3270, in that you describe both presentation and content and send it to a display agent to present to the user.

But compare the 3270 screen with modern displays on the workstations and you begin to see the potential of using a Web browser together with HTML to access the corporate business information on enterprise servers and presenting it to the user. As the Internet world, including HTML and the World Wide Web, is evolving continuously, more possibilities will continue to become available. You may already use formatted text with colors, still or animated images, sound and video to give the most powerful and clear presentation of your data.

You will need to prepare how the application interacts with the user:

- Some sort of entry point for the application this could be a traditional style menu, a set of push-button icons, an image map, or any other HTML construct you wish to use. This would be like a "main menu" to launch the application. You would prepare this in the same way as you prepare any other HTML file, and can be one file or a group of files.
- User input forms if the initial query will be constructed using input from the user, you will need to prepare input forms in HTML. These will be included in DB2 WWW macro files.

  However, you should test them first to ensure they work correctly outside DB2 WWW. You can do your testing in the following way:

  1. Test the look of the form by simply displaying it in a Web browser. Most browsers have a facility such as "Open File" or "Open File in Browser," which you can use to test the look of an HTML page without using a network connection. You should make it a practice to test the look of HTML pages this way as you are developing them. Also, remember that the resolution you use on your screen may be different from the one that your customers have and this could make the page look very different from what you planned. You should also check that the page is correctly displayed by the most important browsers for your users. Each browser behaves differently, and we have made it a practice to look at each page with Netscape, IBM Web Explorer, and Microsoft Internet Explorer at a minimum.
  2. Test the way the form works, including the variables it passes to the server, by connecting it to a simple CGI program that takes the input and returns an HTML page listing "variable=value."

- Report pages

  You need to decide how to present the result to the user. Since DB2 is tabular, the simplest approach is to insert the data into an HTML 3.0 table. Some browsers still do not support these tables, but the number of these back-level browsers is diminishing rapidly, so you can use tables with the confidence that most of the audience will see the output correctly. (However, also see the discussion in 6.3.2.1, "Fast Rendering by Avoiding HTML Tables" on page 161.)

  It is possible, however, to use any other report format that seems appropriate. You are limited only by your imagination. Bear in mind that HTML is evolving rapidly, and the more advanced features you use, the more you restrict the audience that can view the output directly. As with any technology decision, decide what you want to achieve before deciding which HTML markup to use.

- Consider use of JavaScript

  HTML is a powerful formatting language used to write hypermedia documents for the World Wide Web. It is a subset of the Standard Generalized Markup Language (SGML), so anyone who has used any markup language before (such as SCRIPT or GML) to create documents will find it easy to use and adapt to.

  Although HTML is a powerful formatting language, it still has some limitations for presenting application information. JavaScript is a compact, object-based programming language for developing client and server Internet applications. It lets you add a dynamic nature to Web pages. Internet Web browsers that understand JavaScript, for example Netscape Navigator, interpret JavaScript statements embedded in an HTML page. With this interpreted language you will be able to tightly control all browser events, but be careful about what JavaScript constructs you use, as each browser that supports it uses a different implementation with potential syntactic differences. (Also see the discussion in 5.5.1, "JavaScript" on page 137.)

  You will find "JavaScript Resources" at these Netscape and Yahoo Web sites:

  ```
  http://www.netscape.com
  http://dir.yahoo.com/Computers_and_Internet/Programming_Languages/JavaScript
  ```

4.6.1.6 VM Web CD DB2 Sample Applications

We assume that the DB2 World Wide Web Connection Version 1 or DB2 WWW demonstration package is already installed on your Web server. You can get the demonstration package from either:

- Internet URL `http://www.software.ibm.com/data/db2/www`
- VM Web CD file /db2/db2www/db2www.vma
- The loaded VM Web CD files in the SFS directory VMWEBCD.WEBSHARE.DB2WWW directory

The CD also contains the installation guide in file /db2/db2www/install.htm. Follow those instructions if DB2WWW is not already installed on your system.

You also need to move some DB2WWW files to VM:

1. Access the VMWEBCD.WEBSHARE.DB2WWW directory.
2. Copy all D2W files to where you decided to put the demonstration package files (the default is the D2W server subdirectory).
3. Copy the EXEC file on a server-accessed disk or directory.

Since coding the macro files is relatively simple, you can follow the demonstration applications on the VM Web CD to look at the macro source. See the /db2/db2ind.html file for directions to the demonstrations and source code.

The demonstration macros require that you configure two SQL authorization IDs:

- DB2TS to query tables
- DB2AUT to change EMPLOYEE table

The table used by the sample macros is the EMPLOYEE table delivered with the DB2/VM product. It is a DB2/VM sample page, so it should already be installed. You can find its description in *DB2 for VM V5R1 Interactive SQL Guide and Reference*, SC09-2409.

If your database name is not SQLDBA, you will have to change the DATABASE variable in the heading of DB2 WWW macros in the variable ow.

To grant appropriate authorities, the database administrator must install the DB2WWW package and then issue the following DB2 commands:

```
GRANT CONNECT TO WWWTS  IDENTIFIED BY WWWTST
GRANT CONNECT TO WWWAUT IDENTIFIED BY WWWAUTP
GRANT UPDATE  ON SQLDBA.EMPLOYEE TO WWWAUT
```

WWWTS is the “query user” of the application. Since SQLDBA.EMPLOYEE is a PUBLIC table, it just needs the authority to connect to the database to be granted in order to read it.

WWWAUT is the SQL user authorized to modify the table and is only used in DB2 WWW macros updating the employee table.

The db2indx.html file points to five examples. Each example is a refinement in the design of a small application modifying telephone number fields of the database table.

- The first example is shown in Figure 38 on page 89.
- The second link lets you enter an employee number, change the telephone number after SQL has retrieved it, and update the corresponding SQL column. It is the example described in Figure 38 on page 89.
- The third example is a modified version of Figure 38 on page 89.

  This form will only display employees of department E.

  It adds the ability to this macro to select (with a button) an employee and then to call the report part of the second example directly. The %ROW part of %SQL_REPORT has been modified as described in Figure 44.

```
%ROW{
  <TR>
    <TD><INPUT TYPE="radio" NAME="employee" VALUE="$(V1)"></TD>
    <TD>$(V1)</TD>
    <TD>$(V2)</TD>
    <TD>$(V3)</TD>
    <TD>$(V4)</TD>
    <TD>$(V5)</TD>
    <TD>$(V6)</TD>
  </TR>
%}
```

Figure 44. DB2 WWW EMPQRY2 D2W Rows

A button appears in front of each employee row. Selecting one of these buttons sets the employee variable to the first column of the SQL extracted data (V1) for this row.

To call the next DB2 WWW macro, we insert a form definition in the %HTML_REPORT part. See Figure 45 on page 99.

```
<FORM METHOD="POST"
      ACTION="/cgi-bin/db2www/EMPQRYC2.d2w/report">
%EXEC_SQL
<P>
<INPUT TYPE="submit"
       VALUE="Change Phone Number of Selected Employee">
</FORM>
%}
```

Figure 45. DB2 WWW EMPQRY2 D2W Form

The form calls the report part of Figure 40 on page 91 with the employee number set in the employee variable. The %HTML_INPUT paragraph is now useless and is deleted.

- The fourth example adds a new JavaScript in the first panel; for now, this form can be sent without any button checked. JavaScript will check that an employee is selected before submitting the form information. This is shown in Figure 46.

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Hide code from non-js browsers
  function validateChoice() {
     var i=""
     for (i in document.Choice.EMPLOYEE) {
        if (document.Choice.EMPLOYEE·i".checked=="1") {
          return true ;
        }
     }
     alert("You have not selected any employee.");
     return false;
  }
  // end hiding -->
</SCRIPT>
```

Figure 46. EMPQRY3 D2W Selection Validation

The JavaScript routine scans all EMPLOYEE buttons in the Choice form. If one of them is checked, the form is sent to the Web server. If no employee button is selected, an error message pops up on the browser and the form transfer is cancelled.

The telephone modification form is also enhanced in order to ask the users for their user ID and password. This information is passed to the routine that updates the SQL table. EMPCHG3 sets the LOGIN and PASSWORD variables from the previous form, suppressing the inline coding. Moreover, EMPCHG3 introduces two new variables:

```
SHOWSQL  = "Yes"
MYLOG    = %EXEC "LOGMODIF $(EMPLOYEE) $(DBLOGIN)"
```

– SHOWSQL is set to "Yes," which asks DB2 WWW to display the SQL command submitted by the macro.

– MYLOG, which contains a dynamic call to a REXX procedure accessible to the Web server. Each reference to MYLOG in HTML text will execute LOGMODIF EXEC with the employee record modified and the SQL user ID that requested the modification as parameters. LOGMODIF displays a message on the Web server console but you may use such a procedure to do whatever you need...except to include output of called procedures in an HTML document.

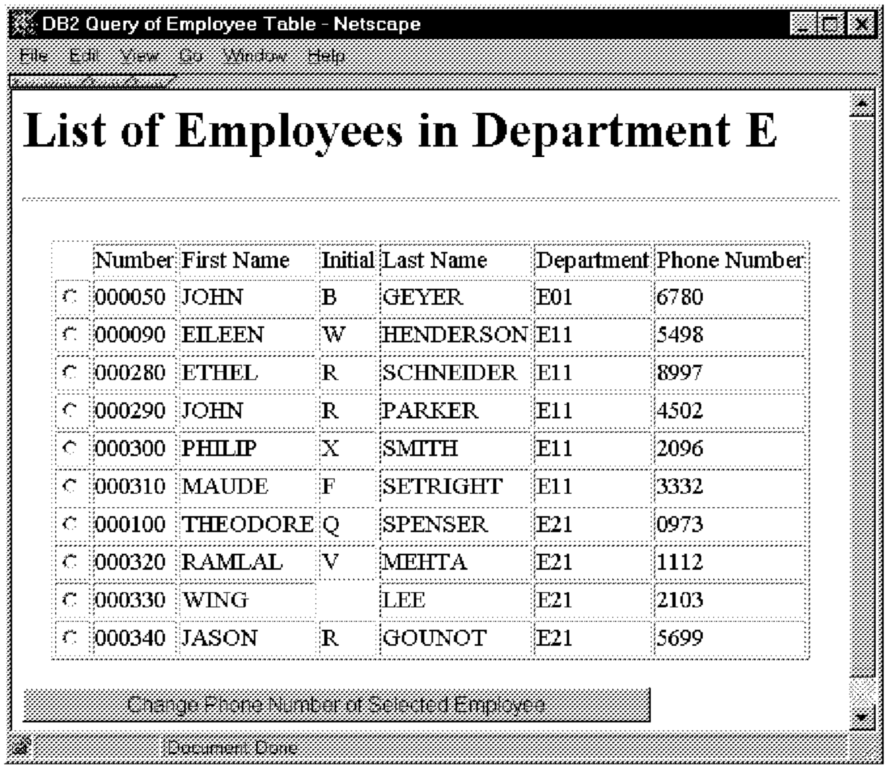
- The fifth version fixes the last two issues in this application:
  - A user might not enter the user ID and password, so SQL update would fail.
  - DB2 WWW needs the user ID and password to be in uppercase.

  The EMPQRYC4 macro contains JavaScript controls to insure that the user ID and password fields are filled and then changes their values to uppercase (Figure 47).

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Hide code from non-js browsers
function validateForm()
{
    formObj = document.Modification;
    if ((formObj.phone.value == "") ||
        (formObj.DBLOGIN.value  == "") ||
        (formObj.DBPASSWORD.value  == "")) {
        alert("You have not filled in all the fields.");
        return false;
    }
    else {
        formObj.DBLOGIN.value = formObj.DBLOGIN.value.toUpperCase() ;
        formObj.DBPASSWORD.value = formObj.DBPASSWORD.value.toUpperCase()
        return true;
    }
}
// end hiding -->
</SCRIPT>
```

Figure 47. DB2 WWW EMPQRYC4 D2W JavaScript

The final application now looks as in Figure 48 on page 101, Figure 49 on page 102, and Figure 50 on page 102.



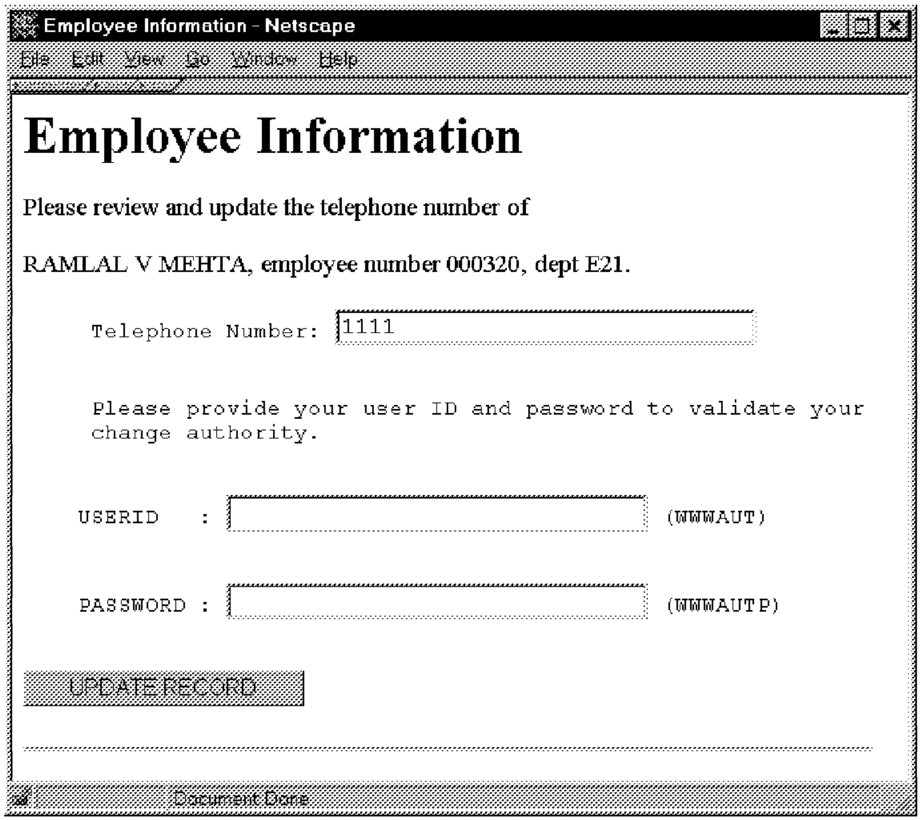
DB2 Query of Employee Table - Netscape

File Edit View Go Window Help

List of Employees in Department E

	Number	First Name	Initial	Last Name	Department	Phone Number
○	000050	JOHN	B	GEYER	E01	6780
○	000090	EILEEN	W	HENDERSON	E11	5498
○	000280	ETHEL	R	SCHNEIDER	E11	8997
○	000290	JOHN	R	PARKER	E11	4502
○	000300	PHILIP	X	SMITH	E11	2096
○	000310	MAUDE	F	SETRIGHT	E11	3332
○	000100	THEODORE	Q	SPENSER	E21	0973
○	000320	RAMLAL	V	MEHTA	E21	1112
○	000330	WING		LEE	E21	2103
○	000340	JASON	R	GOUNOT	E21	5699

Figure 48. Final Application First Page


Employee Information - Netscape
File Edit View Go Window Help

Employee Information

Please review and update the telephone number of

RAMLAL V MEHTA, employee number 000320, dept E21.

Telephone Number: 1111

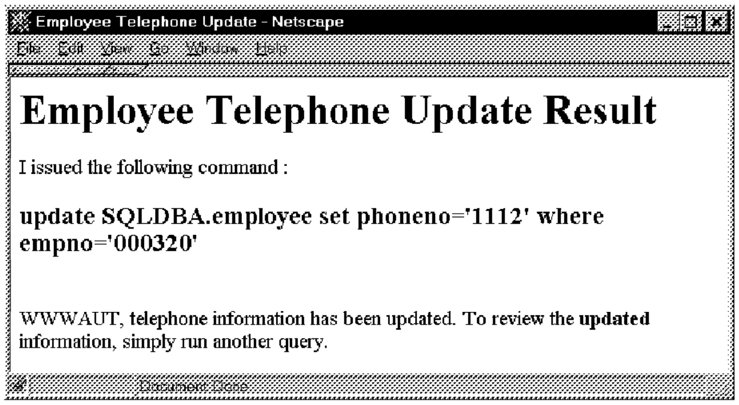
Please provide your user ID and password to validate your change authority.

USERID : (WWWAUT)

PASSWORD : (WWWAUTP)

UPDATE RECORD
Document Done

Figure 49. Final Telephone Update Page


Employee Telephone Update - Netscape
File Edit View Go Window Help

Employee Telephone Update Result

I issued the following command :

update SQLDBA.employee set phoneno='1112' where empno='000320'

WWWAUT, telephone information has been updated. To review the **updated** information, simply run another query.

Figure 50. Final Request Result Page

To access the DB2 WWW sample macros from the browser, we used the following URL:

`http://wtscvmt.itso.ibm.com/~ vmwebcd/db2www/db2indx.html`

4.6.1.7 The Future

Keep an eye on the World Wide Web for updates to DB2 gateway code. Specifically, you should watch `http://www.software.ibm.com/data/db2/www/`.

4.6.2 REXX SQL

As seen in 4.6.1, "DB2 World Wide Web Connection Version 1" on page 86, accessing DB2 tables with DB2 WWW Connection Version 1 does not require new programming knowledge. It is an easy way to extract DB2 data and render it into Web style. But there may still be reasons why you will prefer to write your own DB2 access code, for example:

- You need better control of HTML output.
- You want to create very complicated HTML documents that involve different SQL commands.
- You have skilled REXX SQL programmers.

4.6.2.1 Description of REXX SQL

REXX SQL (RXSQL) is an interface tool that allows REXX programs to access a DB2/VM server using SQL statement packages in embedded RXSQL requests. RXSQL performs some checking on the statements, transforms them into standard runtime SQL operations, and passes them to a database manager. The results of these transactions are returned to your program as REXX variables.

4.6.2.2 Sample of REXX SQL

We will modify the first HTML page of the sample DB2 WWW application shown in Figure 48 on page 101 in order to permit selection of the department on the same Web page. The page will be divided into two parts:

1. The left side will contain the list of all departments found in the SQL table displayed in a selection list. The Web user will select a department and ask a refresh of the employee list display by clicking a button.
2. The right side will display the former screen with the employee update button. The employees list will be controlled by the form on the left side.

The final screen will look like Figure 51 on page 104.

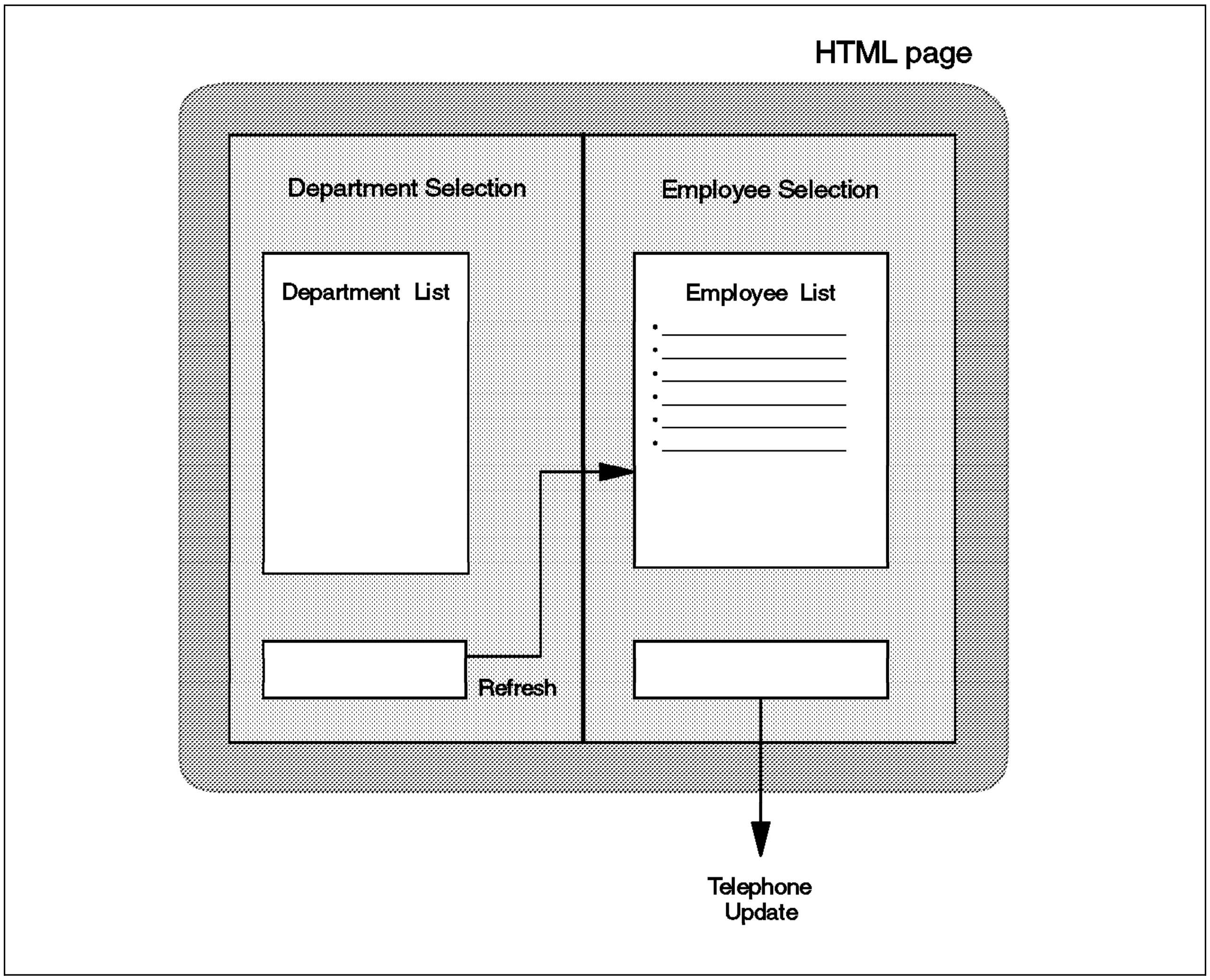


Figure 51. REXX SQL Sample CGI Screen

This modification introduces two difficulties:

1. On the same page, there will be two separated SQL command outputs: The list of the departments and the list of employees in the current department.
2. The list of the departments needs only to be built on the first invocation. When a user asks for a new employee list, we just have to change the right side of the screen. Reading the SQL table for department names would be a waste of resources.

We will address these issues in the sample solution we develop.

Figure 52 on page 105 is the core of our CGI.

```
LOGIN    = "WWWTS"  1
PASSWORD = "WWWTST"
TABLE    = "SQLDBA.EMPLOYEE"
output '<TITLE>RexxSQL Query of Employee Table</TITLE>'

call JScript        2
call GetForm        3

if curdept='_' then title="Complete List of Employees"
  else title = "List of Employees in Department "curdept

output '<h1>'title'</h1>',
       '<hr>',
       '<table>'

call DeptForm       4
call ListHead
call rxs "CONNECT "LOGIN" IDENTIFIED BY "PASSWORD
call rxs "PURGE QNPRET"
call rxs "PREPARE QNPRET SELECT * FROM "TABLE" WHERE WORKDEPT LIKE '"CURDEPT"%'"
call rxs "OPEN QNPRET"
htmlrows=0
/* Main Loop                                                   */
do forever          5
  call rxs "FETCH QNPRET table_row."
  if rc_rxs  =4 then leave                      /* End Of Table */
  output '<TR> ',
    '<TD><INPUT TYPE="radio" NAME="EMPLOYEE" VALUE="'table_row.1'"></TD>',
     '<TD>'table_row.1'</TD>',
     '<TD>'table_row.2'</TD>',
     '<TD>'table_row.3'</TD>',
     '<TD>'table_row.4'</TD>',
     '<TD>'table_row.5'</TD>',
     '<TD>'table_row.6'</TD>',
   '</TR>'
  htmlrows = htmlrows + 1
end
call rxs "CLOSE QNPRET"
call rxs "COMMIT"
call rxs "PURGE QNPRET"
output '</TABLE><BR>'
/* if employees found, then permit modification              */
if htmlrows>0 then ,
  output '<INPUT TYPE="submit"',
         'VALUE="Change Phone Number of Selected Employee">'
else output 'Sorry, there is no employee in this department.'
output '</FORM>  ',
       '</CENTER>',
       '</TD>    ',
       '</TABLE> ',
       '<P>'
exit
```

Figure 52. REXX SQL Sample Main Procedure

It contains the following:

1 The "global" variables settings.

2 The JavaScript control of employee list buttons. The procedure that writes the JavaScript code to the Web server is in Figure 53 on page 106.

```
JScript:
  output '<SCRIPT LANGUAGE="JavaScript">                          ',
         '<!-- Hide code from non-js browsers'
  output '  function validateChoice() {                           ',
         '     var i="";                                          ',
         '     for (i in document.Choice.EMPLOYEE)                ',
         '        if (document.Choice.EMPLOYEE·i“.checked=="1")',
         '          return true ;                                 ',
         '     alert("You have not selected any employee.");      ',
         '     return false;                                      ',
         '  }'
  output '  // end hiding -->                                     ',
         '</SCRIPT>'
return
```

Figure 53. JavaScript Controls

There are three output commands to send the text to the output filter.

When a line is written to the HTML output stream, the Web server automatically adds the "line feed" control sequence at the end of the line before sending it to the Web browser.

As HTML does not interpret line end sequences for formatting purposes, it is possible to send an HTML document in just one line.

But when JavaScript code is embedded into an HTML document, the JavaScript functions have to begin on a new line, as does the end comment. This is the reason why there are three "output" commands.

Of course, in a real programming environment, you may prefer to write JavaScript procedures in single files in order to peek at them with a disk reading filter.

3 The GetForm procedure (Figure 54 on page 107) reads the input stream, decodes it, and sets the passed variables.

```
/* Read the CGI parameters and update corresponding variables */
GetForm:
  'CALLPIPE (name GetForm)',                    /* read parms  */
    ' *:',
    '| xlate 1-* 05 40',
    '| strip',
    '| locate 1',
    '| xlate fieldsep = f1 upper',
    '| change //=PARM./',
    '| console',
    '| varload'
  if SYMBOL('PARM.SELDEPT')='VAR' then do     /* assigned var ? */
    /* Yes : user requested a new employee list               */
    CURDEPT=PARM.SELDEPT   /* displayed department            */
    depttag=PARM.HIDNDEPT  /* HTML code for department change   */
  end
  else do
    /* No  : this is the first CGI call                       */
    CURDEPT='_'            /* display all employees           */
    call List_dept         /* build department selection code  */
  end
return
```

Figure 54. Read Inputs

This makes the difference between the initial call and a user-initiated call to change the displayed department. When a Web user requests a new department, the CGI is called with two variables:

- SELDEPT, which is the selected department to display
- HIDNDEPT, which contains the HTML code of the department selection

At the first invocation, the CGI is called without any parameter so that the procedure sets the current department variable in order to display all department employees, and calls List_dept to build the HTML code controlling department selections as shown in Figure 55 on page 108.

```
    List_dept:
      call rxs "CONNECT "LOGIN" IDENTIFIED BY "PASSWORD

      call rxs "PREPARE LSTDEP SELECT WORKDEPT ",
                       "FROM "TABLE
      call rxs "OPEN LSTDEP"
      depnb=0
      do forever
        call rxs "FETCH LSTDEP dep_row."
        if rc_rxs  =4 then leave                       /* End Of Table   */
        depnb = depnb + 1
        dep.depnb = translate(left(strip(dep_row.1),1))/* 1st char.      */
      end
      dep.0 = depnb
      "CALLPIPE stem dep. | SORT UNIQUE | stem dep."   /* build the list */
      depttag = '<SELECT name="SELDEPT" Size='dep.0+1'>',
                '  <OPTION VALUE=_ SELECTED>All Employees'
      do i=1 to dep.0
        depttag = depttag || '<OPTION VALUE='dep.i'>Dept 'dep.i
      end
      depttag = depttag||'</SELECT>'
      call rxs "CLOSE LSTDEP"
      call rxs "COMMIT"
      call rxs "PURGE LSTDEP"
    return
```

Figure 55. Get the Department List

The List_dept routine reads the SQL table, extracts the department names, and puts them into a selection list. The selection of the “All Employees” list is set as the default selection to ensure that there will always be one selected item: in a selection list, once one item as been selected, you cannot deselect it by clicking again above it. This tip avoids creating a new JavaScript control that would ensure the selection of an item in the selection.

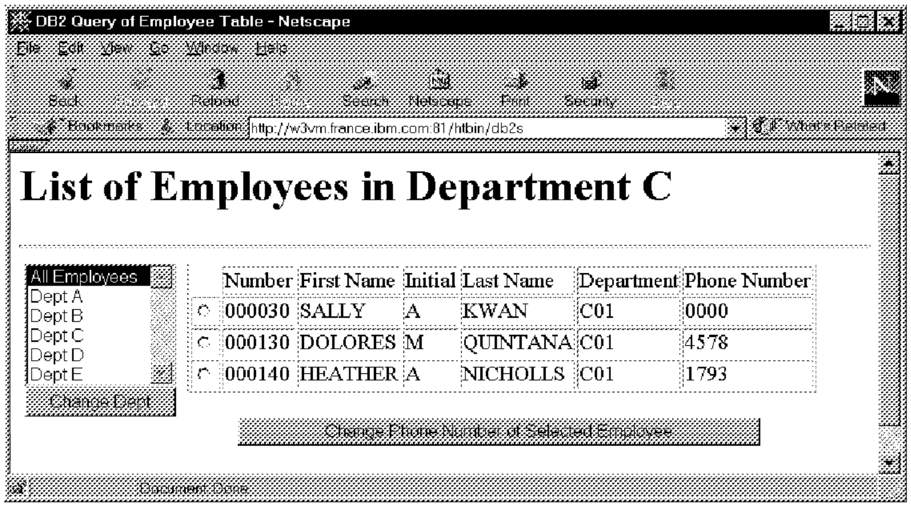
4 DeptForm (Figure 56) builds the department selection form and imbeds the previously set depttag variable containing the selection list.

```
    DeptForm:
      output '<TD VALIGN="TOP">                                      ',
             '<FORM METHOD="POST" name="ChgDpt"                      ',
             '      ACTION="rxsquery">                               ',
             '  <INPUT TYPE="hidden"                                 ',
             '         NAME="HIDNDEPT" VALUE='''depttag'''>          ',
                depttag'<BR>                                         ',
             '  <INPUT TYPE="submit" VALUE="Change Dept">            ',
             '</FORM>                                                ',
             '</TD>'
    return
```

Figure 56. Department Selection HTML Code

5 The main loop that reads employee items and fills the HTML rows.

The first HTML page now appears as in Figure 52 on page 105.


DB2 Query of Employee Table - Netscape

http://w3vm.france.ibm.com:81/htbin/db2s

List of Employees in Department C

	Number	First Name	Initial	Last Name	Department	Phone Number
○	000030	SALLY	A	KWAN	C01	0000
○	000130	DOLORES	M	QUINTANA	C01	4578
○	000140	HEATHER	A	NICHOLLS	C01	1793

Figure 57. Sample RXSQL CGI Output

4.6.2.3 VM Web CD DB2 Sample Application

If you have already installed DB2WWW, you can run the sample directly. If you have not, you need to ensure that your server has the appropriate access to the DB2 server:

- DB2 machine access is given by coding the following commands in the server PROFILE EXEC:

```
"ACCESS VMSYS:SQLMACH.SQL.PRODUCTION. R"           1
"SET LANGUAGE ( USER ADD ARI"                      2
"EXEC SQLINIT DBNAME(SQLDBA)"                      3
```

  1 Access the SQL code disk
  2 Associate the SQL message repository
  3 Prepare the SQL connection to data base SQLDBA

- Grant authorities to the WWWTS DB2 user ID:

```
GRANT CONNECT TO WWWTS  IDENTIFIED BY WWWTST
```

The table used by the sample CGI is the EMPLOYEE table delivered with the DB2/VM product. It is a DB2/VM sample page, so it should already be installed. You can find its description in *DB2 for VM V5R1 Interactive SQL Guide and Reference*, SC09-2409.

If DB2WWW samples are not installed, you cannot run the employee modification form.

To access the REXX SQL sample CGI from the browser, we used the following URL:

```
http://wtscvmt.itso.ibm.com/~ vmwebcd/rxsql/cgi/rxsquery
```

4.6.3 DRDA - VSE Guest Sharing

DB2/VM is part of the supporting databases that implemented the Distributed Relational Database Architecture (DRDA). DRDA is a transparent way for applications to access remote databases. The SQL program uses the local Database Resource Manager to act on remote tables as it does for local tables without any other particular method. The access to remote tables is handled by your local DB2 server using DRDA facilities.

The local server communicates to other relational databases over APPC conversations or TCP/IP networks (only in DB2/VM Version 6).

Once the local database administrator has customized DRDA access to another database, your procedures will be able to use the remote tables just like local tables. The VM database facilities could be used on any other remote DRDA database.

4.7 Office Vision/VM

Two products already exist to access Office Vision/VM (OV/VM) accounts over the Web. They provide a Web interface to OV/VM mail functions.

4.7.1 VM:Webgateway OfficeVision Interface

You will find an overview of VM:Webgateway OfficeVision Interface in *Web Server Solutions for VM/ESA*, SG24-4874.

You can also visit the Sterling Inc. VM Software Division home page for both a description and demonstration of the VM:Webgateway OfficeVision Interface:

`http://www.vm.sterling.com`

4.7.2 EnterpriseWeb Vision

Go to the Beyond Internet site for more information at the following URL:

`http://www.beyond-software.com`

4.8 BookMaster

Refer to 3.5.2, "Processing SCRIPT Files" on page 56 for information and an example of on-demand conversion of BookMaster documents to HTML. The tools used in the example can also be used for conversion of these documents on a one-time basis to make static Web documents from your BookMaster documents.

4.9 MQSeries

MQSeries messaging software enables business applications to exchange information across over twenty different operating system platforms in a way that is straightforward and easy for programmers to implement. The product implementation consists of two portions: The server portion, which takes care of message repositories and management handlers; and the client side, which gives full MQSeries API support and code to communicate with the server part. Implementations may vary according to the platform, depending on what parts are being used. The support included in VM/ESA since Version 2 Release 3 only consists of the MQSeries client part.

MQSeries for VM/ESA is a client that runs on VM′s CMS component. Because it is a client and not a full-function queue manager, the VM client will need to be connected to a server running on another platform (OS/390, MVS/ESA, AIX, OS/2, UNIX, Windows NT).

4.9.1.1 Terms

Some terms need to be defined before going further:

Message	A byte string including application data and MQSeries control data. The MQSeries control data is controlled by API calls but data format is left free to programmers. Except for the maximum length of a message, MQSeries imposes no restrictions on the content or the format of the data part of a message.
Queue	The message repository. It is managed by a queue manager and is identified by a name. It is not a stack. Messages are processed as FIFO queues with priority settings.
Queue Manager	A task running under the MQSeries server. It can manage several queues, and you can implement several in one server.
MQSeries channel	A logical link between the MQSeries server and MQSeries client.

4.9.1.2 VM Implementation

MQSeries VM client code resides on the MAINT 193 minidisk. It includes:

AMQTEXT TXTLIB	Code shared by all languages
AMQTEXTC TXTLIB	C interface
AMQTEXTL TXTLIB	COBOL and PL/I interface
AMQTEXTA TXTLIB	Assembler interface
RXMQV MODULE	REXX interface
CMQC H	C header file
AMQOM MACLIB	COBOL, PL/I and assembler includes

You will also find some samples for each language on the MAINT 193 disk. These samples and the way to compile them are described in Appendix J of *VM/ESA CMS Application Development Guide*, SC24-5761.

APAR

Before testing any MQSeries VM client procedure, check first that APAR VM61665 is applied on your system. You can check required VM product levels in *MQSeries Clients*, GC33-1632.

Using MQSeries API from REXX programs involves three considerations:

1. Setting up the VM environment:
   - You need to access the following disks:

  - MAINT 193 or the disk where MQSeries client code resides
  - Language Environment/370 libraries disk
  - TCPIP disk (592) (must be accessed after the MQSeries code and LE/370 libraries)

- Identify the needed libraries:

  ```
  GLOBAL LOADLIB SCEERUN AMQLLIB
  ```

- Ask CMS to return GETMAIN storage at the end of commands:

  ```
  SET STORECLR ENDCMD
  ```

- Load the RXMQV interface as a nucleus extension:

  ```
  NUCXLOAD RXMQV ( SYSTEM
  ```

2. Setting up the MQSeries environment:

   MQSeries uses CMS GLOBALV facilities to handle MQSeries network interface parameters. The following variables must be declared in GLOBALV:

MQSERVER	Defines the MQSeries interface channel. It indicates: • Server MQSeries channel name • Protocol to access the MQSeries server (IP or APPC) • Server IP address (number or name) • Server port (only needed if you have to connect a port other than the default one, which is 1414)
MQ_User_ID	MQSeries user ID set by the MQSeries server administrator. Only needed when authorization is required on the server side.
MQ_PASSWORD	Password belonging to the MQ_User_ID. Only needed when authorization is required on the server side.
MQCHLTAB	Indicates a file that lists all available MQSeries channels. It is an alternative method to define channels: It is possible to create a channel table on a server system that includes definitions for both the client and server MQI channels, as well as for other channels. This file would then be shipped to a VM identified through the MQCHLTAB variable. The channel table definitions would then be used instead of the MQSERVER definition.
MQCCSID	Character set to use.
MQTRACE	CMS file name in which trace entries have to be written (format is fn.ft.fm).

   If you need to change the MQSeries interface or server in a procedure, you need to change those parameters because they are unique. This means a program cannot work with two different channels or servers at the same time. It must terminate the running interface before connecting to the new one. This limitation can be worked around in a number of different ways because you can still get to multiple queue managers on a single host, and if remote queues (queues belonging to other queue managers, whether they are on the same node of the network or on a remote node) are defined on that particular host, then you can get to those as well.

   Another exposure of the use of GLOBALV is running two MQSeries CGIs at the same time in a multitasking server. (See 5.6.6, “Reentrant and Serially Reusable Resources and CGIs” on page 144 for a full discussion of this subject.) If you run VM:Webgateway you can avoid such a situation by using use the worker machine environment for your CGI.

3. Writing and running the procedure itself (that is, using the API functions).

   These functions are described in *MQSeries: Technical Reference*, SC33-0850 and *MQSeries: Application Programming Reference*, SC33-1673.

   The most used functions are:

INIT	Initializes the REXX interface code
CONNECT	Connects to the remote queue manager
OPEN	Opens the remote queue
PUT	Puts a message in the remote queue
GET	Gets a message from the remote queue
CLOSE	Closes the remote queue
DISCONNECT	Disconnects from the remote queue manager
TERMINATE	Ends the REXX interface calls

In order to run the sample programs or to use your own procedures, you have to see your local MQSeries server administrator to get the following information related to a TCP/IP connection:

Table 7. MQSeries Client TCP/IP Connection Basic Parameters. Parameters needed to write and run an MQSeries application using a TCP MQSeries interface.

Information	**GLOBALV variable**	**API call**	**Set by administrator**
Server address	MQSERVER		
Server port number	MQSERVER		Yes
Channel interface name	MQSERVER		Yes
Queue manager name		Yes	Yes
Queue name		Yes	Yes
MQ_User_ID		Yes	
MQSeries password	MQ_PASSWORD		Yes

4.9.1.3 Possible Use of MQSeries VM Client

MQSeries VM Client allows you to connect to any other MQSeries application that is running on a host that supports MQSeries client API, or is accessible through a host that supports clients. The application you dialog with does not have to know it dialogs with a VM program, it just has to exchange MQSeries messages through message queueing facilities.

You can use MQSeries VM Client in the following ways:

- Put a message to be processed later by another application
- Put a message and scan an output queue to get the result
- Establish a dialog using MQSeries queues
- Browse a queue dynamically modified by a remote application

The most current implementation is the “put and get” queue application. It offers good flexibility. The only matter to decide is the message format and protocol. You do not even have to know what the other application is that you talk to. The message flow is explained in Figure 58 on page 114.

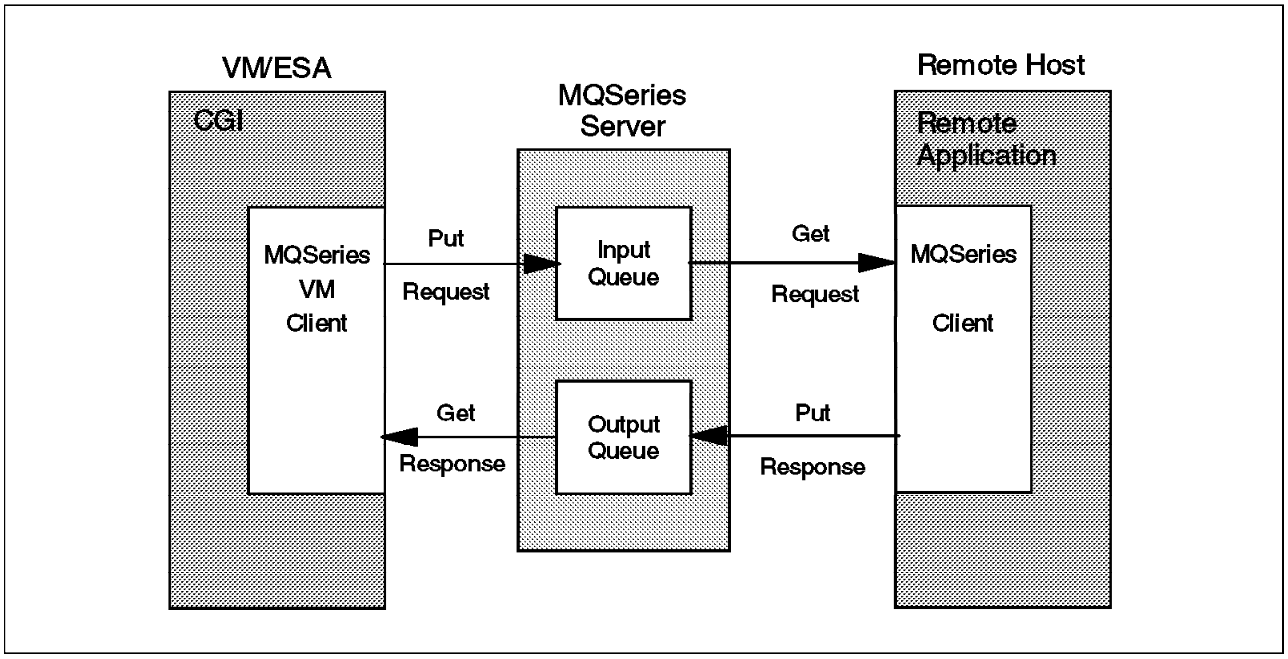


Figure 58. Message Exchange between Two MQSeries Applications

MQSeries bridges are available to help interface with other platform applications. The MQSeries Client sends an MQSeries message that will be processed by the bridge, locally contacting the application and getting back the application results to an MQSeries output queue. For example, you may use the MQSeries/CICS bridge or the MQSeries/IMS bridge to access OS/390 (MVS) and VSE data. This allows you to:

- Access the data where it is.
- Retrieve data with a regular access method.
- Need only light interface development using a simple MQSeries API.
- Have the opportunity to use VM for easy and fast development upon OS/390 (MVS) and VSE large repositories of data. Take the advantages each system offers.

The application you communicate with through MQSeries can also be a VM application. This application might be local to your system, or to another system. This is achieved by having both your code and the VM application make use of message queues hosted on an MQSeries server. Your application is a client placing requests on a queue. The VM application you are communicating with is a client taking requests off a queue. This is illustrated in Figure 58.

4.9.1.4 Put a Message in a Queue

The first step in this communication is to know how to post requests. Figure 59 on page 115 shows you how to put a message in an MQSeries queue.

```
/* MQSMPUT  CGI  : PUT A MESSAGE IN MQSERIES QUEUE           */
  mqserver=translate("MVS")
  Output '<HTML><TITLE>MQSeries Put</TITLE>'
  call GetForm
  call mqput mqserver PutMsg
  rcc7=word(RXMQV('DISC'),1)
  rcc8=word(RXMQV('TERM'),1)
  Output '</HTML>'
exit
/* Read the CGI parameters and update corresponding variables */
GetForm:
  'CALLPIPE (name GetForm)',                  /* read parms  */
    ' *:',
    '| xlate 1-* 05 40',
    '| strip',
    '| locate 1',
    '| xlate fieldsep = f1 upper',
    '| change //=PARM./',
    '| varload'
  PutMsg=PARM.PUTMSG
return
mqput:
  parse arg server themessage
  server=translate(server)

  select                                                      1
     when server="MVS" then do
       mqserver = "9.12.14.227"
       mqport   = "1414"
       mquser   = "MQS2"
       mqchan   = "MQICLIEN"
       qm       = "CSQ2"
       qn       = "SYSTEM.DEFAULT.LOCAL.QUEUE"
     end
     otherwise exit 8
  end
```

Figure 59 (Part 1 of 2). MQSMPUT CGI

```
    address command "GLOBALV SELECT CENV PURGE"
    address command "GLOBALV SELECT CENV SET   MQSERVER "||,    2
                    mqchan"/TCP/"mqserver"("mqport")"
    address command "GLOBALV SELECT CENV SET   MQ_USER_ID "mquser
    Output " Queue manager: "qm"      Queue : "qn"<BR>"
    rcc1=rxmqv('INIT')                                        3
    if ( word(rcc1,1) <> 0 ) then do
      Output "Rc Initializing MQSeries interface "rcc1
      return ; end
    rcc2=rxmqv('CONN',qm)                                     4
    if ( word(rcc2,1) <> 0 ) then do
      Output "Rc Connecting to MQI: "rcc2
      return ; end
    oo = mqoo_output+mqoo_fail_if_quiescing                   5
    rcc3=rxmqv('OPEN',qn,oo,'hqn','ood.')
    if ( word(rcc3,1) <> 0 ) then do
      Output "Rc Opening queue" rcc3
      return ; end
    d.0       = length(themessage)                            6
    d.1       = themessage
    imd.PER = MQPER_PERSISTENT
    imd.FORM = MQFMT_STRING
    imd.MSGTYPE = MQMT_DATAGRAM
    imd.REPLYTOQ = ' '
    imd.REPLYTOQMGR = ' '
    imd.MSGID = MQMI_NONE
    imd.CORRELID = MQCI_NONE
    ipmo.opt = MQPMO_NO_SYNCPOINT
    rcc5 = rxmqv('PUT', hqn,'d.','imd.','omd.','ipmo.','opmo.')
    if (word(rcc5,1) <> 0) then Output "Rc Error sending message "rcc3
    else ,
      Output "The message:<PRE>"themessage"</PRE> was successfully",
             "put in queue "qn"."                             7
    rcc6=word(RXMQV('CLOSE', hqn, MQCO_NONE ),1)
    drop hqn d. imd. omd. ipmo. opmo.
  return
```

Figure 59 (Part 2 of 2). MQSMPUT CGI

The MQSMPUT CGI is called from an HTML form, which sends the message to add on the queue in the variable PutMsg. After retrieving the PutMsg content, the CGI stores it in a queue, as follows:

1 Setting up of the environment. This part of the program contains all the MQSeries parameters that will be used. To run in your MQSeries environment you just have to add your own MQSeries server parameters and change the mqserver variable on the first line of the program.

2 Setting the CMS GLOBALV variables. It is important to note the concatenation string (||). There must be only one space between the name of the variable and the value you assign to it.

3 Initializing the REXX interface. This is a local initialization so it does not need any queue name or queue manager name.

4 Connecting to the queue manager. This is the first time the definitions are used.

5 Before opening the queue, the open option variable (oo) is computed. The different options are described in *MQSeries: Application Programming Reference*, SC33-1673. You can also find a list of them in the CMQC H header file on the MAINT 193 disk.

The OPEN function needs four parameters:

- The queue name that is supposed to be opened
- The open option variable
- The name of the variable that will receive the connection handle

  This handle represents the connection to the message queuing queue manager. It must be specified on all subsequent message queuing calls issued by the application. It ceases to be valid when the MQDISC call is issued.
- The name of the variable that will contain the open object descriptor

6 Until the queue is available, the CGI prepares to put the message in the queue by copying it into a stem variable and preparing the PUT parameters, which are:

- The connection handle
- The stem containing the variable (index 1) and its length (index 0)
- The stem to use as input message descriptor
- The stem to write the final message descriptor used
- The stem to use as options to the PUT command
- The stem to write the final options used on the PUT command

7 The queue is closed with a successful return code.

4.9.1.5 Get a Message from the Queue

Once you send an order message in a queue, you will need to read the response. Figure 60 on page 118 shows you how to read a queue.

```
/* MQSMGET  CGI  : Get messages from MQSeries Queue          */
server = translate("MVS")
wtime  = 2
output '<HTML><TITLE>MQSeries Put</TITLE>'
Output "<HR>Queue content : <BR><PRE>"
finalrc=4
select
   when server="MVS" then do
     ...
   end
   otherwise exit 16
end

address command "GLOBALV SELECT CENV PURGE"
address command "GLOBALV SELECT CENV SET   MQSERVER "||,
                mqchan"/TCP/"mqserver"("mqport")"
address command "GLOBALV SELECT CENV SET   MQ_USER_ID "mquser
rcc1 = RXMQV('INIT')
if ( word(rcc1,1) <> 0 ) then do
  Output "Rc Initializing MQSeries interface "rcc1
  signal mqdisc ; end

rcc2 = RXMQV('CONN', qm )
if ( word(rcc2,1) <> 0 ) then do
  Output "Rc Connecting to queue manager "rcc2
  signal MqDisc ; end

oo  = mqoo_input_as_q_def+mqoo_fail_if_quiescing
rcc3 = RXMQV('OPEN', qn, oo, 'hqn', 'ood.' )
if ( word(rcc3,1) <> 0 ) then do
  Output "Rc Opening queue" rcc3
  signal MqDisc ; end

i=0
do forever
   g.0 = 200
   g.1 = ''
   igmo.opt = MQGMO_WAIT + MQGMO_CONVERT          1
   igmo.WAIT =  1000 * wtime
                /* searching for messages                    */
   rcc5 = RXMQV('GET', hqn,'g.','igmd.','ogmd.','igmo.','ogmo.')
   if word(rcc5,1) = 2033 then leave
   if ( word(rcc5,1) <> 0 ) then do
     Output "Rc Reading message "rcc5
     finalrc=8
     leave
   end
   i=i+1
   Output g.1
   finalrc=0
end
Output "</PRE>"

if finalrc>4 then Output "Error occurred while retrieving messages."
if finalrc=4 then Output "Sorry, no messages in queue."
if finalrc=0 then Output i" messages retrieved."
```

Figure 60 (Part 1 of 2). MQSMGET CGI

```
      MqDisc:
        rcc6=word(RXMQV('CLOSE', hqn, MQCO_NONE ),1)
        rcc7=word(RXMQV('DISC'),1)
        rcc8=word(RXMQV('TERM'),1)
        Output "</HTML>"
      exit
```

Figure 60 (Part 2 of 2). MQSMGET CGI

1 The new parameter when reading a queue is the timer indicating how long to wait for a message. In this program we will read all queued messages until no new message arrives after waiting for two seconds.

The MQSMGET CGI reads messages so they are removed from the queue when read. MQSeries allows you to just browse messages using different option parameters. MQSMBRW CGI uses browsing options which changed the message scanning routine as detailed in Figure 61.

```
oo  = mqoo_browse+mqoo_fail_if_quiescing            1
rcc3 = RXMQV('OPEN', qn, oo, 'hqn', 'ood.' )
if ( word(rcc3,1) <> 0 ) then do
  Output "Rc Opening queue" rcc3
  signal mqdisc; end
i=0
do forever
   g.0 = 212
   g.1 = ''                                         2
   igmo.opt = MQGMO_NO_WAIT + MQGMO_BROWSE_NEXT+MQGMO_ACCEPT_TRUNCATED_MSG

                /* searching for messages                           */
   rcc5 = RXMQV('GET', hqn,'g.','igmd.','ogmd.','igmo.','ogmo.')
   if word(rcc5,1) = 2033 then leave
   if ( word(rcc5,1) <> 0 ) then do
     Output "Error Browsing Queue "qn" Return code : "rcc5
     finalrc=8
     leave
   end
   i=i+1
   Output g.1
   if debug="Y" then Say "Read message : "g.1
   finalrc=0
end
```

Figure 61. MQSMBRW CGI Get Section

1 The queue is opened in browse mode.

2 MQGMO_NO_WAIT disables the timer mechanism.

Browsing an MQSeries queue is similar to the “peekto” CMS Pipelines filter command. The message format can be checked by a browse section of the program, which will choose whether to call the processing section (this is the way the MQSeries/CICS bridge works). Another use is to read the message in browse mode, process it, and only delete it from the queue if the process completed successfully.

Here are the most common error messages you might encounter:

2012 MQRC_ENVIRONMENT_ERROR: At connect time, it generally indicates you are not able to connect the MQSeries server. If your GLOBALV variables are correct, you need to check with the MQSeries server administrator if the MQSeries server is up, especially the TCP/IP port listener task. Verify also that your last program disconnected properly from the MQSeries interface.

2042 MQRC_OBJECT_IN_USE: Trying to open the queue means that the queue is not set as shared and another process has locked the queue access after opening it. Queue property can be changed to permit concurrent access to the same queue.

2079 MQRC_TRUNCATED_MSG_ACCEPTED: You tried to read a message larger than the estimated length you expected. Change your application to fit the message lengths.

2085 MQRC_UNKNOWN_OBJECT_NAME: Obvious!

To get the return code name without searching for it in the *MQSeries: Technical Reference*, SC33-0850, you can edit the CMQC H header file and issue a find on the decimal message number value returned by the MQSeries REXX interface. To output this string in a program, you just have to add the following function in your programs:

```
Find_MQ_RC: procedure
  parse arg mqrc .
  msgtxt=""
  "CALLPIPE < CMQC H * ",
         "| LOCATE 10.5 /MQRC_/ ",
         "| LOCATE /"mqrc"L/ ",
         "| specs w2 1 ",
         "| var msgtxt"
return msgtxt
```

4.9.1.6 MQSeries/CICS Bridge

The MQSeries/CICS Bridge enables an application not running in a CICS environment to run a program or transaction on CICS/ESA and get a response back. This non-CICS application can be run from any environment that has access to an MQSeries network that encompasses MQSeries for OS/390 (MVS/ESA).

The MQSeries/CICS Bridge is supplied in a SupportPac you can find at `http://www.software.ibm.com/ts/mqseries/txppacs/ma1e.html` or as a separately installable feature of MQSeries for MVS/ESA Version 1.2.

Documentation for the bridge is provided in the User's Guide in this SupportPac, or in the seventh edition of the *MQSeries for MVS/ESA System Management Guide*, SC33-0806.

A program is a CICS program if it can be invoked using the EXEC CICS LINK command. It must conform to the Distributed Program Link (DPL) subset of the CICS API, that is, it must not use CICS terminal or syncpoint facilities.

A CICS transaction is designed to run on a terminal. This transaction can use BMS or TC commands. It can be conversational or part of a pseudoconversation. It is permitted to issue syncpoints.

4.9.1.7 MQSeries/CICS Bridge Modes

The CICS bridge allows an application to run a single CICS program or a “set” of CICS programs (often referred to as a unit of work). It caters to the application that waits for a response to come back before it runs the next CICS program (synchronous processing), and to the application that requests one or more CICS programs to run, but does not wait for a response (asynchronous processing).

The CICS bridge also allows an application to run a 3270-based CICS transaction, without knowledge of the 3270 data stream.

The CICS bridge uses standard CICS and MQSeries security features and can be configured to authenticate, trust, or ignore the requestor′s user ID.

4.9.1.8 Running CICS DPL Programs

Data necessary to run the program is provided in the MQSeries message. The bridge builds a COMMAREA from this data, and runs the program using EXEC CICS LINK.

Figure 62 shows the step sequence to process a single message to run a CICS DPL program.

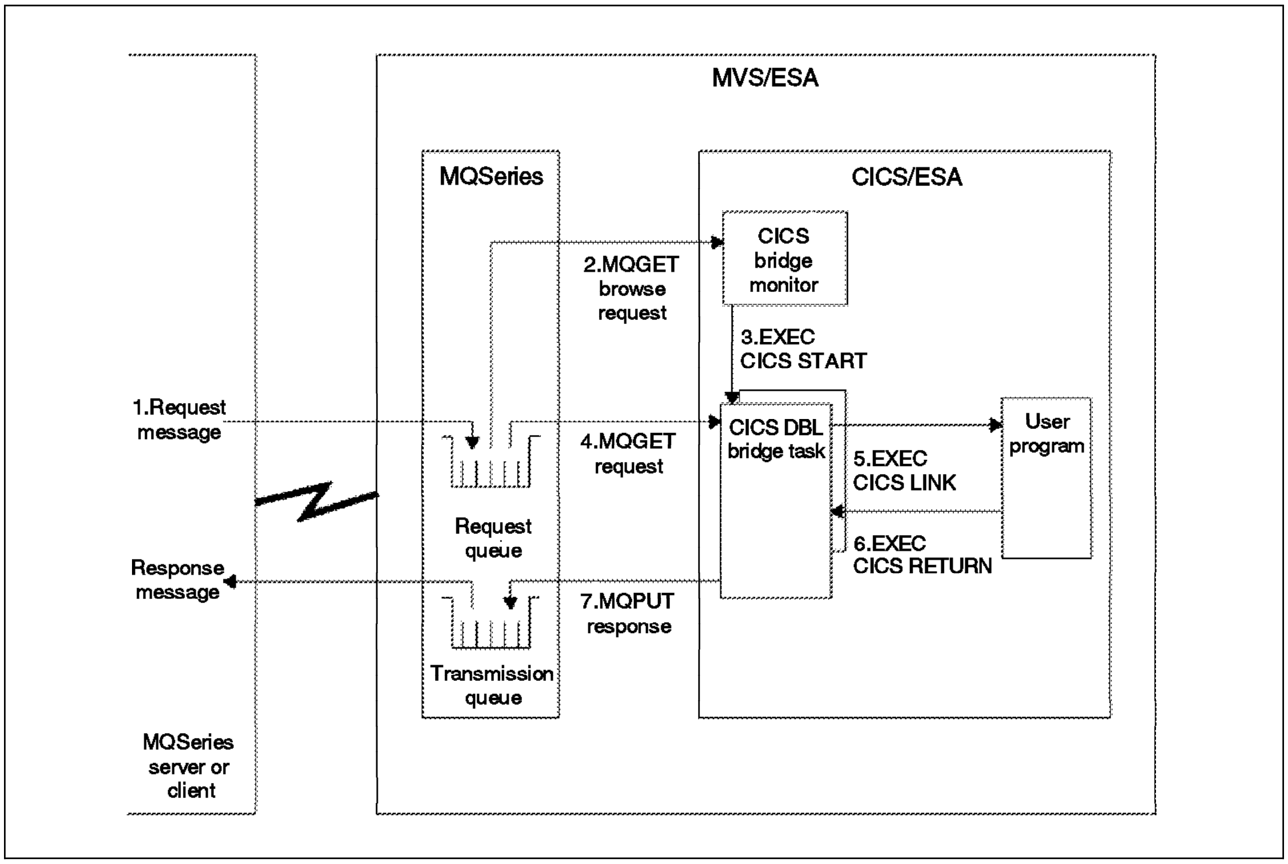


Figure 62. MQSeries/CICS Bridge: Execution of a DPL Program

These steps are:

1. A message, with a request to run a CICS program, is put on the request queue.

2. The CICS bridge monitor task, which is constantly browsing the queue, recognizes that a “start unit of work” message is waiting (CorrelId=MQCI_NEW_SESSION).
3. Relevant authentication checks are made, and a CICS DPL bridge task is started with the appropriate authority.
4. The CICS DPL bridge task removes the message from the request queue.
5. The CICS DPL bridge task builds a COMMAREA from the data in the message and issues an EXEC CICS LINK for the program requested in the message.
6. The program returns the response in the COMMAREA used by the request.
7. The CICS DPL bridge task reads the COMMAREA, creates a message, and puts it on the reply-to queue specified in the request message. All response messages (normal and error, requests and replies) are put on the reply-to queue with default context.
8. The CICS DPL bridge task ends.

In this scenario, a unit of work that is made up of many messages works in the same way, with the exception that the CICS bridge task waits for the next request message in the final step unless it is the last message in the unit of work.

The structure a DPL bridge message must take is:

1. MQMD (MQSeries message descriptor)
2. MQCIH (CICS bridge header), except if you want to run a single DPL program where AUTH is set to LOCAL or IDENTIFY
3. Program name on eight first positions
4. COMMAREA in message data beginning at the ninth position

There is another queue to consider: the dead letter queue. It contains all messages rejected by the CICS bridge monitor.

4.9.1.9 MQSeries CICS DPL CGI Call Sample

See Figure 63 on page 123.

```
/* MQSMCICS CGI  : Run a DPL CICS program and Display the response */
  NEW_SESSION = "414d51214e45575f53455353494f4e5f434f5252454c494400"
  MQCI_NEW_SESSION = x2c(NEW_SESSION)                    1
  ttime=5

  call GetForm
  call mqcon mqsrv
  call mqput PutMsg
  Output "<HR>"
  if rcput=0 then do
     call mqget
     if finalrc=4 then call mqdead
  end
  call mqdisc
exit
mqput:
   parse arg CICSDPL COMMAREA
   OPEN..
   themessage = left(left(CICSDPL,8)||COMMAREA,300) 2
   d.0        = length(themessage)
   d.1        = themessage
   ...                                          3
   imd.RTOQ = qni                                        /* ReplyToQ */
   imd.REPLYTOQMGR = qm
   imd.MSGID       = MQMI_NONE
   imd.CID         = MQCI_NEW_SESSION                    /* CorrelId */
   rcc5 = rxmqv('PUT', hqn,'d.','imd.','omd.','ipmo.','opmo.')
   ...
return
```

Figure 63 (Part 1 of 3). MQSMCICS CGI

```
mqget:
  OPEN...
  zero=time("E")
  wtime=ttime*1000                                       4
  do forever
     g.0 = 1000
     g.1 = ''
     igmo.opt = MQGMO_WAIT + MQGMO_CONVERT
     igmo.WAIT = wtime
     rcc5 = RXMQV('GET', hqn,'g.','igmd.','ogmd.','igmo.','ogmo.')
     if word(rcc5,1) = 2033 then leave
     if ( word(rcc5,1) <> 0 ) then Error message...
     if left(g.1,4)="CIH " then ,                        5
         Output "Error: "translate(substr(g.1,181)," ",x2c("00"))
       else Output "Response: "substr(g.1,9)
     wtime = wtime - format(time("R") * 1000,,0)
     if wtime<=0 then leave
  end
return
```

Figure 63 (Part 2 of 3). MQSMCICS CGI

```
/* Get messages from MQSeries Dead Message Queue     */
mqdead:
  ...
return
```

Figure 63 (Part 3 of 3). MQSMCICS CGI

1 For the bridge to start to process your message, you need to put it on the CICS bridge input queue, but moreover you need to fill the correlation ID message option variable with the MQCI_NEW_SESSION. Otherwise, your messages will not be processed and you will find dead letter messages about them in the dead letter queue.

2 The format of the DPL request needs eight first characters to be the program name. The CGI also forces the request message length to 300 because the CICS response buffer size is the same as the request message length. If your request message is smaller than CICS output, data is truncated without any error message.

3 The message options are:

- RTOQ is the queue where the CICS bridge will put the response message. Unlike the C variable name (ReplyToQ), the REXX interface variable is RTOQ.
- As you needed to indicate the reply queue, you also need to indicate the queue manager it belongs to.
- MSGID is the message identifier and is null in the sample, but in a regular MQSeries application you may give it a value. As a result you will indicate the value of this message ID in the GET message input option, and MQSeries will only scan the messages with the same ID. This is the normal way to share an MQSeries queue: All users send messages to the same queue and read only their own messages (based on MSGID and CorrelID).
- The correlation ID is generally used like MSGID, to restrict the queue scan to only your message. Forcing the user to fill it with the MQCI_NEW_SESSION value permits the CICS bridge to wake up only on its messages. The REXX interface variable for the correlation ID is CID (the C variable is CorrelID).

4 The previous example of the GET method defined a wait interval of five seconds, waited for five seconds for a message and waited again for five seconds for each new message. As a consequence, the total wait time was unpredictable. This sample shows how to set a maximum time interval. The REXX elapsed time facility is used to subtract from a five seconds counter (ttime variable). As the program does not know how many response messages it will receive, it receives all messages during this five-second interval.

Another method to wait for a response is to set a fairly short time-out interval and go through a loop a number of times. This allows the CGI to output progress messages letting the end user know what is happening.

5 Once a message is received, the procedure checks if it is a message prefixed with a CICS header message. If this header is present, it means an error occurred during CICS bridge processing, so the program displays the error message. If no such header is present, the message is a CICS program response.

The CEDF CICS command may be used to trace the transaction flow across the MQSeries bridge. It makes it possible to display the COMMAREA passed to CICS programs. The CICS task ID to trace is CKBP.

4.9.1.10 Running CICS 3270 Transactions

Data necessary to run the transaction is provided in the MQSeries message. The CICS transaction runs as if it has a real 3270 terminal, but instead uses one or more MQ messages to communicate between the CICS transaction and the MQSeries application.

Unlike traditional 3270 emulators, the bridge does not work by replacing the VTAM flows with MQSeries messages. Instead, the message consists of a number of parts called vectors, each of which corresponds to an EXEC CICS request. Therefore, the application is talking directly to the CICS transaction rather than via an emulator, using the actual data used by the transaction (known as application data structures or ADSs). Figure 64 shows the sequence of steps taken to process a single message to run a CICS 3270 transaction.

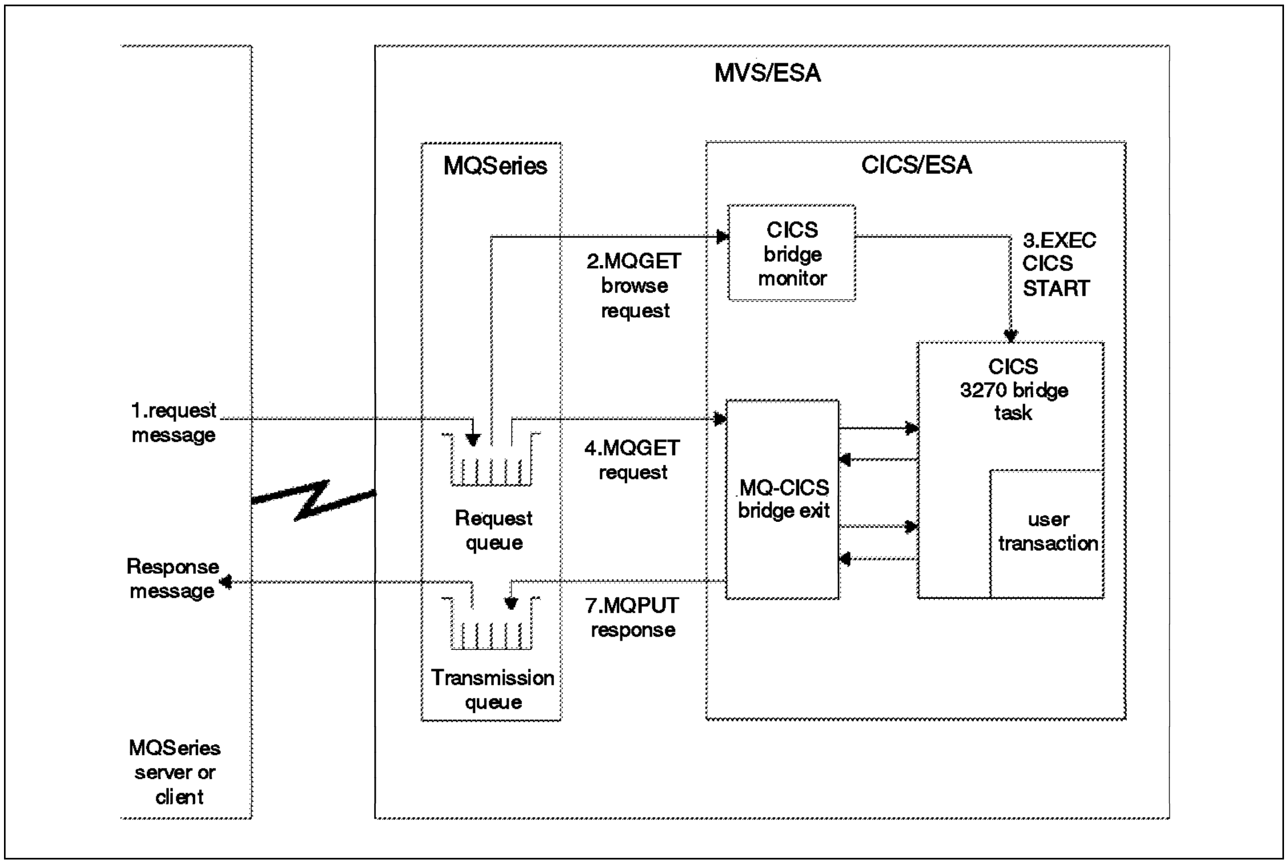


Figure 64. MQSeries/CICS Bridge: Execution of a 3270 Transaction

1. A message with a request to run a CICS program is put on the request queue.
2. The CICS bridge monitor task, which is constantly browsing the queue, recognizes that a `start unit of work` message is waiting (CorrelId=MQCI_NEW_SESSION).

3. Relevant authentication checks are made, and a CICS 3270 bridge task is started with the appropriate authority.
4. The CICS DPL bridge task removes the message from the request queue and changes the task to run a user transaction.
5. Vectors in the message provide data to answer all terminal-related input EXEC CICS requests in the transaction.
6. Terminal-related output EXEC CICS requests result in output vectors being built.
7. The MQ-CICS bridge exit builds all the output vectors into a single message and puts this on the reply-to queue.
8. The CICS 3270 bridge task ends.

4.9.1.11 MQSeries/IMS Bridge

MQSeries for MVS/ESA provides support in the IMS environment for online message processing programs (MPPs), interactive fast path programs (IFPs), and batch message processing programs (BMPs).

The MQSeries-IMS bridge is the component of MQSeries for OS/390 (MVS/ESA) that allows direct access from MQSeries applications to applications on your IMS system. This means that you can reengineer legacy applications that were controlled by 3270-connected terminals to be controlled by MQSeries messages. The bridge is an IMS Open Transaction Manager Access (OTMA) client (see *IMS/ESA V6 OTMA Guide and Reference*, SC26-8743 for more information).

To submit an IMS transaction that uses the bridge, applications put messages on an MQSeries queue as usual. The messages contain IMS transaction data; they can have an IMS header (the MQIIH structure) or allow the MQSeries-IMS bridge to make assumptions about the data in the message.

In VMWEBCD.WEBSHARE.MQSERIES, you can find a sample CGI to call IMS programs: MQSMPIMS CGI. The overall structure is the same as the MQSeries/CICS bridge. The only difference is that the CGI will build an IMS Information Header (IIH), and will extract IMS responses from the IIH messages returned back by the MQSeries/IMS bridge.

Table 8 (Page 1 of 2). IIH Header Content. MQSeries/IMS Bridge IIH header format.

Data type	Length	Name	Description
String	4	StrucId	Structure identifier
Integer	4	Version	Structure version number
Integer	4	StrucLength	Length of MQIIH structure
Integer	4	Encoding	Reserved
Integer	4	CodedCharSetId	Reserved
String	8	Format	Format name
Integer	4	Flags	Reserved
String	8	LTermOverride	Logical terminal override
String	8	MFSMapName	Message format services map name
String	8	ReplyToFormat	Format name of reply message
String	8	Authenticator	RACF password or pass ticket

Table 8 (Page 2 of 2). IIH Header Content. MQSeries/IMS Bridge IIH header format.

Data type	Length	Name	Description
Bytes	16	TranInstanceId	Transaction instance id
Byte	1	TranState	Transaction state
Byte	1	CommitMode	Commit mode
Byte	1	SecurityScope	Security scope
Byte	1	Reserved	Reserved

Documentation for the bridge is provided in:

- *MQSeries for MVS/ESA System Management Guide*, SC33-0806
- *MQSeries: Application Programming Guide*, SC33-0807
- *MQSeries: Technical Reference*, SC33-0850

4.9.1.12 MQSeries Bridge Tools

You will find several tools that help design an MQSeries bridge in the VMWEBCD.WEBSHARE.MQSERIES directory These tools are as follows:

MGET MODULE Read messages and format them in various ways. Its syntax is:

```
MGET <options> qname <qmgr>
where options can be:
-b      = browse queue
-c      = convert data
-d      = assume the message data begins with a MQDLH
          and print the MQDLH header.
-h      = print MQMD header for each msg
-n<xxx> = get xxx messages, default value
          of xxx is 1, default if -n flag
          not specified is 999999 messages
-i<x>   = open options: -iq _INPUT_AS_Q_DEF (default)
                        -is _INPUT_SHARED
                        -ie _INPUT_EXCLUSIVE
-q      = quiet mode, that is, do not print
          messages
-t<xxx> = set buffer size to xxx bytes.
          Default of xxx is 1000.
-u      = translate queue and qmgr to uppercase.
-w<xxx> = set wait interval xxx seconds. Default
          of xxx is 5.
```

MGET C C source of MGET module.

MGET OUTPUT Sample MGET output.

MGETH OUTPUT Shows MQSeries message fields.

PROGTSVM MODULE A command processor to send IMS bridge messages. Look in the C source prolog for a complete description.

SAMPLE DAT An input example to use with PROGTSVM.

PROGTSVM EXEC Sample EXEC to call PROGTSVM. You have to customize it with your server parameters, then call it with the server name and the DAT file you created.

PROGTSVM C C source of PROGTSVM module.

READIMSG MODULE.	Reads messages from the IMS/Bridge output queue and displays the MQSeries fields and IIH structure.
READIMSG OUTPUT	Sample OUTPUT from READIMSG.
READIMSG C	Source file of the READIMSG MODULE.
CMQCBC H	Header file for C compilation. Contains additional structure definitions and message constants not provided in CMQC H VM header file.
CMQ EXEC	Sample REXX procedure to compile C programs (ignore warning messages).

To use these programs, issue the following command:

```
GLOBAL LOADLIB SCEERUN AMQLLIB
```

Original C programs are available on:

```
http://www.software.ibm.com/ts/mqseries/txppacs/ma1c.html
```

In the ma1c package, you will also find the IAPMDI28 IMS program we used to test the CGI.

We also used the sample IMS PART application (delivered with the IMS product). The IMS command PART AN960C10 calls the PART IMS application that will inquire a DB2 table and send back the description of a product.

4.9.1.13 VM Web CD MQSeries Sample Applications

To run the MQSeries samples:

1. Update the PROFILE EXEC of the virtual machine that will run the MQSeries CGI according to MQWEB EXEC in directory VMWEBCD.WEBSHARE.MQSeries.
2. Each CGI contains a sequence like the following:

```
select
   when server="MVS" then do                              1
     mqserver = "x.xx.xx.xxx"
     mqport   = "1414"
     mquser   = "MQU"
     mqchan   = "MQCLIENT"
     qm       = "MYQ"
     qn       = "SYSTEM.DEFAULT.LOCAL.QUEUE"
   end
   otherwise exit 8
end
```

   1 Duplicate the when section according to your own settings and change the server variable at the beginning. You must do this for every CGI. These fields represent the basic information the MQServer administrator should give you.

3. The MQSMBRW HTML file contains a list of queues that can be selected for browsing. Change this list to your local queue names.
4. Bridge sample CGIs, MQSMCICS and MQSMPIMS need three more variables to be customized:
   - qno: the output queue for the program, that is, the bridge input queue
   - qni: the input queue for the program, that is, the bridge output queue

- qnd: the dead letter bridge queue

These parameters are customized by the bridge administrator.

To access the MQSeries sample CGIs from the browser, we used this URL:

```
http://wtscvmt.itso.ibm.com/~ vmwebcd/mqseries/mqmain.html
```

Chapter 5. Security Issues

You are now an expert in Web-enabling VM resources. But have you considered all of the security implications of making your corporate resources accessible via the Web? In this chapter we review some technologies, and then show you how to apply these technologies and related techniques to address these security issues. We also learn that in questions of security, as with questions of performance, there is only one correct answer: "it depends."

We assume that you have already read the security section of *Web Server Solutions for VM/ESA*, SG24-4874, which covers such concepts as the Secure Socket Layer (SSL) protocol and firewalls.

In general, all points and issues raised in this chapter apply equally to all of Webshare, EnterpriseWeb/VM, and VM:Webgateway (although there may be differences in details). We have attempted to flag all cases that are particular to only a subset of these products. We also suggest that you review the security section of the "VM/ESA Web Server Feature Summary" in *Web Server Solutions for VM/ESA*, SG24-4874.

You will find that these issues, and countermeasures, will in some form also be applicable to your non-VM systems. Do not allow our listing of these issues as they relate to VM in any way to lead you to believe that the problems non-VM systems face in this area are any less severe. In our opinion most of these exposures are actually less severe and/or more easily addressed on VM systems.

It is also our opinion that many of the exposures presented in this chapter will only apply to the most secure sites. We are providing you with a pessimistic view that deals with the worst-case scenarios.

5.1 Security Concepts

One of the biggest problems with security is knowing how much is enough. Take the example of a private house. You can imagine a series of increasingly secure features:

- Curtains on the windows to prevent people from seeing in
- Locks on the doors, to stop a thief from walking in
- A big, ugly dog to keep unwanted visitors away
- An alarm system to detect intruders
- An electric fence, mine field and armed guards

Clearly, it is possible to have too much security. In general you should try to aim for an *appropriate* level of security, based on the following three factors:

1. The *threat* (what kind of neighborhood do you live in?)
2. The *value* of what you are protecting (how many Van Gogh's do you have?)
3. The *objective* of your security measures

This last factor is less obvious than the other two, but equally important. To go back to the example of the house: If the objective we are aiming for is "to stop a thief from walking in," the most appropriate security measure may well be the locks on the doors.

In this book we are interested in creating an appropriate level of security that prevents unauthorized people from accessing your Web server or other services or data that might be available on the platform your are running your Web server on.

The *value* of the data we are protecting varies enormously, so we have to be constantly alert to make sure that our security level is appropriate.

The *objectives* of our security measures depend on what type of data we are protecting. It is important to use consistent language for describing these objectives, because the terms can be ambiguous. For example, if we talk about a message being “authentic,” do we mean that we know it has not been changed, or that we know where it came from?

5.2 Types of Attacks

The Internet is home to a variety of criminals who pose threats to the security of World Wide Web communications. They may attempt a number of different types of attacks. For example:

Passive Attacks
In a passive attack the perpetrator simply monitors the traffic being sent to try to learn secrets. Such attacks can be either network based (tracing the communications links) or system based (replacing a system component with a *Trojan Horse* that captures data insidiously). Passive attacks are the most difficult to detect. You should assume that someone is eavesdropping on everything you send across the Internet.

Active Attacks
In these, the attacker is trying to break through your defenses. There are several types of active attack. For example:

- System access attempts, where the attacker aims to exploit security loopholes to gain access and control over a client or server system.
- Spoofing, where the attacker masquerades as a trusted system to try to persuade you to send him secret information.
- Cryptographic attacks, where the attacker attempts to break your passwords or decrypt some of your data.

Denial of Service Attacks
In this case the attacker is not so much trying to learn your secrets as to prevent your operation, by redirecting traffic or bombarding you with junk.

Social Engineering Attacks
One active attack method that has proven highly successful for hackers is popularly know as the *social engineering* technique. This involves persuading someone in an organization to part with sensitive access control information, such as user IDs and passwords.

Several forms of this attack have been recorded, for example:

- Pulling rank: The attacker identifies new recruits in the organization and telephones them, claiming to be a high-ranking official who is out of the office. The targets are so nervous about creating a good impression that they will give out secret information rather than appear to be obstructive.

- One of us: The attacker claims that a genuine systems administrator told him to get in touch and arrange a guest account or some other access. This needs an understanding of the system support departments. By appearing to be just "one of the gang" the attacker can persuade the targets to lower their guard.

Social engineering attacks are the realm of the con artist, rather than the cunning technician. Indeed, anyone could attempt them, given an organization chart and a convincing telephone manner. As loopholes in the software are progressively identified and patched up, you can expect this kind of attack to become more common. The only defense is to put good administrative procedures in place, and to apply them rigidly.

5.3 Five Basic Components of Security

As you learned in *Web Server Solutions for VM/ESA*, SG24-4874, there are five basic components to security. They are so basic to this topic that they are worth repeating and expanding upon here. Note that to some degree each of these components of security often intimately interrelates with the other components.

Information exchanges are secure if all of the following are true:

Authentication
You can identify with certainty both parties in the exchange. Not only was the contract signed, but it was signed by the proper person. The sender knows the receiver is authentic and not someone masquerading as the receiver. Authentication ensures that someone is who they say they are.

Authorization
The parties are actually allowed to share the information. Control of access to the data has not been compromised. The server is allowed to share this data with this client. In addition, when data is being modified on the server, the client is allowed to modify this data. Authorization often relies upon authentication to know what to authorize.

Integrity
The information is not altered as it flows across the network. The message received is the same as the message sent. For example, financial transactions are unchanged. Encryption and digital signatures ensure integrity.

Confidentiality
Your message content is private and not available to others as your messages flow through the Internet. Encryption is used to ensure that the message content is confidential and no one eavesdrops on your communications.

Accountability
Neither the sender nor the receiver can deny that the exchange took place. They both agree they took part in the exchange, and can prove that the other did so also. For example, the receiver knows that the sender signed the contract. Digital signatures assure

accountability. Accountability often requires both authentication and data integrity.

As you consider the questions of what to secure, why to secure it, and how to secure it, you should always consider them in the light of these five basic aspects of security.

Table 9 contains a summary of the security components addressed by each section of this chapter. You may find some of the assignments in this table to be relatively arbitrary due to the fuzzy nature of attempting to categorize some of these threats and considerations. For a table of feature comparisons by product, see the feature summary chapter of *Web Server Solutions for VM/ESA*, SG24-4874.

Table 9 (Page 1 of 2). Security Issue Cross-Reference

Discussion/Technique	**Authentication**	**Authorization**	**Integrity**	**Confidentiality**	**Accountability**
5.5.1, "JavaScript" on page 137	√		√		
5.5.2, "Can You Trust the Displayed Information in a Frame" on page 138	√		√		
5.5.3, "Can You Trust an IP Address or DNS Name" on page 138	√				
5.5.3.1, "Trusting IP Addresses" on page 138	√				
5.5.3.2, "Trusting DNS Names" on page 139	√				
5.5.4, "Client and Server Authentication" on page 139	√				
5.5.4.1, "Means for Servers to Authenticate Clients" on page 140	√				
5.5.4.2, "Means for Clients to Authenticate Servers" on page 140	√				
5.6.1, "Authorization" on page 141		√			
5.6.2, "Do Not Trust Incoming Data Validity" on page 142	√	√	√		
5.6.3, "Forms: Get or Post" on page 142				√	
5.6.4, "Denial of Service" on page 143		√			
5.6.5, "Setting Security Profiles for URL Trees" on page 143	√	√			√
5.6.6, "Reentrant and Serially Reusable Resources and CGIs" on page 144			√	√	

Table 9 (Page 2 of 2). Security Issue Cross-Reference

Discussion/Technique	Authentication	Authorization	Integrity	Confidentiality	Accountability
5.7.1, “Secure Sockets Layer (SSL)” on page 146	√		√	√	√
5.7.2, “Are Your CGIs Safe” on page 147	√	√			
5.7.2.1, “Overly Strong VM Privileges” on page 147		√			
5.7.2.2, “Overly Strong Data Access” on page 148		√			
5.7.2.3, “General Rules of Cleanliness” on page 149		√	√	√	
5.7.2.4, “Monopolization of Web Server SVM Resources” on page 150		√	√		
5.7.2.5, “Full Access to HTTP Authorization Header and Password Data Files” on page 150	√	√			√
5.7.3, “Restricting the Ability to Run CGIs” on page 150		√			
5.7.4, “VM:Webgateway′s SVMWEBSHARE CGI Environment” on page 151		√			
5.7.5, “Webshare and Security” on page 152	√	√	√	√	√
5.7.6, “Initial Access Control Conditions” on page 152	√	√			√
5.7.7, “Additional Server Configuration Suggestions” on page 153	√	√	√	√	√
5.8.1, “Firewall Systems” on page 153		√		√	
5.8.2, “Ensure Server on TCP Port Is the Web Server” on page 154	√	√		√	√

5.4 What Data to Secure

One of the first questions to ask yourself is: am I protecting the right data? All too often, the approach is to protect either everything, or nothing. While at times this will be the correct answer, often it is not. Take time to examine what data you are publishing to and receiving from the World Wide Web, asking yourself these questions about it as you go:

- What is the value of the data that is at risk?

  - Is this data already available by other delivery mechanisms? If so, what are the characteristics of those delivery mechanisms? For instance, are you giving away data on the Web for free that you are attempting to sell via other delivery channels? (Note that this is not always a problem, simply something that you should be aware of and make conscious decisions about rather than simply letting it occur.) You need to maintain a global picture of what data is being provided through all your delivery channels. You may find data that you want to remove from other channels (due to security considerations). You may also find data being secured that does not need to be secured.
  - What is the scope of its value and interest? For instance, staff, customers, competitors. Are you publishing to the world information that properly belongs only on your intranet? Are you making available to your competitors information that will assist them and yet not aid you in your marketing and sales efforts?
  - What is the cost of its exposure to being read by unauthorized parties? Compare this with the opportunities generated by making the information available on the Web. Some considerations in this area might include, will this cause lost business opportunities or create new ones? Similarly, will it cause law suits or help to avoid them?
  - What is the cost of the potential of it being modified by unauthorized parties?
  - Do we need to be able to prove that this information was sent from or to the client?
  - Do we need to be able to prove who the other party in a transaction was?
- What will it cost to protect this data? Be sure to include all additional costs, such as: additional hardware (for instance firewalls), extra system resource consumed, and administrative costs to maintain the protections.
- What are the negative impacts of this protection? Include areas such as: system performance and impact to other applications; application functionality, and user productivity.
- How effective will the protections be?

Once you have the answers to these questions, you not only know whether the data is worth protecting, but also whether protection is even appropriate.

5.5 Security Issues on the Browser

As the end consumer of information, the browser user often has what appears to be the easiest time when it comes to security. However, upon taking a closer look, this fallacy is quickly dispelled. Today we are all subject to many invasions of our private lives. Many of the suppliers of information and other products will in fact take such questionable steps as to sell information they have collected about us to other suppliers. In addition, we are often the target for various forms of con games and fraud. There is much to watch out for!

In addition to the issues we raise in this section, the security conscious consumer should know what the benefits of an SSL secured connection are, as well as what problems are not solved by SSL. See section 5.7.1, “Secure Sockets Layer (SSL)” on page 146 for more information.

5.5.1 JavaScript

JavaScript may represent one of the single largest security exposures to the browsing user of any facility on the Web. Using this scripting language, it is possible for the nefarious CGI programmer or HTML author to completely take over the browser from the end user.

All fields displayed by the browser can be manipulated by JavaScript, thus invalidating their believability. This includes the typical "status" line on the bottom of the browser′s window, typically used to report the URL that a link on the page will retrieve. It also includes the "location" text box on the top of the browser′s window, which reports the URL of the current document being displayed.

These two fields in particular are of key importance in allowing browser users to know where they are and where they are going. (See 5.5.3, "Can You Trust an IP Address or DNS Name" on page 138 for more on this topic.) The fact that they can no longer be believed when JavaScript is active, combined with the fact that the browser user can only either turn off all JavaScript (if even that) or has to accept all JavaScript (with no warning when a downloaded page contains JavaScript), makes allowing JavaScript programs to run at all when surfing the Internet extremely dangerous. See the following URL for an article on a variation of this problem:

```
http://www.princetoninfo.com/felten.html
```

This problem and the problem outlined in 5.5.3.2, "Trusting DNS Names" on page 139 are more fully described in the paper *Web Spoofing: An Internet Con Game* by Edward W. Felten, Dirk Balfanz, Drew Dean and Dan S. Wallach. You can find this paper at:

```
http://www.cs.princeton.edu/sip/pub/spoofing.pdf
```

The exposure outlined in 5.5.2, "Can You Trust the Displayed Information in a Frame" on page 138 is a closely related variation on this theme which both demonstrates how JavaScript can be abused, and how easily mislead users can be by what they see in their browser′s window.

To add insult to injury, all action buttons can be taken over also, allowing even attempts to close down the browser′s window to be intercepted and overridden. This allows the nefarious CGI programmer or HTML author to complete the takeover of the browser from the end user. Now, even attempts to close the browser window can lead to it being reopened immediately with a URL of the JavaScript programmer′s choice. Having been the victim of such a page, we can attest to the panic and fear it generates.

The security-aware browser users will configure their browser to disallow the ability to run JavaScript programs. Unfortunately, this will not only disallow use of these facilities by those who wish to take advantage of you, but also by those who wish to legitimately use these facilities to enhance the presentation of their information to you.

5.5.2 Can You Trust the Displayed Information in a Frame

Many sites on the Internet make heavy use of HTML frames. While some users love them, others hate them. But few would question their content, fully believing that content does indeed come from the site that put up the frame. According to *Tasty Bits from the Technology Front* in the November 17, 1998 issue, you cannot trust this information. (You can find this at `http://tbtf.com/`.) It is possible for a nefarious site to literally "take over" a frame of an innocent site. To quote from TBTF: "The ′frame spoof′ vulnerability is breathtaking in its scope and simplicity. It represents not so much a bug in the browsers′ code as a flaw in the security policy they implement."

This flaw can be taken advantage of both with JavaScript as well as with plain HTML code. Both paths allow unauthorized information to be displayed in the frame of an unsuspecting site.

5.5.3 Can You Trust an IP Address or DNS Name

The URL displayed in the browser′s location field is the primary method used by end users to validate that they are interacting with the organization that the Web page claims to be. As we learned in 5.5.1, "JavaScript" on page 137, this data may be suspect due to modifications by the nefarious CGI programmer. However, can and should the information in the browser′s location field be believed at all?

Some people will answer this simple question with a resounding "no." Others will almost as strongly argue "yes." It would seem from reading 5.5.1, "JavaScript" on page 137 and 5.5.2, "Can You Trust the Displayed Information in a Frame" that the no votes have it. But, as we said at the beginning of this chapter, there is only one correct answer to most questions of security: "It depends." It is true, you cannot place complete trust in either of these values. It is also true that you cannot place complete trust in most other forms of authentication. There is always some element of risk, of not being able to be one hundred percent sure. So the question should not be, are we completely sure, but rather, are we confident enough. This question of degree is always present in any question of trust.

5.5.3.1 Trusting IP Addresses

It has often been pointed out that "any workstation can change the IP address to whatever it likes." However, this is not necessarily the case, or even of interest. If a workstation can make such a change to its IP configuration, it still needs to actually successfully exchange IP packets with a host in order to take advantage of this subterfuge. In order to accomplish this, the workstation either needs to be on the same LAN as the target host, or it needs to have the cooperation of the LAN bridges and IP routers between it and the target host. Normally this is not as easy to achieve as the initial subterfuge was. Remember, in order to take active advantage of this subterfuge, it must be possible for a full TCP connection to be achieved, and for this to be done the data must flow in both directions. Thus, most of the time, you can indeed trust the IP address to be correct, because the cost of not doing so is larger than the cost of doing so.

In those cases where the cost of being wrong is too great to accept the risk, there are other alternatives, such as the use of client certificates as discussed in 5.7.1, "Secure Sockets Layer (SSL)" on page 146. For a further reduction of risk, and slightly higher cost, one can even combine multiple techniques for a higher confidence level. For instance, combining a low trust level IP address test with a test of knowledge of a user ID and password pair, or a test of possession of a client certificate leads to an even lower risk of being compromised.

5.5.3.2 Trusting DNS Names

The same basic analysis applies to performing security tests based on the TCP/IP DNS name. The DNS name is resolved by performing a search of a distributed database to map the IP address to the DNS name. While it is possible to compromise this distributed database, it is not common to find such compromised data in it. When combining tests based upon the DNS name with other tests (as outlined for IP address testing), one can have a higher level of confidence in the results than obtained by a single test alone.

However, of the two tests, it is fairly clear that the DNS name for an IP address is far less reliable than the IP address itself. This is true for two reasons. First, the DNS name is derived from the IP address, and thus inherits any questions of the IP address's validity. Second, the DNS database's association between the IP address and DNS name is a second place where incorrect information can be injected. This makes the resulting name more suspect than the original, potentially suspect IP address.

In general, it is also possible with both IP addresses and DNS names to place greater trust in information that is about your firewall-protected intranet than information that is about the Internet. However, as before, if the risk is high, then even for intranets, additional authentication checks are justified to mitigate that risk.

5.5.4 Client and Server Authentication

Authentication is the process of verifying that all parties in the communication are who they say they are. While it is not always necessary to perform authentication for all transactions, there are many cases where it is wise for the server to at least authenticate the client, as in the following:

- Prior to checking client authorization (see 5.6.1, "Authorization" on page 141)
- Whenever data is being accepted from the client to modify the server
- When tracking of resource use by client (not just by the client's computer system) is desired

Likewise, there are cases when the client should authenticate that the server is who it claims to be. While it is relatively unlikely that the traffic destined for a given DNS name will be diverted to a system other than the one it is supposed to be associated with, this is possible. More common is the question of what organization the DNS name in question is really associated with. For instance, in many character sets the letter O and the numeric digit 0 are indistinguishable. Likewise, there can be confusion over the uppercase letter I and the numeric digit 1.

Thus, can you notice the difference between these two URLs?

```
http://WWW.IBM.Com/
http://WWW.1BM.Com/
```

Or between these two (a true case):

```
http://MICROSOFT.Com/   - the company
http://MICR0SOFT.Com/   - a site that spoofed the company
```

It is just as difficult to know what company a cryptic domain name is associated with. For instance, are both, or either, of these URLs associated with Merrill Lynch, the Wall Street company?

```
http://www.merlyn.com/
http://www.ml.com/
```

They both put up pages that claim to be Merrill Lynch, but are they really? This may not be important for casual work, but if you are about to make a financial transaction, this could be the difference between making a valid investment in the market and potentially succumbing to a con artist posing as a legitimate corporation.

5.5.4.1 Means for Servers to Authenticate Clients

A server or CGI can authenticate its clients using any of the following techniques. It is often possible, and desirable, to combine more than one of these techniques for additional levels of confidence.

- Confirmation of the end user′s knowledge of a user ID and password:
  - VM directory or other database of user IDs and passwords.
  - External Security Manager (ESM) (for instance RACF or VM:Secure) based authentication. Such testing might include use of interfaces such as RACROUTE or VM diagnose X′A0′.

  See 5.6.5, “Setting Security Profiles for URL Trees” on page 143 for more information.
- Based upon confirmed knowledge of a user ID and its password, this knowledge may be used to further categorize the user ID into an ACI (or ACI-like) grouping. Examples include VM:Webgateway′s ability to identify system administrators and system operators. See 5.6.5, “Setting Security Profiles for URL Trees” on page 143 for more information.
- Client certificates. See 5.7.1, “Secure Sockets Layer (SSL)” on page 146 for more information.
- Network location of the workstation the browser resides on, by using its IP address. A related view is the DNS name associated with that IP address, which may reveal organizational associations. See 5.6.5, “Setting Security Profiles for URL Trees” on page 143 and 5.5.3, “Can You Trust an IP Address or DNS Name” on page 138 for more information.

5.5.4.2 Means for Clients to Authenticate Servers

A client has fewer options for validating the identity of the server it is communicating with (or, conversely, for the server to give the client validation of who it is). These options are:

Trust of TCP/IP DNS resolution, and the correct delivery of the IP packets to the correct host:
This includes trust in the ability of the browser user to recognize a valid host name. This is a very weak method of validation, not so much because it is easy to spoof either the DNS system or IP addressing (see also 5.5.3, “Can You Trust an IP Address or DNS Name” on page 138), but rather because it is trivially easy for the nefarious CGI programmer to write a JavaScript program as part of the data served and have that program override all browser displays (including the location field). (See 5.5.1, “JavaScript” on page 137 for more information on this exposure.)

Require an SSL connection:
As part of establishing an SSL connection, the server must validate itself to the client. The browser user can then ask the browser to

display the information from the server certificate that was used to establish the connection, or the user can configure the browser to prompt the user before even accepting the certificate (details depend upon the browser in use). See 5.7.1, “Secure Sockets Layer (SSL)” on page 146 for more information.

5.6 Security for Application Writers

In addition to having to face all of the same issues that application users have to face (see 5.5, “Security Issues on the Browser” on page 136), the application writer has additional issues to contend with.

In addition to the issues we raise in this section, the security conscious consumer should know what the benefits of an SSL-secured connection are, as well as what problems are not solved by SSL. See section 5.7.1, “Secure Sockets Layer (SSL)” on page 146 for more information.

As application writers, we find that the easiest technique for performing CGI development includes the use of a dedicated Web server environment for each application writer. This not only simplifies work such as debugging your CGIs, but it also makes it possible for you to explore the effects of running your CGI in various security profiles. If you are blessed with such a development environment, you will also want to review the information in 5.7, “Security Issues for Web Server Administrators” on page 146.

5.6.1 Authorization

Once you know who the client is (authentication), the next task is to determine if the client is allowed to perform the requested task (authorization, or access control). There are many techniques that can be used to do this, each with its own strengths, weaknesses and applicabilities. Some techniques of potential use include:

- Server facilities to constrain access to the URL by server-supported authorization tests. See 5.6.5, “Setting Security Profiles for URL Trees” on page 143 for more information.
- A simple database of authorized user IDs (or any other authenticated value, such as IP addresses, or a field from a client certificate), often referred to as an Access Control List (ACL). This may be as simple as a hard coded test for a user ID or ACI group name, or a small flat file listing authorized user IDs or ACI groups. SFS′s authorization list (as displayed by QUERY AUTH) is an example of this approach.
- Use of RACROUTE to query an ACL maintained by RACF.
- VM:Webgateway′s Dynamic Workers and normal VM access control facilities. This provides only the authority of the data owner to data access. It does not provide checking of the client for access. The authorization is based upon that of the data owner, as controlled by existing normal VM/ESA access control facilities.
- VM:Webgateway CGI Extension′s line mode and 3270 application interfaces allow an application to run with the authorities of the transaction′s client. The authorization is based upon that of the accessing client, as controlled by existing normal VM/ESA access control facilities.

5.6.2 Do Not Trust Incoming Data Validity

Realize that data that comes from the browser is always suspect. It should always be validated by the CGI program itself. You cannot count on any validation by the client, as you do not have any control over what that client is. It is easy for a malicious client to present any data it wishes in its transaction. This includes checks for missing and extra input, as well as ill-formed input such as bad lengths, data types, and value boundaries. Validation checks should be done on all data received, including the data in a hidden form variable, as they all may be modified. See the note on page 39 for additional information. Of particular interest is any value that is used directly in a program in such a way as to potentially modify the behavior of that program when bad data is sent.

An example is a string that is used as part of a pipeline specification. In this case, if the string contains the pipe separator character, the program will have been altered to include a new, unintended stage. Such a stage might, for instance, issue a CP command or otherwise harm the execution of the server. An example might be:

```
'CallPipe',
   '  <' fileid_provided_by_client,
   '| ...'
```

which is fine, until it is passed a value containing the current pipe command separator, such as:

```
fileid_provided_by_client = 'INPUT FILE A | CP SLEEP'
```

which, in this case, would serve to completely hang this server. This could be avoided by adding code to clean up the input before its use, in this case something like the following:

```
Parse Var fileid_provided_by_client fileid_provided_by_client '|'  rest
If rest<>''  Then Signal BadInput
If Words(fileid_provided_by_client)<>3 Then Signal BadInput
```

Another problem is the use constructs, such as the following REXX statements:

```
Interpret data_from_client
Address 'COMMAND' data_from_client
```

This exposure, and similar ones, are also discussed in *Writing WWW CGI Script in REXX* by R. L. A. Cottrell, available on the Web at:

```
http://www.slac.stanford.edu/~cottrell/rexx/share/
```

While this paper assumes a non-VM runtime environment, most of the techniques and issues that he raises are also valid in VM.

As with input from the client, also be suspect of any data that comes from a file that can be modified by unauthorized parties.

5.6.3 Forms: Get or Post

As you design your forms, you should consider the sensitivity of the data which you expect to receive from that form. When you use the GET method in a form (see 3.3.2, "Enhancing Our Sample Program" on page 31), you receive the form variables as a query string on the URL. When you use the POST method in a form (see 3.3.3, "Using a FORM with POST" on page 35) you receive the form variables as an attached document. While these two variations do not affect the vulnerability of the data on the network, they do have very different characteristics in both what is recorded at the server and what the browser user will see. These differences may be motivation for selection between these two methods.

When GET is used, the query string with your encoded form data (including hidden variable values on the form) is visible to both the browser user (as the URL being displayed when the browser renders the resulting page of data from your CGI) and to the Web server′s system administrator(s) (as the URL is recorded on the SVM′s console and/or in the HTTP logs of the SVM). When POST is used, the attached document will contain your encoded form data. This attached document is not visible at the browser (although the browser does typically have the ability to display the source document, and thus any hidden variable values), and is not normally recorded on the SVM (except possibly through debugging diagnostic output).

If the form contains data which you do not wish to have seen by the casual browser user or to have recorded on the SVM, you should use the POST method for your forms.

5.6.4 Denial of Service

When a server becomes so overloaded with work that it can no longer service all incoming requests for work, there is a denial of service. This can occur both innocently (for instance, due to a server that is too small for the presented legitimate workload), and as a form of security attack. The denial of service can also take on several forms. For instance, it may take the form of an overwhelming number of requests, or it may take the form of causing the server to handle no further requests for some period of time by monopolizing the server, or causing it to crash, or otherwise not accept new incoming work.

Additionally, when a CGI program that runs in a Webshare compatibility mode is executing on Webshare, EnterpriseWeb/VM, or in SVMWEBSHARE mode on VM:Webgateway, the server can perform no other processing. A CGI running on VM:Webgateway in SVMEXEC mode also monopolizes the server′s resources, allowing other URLs to be serviced only when it calls the CGI command or (under CMS 14 and later) when CMS Pipelines performs a thread blocking operation. Either of these situations can lead to a denial of services to other URLs.

This exposure can be somewhat addressed by the use of large numbers of servers (for Webshare and EnterpriseWeb/VM), the use of large numbers of workers and a worker-based CGI environment (VM:Webgateway), and the use of VM:Webgateway CGI Extension′s interfaces to move application processing to the end users′ virtual machines. Also refer to topics in Chapter 6, "Performance Issues" on page 157 for more information on reducing the cost of serving a URL (and thus the chance of experiencing a denial of service situation).

5.6.5 Setting Security Profiles for URL Trees

An important part of securing your CGIs is the appropriate securing of each part of the URL tree in both the server′s root and in the user′s ID USERROOTs. While Webshare has no real support in this area, both EnterpriseWeb/VM and VM:Webgateway have extensive support for this important function.

EnterpriseWeb/VM′s support is based upon the existing network pseudo-standard of HTACCESS files. The syntax and semantics of these files are covered in the product documentation, *EnterpriseWeb Secure/VM Installation*

Guide and Reference, and may be familiar to users of other non-VM-based Web servers. This support allows access to be granted or denied based upon criteria of:

- The HTTP method
- The IP address
- The DNS name
- The user ID
- A security group associated with the user ID
- A user-written exit
- The SSL X.509 client certificate (see 5.7.1, "Secure Sockets Layer (SSL)" on page 146 for more information)

The access granted or denied is "all or nothing" on a directory-by-directory level. No support is documented for finer grain access control.

We find that VM:Webgateway's support is more functional and has easier to understand syntax and semantics than a system based on HTACCESS files. This is especially true for users who are familiar with CMS and REXX. The online documentation for VM:Webgateway fully covers this topic with both its reference materials and its task oriented materials. This support allows access to be granted or denied based upon criteria of:

- The HTTP method
- The IP address
- The DNS name
- The user ID
- A security group associated with the user ID
- Whether the user ID is a product system administrator or operator
- The file ID being served
- The URL of the page this URL was found on (the HTTP Referer header)
- Whether there is a known DNS name for this IP address or not
- The return code of a user exit routine that will be passed a list of criteria for its own analysis
- The standard SSL client certificate tag values (see 5.7.1, "Secure Sockets Layer (SSL)" on page 146 for more information)

Access granted or denied is on a file-by-file basis. As with EnterpriseWeb/VM, this access is built up by analysis of access control files at each directory level of the URL. However, unlike EnterpriseWeb/VM, in VM:Webgateway the data owner may optionally force the termination of further analysis of access control records at any time.

For initial access control conditions you may want to see 5.7.6, "Initial Access Control Conditions" on page 152 for more information about how the initial access control status can be controlled.

For more information on this topic, refer to *Web Server Solutions for VM/ESA*, SG24-4874 and to the specific product documentation.

5.6.6 Reentrant and Serially Reusable Resources and CGIs

CMS has many resources that are implicitly shared among all of the programs concurrently or serially used in the virtual machine. A partial list of these resources includes:

- GLOBALV settings
- GLOBAL xxxLIB lists

- NUCXLOADed programs
- EXECLOADed programs and data files
- Virtual devices such as tape drives
- Unit record devices (readers, punches, printers, the console)
- The CMS file system search order
- Any statically named files, for instance profiles, configuration files and work files
- VM′s *System services, such as *MSG and *MONITOR

Often modification of any of these resources by one program will affect the next program′s execution. In addition, when running in a multithreaded, multiprogrammed virtual machine, such as the VM:Webgateway server, changes to one of these resources can actually affect other currently running programs. These interactions must be considered as part of the security (and operational) implications of allowing CGIs to be run at your site.

There are four classes of execution environments. These classes are listed here in order of increasing complexity.

1. The simplest, and safest, alternative with respect to this issue is to make use of VM:Webgateway′s worker-based CGI environments. In this alternative there is a simple, single-threaded, single-process mode of execution. You are also provided with exit points to perform setup and cleanup of the worker virtual machine′s environment. In the extreme, the cleanup exit can even be used to issue a CP LOGOFF to perform the ultimate in cleanup. We highly recommend the VM:Webgateway worker-based CGI environments for sites that are highly security conscious, and in general (for both security and non-security reasons) for all sites that allow CGIs on user pages.

   In addition, we recommend that the site implement a good cleanup exit (VIWEXIT2) when using this environment. This exit should be used to clean up all resources you are concerned with, or that are known to be potentially left allocated by a misbehaving application environment. An extreme version of this exit, for the most complex environments, is to simply issue a CP LOGOFF in the exit, exercising the ultimate in VM cleanup for a logged-on user ID.

2. The next most complex of the environments is that of the WEBSHARE-based servers, EnterpriseWeb/VM and Webshare. In these servers each CGI runs in an separate virtual machine. However, the CGI runs serially to the serving of other URLs, including the running of other CGIs in the same virtual machine. In other words, each URL is served by a dedicated server that is currently processing only that one URL. The Web server does not need to worry about multiple CGIs running in one user ID at one time. However, this can still expose the CGI to changes in the execution environment (resources such as those listed at the start of this section) that purposefully or accidentally occur.

   We recommend that this class of CGI environments only be used by sites that can exercise careful control over the CGIs that are to be run, primarily only those that run from the server′s URL root tree. We do not recommend this class of CGI environments for sites which allow CGIs to run from user pages.

3. The next most complex of the environments relative to this issue are VM:Webgateway′s SVM-based CGI environments. In these environments there are multiple threads of execution present, and actions on one thread will undoubtedly affect the execution of other threads. In addition,

VM:Webgateway has requirements for more than simple CLASS G VM privileges. This means that all CGIs that run in the server are executing on a privileged user ID. While VM/ESA is a very secure operating system, once you are executing in a virtual machine with more than simple CLASS G privileges, the degree of security is greatly compromised.

We recommend that these environments only be used for sites that have a high degree of trust in their enabled CGIUSER user IDs and those that run only system administrator-provided CGIs (those on the server′s root).

4. VM:Webgateway CGI Extension-based CGIs are an interesting mix of both class 1 and class 3. Since the application is running in a dedicated user ID, much of the description applies to these client user IDs. Also, since the controlling CGI is running in a class 3 environment, all of the complexities of that environment apply. In addition to those complexities, there are new ones associated with the maintenance of a long-term session on the client user ID in use.

5.7 Security Issues for Web Server Administrators

In addition to having to face all of the same issues that application users (see 5.5, "Security Issues on the Browser" on page 136) and application writers (see 5.6, "Security for Application Writers" on page 141) have to face, the systems programmer or system administrator has additional issues to contend with.

5.7.1 Secure Sockets Layer (SSL)

The SSL protocol was introduced in *Web Server Solutions for VM/ESA*, SG24-4874. Use of SSL will address several of the basic security issues. In particular, it addresses the issues of confidentiality and integrity. It may also, by use of its client certificates, be used to address the questions of authentication and accountability. It does not address the question of authorization.

In order to use SSL to secure the data served by a URL, you must take two steps:

- Make sure that the data is not served by the HTTP protocol, which does not make use of SSL, and thus represents an unsecured serving of the data. This can be accomplished by not making the data available to a server that can serve the HTTP protocol. Alternately, it can be accomplished by extending trust that the data in question is never served with the HTTP protocol either by explicit configuration of the server, or by any CGI running in the server.
- Configure the server to serve this data on a port configured for SSL (normally port 443) and using the HTTPS protocol. See *Web Server Solutions for VM/ESA*, SG24-4874 and your server′s documentation for detailed information on how to configure your server for SSL.

No further effort is necessary to take advantage of this form of security. CGIs do not need to know if they are running in an SSL or non-SSL environment, as the interfaces, the data they read, and the data they write are all handled in a consistent manner in both cases.

CGIs may want to make sure that they are running with SSL enabled before they access sensitive data. EnterpriseWeb Secure/VM and VM:Webgateway Release 2.2 allows CGIs to test both SSL and the existence of a client certificate.

The SSL protocol is only supported by EnterpriseWeb Secure/VM and VM:Webgateway. It is not supported by EnterpriseWeb/VM or Webshare.

EnterpriseWeb Secure/VM and VM:Webgateway Release 2.2 has support for the fetching of client certificates, making basic security (authorization) decisions based upon those client certificates, and presenting their values as CGI variables. This support can be configured to either never fetch the client certificate or to always require a valid client certificate. When client certificates are being fetched, the decision to reject SSL connections that are not associated with a valid client certificate is left up to the data owner.

For more information on EnterpriseWeb Secure/VM refer to the “Encryption and SSL” chapter of *EnterpriseWeb Secure/VM Installation Guide and Reference* for more information. left up to the data owner.

The VM:Webgateway support for client certificates is not documented in Release 2.2, but is expected to be fully documented in the next release of the product (Release 3.0 is expected to be available in early 1999). Until that time, customers may contact Sterling Software Customer Services for further information on the existing support in Release 2.2.

Before implementing SSL, you should consider the performance implications of its use. See 6.3.1, “Performance Implications of SSL” on page 159 for more information.

5.7.2 Are Your CGIs Safe

One of the glories of VM/ESA is how easy it is to write a REXX program. However, perhaps the single largest exposure to the security of your system′s data are the CGIs you use to provide access to that data on the Web. This is especially true when you allow information providers to create their own CGIs for their user pages, as these CGIs typically will not have been validated to be safe by the system administrator. Some of the more critical exposures posed by CGI programs are outlined in the following sections.

5.7.2.1 Overly Strong VM Privileges

All programs that run in a virtual machine have full access to the full power of that virtual machine. This means that all CGIs running in the Web server can make use of all CP commands and interfaces which the server has access to, based upon that server′s privilege class.

EnterpriseWeb/VM Installation Guide and Reference and *EnterpriseWeb Secure/VM Installation Guide and Reference* list VM class B as being required in order to gain access to VM′s diagnose X′84′ for password checking. The EWEBLOG userid need CLASS B MSGNOH for the logging server to use the MSGNOH command. If CLASS B is not present the logging server will use MSG instead. If a ESM package is used EnterpriseWeb/VM and EnterpriseWeb Secure/VM defer the password checking to the ESM package and therefore EnterpriseWeb/VM adn EnterpriseWeb Secure/VM should be fine with VM class G.

VM:Webgateway Getting Started indicates that VM:Webgateway requires VM classes BEG for operation, without explicitly listing the actual interfaces used. In addition, the CP FORCE command is necessary for smooth operation of VM:Webgateway workers. (We suggest that you call Sterling Software Customer Services to request an explicit list of the interfaces actually required.)

Specific ESM authorization will also be needed by both EnterpriseWeb/VM and VM:Webgateway when they are running in an ESM environment. Compared to Webshare′s ability to run in a class G user ID, this represents an additional security exposure for any CGI run in the server on either EnterpriseWeb/VM or VM:Webgateway.

EnterpriseWeb/VM has not addressed this exposure. VM:Webgateway addresses this exposure with its dynamic worker facility, which allows CGIs to run in a simple class G user ID.

We recommend that the VM privileged interfaces used by your Web server be granted to that server by making use of VM/ESA′s class override facility. In addition, we recommend the use of the worker facility in VM:Webgateway. This way, only those specific CP commands and diagnoses that the server requires are available to the server. Other, unneeded interfaces are thus not available for potential misuse and abuse by CGIs. Contact your product′s provider for an explicit list of all such required interfaces.

5.7.2.2 Overly Strong Data Access

As we said in 5.7.2.1, “Overly Strong VM Privileges” on page 147, all programs that run in a virtual machine have full access to the full power of that virtual machine. This means that all CGIs running in the Web server can make use of all data access rights of that server. Thus, each CGI has access to the superset of all data authorizations, both read and write, of all CGIs that might be run on the server. As a part of this, they have the ability to write modified data into EXECLOADed storage and the server′s own CMS search order. They could serve (or modify, if the server has write access) data other than the data which is intended to be served, either inadvertently through a programming error, or maliciously. (See 5.6.6, “Reentrant and Serially Reusable Resources and CGIs” on page 144 for other related exposures.)

This means that all data that has PUBLIC read authority in what was formerly a closed shop, now implicitly might be readable by the world (given a cooperating CGI). Likewise, the same exposure exists for data that has weak write protection, such as PUBLIC write in SFS or an ALL write password on a minidisk (again, given a cooperating CGI). In addition, a malicious programmer could modify other programs to perform this work, and thus disguise the source of the violations.

While these exposures should not be taken lightly, there are techniques that you can use to address them. These techniques include:

- Never use ALL passwords on minidisks or PUBLIC authorizations in SFS; use ACI group-based authorizations and an ESM instead. For instance, RACF/VM and SafeSFS from Safe Software, Inc. are ESMs for SFS that provides ACI group-based access control lists (ACLs) (see `http://www.safesoftware.com/` for more information). Likewise, ESMs like RACF and VM:Secure are good solutions for minidisk-based systems.
- Only give the Web server (or the data owner, if workers are used in VM:Webgateway; or the client, if VM:Webgateway CGI Extension interfaces are in use) authority to the resources it requires. For instance, do not enroll them in filepools they should not access, and do not give them access to minidisks that are sensitive.
- Enroll the Web server in an SFS filepool dedicated to holding materials to be served on the World Wide Web. Have other VM-based applications which

also access and/or maintain this information do so directly in this filepool, thus avoiding the *duplicate data* problems.

- Make use of VM:Webgateway′s worker facilities and VM:Webgateway CGI Extension′s interfaces to completely avoid the need to give the Web server SVM access to these sensitive applications and their data.
- For EnterpriseWeb/VM and Webshare, since you cannot use worker facilities or client-based authorizations to contain a CGI′s ability to access data that was not intended for it, you must take more extreme steps. We suggest isolating such sensitive environments to a separate TCP port served by a separate, isolated set of Web server SVMs. While this will incur greater administrative overhead costs for your Web environment, it is the only way to contain such access reliably.

5.7.2.3 General Rules of Cleanliness

The following is a list of general rules and guidelines on how to achieve and maintain a clean and safe set of CGIs in your Web server. These rules are very similar to the steps most sites take when deploying any new application or modification to an existing application.

- Institute and enforce standards and guidelines for CGI programs.
- Perform code reviews of all CGIs run on the Web server. As a part of this, you will need to control access to the file system areas that are allowed to contain CGIs. This ensures that no unauthorized parties can create CGIs in the server′s URL tree. See 5.7.3, “Restricting the Ability to Run CGIs” on page 150 for additional information on authorization of users to run CGIs.
- Do not allow the CGI program to be read by those who are not authorized to modify it. Such read access may lead to a form of “security by obscurity.” It is often the case that the way a shortcoming in a CGI program is discovered by reading its source to find its deficiencies.
- Provide and require the use of test servers to aid in validation of the runtime behavior of CGI programs.
- Make use of the security features built into the Web server. An excellent example is VM:Webgateway′s dynamic worker facility, which allows a CGI to run with only the VM authorities of the CGI owner, not the full authority of the Web server. Workers eliminate the exposure to the misuse of both the server′s CP privilege class and its wider access to file system materials.
- Inspect all disks and directories that are accessed by a CGI program using CMS ACCESS. Make sure that no system commands are overridden and make sure the CMS file mode is released at CGI end, including all error exits from the CGI such as REXX NoValue and Syntax exit paths.

  Since WEBSHARE style CGIs by definition include the accessing of the disk or directory that they reside on, it follows that a WEBSHARE mode CGI should never be allowed to run on the Web server from a disk or directory that is not under direct control of these rules.
- When denying access to a resource, do not reveal who is authorized to access that resource. An example is when a CGI contains additional authorization checks for its use. In general, do not give overly detailed reasons for any access denial. While this is in direct conflict with good human factors and usability recommendations (which would have you give explicit and detailed reasons and explanations for the denial), such decisions

are sometimes necessary to preserve good security. The challenge is to attempt to meet the needs of both of these considerations.

5.7.2.4 Monopolization of Web Server SVM Resources

A CGI program can monopolize the server′s resources, such as CPU cycles, DASD I/O, DASD space, network I/O and logical units of work on database servers (for instance locking a database record for update or just opening an SFS file under a work unit that is never closed). In addition, in a multithreaded server, such as VM:Webgateway, any machine synchronous operation or wait state that is misused can severely impact the server and other URLs. An extreme case of this, which is very easy to inadvertently code, is the CGI that uses the CP SLEEP command to wait for some condition. While this is relatively harmless in VM:Webgateway workers and Webshare-based servers, it is a disaster for a VM:Webgateway server. For an example of such a CGI, refer to 3.4.8, “Server Push” on page 53.

In our opinion, the best solutions for this class of exposures are use of single-threaded CGI execution environments (such as VM:Webgateway workers and the Webshare-based servers) and code review of your CGIs. In addition, we recommend that the site implement good clean-up procedures to be used after each CGI runs. See 5.6.6, “Reentrant and Serially Reusable Resources and CGIs” on page 144 and 6.3.6, “Serializable Server Resource Access” on page 170 for more discussion on this topic.

5.7.2.5 Full Access to HTTP Authorization Header and Password Data Files

The CGI program could collect the information from any HTTP AUTHORIZATION header present on the request. This would allow it to potentially collect user ID and password pairs from any password repository that it might use for end-user authentication.

Disable the ability of a CGI to see the HTTP AUTHORIZATION header in VM:Webgateway by coding the AUTHHEADERPASS NO characteristic for CGIs for which you do not wish to allow access to user ID and password information.

If you are running VM:Webgateway, we recommend that you take advantage of VM′s ability to encrypt the user ID passwords in the CP directory. This prevents a CGI from simply reading unencrypted passwords. It does not prevent a copy of the directory from being exported, or exhaustive search techniques by the CGI itself. For these exposures, as well as the CGI exporting or examining the contents of a password file, you will need to make use of VM:Webgateway′s worker facilities to isolate the CGI code from these data sources. Also see 5.7.2.2, “Overly Strong Data Access” on page 148 for more information on this topic.

5.7.3 Restricting the Ability to Run CGIs

If all CGIs are only resident in the server′s URL tree, then there is less opportunity for an errant CGI to be run. This is similar to not allowing every end user to write programs to access your databases, choosing instead to only allow those programs which are validated and made available for all to use. Each of the Web servers offers a method to restrict which user IDs can run CGIs.

In Webshare and EnterpriseWeb/VM this is an "all or nothing" configuration setting made with the CGIUSERS configuration statement. This statement takes the form:

```
CGIUSERS userid1 userid2 userid3 ...
```

to provide a list of user IDs that are allowed to use any configured CGI file type to run a CGI program in the server.

In VM:Webgateway there is a finer granularity method, with higher administrative cost, to enable who may run CGIs from their private URL trees. The CONFIG CGIUSER command explicitly lists each filetype that a user ID may run as a CGI. The syntax of this command is:

```
CONFIG CGIUSER ADD userid filetype
```

5.7.4 VM:Webgateway′s SVMWEBSHARE CGI Environment

VM:Webgateway has two means of running Webshare compatibility mode CGIs: in the server (CGI environment SVMWEBSHARE), and in a worker (CGI environment WORKERWEBSHARE). While the programming APIs for these two environments are essentially identical, there are significant differences in the security profiles of these two modes of operation.

When running in a WORKERWEBSHARE environment, the running CGI is isolated in a worker virtual machine. This worker virtual machine has only been given the security profile of the CGI′s owner (by making use of VM/ESA′s diagnose X′D4′ support). There are no other threads of execution when running in this mode. The CGI has almost complete control over the virtual machine. This is tighter security than the CGI environment of either a Webshare or EnterpriseWeb/VM-based server.

When running in a SVMWEBSHARE environment, the running CGI is in a shared, threaded, multiprocessing environment, the VM:Webgateway server virtual machine. This virtual machine has a security profile that is much more open than that of a worker. As with any CGI on Webshare or EnterpriseWeb/VM, and all other SVM-based CGI environments on VM:Webgateway (currently only SVMEXEC), an SVMWEBSHARE environment CGI has the full superset of VM privileges needed to access and run all of the data and CGIs that it may be called upon to serve. In addition, as with all other SVM-based CGI environments on VM:Webgateway, the CGI program needs to share the virtual machine′s resources with other CGIs and the serving of other static content files. This leads to some special security considerations for this SVMWEBSHARE CGI environment.

An implicit part of the definition of the environment of a WEBSHARE-compatible CGI environment is that the file system that the CGI was found on will, if necessary and possible, be added to the list of CMS ACCESSed minidisks and SFS directories. The action of fulfilling this assumption modifies a key part of the execution environment for the virtual machine, specifically the CMS search order. This can lead to incorrect results if this newly accessed file system contains file IDs that were used from later in the CMS search order and are referenced during the execution of the CGI. When running in a single-threaded, single-processing environment (such as Webshare and EnterpriseWeb/VM), this does not pose a major security issue. However, when run in a multiprocessing virtual machine, such as VM:Webgateway, this can materially affect other work in the server. This is especially true if the server is configured with a URL root of the CMS search order.

We recommend against the use of the VM:Webgateway SVMWEBSHARE CGI environment by CGIUSER-authorized user IDs. We recommend that the more secure VM:Webgateway WORKERWEBSHARE CGI environment be used instead.

5.7.5 Webshare and Security

The base Webshare package contains essentially no support for security. (See *Web Server Solutions for VM/ESA*, SG24-4874, for an overview of its facilities.) However, there are local modifications to the product which add some very basic security features (as well as rudimentary SSI support). The document *Server Side Processing for WebShare on VM/CMS* available from `http://miamiu.muohio.edu/~jdkinne/ssp/ssp.html` describes modifications to Webshare that add Server Side Include (SSI) processing, including the ability to:

- Restrict the IP addresses to which an HTML document can be delivered
- Include dynamic information in documents, such as:
  - The number of times a document has been served
  - The last modification date of a document
  - The current date
- Include the contents of other files

While these extensions are important and useful, we cannot in good faith recommend the use of Webshare or these extensions to any site that has security concerns that are not addressed by the base package. We base this decision on the statement contained in the disclaimer for these extensions to the package. It reads in part:

"Warranty of WebShare Server Side Processing: This software is provided with no warranty. It is not warranted for any purpose. Use it at your own risk. Neither the author nor any other party shall be held liable for any damages resulting from the use of this software.... Actually making the modifications to Webshare discussed in this paper and distribution, publication or sale of such modifications is prohibited without the approval of Beyond Software..."

We do not believe that any site with security concerns should accept these terms.

5.7.6 Initial Access Control Conditions

In EnterpriseWeb/VM the initial assumption is that a file may be served. All access control that may be applied by the techniques outlined in 5.6.5, "Setting Security Profiles for URL Trees" on page 143 is due to the explicit actions of the data owner who creates and maintains the access control files.

In VM:Webgateway the same is true for both the server's root and, by default, for USERPAGEs. However, the system administrator of VM:Webgateway can choose to configure an initial access control file that contains standard product access control file records to set the starting condition for allowing or denying service. See VM:Webgateway's online documentation on "Initial ACCESS File" for more information.

5.7.7 Additional Server Configuration Suggestions

You should be very careful when configuring your Web server. In addition to the points raised elsewhere in this chapter, consider the following aspects of your configuration:

EnterpriseWeb/VM Server Configuration Suggestions

- Configure AUTOINDEX OFF to prevent the dynamic generation of an index of files on a minidisk with no INDEX HTML file present.
- Configure USERWEBSPECIFY OFF to prevent the browser from being able to surf your site for minidisks that may not be intended for serving on the Web.
- Do not configure USERWEBLINKP CHALLENGE COOKIE to prevent building cookies containing the link password of minidisks.
- Consider not serving user data files at all. This is accomplished by the configuration setting of USERWEBS OFF.
- We recommend that you use great care when configuring the ability to map a file type to a command via the product′s media map file. The lack of built-in sanity and security testing of URL-provided parameters and arguments makes this a very powerful facility, which could be easily misused in the ways we outline in 5.6.2, "Do Not Trust Incoming Data Validity" on page 142.

VM:Webgateway Server Configuration Suggestions

- Consider not serving user data files at all. This is accomplished by the CONFIG USERPAGES OFF command.

5.8 Security for Network Programmers

In many ways, while the network programmers′ life may be a harried one, they can rest easier when it comes to security and the World Wide Web. In large part, this is because the amount of control that they can exercise over this niche of their world is limited.

5.8.1 Firewall Systems

In a car or a building, a firewall is a fireproof wall that is used to prevent the spread of fire from one place to another. Firewalls in networking are similar to firewalls in cars and homes. They are systems that act as barriers to prevent or control the transmission of data. They protect resources from access by unauthorized remote users. There are several types of firewalls, but in general they all accomplish this same shared goal. Their primary difference lies in how they decide what data is allowed to pass and how they let that data pass. Firewall systems are typically set up at the corporate level of network connectivity, and maintained by the corporation′s network programmers. They serve both to protect the corporate intranet data from being exposed to the outside world, and to isolate portions of the corporate network from access to the outside world. As such, they are often an appropriate and simple means to provide security for your Web transactions. They provide an easy path to simple "all or nothing" security solutions.

Use of firewalls addresses aspects of several of the basic security issues. In particular, it partially addresses the issues of confidentiality (only those inside of firewall can see the data it is protecting), authentication (the client and server can both be known to be either inside the firewall, or authenticated by the firewall), and authorization (due to its form of partial authentication). It does not address the question of integrity or accountability.

For a further discussion of this topic see *Web Server Solutions for VM/ESA*, SG24-4874.

5.8.2 Ensure Server on TCP Port Is the Web Server

In VM/ESA′s TCP/IP product it is possible for any user ID to connect to any port that is not reserved for use by a configured user ID. Thus, if you start a Web server without reserving a port for its user ID in the TCP/IP configuration, you are opening yourself to the potential of another user ID on the system gaining control of the port, and thus subverting the data flow. This could allow that user ID to gain access to sensitive information such as the client′s user ID and password. This also would allow for a class of *denial of services* attack.

These problems will be avoided if the user ID of the Web server is bound to a specific port in the TCP/IP configuration file, PROFILE TCPIP.

In addition, another form of denial of service related to the use of the VM/ESA TCP/IP port reservation interface is to prevent the server from listening on the reserved port for new connections. This can be solved by making use of the AUTOLOG configuration records in PROFILE TCPIP. VM TCP/IP starts all virtual machines on its current AUTOLOG list when it starts execution. If a virtual machine on the AUTOLOG list has reserved a TCP port with the PORT statement, but is not accepting connections on that port, TCP/IP attempts to FORCE that virtual machine and start it again. An exception is the case where the PORT statement specifies NOAUTOLOG.

We suggest that you always reserve the ports in use by your Web server virtual machine, and that this virtual machine be placed in the AUTOLOG list in the PROFILE TCPIP configuration file.

5.9 Security Summary

While we have presented you here with a list of pundits′ questions and "rules of thumb" suggested answers, remember that few of these should be applied to your site without first performing your own analysis of both questions and answers.

The single most important question, of course, is: what is right for your site. Remember that this is not an area of static questions or answers; you should reexamine both on a regular basis, because "things change." We also hope that you will approach the problem creatively.

5.10 References

The following are additional references on security on the World Wide Web:

- *Web Server Solutions for VM/ESA*, SG24-4874 gives an introduction to several Web servers on VM/ESA, including the security features they offer. In addition, it provides a basic introduction to security issues in a Web environment.

- *The World Wide Web Security FAQ* at: http://www.w3.org/Security/Faq/www-security-faq.html
- A general overview of writing CGIs in REXX, including security concerns, is *Writing WWW CGI Script in REXX* by R. L. A. Cottrell, available on the Web at: http://www.slac.stanford.edu/~ cottrell/rexx/share/
- http://www.vm.sterling.com/conferences/teleconf.html#ssl is the transcript of the Sterling Software teleconference on *Addressing Web Security Issues* from October 21, 1997.
- http://www.safesoftware.com/ for information on SafeSFS, an ESM for SFS that provides ACI group-based access control lists (ACLs).
- The Princeton University Secure Internet Programming Web site at http://www.cs.princeton.edu/sip/ is an excellent source of information on exposures found in mobile code systems such as Java, JavaScript, and ActiveX.
- The *Tasty Bits from the Technology Front* Web pages at http://tbtf.com offer a wonderful source of information on new technology. You can often find mention of new security issues for the Web mentioned at this site. The site is fully searchable, and an e-mail list exists for receiving regular posting of new information.

Chapter 6. Performance Issues

You are now an expert at Web-enabling VM resources, and you know that your data is appropriately secured. You are now running a Web site that is rich in content and you have an ever-growing base of consumers of its contents. But, you are now hearing complaints from those once happy consumers about how long it takes to access data from your site, and your capacity planners are asking if you plan to continue to grow in your capacity needs as rapidly as you have recently been growing. Perhaps it is time to investigate where your resources are being consumed, and improve the efficiency of your more resource-intensive CGIs.

As you read this chapter, remember that when it comes to questions of performance, there is only one correct answer: "It depends."

6.1 What to Optimize

Everyone hears something different when they hear the term "performance." What do we mean when we use the term here? It depends. Yes, as you may have feared, every one of these many different aspects of performance is important in its own context. And just to make things interesting, these many aspects of performance are sometimes in direct conflict with each other or with other important considerations such as usability, human factors, and security. A partial list of interesting views of performance includes:

- Browsing user′s wall clock time
- Browser resources
- Network bandwidth and resources
- Server CPU use
- Server memory use, both allocated and working set size
- Server I/O
- Server elapsed wall clock time
- Serialization of access to serializable server resources
- Load presented to secondary VM SVMs such as SFS, DB2, and the ESM
- Server′s system-wide CPU use
- Server′s system-wide memory use, both allocated and working set size
- Server′s system-wide I/O
- Load presented by the server to other non-VM-based servers, such as MQSeries servers and other TCP/IP-based systems

6.2 Performance Issues of Browser Configuration

As the end consumer of information, it may seem that the browser user can do little to affect performance. However, there are several basic aspects of browser configuration, all relating to document caching, that can have a great effect on the load it places upon your applications, the Web server and the network.

6.2.1 Browser Document Caching

In our opinion, the single largest knob on Web server performance is how the Web browser cache is configured. Every document that can be served out of the browser′s cache is a URL that does not need to be serviced by the Web server. There are two basic knobs on most browsers′ cache settings (although you may find variations):

1. Cache size
   How much memory and DASD storage should be used to hold copies of the documents you fetch from the Web server? In general, the larger the setting, the better performing the Web browser will tend to be. This is because if a copy of a document is resident locally in the browser′s cache, there is no need to fetch the document across the network at what is typically lower speed than local DASD access. A document is removed from the cache when the cache becomes full (or shortly thereafter), typically based upon the date and time of last use of the document (that is, the least recently used documents would be removed until the cache′s configured size is no longer exceeded).
2. Cache validation timing
   Whenever there is a cache, there is implicitly a cache validation algorithm in place to provide a way to make sure that the local cached copy is the same as the copy that would have been retrieved from the network if that cached copy did not exist. In HTTP the algorithm consists of asking the Web server if a document has changed since the last time it was modified. (See the Last-Modified and Expires headers in Table 3 on page 13 and the If-Modified-Since header in Table 1 on page 11 for more information.) Typical browsers allow the browser user to select when this algorithm is invoked. Three common settings, in order of increasingly higher performance (and lower exposure to the use of bad cache data), are:
   a. Every time the URL is to be displayed. This setting is the only one that can hope to maintain a valid cache.
   b. The first time a URL is to be displayed during a browser session (that is, since the browser application was started).
   c. Never. This setting is almost guaranteed to often display out of date cached copies of data.

   The browser will typically provide its user with one or more methods to override this algorithm and either explicitly check the cached copy′s validity or unconditionally fetch a new copy of the document from the Web server.

   Also refer to 3.4.3, "Making Efficient Use of the Browser′s Cache" on page 44 for additional information.

Some browsers also include separate configuration controls for documents that were received over an SSL connection. These documents will typically not be stored on DASD, under the assumption that this might represent an unreasonable potential for their content to be compromised.

One last consideration for these caches and their size is the effects they will have on the backups of the system that the browser resides upon. Typically the contents of this cache (usually represented as multiple files in a single dedicated directory or folder) have no long term interest, tend to contain large numbers of frequently changing files, and have no real value if they were to be lost or restored. We suggest that you strongly investigate configuring your PC backup facilities (such as the ADSTAR Distributed Storage Manager (ADSM) and Norton Utilities) to not back up the browser cache′s folder.

6.2.2 Browser Use of a Caching Proxy Server

If your site has a caching proxy server, it is wise to configure your browser to make use of this resource. You will typically find that the response of such a local service is far better than the response of a remote service by the true data owner. This is due to the tendency for a “locality of reference” that occurs when many browsers are seeking the same data, for instance the data on an intranet site, which may actually reside in a non-local host.

6.3 Performance Issues for Application Writers

In addition to having to face all of the same issues that application users have to face (see 6.2, “Performance Issues of Browser Configuration” on page 157), the application writer has additional issues to contend with.

As application writers, we find that the easiest technique for performing CGI development includes the use of a dedicated Web server environment for each application writer. This not only simplifies work such as debugging your CGIs, but it also makes it possible for you to explore the performance implications of various changes to your CGIs. If you are blessed with such a development environment, you will also want to review the information in 6.4, “Performance Issues for Web Server Administrators” on page 171.

6.3.1 Performance Implications of SSL

In 5.7.1, “Secure Sockets Layer (SSL)” on page 146 you learned how SSL can address some of your security issues. However, SSL also has negative performance implications.

The SSL protocol is session oriented. This is done to help improve performance by reducing the cost of the encryption work and amortizing the cost of authentication over multiple transactions. A session has two phases:

1. Initialization

   This phase consists of all of the one-time start-up costs for the SSL session. This includes:
   - Establish authentication of server to client.
   - Optionally fetch a client certificate from client for use by code on the server.
   - Use server′s public/private RSA key pair to exchange a shared secret key.

   This is a relatively high-cost phase in resources consumed due to the many request/response data transfers and the cost of public key encryption.
2. Transparent data transfer, making use of the shared secret key for symmetric encryption

   During this phase multiple HTTP transactions (request URLs and response documents) are exchanged over one or more TCP connections. Each new TCP connection requires a relatively fast, low-cost “reconnection” of the new TCP connection to the existing SSL session. This is a relatively low-cost phase compared to phase one, both due to the low cost of reconnection to an existing SSL session compared to establishment of a new one, and the low cost of symmetric key encryption compared to public key encryption.

Given the small, low-cost nature of serving many URLs, phase one costs can easily be double the cost of a typical URL. However, by being able to amortize this cost over the serving of many URLs, the incremental cost per URL can easily be reduced to an acceptable value. Luckily, the cost of phase two represents a relatively low incremental cost over that of serving the same URL without SSL. Laboratory measurements have shown that an increase in server CPU of 10-20% can be expected for the use of HTTPS compared to HTTP. This is consistent with the measurements by Goldberg, Buff, and Schmitt in *A Comparison of HTTP and HTTPS Performance*, available at: `http://www.cs.nyu.edu/artg/research/comparison/comparison.html`.

Based upon these results, we make the following recommendations:

- Use SSL only when you need it. Do not use SSL when you do not need the facilities that it provides.
- Due to the high cost of phase one of SSL session creation it is critically important to make use of an SSL session caching scheme. For EnterpriseWeb/VM this is an optional additional server (EnterpriseWeb Secure/VM) that we strongly recommend that you install. For VM:Webgateway this is automatically handled by the server when SSL is enabled.
- For the same reason, it is critical that the session cache size and expiration configuration be liberal enough to allow significant work to be accomplished during each session. The goal is to not require repeated establishment of a session with the same client. In EnterpriseWeb/VM there is no apparent knob for the cache size (which is coded to 10 MB), but cache life is controlled by the SSLSessionKeyLife configuration setting. The default setting of 100 seconds for SSLSessionKeyLife seems too small to us, and we recommend that you increase it significantly. In VM:Webgateway there are no knobs for either of these controls. We suggest that you request such knobs from Sterling Software Customer Services.
- There is a nontrivial cost to both the fetching and the analysis of client certificates. We recommend that you not configure for their use unless your site′s applications require the benefits they bring. Both EnterpriseWeb/VM and VM:Webgateway have configuration options to allow for this. However, these two products have significantly different support in this area. EnterpriseWeb/VM′s option only allows for the client certificate to either not be fetched at all, or to require that it be present. VM:Webgateway′s option allows for the client certificate to either not be fetched at all, or to optionally fetch it if it exists, leaving the choice of how to handle an SSL connection with no client certificate up to the data owner to accept or reject.
- As with all HTTP configurations, the use of caching by the browser is of critical importance. For more information see 6.2.1, “Browser Document Caching” on page 158.

6.3.2 Reducing Browser Rendering Time

Let us be realistic: all that your end users really care about is the wall clock time it takes for their browser to display the information they requested. They could care less how much resource is consumed on any of their workstations, the network, or the Web server. It is a very simple equation for them: less wall clock time is good. What is more, they do not even care if the request is being met out of their local cache or from your Web server. Even when it is a local cache copy, they still need to have the document rendered in a timely and efficient manner. There are several steps you can take to address reducing the cost of the rendering time for your documents.

6.3.2.1 Fast Rendering by Avoiding HTML Tables

Limit the use, size and complexity of tables, as in our experience they tend to be slow to render. But, do use them when they are important to the presentation of the material. There is no value in having a page that is fast to render, but whose content is not easily understood.

6.3.2.2 Fast Delivery of Start of Document

An important step towards the appearance of quick service time is to reduce the time from initiation of the request to the first visual response to the request. If the end users can see the browser render a title and the first few lines of your page quickly, they are more willing to wait longer for the reception of the last few words of your page. One of the easiest techniques for reducing this time to initial response is to be sure to send all headers and the first few lines of your output early during your processing. See the performance tip on page 26 for more information.

This may also be in direct conflict with the goal of reducing the server′s CPU, especially as achieved by blocking the output. See 6.3.3, "Reducing Web Server CPU" on page 162 and in particular 6.3.3.4, "Reduce CPU Cost of CGIs" on page 167 for more information.

6.3.2.3 Avoid Multiple Fetches of the Same URL

Browsers need to use a pessimistic scheme for mapping URLs to cache entries. They must do this to ensure that they do not use a cache entry that actually represents data for a different URL than the one entered.

Most Web servers treat their URLs as a case sensitive string. This makes sense when mapping a URL into a file system that has a case sensitive file naming scheme, as with CMS′s BFS file system. Thus, as a part of proper cache management, browsers treat URLs as case sensitive mixed case strings. However, CMS′s minidisk and SFS file systems are used in CMS in a case insensitive manner, specifically by folding their file IDs to upper case. All three of the VM Web servers treat these two file systems the same way, as strictly upper case. This means that many different mixed case URLs will actually resolve to the same upper case file ID.

In addition, while DNS names are by definition case insensitive, they are sensitive to the exact sequence of characters used. Thus, even though it appears to you that two names are "the same" (for instance WTSCPOK and WTSCPOK.ITSO.IBM.Com), the browser will treat them as being different and unique for the purposes of its cache management. Some browsers may even treat this name in a case sensitive manner, even though this may not be required.

Always refer to URLs in both HREF tags and the SRC attribute of IMG tags using the identical character string for links to the same data. This includes both the same case for all characters in the string, and the same host name form. This allows the browser to recognize when a URL matches data in its cache.

6.3.2.4 Efficient Rendering of Images

Help the browser to display as much information to its users as soon as possible:

- Keep the count of images on a page, and their size (in bytes transferred) to a minimum. This will reduce the number of HTTP connections created (in HTTP 1.0) and the number of URLs to be served. It will also minimize network bandwidth consumed and delays incurred.
- Allow clients to see full graphical images before all of the image data is sent by using “interlaced GIF” files. See the discussion of images in HTML in Chapter 1 of *Web Server Solutions for VM/ESA*, SG24-4874 for more information on this topic.
- Tell the browser the size of images. This reduces the work to render pages, allowing the browser to perform basic layouts and rendering of text before it actually fetches the image. Thus, the end user can often gather the information of interest from the page before the images are fully fetched, (when working on a slow link). To accomplish this, always include the HEIGHT and WIDTH attributes on HTML IMG tags. This allows the browser to render the text of a page before the images are completely fetched from the server. See *Web Server Solutions for VM/ESA*, SG24-4874 for more information on this topic.

  EnterpriseWeb/VM includes a pipeline filter utility named EWIMGSIZ, which scans source HTML looking for IMG tags. For each tag found, it attempts to add these additional attributes to help improve the performance of the page. It supports the GIF, JPEG, and PNG formats (with file types of GIF, JPG, and PNG, respectfully). Refer to the *EnterpriseWeb Secure/VM Installation Guide and Reference* for more information.

You can also use your browser′s view pageinfo facilities to potentially identify these image characteristics.

6.3.2.5 Getting the Browser to Display ″nn% of ##K″ Progress Messages

While not strictly a performance improvement, a good human factor is to include a Content-Length header in outgoing HTML data streams. As explained in 3.4.1, “Sending Header Fields” on page 43, this allows the browser to display a percentage retrieved number while the document is being served to it. However, this means that you cannot generate the header until all of the data is generated, and that you will have to completely buffer all of that output data. This is not a good idea. It is far better to start serving the data to the browser as soon as possible, and to avoid having to buffer all output before starting to send any data to the browser. By avoiding this unnecessary buffering (just to send this header), you get the advantage of overlapping some of the network transfer time and browser rendering time, with the time to generate the data stream. This advantage far outweighs the disadvantage of the browser not having this byte count.

6.3.3 Reducing Web Server CPU

One of the first areas people think about when discussing server performance is the CPU utilization of the server. Reducing the CPU utilization typically reduces the total response time. However, the CPU component of the response time is often not the dominant factor. Often other delays, such as I/O time, server queuing time, and network time dominate the total response time equation. Thus, we hope that you do not dwell completely upon this one factor in the performance of your Web environment. But that said, reduction of CPU utilization is still a laudable goal, even if it results in some trade-offs where other resource utilizations, such as server memory use, may be increased.

6.3.3.1 Use a REXX Compiler

If your CGIs contain much REXX logic (as opposed to being primarily CMS Pipelines CALLPIPE commands), they will very likely benefit greatly from compilation with one of the available CMS REXX compilers. In addition, the code of all three VM Web servers is known to benefit from compilation. Sterling Software, Inc. and Beyond Software Inc. are distributing their Web server in compiled form, as well as in source form, to facilitate both product maintenance and optional compilation with an alternate compiler.

There are four execution environments for your REXX code:

- IBM REXX/370 Compiler, 5695-013
  This REXX environment tends to produce the fastest running object code. However, use of compile time options to allow for runtime tracing of the program tends to greatly impact the runtime speed of the code it generates. In our experience this impact tends to make that code run slower than the interpreted code environment.
- Sterling Software, Inc.′s VM:ProRexx product
  This REXX compiler tends to be the fastest at compile time, but does not produce quite as fast a runtime program. Allowing for runtime tracing of the compiled program tends to produce programs that run faster than any of the other environments. However, its runtime tracing facilities are limited to TRACE C output. This product is no longer marketed by Sterling Software, Inc..
- Sterling Software, Inc.′s VRXUTIL tool
  This compiler is shipped with their Automated Install Manager (AIM) system and has the same compile time and runtime characteristics as the VM:ProRexx product.
- VM/ESA CMS interpreted execution of REXX source
  The default execution environment. For some classes of REXX programs, this environment is still the highest performing one.

We recommend you take the following steps to compile your code:

- Compile complex, large, and long-running CGIs with a REXX compiler. Then, measure the effects to make sure it is faster, as in some cases the resulting code can actually be slower than the interpreted REXX. In our experience with REXX compilers, we have seen results that range on the down side from a doubling in CPU consumed to on the up side a 90% reduction in CPU consumed. In general, the more application logic that exists as REXX clauses, the better the results of compilation. This includes logic for significant string handling, mathematical operations, and significant flow of control structure. On the other hand, code that is characterized as straight line flow of control to invoke a small number of host commands will typically not benefit from compilation.
- Compile all product code. Both Beyond Software Inc. and Sterling Software, Inc. have stated that their product code runs fastest when compiled. Use IBM′s REXX compiler if available, otherwise for VM:Webgateway use VM:ProRexx from Sterling Software, Inc. or the VRXUTIL compiler that they have built into recent releases of most of their product line.
- Install the runtime support for the compiler of your choice into a DCSS. This reduces the total system-wide storage needs, by allowing this storage to be

shared. Instructions for this installation come with each of the compilers. If you use more than one of these compilers at your site, be sure to not overlap these DCSSes, as you may need all of them to be available in a single virtual machine at a single time.

- When compiling with the IBM REXX compiler, do not enable the support of the TRACE statement. Use of the NOTRACE option is recommended for the best performance. (But, as always, be sure to make measurements to confirm this.)

6.3.3.2 Reduce CPU Cost of URL Resolution and Basic Security

Every piece of work your Web server performs includes resolving a URL to a local file to serve or CGI to run. Whatever steps you can take to streamline this process will result in lower CPU and I/O costs to the Web server.

- Optimize VM:Webgateway′s DIRMAP and Webshare-based products′ FILELIST files.
  - In general, minimize the number of lines in the files.
  - Reduce the count of, or (in the extreme) eliminate comments in DIRMAPs and FILELISTs. Keep whatever comments are important for the long term understandability and maintainability of the file. We do not recommend making copies of the file with comments in them to be maintained in parallel. We do recommend combining block comments into a single long line rather than multiple short lines. This reduces the number of lines to process and maintains the information for long-term understanding of the file. The following example shows both good and bad comment forms:

```
* This is a good way to write your comments, all on one line.       .
* Empty lines and lines with just ″*″ are not recommended.

*
* Lines
* with
* few
* words
* on
* them
* are
* not
* recommended.
```

  - In EnterpriseWeb/VM, use EWCOMP to compile all password, HTACCESS, and FILELIST files. See the *EnterpriseWeb Secure/VM Installation Guide and Reference* for more information.
  - When serving data out of SFS or BFS, it is preferable to make use of the native file system′s hierarchical nature rather than making use of either DIRMAPs or FILELISTs. We know that this works for VM:Webgateway and believe it to work in EnterpriseWeb/VM (although we could not find positive confirmation of this intention in *EnterpriseWeb Secure/VM Installation Guide and Reference*).
  - On VM:Webgateway, make use of the wild card character support in DIRMAPs whenever possible to reduce the number of FILE statements present. This will both make maintenance of the DIRMAP easier, and significantly speed up URL resolution (compared to parsing large numbers of FILE records).
  - Order your DIRMAP and FILELIST files by the rate of URL reference (most frequently referenced at the top, least frequently at the bottom).

- Use as simple a security profile as you can to meet your needs. The more security statements needed, the slower the URL resolution for every URL that is in a directory at or below their level in the analysis tree. Place the security profiles as close to the URL tree leaves as possible, leaving the nodes and files with no security needs that are near the URL tree root free from the cost of the analysis of the security profiles. See 5.6.5, "Setting Security Profiles for URL Trees" on page 143 for more information.
- Use a flat URL directory structure (little depth to the file system′s tree to minimize directory depth). If DIRMAPs are used, beware that this may slow things down unless wild cards are used heavily to reduce the size of the DIRMAP.

  This flat nature is often in direct conflict with usability and maintainability considerations. You will need to decide which is more important for your site and the application in question. In addition, when very large numbers of files exist, it may be faster to segregate these into many smaller directories rather than making use of a flat directory structure. The only way to be sure is to experiment when you have performance issues.
- Avoid a "residual path" on a CGI′s URL. A residual path is the portion of the path that was not used to resolve the URL to the CGI′s file ID. This residual path is communicated to the CGI via the CGI variables:

PATH_INFO	The string representing the portion of the URL path left over.
PATH_TRANSLATED	The resulting file ID from performing URL resolution on the PATH_INFO value.

  See Table 4 on page 14 for more information on these variables.

  From these definitions you can see that when such a residual exists, it must be resolved as a URL (an expensive process) to set the required CGI variable PATH_TRANSLATED. You should only use this technique when the string to be passed is really a URL and you want to know in your CGI how that URL resolves. In all other cases, it is preferable to pass this information as a URL QUERY_STRING (in CGI variable terms).

6.3.3.3 Reduce CPU Cost of Static Documents

While a VM Web server will likely tend to have more content generated by CGIs than from static documents, the following points are still worth considering as optimizations:

- Avoid Server Side Include (SSI) processing. In VM:Webgateway this is invoked by serving a file with a file type associated with the SSI characteristic. In EnterpriseWeb/VM this is invoked by serving a file with a file type that includes the SSIPSV, TEMPLATE or TEMPLATEPSW transport filter in addition to the server-wide setting of the SSI configuration records.

  SSI processing requires that the server perform analysis of the HTML source looking for SSI directives. (See *Web Server Solutions for VM/ESA*, SG24-4874 for more information on SSI.) This processing not only causes extra server overhead, but it also prevents the browser from being able to cache a resulting page of data. This may then require the server to re-serve the same data repeatedly, which in turn also causes extra server overhead.

  For those files that you do wish to use SSI for, consider the following techniques to improve the performance of serving them:

  – On VM:Webgateway, use a directory that is close to the root of the directory structure the included file resides in. The security profile should be as simple as possible, ideally with open access for all. This will minimize the cost of URL resolution and basic security analysis for these includes.

– For documents where delivery performance is paramount, "compile" the parsable document into its equivalent static document. This can be done in one of two ways:
  - Create a simple program to resolve these SSI directives and create an output document that will then be used on the server as the file to be served. Such a program, and its source, would not even need to support the full official SSI language. It would only need to perform the processing needed for your documents and could use any directive language you care to formulate.
  - Fetch the document through the browser with full SSI resolution enabled. The browser can then be instructed to save the document′s source to a file (or the file can be fetched from the browser′s document cache). This saved, fully resolved copy of the file can then be uploaded to the Web server and served as a normal non-SSI static document. A VM-based browser, such as Charlotte, which may be obtained from Beyond Software Inc., is ideal for this type of processing.

• When serving binary data, such as GIF and JPEG images, audio files, and movie files such as MPEGs, store the data in the file system in as large a record size as is practical. This will reduce the work spent processing the data on the server. A simple method of achieving this is to reblock these source data files in the CMS file system using a CMS Pipelines command such as:

```
'PIPE',
   '  <' fn ft fm,
   '| fblock 65535',
   '| >' fn ft fm
```

This will reduce the number of records processed by CMS and the server, and thus reduce the total server resource consumption to serve the file.

You may wish to consider performing the same sort of optimization to your static HTML files, although since you cannot break the lines at arbitrary points, you will want to exercise more care on how they are blocked. For instance, you could use:

```
'PIPE',
   '  <' fn ft fm,
   '| join * x0D25 65535',
   '| >' fn ft fm
```

The stage `join * x0D25 65535` takes all the records from the input file and joins them together into one record. We have the JOIN stage insert a CRLF (X′0D25′) between each record (as would be done when serving a file containing EBCDIC records) to help make the document more readable with the browser′s VIEW DOCUMENT facilities. The number 65535 means that join will not create a record larger than 65535 bytes; it will create another output record for the additional records. This is essentially a *blocking* stage that takes the small records from the file and makes a large "block" of data to write to the Web server. The number 65535 was chosen as a trade-off between storage use and Web server overhead, and because it is the longest variable format file logical record length that CMS supports. See the performance tip on page 26 for additional information.

6.3.3.4 Reduce CPU Cost of CGIs

In addition to the recommendations in 6.3.3.1, "Use a REXX Compiler" on page 163, there are steps particular to CGIs that you can take to minimize their CPU costs:.

- Buffer output data into large blocks before sending it from the CGI to its output streams. This has two benefits to reduce CPU utilization. First, it minimizes the number of commands issued to transfer data. Second, it allows the VM TCP/IP server to process large blocks of data at a single time. Both of these aspects have the effect of amortizing fixed costs over a larger number of bytes, achieving a lower cost per byte for processing the data (the basic "buffering is good for performance" rule).

  EnterpriseWeb/VM will perform this operation for you, as controlled by its DEFAULTBLOCK configuration record and the "blocking" field of the MEDIAMAP file (for all types of data). For VM:Webgateway each invocation of CGI WRITE DOCUMENT is a write to the network; for efficiency, you should block up data for this write. See the performance tip on page 26 and 6.3.3.3, "Reduce CPU Cost of Static Documents" on page 165 for more information.

  Do not delay output too long to accomplish this. Be sure to send the headers without waiting for the document body. If there is a stage in the processing that takes substantial wall clock time, flush existing data to the browser to take advantage of the parallel processing nature of the server/network/browser configuration. See the discussion in 6.3.2.2, "Fast Delivery of Start of Document" on page 161 for more information.
- Avoid use of CMS STACK if a REXX variable interface exists. In our experience, most commands that provide both of these interfaces perform better with the REXX variable interface.

VM:Webgateway Specific Considerations The following recommendations apply only to VM:Webgateway CGI environments.

- Run CPU and I/O intensive CGIs in a worker environment. You can use the accounting support in VM:Webgateway to help identify CGIs that have these characteristics.
- Use CGI GETVAR * if more than five CGI variables are needed by the CGI. This interface will fetch all the CGI variables to the local REXX variable pool in about the same amount of CPU utilization as it takes to fetch just five variables. The other variables are thus fetched "for free."
- Avoid fetching CGI variables more than once. It is more efficient to share these variables among your subroutines. If your CGI is completely contained within one source file, then we suggest making use of global variables that are exposed in all of your procedures. For example:

```
/* My Sample CGI.                                         */
/* Show how to share REXX variables globally.             */
Address ''
Signal On Novalue

/* Initialize a global data structure.                    */
Global. = ''

/* And some specific option settings we will use globally.   */
/* We use the convention of never assigning a value to a     */
/* variable name which begins with an "_", thus we can always */
/* use them as part of a stem based variable and know that    */
/* the value fetched from the stem will be the one we expect. */
Global._Debug = (1=1)
Global._Owner = 'LS8105NG'

/* Fetch our CGI variables for everyone to use.             */
'CGI GETVAR * ( STEM GLOBAL._CGIVARS.?'

/* ... main line code ...                                   */
/* Call a subroutine to do some work...                     */
Call Subr1

Exit 0


/************************************************************/
/* Subroutine 1 optionally displays all incoming CGI vars.   */
/*                                                          */
/* We need access to our global data structures, expose them. */
/************************************************************/
Subr1: Procedure Expose Global.
/* Check a global variable to see if debugging is enabled.   */
If Global._Debug Then Do
   /* Make use of the existing global. stem rather than      */
   /* fetching the values again.                             */
   /* Display all the CGI variables on the console.          */
   Tails = Global._CGIVars.?0
   Do While Tails<>''
      Parse Var Tails Tail Tails
      xTail = '?'Tail
      'CP MSG' Global._Owner Tail '= "'Global._CGIVars.xTail'"'
      End /* While Tails<>'' */
   End /* Global._Debug */
Return
```

For a further discussion of this topic and VM:Webgateway CGI Extension programming, see the note on page 72.

6.3.3.5 Reduce CPU Cost of Active Images

When you place an HTML IMG tag inside an A tag, you create an image that you can click to invoke a URL. When you also code an ISMAP or USEMAP attribute on that IMG tag, you create an image that will result in the invoking of one of several URLs. The image becomes an active map for resolving to any of a number of target URLs. The costs of these two attributes vary tremendously.

- Make use of client-side active images in preference to server-side active images. The proposed USEMAP attribute of the IMG tag used in combination with the MAP tag as supported by some browsers (and a part of HTML proposed standards) is more efficient than the ISMAP attribute. USEMAP allows the browser to directly map the x,y coordinate of the image to a URL to fetch. ISMAP requires the invoking of a URL on a Web server to perform this mapping. Thus, compared to ISMAP's server-side active image processing, the USEMAP client-side active image offloads processing from the Web server to the browser and avoids the need to process an extra URL for the selection process.

  The USEMAP approach only works if all browsers that will render the page support this capability. The only way to be sure if the browsers of interest to you support this construct is to test it on each of them. Some HTML references may provide this information to you also. Refer to an HTML reference for more information on the details of these HTML constructs. One such reference can be found online at `http://www.w3.org/TR/REC-html32.html`.

You can find programs on the Web to help you convert your server-side image maps into the directives appropriate for client-side use. For instance, see http://www.popco.com/popco/convertmaps.html.

- When using server-side imagemap processing, order the contents of the imagemap such that more frequently used areas of the image appear earlier in the map file (within the bounds of ability to do so, due to the fact that the map is by definition a “first match” mapping).

6.3.4 Reduce CGI′s I/O

Real I/O, even on today′s highly cached controllers, is a necessary evil to be minimized:

- Make use of the INSTORE and OUTSTORE stages in CMS Pipelines-based CGIs which read a file repeatedly in one invocation. Refer to Melinda Varian′s *Streamlining Your Pipelines* paper, found at http://pucc.princeton.edu/~pipeline/, for more information.
- EXECLOAD data files read repeatedly by CMS Pipelines-based CGIs, which read a file repeatedly across many invocations. Make use of EnterpriseWeb/VM′s PRELOAD configuration directive to help achieve this. In VM:Webgateway you will need to perform such operations either in the CGI itself, or in the PROFILE for the Web server. However, beware of the issues raised in 5.6.6, “Reentrant and Serially Reusable Resources and CGIs” on page 144.
- Make use of REXX′s associative storage rather than making multiple passes over your data files.

6.3.5 Reduce Amount of Data Sent to the Browser

Reducing the amount of data sent to the browser has several benefits. It will tend to reduce all of the following:

- The CPU used by the server (see also 6.3.3, “Reducing Web Server CPU” on page 162 for more information)
- The time to render the document on the browser (see also 6.3.2, “Reducing Browser Rendering Time” on page 160 for more information)
- The CPU used by the VM TCP/IP server (see also 6.4.3.1, “VM/ESA TCP/IP Server” on page 175 for more information)
- The network resources consumed

Send as little data as possible; avoid sending data that does not contribute to the page′s content. These steps will help reduce the time the end users spend waiting for your information to be rendered on their browser. They will also reduce the resources consumed on shared facilities such as the network and the server. When the end user is on an old, low-power PC (for instance a 100 MHz PC or even a 20 MHz MacII), or connected to the network via a low-speed LAN or dial-up connection, these steps can make the difference between your Web site being usable for the user and a frustrated user that will not return to your site no matter how valuable the data you have to share.

- Do not send extra white space (spaces that will be eliminated by the browser when it renders the document). The code in Figure 53 on page 106 and Figure 56 on page 108) is a good example of what not to do. In these examples, by eliminating the trailing blanks from each line we maintain the readability of the code, and yet reduce the amount of data sent by one third to one half. Even more of a reduction can be had if leading blanks are eliminated, which can be accomplished with no readability impact simply by moving the leading quote to precede the first non-blank character on each line. Making these changes to Figure 56 on page 108 would result in the code shown in Figure 65 on page 170, which is just as readable in your CGI as the original code is.

```
DeptForm:
output '<TD VALIGN="TOP">',
       '<FORM METHOD="POST" name="ChgDpt"',
            'ACTION="rxsquery">',
         '<INPUT TYPE="hidden"',
               'NAME="HIDNDEPT" VALUE='''depttag'''>',
          depttag'<BR>',
         '<INPUT TYPE="submit" VALUE="Change Dept">',
       '</FORM>',
       '</TD>'
return
```

Figure 65. Rewritten Department Selection HTML Code

- Do not send HTML comments.
- If appropriate, send an HTTP Last-Modified header and respond to HTTP If-Modified-Since headers (available in the CGI variable HTTP_IF_MODIFIED_SINCE). See the Last-Modified and Expires headers in Table 3 on page 13 and the If-Modified-Since header in Table 1 on page 11 for more information. Also refer to 3.5.2, "Processing SCRIPT Files" on page 56 for an example of their use.
- Audio and video data files tend to be very large, as are animated GIF files. When appropriate, make references to such data files optional, requiring the client to actually request them explicitly via a link rather than inline references. This allows the client to decide if they are worth waiting for.
- Do not serve static content using a CGI or an SSI filetype characteristic. The browser will not be able to cache these results, and thus may require retransmission of the same data repeatedly as a page is revisited by the client.
- Use image tuning software such as GIF Lube or GIF Wizard to reduce the size of your graphics with no loss of quality.

6.3.6 Serializable Server Resource Access

As discussed in 5.6.6, "Reentrant and Serially Reusable Resources and CGIs" on page 144, most of the resources in a virtual machine are not owned by an execution thread, but rather belong to the virtual machine as a whole. When accessing such resources, it is important to do so in an environment where you truly have complete control of the resource, and are not delayed excessively long by a resource allocator waiting for other threads to free up access to the resource.

For example, in VM:Webgateway the server's code makes use of the *MSG system service. As such, it is impossible for a CGI to make use of this resource in the server. However, the resource is available for use in the VM:Webgateway worker-based CGI environments. This type of application is thus forced to run in workers on VM:Webgateway.

Another example is waiting for access to an SFS file that another thread (or user ID) has locked for write. In this case, access to the resource is controlled by an SFS work unit. There can be many of these work units in a single virtual machine at one time. There are no issues of tying up the entire virtual machine by the temporary consumption of this resource. A CGI programmer can make decisions about other aspects of resource control and locating the execution of the CGI in a worker or a server based upon the other characteristics of the program. For instance, if the wait for the resource is expected to be long, and there are no other considerations that would require use of a worker-based execution environment, it would be wise to locate this type of CGI in the server, where it can happily coexist with other similar CGIs without tying up the serially reusable resource of a worker user ID.

A final example involves use of machine synchronous interfaces such as diagnoses and CP commands. If a long-running CP command is going to be run (for instance the CP VMDUMP command as part of diagnosing a CGI, or some of the long-running VM class D commands like CP SPTAPE), this will completely block all other execution on the user ID for the duration of the command. Use of such facilities such should be restricted to a single-threaded execution environment, such as a VM:Webgateway worker or EnterpriseWeb/VM servers.

For further discussion of related issues refer to 5.6.4, “Denial of Service” on page 143 and 5.7.2.4, “Monopolization of Web Server SVM Resources” on page 150.

6.3.7 Reduce CGI′s Storage Needs

This is a relatively low priority goal on most systems today. Typically you will achieve greater benefit by performing other optimizations, and may even want to make use of algorithms that consume more storage in order to achieve a lower runtime. However, even so, the best solution is one that is efficient in both CPU time and storage space.

Our recommendation is to avoid buffering large amounts of data and control structure in your program′s data structures. In CMS Pipelines, try to avoid buffering stages such as BUFFER and SORT when processing large amounts of data. In REXX, avoid large numbers of variables, especially large stems, and storing large values (that is, avoid long strings).

6.4 Performance Issues for Web Server Administrators

In addition to having to face all of the same issues that application users (see 6.2, “Performance Issues of Browser Configuration” on page 157) and application writers (see 6.3, “Performance Issues for Application Writers” on page 159) have to face, the systems programmer or systems administrator has additional issues to contend with.

6.4.1 Install Most Recent Software Levels

Software development in the Web area is moving fast. You will typically not only find advances in function, but also in capacity, efficiency and reliability characteristics. You should endeavor to keep up with recent releases of software not only in the base operating system and its services (such as TCP/IP), but also in all application server environments (such as the Web server and data base engines). In particular, we strongly recommend upgrading your VM TCP/IP server to Level 310 (or later). (See 1.8, “Environment” on page 6 for information on the releases of software the work in this book is based on.)

6.4.2 Performance Data Analysis

Compared to many other systems, VM/ESA applications are blessed with a wealth of information sources about aspects of their performance. Each of these sources provides a slightly different view of the performance and resource consumption aspects of an application. They each also have their own impacts upon the system being measured, because the very act of measuring the performance and resource utilization of a system tends to, in some way, alter that performance and resource utilization. Thus, in choosing an information source for your data analysis, you need to consider not only which of these sources are most easily available for your site and the application in question, but also which will cause the smallest perturbation in the parameters you wish to measure.

Often the simplest and most direct measures are forgotten or overlooked. But they can often provide a wealth of information and insight, especially when performance problems are caused by error conditions. Thus, it is common to find quick, simple solutions to the class of problems revealed by these sources.

6.4.2.1 Console Logs

Console logs contain just such information. They will typically record error conditions that arise during the servicing of a request. Error handling and recovery is often an expensive and time consuming process. In addition, it is common to perceive a failure to deliver a document in a timely manner as a performance problem, when in fact it may be a more basic failure of the application to ever return the results due to error conditions. These failures may show up as informational or warning messages during startup of the server, or they may show up during the actual serving of a transaction. Alternately, you may simply collect information on which transactions are taking too long from information such as time stamps in the console. Keep your eyes open for any and all hints from this source, as insight can come from almost any of the normal messages, as well as the abnormal ones. One common case of this class is failures in the configuration and operation of the DNS server. Failures in this area of a Web server environment can often add 30 seconds to the servicing of a transaction, which, except for this delay, runs correctly. (See 6.4.8, "DNS Impacts on Web Server Performance" on page 181 for more information.) One of your first and most important sources of information is thus one of the easiest to obtain, and comes at a cost that almost everyone accepts as a given.

6.4.2.2 HTTP Transaction Logs and Analysis Tools

For a Web server, the next most obvious source of information are your HTTP logs. Both EnterpriseWeb/VM and VM:Webgateway record this information in the CERN/NCSA standard log format. With EnterpriseWeb/VM, how this information is recorded is controlled by the LOGPIPE configuration directive, and what is recorded (standard or extended log format as defined by NCSA) is controlled by the LOGFORMAT directive. With EnterpriseWeb/VM the information is recorded as one record in the log file, just like on UNIX-based servers. With VM:Webgateway the information is recorded in multiple console messages in the extended format as defined by NCSA′s server. Since a single message rarely holds the entire contents of a log file record, Sterling Software, Inc. provides the VIWLOGEX tool to process a server console and reconstitute the multiple messages into a single record.

As with the console logs, you can often easily discern interesting and revealing information from direct reading of these logs. Correlation of the information in them with other messages recorded on the console may also make the information in the log more useful. Unfortunately, neither vendor documents the format of these logs, or where to find a formal definition of this format. Luckily, we have the Web itself to help us discern this information, although we found no definitive definition of the format for all servers. In general, a log record is composed of the following values, in order:

1. Client host, recorded as either the DNS name or the IP number
2. RFC 931 "IDENT" user ID value or "-"
3. User ID from any HTTP authorization header present in request or "-"
4. Date and time of request, enclosed in square brackets, with the value in GMT or the local time zone depending upon the server in question
5. HTTP request line, enclosed in double quotes and potentially containing blanks
6. HTTP response status code
7. Bytes of response sent, where the count may or may not include the data in the HTTP response header depending upon the server in question
8. Optionally present "extended" fields, enclosed in quotes:
    a. HTTP Referer header from request or ""
    b. HTTP User-Agent header from request or ""

In addition to manual inspection of the raw log data, you can also gain information by performing statistical analysis of the data. For instance, such analysis will quickly reveal which URLs are most frequently visited. This allows you to concentrate your efforts on what may be the more important CGIs and tuning parameters, rather than on ones which may not even be used. In addition, analysis of frequency of HTTP response codes may reveal poor use of caching in browsers, and similar site-wide issues that are not directly related to the Web server's configuration. Given the information on the format of these records, it is possible to construct your own data reduction tools to perform this statistical analysis. However, you may find that such tools already exist on the network, and that it is quicker to pick up such a free tool rather than developing your own or buying a commercial package. A quick search of the Web reveals that there are many such tools available. One that caught our eye is Analog, which, based upon network surveys, describes itself as "The most popular log file analyzer in the world." More information on this tool can be found at `http://www.statslab.cam.ac.uk/~sret1/analog/`. Additional information on log file analysis tools can be found at

```
http://dir.yahoo.com/Computers_and_Internet/Software/Internet...
   .../World_Wide_Web/Servers/Log_Analysis_Tools
```

or with your favorite search engine.

6.4.2.3 Accounting Cards and Resource Consumption Data

An often ignored source of information are accounting cards. While the normal VM LOGON and LOGOFF records are not of much use in this particular case, both EnterpriseWeb/VM and VM:Webgateway allow you to generate application-based accounting cards that could be of great value.

VM:Webgateway provides cards containing the following information:

- User ID of the Web server SVM
- User ID that owns the data that was served
- Date and time of the record
- User ID supplied by the Web browser user, with an indication if it was validated against the VM directory

- IP address and TCP port of the client and server
- A unique counter for use in correlation with the server′s console
- Virtual CPU and I/O used by the SVM and any worker that may have been used
- Byte count of data served from the TCP port
- Additional flag bits and bytes as described in the product documentation

For EnterpriseWeb/VM there is no standard definition of what an accounting card contains because their creation is completely performed by site-provided code. While this provides you with more freedom to make the card useful for your sites′ needs, it also presents you with more implementation work to get benefits from this source of information. However, it is easy to see how most of the information provided by VM:Webgateway could also be collected and summarized in EnterpriseWeb/VM.

With this information and the information from the console log, one can quickly zero in on the URLs that are consuming the most resources either individually or in aggregate. Unfortunately, no analysis tools exist to aid in this process.

Further information on accounting card generation can be found in your product′s documentation and in 6.4.9, "Accounting Card Generation Cost" on page 182.

6.4.2.4 Real Time Monitor VM/ESA (RTM VM/ESA)

We are mentioning this data source for completeness, as no source of information should be ignored. However, we do not believe that it will provide you with any significant information, primarily due to its real-time nature and lack of the ability to easily correlate its information with the workload details in your Web servers. That said, you may still find its information of use, as it may help to identify in real time when your Web servers are incurring heavy load, and thus point to time frames for further analysis of other data sources.

6.4.2.5 VM Monitor Data and Reduction Tools

The VM monitor data stream contains information from two basic sources, CP and monitor-enabled applications running in SVMs.

The CP monitor data stream and its reduction tools provide a wealth of data on the operation of your VM system. All of the normal performance analysis techniques apply for Web servers when using these tools, just as they would for any other SVM on your system. The details of these are beyond the scope of this book, and we refer you to the normal sources for VM measurement based upon VM monitor data for further information.

The application data which some SVMs add to the VM monitor data stream is also a potentially wealthy source of information. Whether this potential is realized or not depends upon two factors, the SVM actually placing valuable, documented data into the data stream, and suitable analysis programs for its reduction. It is these two points that make the CP data in the data stream of so much valuable use, and the same can be said of selected other applications (such as SFS and VM TCP/IP), which make use of the data stream. It is also these two points that make this data source for Web servers of questionable use.

While EnterpriseWeb/VM has the ability to produce the data, Beyond Software Inc. has not documented what information it is placing in the data stream (stating that it is proprietary information), and only one reduction tool vendor, Velocity

Software, is capable of performing analysis of the data. As a consequence, while the data exists, it has no value as useful information unless you explicitly purchase products from these two vendors, and then only assuming that the reports produced by Velocity′s product are fully documented (unlike the raw data which Beyond Software Inc.′s product produces). Since we did not have access to the Velocity product, we could not evaluate if the potential value of this information source has been realized by these reports.

VM:Webgateway from Sterling Software, Inc. does not today make use of this data stream.

We hope that in the future both of these vendors will not only produce a fully documented data stream, but that multiple data reduction tools will also exist for their products.

6.4.3 VM Tuning Knobs for Service Virtual Machines

As always, it is critical that you properly set VM′s tuning knobs for all of your SVMs, including the ones associated with Web serving:

- Set CP QUICKDSP ON for all servers in any way associated with interactive or multiuser work loads. These include:
  - All of the Web server user IDs
  - All worker user IDs on VM:Webgateway
  - All networking user IDs such as TCPIP and VTAM
  - All database user IDs such as DB2
  - All SFS server user IDs
- Set CP SHARE

  VM:Webgateway

  As you would for other multiuser servers. While we found that a setting of 100 on our idle test system worked fine in a production or more heavily loaded environment, a higher setting is desirable.

  EnterpriseWeb/VM (and Webshare)

  Beyond Software Inc.′s recommended 1200 is very likely far too high. Since these are single-URL-at-a-time servers, we recommend leaving the SHARE settings for these servers at 100 or perhaps slightly higher.
- If you are analyzing your CP MONITOR data, run with a CP MONITOR sample rate of 1 minute. If you are not analyzing it, do you still want to collect it?
- When running on VM/ESA 1.1.0 up through (at least) 2.3.0 avoid using Single Console Image Facility (SCIF, VM′s “secondary user” support) for user IDs that generate high volumes of console output. CP injects a hard 1-second wait when more than 23 I/O ops/sec are attempted to such a user ID.

6.4.3.1 VM/ESA TCP/IP Server

As you add load to your Web server, you are also adding load to your VM TCP/IP SVM. You should review your tuning of this machine on a regular basis. Refer to the information in Appendix A of *Web Server Solutions for VM/ESA*, SG24-4874, the IBM documentation on this product and the presentations of Bill Bitner found at http://www.ibm.com/s390/vm/devpages/BITNER/ for more information on this topic. Also see the OS/390 informational APARs II08848 and II08849 for some general TCP/IP performance tuning tips. Some knobs to look at include:

- Sufficient buffer allocations -- the negative effects of over-allocation are eliminated with VM TCP/IP level 310.
- Maximum Segment Size -- generally, bigger is better.

- Datagram size -- normally, bigger is better, especially in an intranet environment.
- Network attachment device settings (if any are externalized).
- Server CP settings.

6.4.4 Tune Your I/O Systems, Especially SFS

No matter what else you do to improve your system′s performance, if your I/O system′s performance is off, you will suffer, especially in a Web server environment.

- We recommend that you serve documents out of local SFS servers rather than remote SFS servers. We have observed that the APPC communication with the SFS server is, for certain operations, virtual machine synchronous. This showed up even in an environment where asynchronous interfaces to SFS are being used. In our experience, this communication can represent a significant percentage of the wall clock time to serve a URL when serving data from remote SFS servers.
- Take steps to reduce and optimize normal CMS file system I/O. All minidisk I/O done by CMS is via VM diagnoses, and all SFS I/O done by CMS to files that are not in a data space is done with VM′s IUCV or APPC interfaces. In both cases, CMS performs this I/O in a virtual machine synchronous manner. This means that for normal CMS-driven I/O, there is no overlapping of any I/O wait time with other processing in the CMS virtual machine.

  You can reduce or eliminate these waits by a number of techniques:

  – In SFS use DIRCONTROL directories for CMS ACCESSed SFS directories and VM′s data space support on the SFS and Web server SVMs. This will have the largest effect for Webshare, EnterpriseWeb/VM and WEBSHARE compatibility mode CGIs and CMS domains in VM:Webgateway. It will have no real benefit for VM:Webgateway in other configurations. Be aware that while improving performance of I/O, this may have negative side effects. For instance, your Web server SVM will no longer immediately notice changes to files on these directories the way it does with FILECONTROL directories. Lastly, the use of data spaces in VM may have adverse effects upon your system′s real storage utilization, leading to higher overheads in other aspects of your system′s performance. As always, use caution when making such system tuning decisions.

    For these reasons we recommend against the use of DIRCONTROL directories. See 6.4.7.1, “EnterpriseWeb/VM Configuration” on page 180 for more information and additional recommendations in this area.

  – Use VM′s Mini Disk Caching (MDC) support or controller hardware cache for all high use and important application minidisks.

  – Make use of V-disks as appropriate, especially for both Web server and worker CMS A-disks.

  – Tune the server for SFS and BFS use Refer to the information in Appendix B of *Web Server Solutions for VM/ESA*, SG24-4874 for more information on this topic. Also refer to the many fine presentations by Bill Bitner that you can find at `http://www.ibm.com/s390/vm/devpages/BITNER/`.

  – EXECLOAD all server REXX code. You will need to determine what to load for Webshare. EnterpriseWeb/VM provides a PRELOAD configuration directive to aid in this process. VM:Webgateway completely automates this processing for you.

  – EXECLOAD all “user exit” code that is REXX based.

– EXECLOAD data files read repeatedly by CMS Pipelines from the CMS search order. EnterpriseWeb/VM provides a PRELOAD configuration directive to aid in this process. However, beware of the issues raised in 5.6.6, “Reentrant and Serially Reusable Resources and CGIs” on page 144.

VM:Webgateway avoids many of these delays by using a thread synchronous I/O technique for all I/O performed in service of URL roots in the SFS, BFS and MDISK domains. This means that while the server is waiting for I/O to complete for the servicing of one URL, it can perform work for the serving of other URLs in parallel. I/O for the CMS domain in VM:Webgateway is still subject to these considerations.

EnterpriseWeb/VM runs one thread per virtual machine. By doing this, it can take advantage of CP′s scheduler, which is extremely efficient. Also, since each is a separate virtual machine, I/O and/or processing can be done in parallel; this takes full advantage of S/390 architecture and N-way processors.

6.4.5 Tuning the Web Server′s Virtual Storage Size

Be certain to set the virtual storage size of your Web server and worker virtual machines to be large enough to handle peek needs. For EnterpriseWeb/VM and Webshare, since both of these Web servers are single-threaded virtual machines, this is relatively straight forward. You simply need to establish the storage needs of the CGI with the largest storage needs plus the CMS and product code overheads and use the maximum of that number or the vendor′s recommended storage size. The same can be said for worker virtual machines for VM:Webgateway, although we recommend that you treat these machines as you would VM:Webgateway SVM that is single threaded (as outlined below). The VM:Webgateway SVM is more complex, as it is a multithreaded virtual machine.

There are two basic approaches for sizing a virtual machine for an application environment. The simplest is to fire up the application and verify that it works in some virtual machine size, and then to declare that size as the right one. This is quick and easy, and often incorrect. However, the second requires much more effort, as you must answer the following questions:

- What are CMS′s storage needs, including its low storage use (NUCON primarily), the CMS NSS, the CMSINST DCSS, FSTs for accessed disks and directories, and its storage management tables (in a 2 GB virtual machine this adds up to approximately 4 MB). In today′s CMS a reasonable start estimate for a typical server is 8 MB.
- What DCSSs will be used, and what are their sizes and locations (to verify that two DCSS that are simultaneously needed do not overlap, and to verify that CMS′s storage layout is not so fragmented as to make it incapable of returning large blocks of storage as needed).

Tip

Since you have now identified all of the DCSSs that your virtual machine is known to need, update the user ID′s PROFILE EXEC to either SEGMENT RESERVE them or to actually attach them by making use of the appropriate product interface (as is required by products such as VM:ProRexx). This will insure that CMS does not make use of the required storage locations before the DCSS is used in the application, and thus insure that the DCSS can be attached when it is needed.

- What are the storage needs of the Web server code? This will include not only the code, but also all EXECLOADed programs and data, data and database caches, network buffers, control data structures, and similar considerations.
- What is the maximum simultaneous active URL resolution count? In Webshare-based servers and VM:Webgateway′s workers, this is 1.
- What is the maximum storage need of the most storage-intensive CGI to be run?

These values can be used in the following formula to estimate the maximum storage needs for your server:

Virtual size = CMSS + DCSS + (URLmax * CGImax)

where:

- CMSS = CMS′s storage needs
- DCSS = DCSS storage needs
- URLmax = maximum URL count
- CGImax = maximum CGI storage

How to reduce the virtual machine size needed for a server?

- Eliminate SEGMENTs that are not actually needed, especially those below the 16 MB line, as this storage is often in short supply and is required for certain classes of storage requests.
- Reduce the storage needs of your most storage-intensive CGIs. See 6.3.7, "Reduce CGI′s Storage Needs" on page 171 for more information.

6.4.5.1 VM:Webgateway SVM Virtual Storage Considerations

The *VM:Webgateway Getting Started* manual from Sterling Software, Inc. contains an appendix that will lead you through the complete analysis of sizing this virtual machine. As you work through this process, remember that virtual storage in VM is cheap, and it is far better to make the virtual machine too large than too small.

It is critical for this server′s performance to have a data base cache size that is greater than the real data base′s size on disk. This is controlled by the DATABASE record in the VMWEBSRV CONFIG file. Since the typical size of the data base is 60-100 records, the shipped default of 1000 is more than sufficient for the server′s performance. However, this default also wastes a large amount of virtual address space.

If you wish to recover some of that address space, set the DATABASE record′s CACHE to the next higher multiple of 100 greater than the record count for the VMWEBSRV PFMDB file, with a minimum size of 100 and a maximum of 8192.

6.4.6 Tuning the Number of Servers and Workers

As stated in *EnterpriseWeb Secure/VM Installation Guide and Reference*, "the single most effective thing that you can do to improve EnterpriseWeb/VM performance is to ensure that you have enough servers to handle the load." With EnterpriseWeb/VM and Webshare we suggest that you start with more servers than you believe you will ever need. The reason for this is that each server will need to be authorized for access to the data (SFS authorizations and similar considerations). Once you have an existing number of servers, it is difficult to add more to it and get all of the needed authorizations correct. Beyond Software Inc. suggests the following formula for estimating the correct number of servers:

Number of servers = avnumopp * numofsimuser

where:

- avnumopp = average number of objects per page
- numofsimuser = number of simultaneous users

We believe that this is too conservative for this product, and that the formula should be modified to read:

Number of servers > (maxumopp * maxnumofsimuser)

where:

- maxnumopp = maximum number of objects per page
- maxnumofsimuser = maximum number of simultaneous users

With VM:Webgateway we suggest that you start with one server and rely heavily upon the use of workers. We suggest you add sufficient workers so as to always have at least one worker unallocated.[9] These workers can be added as needed, as they do not need any authorization to access data (since they gain this authorization by taking on the authorizations of the CGI owner). If you have performance problems in this environment, we suggest that you contact Sterling Software Customer Services for aid in resolving them. We suggest that the following very conservative formula be used to determine the number of workers to configure:

Number of workers > (maxnumopp * maxnumofsimuser)

where:

- maxnumopp = maximum number of objects per page
- maxnumofsimuser = maximum number of simultaneous users

We suggest that you run VM:Webgateway OfficeVision Interface-based services out of a separate VM:Webgateway server from the other load on your system. Likewise, if you have another major application, you may wish to create a dedicated VM:Webgateway server for its use. We suggest that for good human factors you make use of *virtual hosting* to create a separate DNS name for each of these servers, allowing them to still be served out of the default TCP port number, and even to be moved to a separate VM host without any impact upon the externals of their DNS host name.

Also refer to the information in Appendix A of *Web Server Solutions for VM/ESA*, SG24-4874 for more information on this topic.

[9] At the time of this writing there is no easy method for determining if VM:Webgateway has run short of workers to meet current needs. We suggest that you call Sterling Software Customer Services if you feel that you need such a facility.

6.4.7 Web Server SVM Configuration

Each of the Web servers has unique configuration options that present possible knobs for controlling the performance characteristics of the server. Some of these knobs have been discussed elsewhere in this chapter. We have created this section to touch upon those that are not otherwise covered.

Both EnterpriseWeb/VM and VM:Webgateway have mechanisms to disable generation of the CERN/NCSA logs. *EnterpriseWeb Secure/VM Installation Guide and Reference* documents the process for EnterpriseWeb/VM. You will need to contact Sterling Software Customer Services for further information for VM:Webgateway, which uses the same method as in VM:Secure. Even though this saves some small amount of server resource, we do not recommend that you disable the collection of this information. We believe that you and the customer support organizations of these two products will find this information too useful to justify the marginal potential performance benefits of disabling it.

6.4.7.1 EnterpriseWeb/VM Configuration

Many of the following configuration recommendations are the defaults and the settings recommended by Beyond Software Inc.. We have attempted to include all of them for completeness.

- Configure as short a list as possible in the USERFILEPOOL and USERWEBDISKS configuration records. Order the entries in these to place the most likely to succeed first. Consider the use of a POOLEXIT as an alternative. This would be set by a USEREXIT FILEPOOL record in a EWXFPOOL EXEC.
- Also see the information in the EnterpriseWeb/VM “Turbo” Mode section of *EnterpriseWeb Secure/VM Installation Guide and Reference* for further ideas on improving the performance of this product.
- Install the product into a shared segment.
- Be sure to use multiple servers. See 6.4.6, “Tuning the Number of Servers and Workers” on page 178 for more information.
- Be sure to make use of the SSL Session ID caching server.
- Configure ACCESSINT to the number of seconds you would like to check minidisks and SFS DIRCONTROL directories for new or changed information. Every ACCESSINT time interval a assembler routine runs to check without reaccessing the minidisks or SFS DIRCONTROL directories if there are new or changed data. EnterpriseWeb/VM only reaccesses the minidisks and SFS DIRCONTROL directories if there are new or changed data.
- Configure APPLMON OFF, unless you really need and want the CP MONITOR data it generates. See 6.4.2, “Performance Data Analysis” on page 172 for other considerations.
- Configure DEFAULTBLOCK 61440 (the maximum, and default value). This will significantly reduce server and VM TCP/IP overheads. Be sure to also be careful about overriding this setting on any entries in your $EWEB MEDIAMAP file. See 3.1.3, “Sample CGI Program for VM:Webgateway” on page 24 for more information on this topic.
- Configure IDENT OFF.
- Configure IF_MODIFIED_SINCE ON.
- Configure LAST_MODIFIED ON.

6.4.7.2 VM:Webgateway Configuration

Most of the recommendations for this product have been covered in other sections of this chapter.

Run the listener in a “bound” mode for a single local IP address (running multiple listeners, one for each possible IP address, if necessary). This eliminates the work of looking up the local IP address (and possibly the DNS name for it) for each connection (as is required for a listener that is not bound to a specific local IP address). Alternately, consider setting up the product with a modified TCPIP DATA file to disable DNS resolution. See 6.4.8, “DNS Impacts on Web Server Performance” for more information on this topic.

6.4.8 DNS Impacts on Web Server Performance

Poor performance of your Domain Name System (DNS) server will have dramatic effects on performance of your system′s Web server. In a Web server, the DNS server is used to perform what is known as a “reverse lookup” to resolve an IP address back to a DNS name. This is accomplished via the configuration of special records in the DNS server to perform this mapping. It is not uncommon to find that the DNS server may not be configured to respond to such requests in a timely manner. More basically, the DNS server may not even be configured for good performance for any requests, or may easily become overloaded. Lastly, it is possible that the configuration of VM/ESA′s TCP/IP product may be incorrect, leading to the queries being directed to the wrong server. Any or all of these can lead to long service times for all URLs.

If you do not wish to perform security based upon DNS names (see 5.5.3, “Can You Trust an IP Address or DNS Name” on page 138 and 5.6.5, “Setting Security Profiles for URL Trees” on page 143 for more information on this topic), and do not care about having these names in either the CERN/NCSA log or in the URLs seen in some redirections, then disable DNS name resolution. This is an extreme step, and we do not recommend it. We believe that it is far more reasonable and cost effective to put time and effort into establishing a fast and reliable name service for your site.

In VM the configuration of what DNS server to use is accomplished by configuration records in the TCPIP DATA file. They are:

NSINTERADDR
Specifies the Internet IP address of the name server. LOOPBACK (14.0.0.0) is the default value, and represents a name server on the local VM system. If a name server will not be used, then do not code an NSINTERADDR statement to disable DNS resolution.

NSPORTADDR
Specifies the foreign port of the Name Server. It has a default value of 53, which is normally correct for a site.

RESOLVEVIA
Specifies how the resolver is to communicate with the name server. TCP indicates use of TCP virtual circuits. UDP indicates use of UDP datagrams. The default is UDP. UDP is the lower cost alternative and is typically the right setting.

RESOLVERTIMEOUT
Specifies the time in seconds that the resolver will wait to complete an open to the name server (either UDP or TCP). The default is 30 seconds. This is typically unimportant in a properly configured DNS

system, as requests will never time out. However, if there are problems in the DNS system or its configuration on VM, this value will lead to unacceptable Web server performance. Changes to this value should be considered in the light of all the TCP/IP servers and clients you run on VM. While a small value may be appropriate for Web servers, it may be inappropriate for an e-mail server such as SMTP.

RESOLVERUDPRETRIES
Specifies the number of times the resolver should try to connect to the name server when using UDP datagrams. The default is 1. As with RESOLVERTIMEOUT, changes should be considered in light of the total needs of your system.

You may find it appropriate to provide a different and unique copy of TCPIP DATA for servers and clients with special needs. The long term maintenance cost of such a decision should also be considered, as the cost of diagnosing such a file that was not kept in synchronization with the master copy can quickly override the benefit of its original creation. It is better to expend resources in creating and maintaining a fast, reliable DNS server for your site than going down the path of working around a problematic DNS server.

For EnterpriseWeb/VM the REVERSEDNS configuration record is an additional level of control if these mappings will be attempted.

6.4.9 Accounting Card Generation Cost

In general, if you can avoid some processing overhead in a Web server, you should do so. Accounting card generation falls in this category. If you do not need accounting cards, they by all means do not generate them However, if you do, then you should do so in a manner that generates as little additional overhead as possible.

For EnterpriseWeb/VM you will need to generate your own code to collect and build accounting cards. This is accomplished in the TRANEXIT (EWXTRANS EXEC). As with all exits on any product, you should exercise care in making sure that the coding of this exit is not only reliable and robust, but also efficient.

For VM:Webgateway these matters have already been addressed for you. To enable accounting you need to simply use the CONFIG ACCOUNT command, as documented in the product′s online documentation.

See 6.4.2, “Performance Data Analysis” on page 172 for other considerations.

6.5 Performance Summary

While we have presented you here with a list of pundits′ questions and “rules of thumb” suggested answers, remember that few of these should be applied to your site without first performing your own analysis of both questions and answers.

The single most important question, of course, is: what is right for your site. Remember that this is not an area of static questions or answers; you should reexamine both on a regular basis, because “things change.” We also hope that you will approach the problem creatively.

6.6 References

The following are sources that you may also find of use. While some of these resources are non-VM based, many of their tips and techniques apply in a VM environment.

- *Web Server Solutions for VM/ESA*, SG24-4874.
- *Exploiting Recent CMS Function: A User's Guide to CMS Application Multitasking*, SG24-5164
- *A Comparison of HTTP and HTTPS Performance* by Goldberg, Buff, and Schmitt, available at `http://www.cs.nyu.edu/artg/research/comparison/comparison.html`.
- *HTML 3.2 Reference Specification*, available at `http://www.w3.org/TR/REC-html32.html`. While this is not an Internet standard, it is a reasonable overview of the HTML language.
- *HTML 4.0 Reference Specification*, available at `http://www.w3.org/TR/REC-html40.html`. While this is not an Internet standard, it is a reasonable overview of the HTML language.
- *DNS and BIND*; by Albitz, Paul and Cricket, Liu; published by O'Reilly.
- `http://www.ibm.com/s390/vm/perf/`
- `http://www.ibm.com/s390/vm/devpages/BITNER/`
- Here is an example of what IBM has done in this area for one of their OS/390 servers: `http://www.networking.ibm.com/icserver/pub/icstun46.htm`.
- You will find extensive information on tuning CMS Pipelines based applications at `http://pucc.princeton.edu/~ pipeline/`

Chapter 7. Desktop Web Publishing to VM Web Servers

This chapter focuses on how to publish Web pages, from desktop Web publishing tools to the VM Byte File System (BFS), even though it is possible to publish to SFS and minidisks. Publishing to the BFS will be accomplished through the use of either the Network File System (NFS) server or the File Transfer Protocol (FTP) server. We discuss the following desktop Web publishing products:

- Netscape Composer
- Microsoft FrontPage Express
- NetObjects Fusion

Web page or Web site development with the use of a desktop Web publishing tool is much easier than writing HTML tags. These PC-based tools give you the possibility of designing a whole Web site or Web page without writing any HTML tags at all. Furthermore, you have the What You See Is What You Get (WYSIWYG) possibilities on the desktop, so you can see immediately how your just-developed Web site or Web page looks. The use of such a Web publishing tool does not limit the actual Web site to be served from a desktop Web server. We will show you how to publish from the desktop to a VM Web server utilizing NFS and BFS. The Network File System server (VMNFS), available with TCP/IP 310, supports the mounting of Shared File System (SFS) and (BFS) directories, in addition to CMS minidisks. Current information on NFS on VM can be found at http://www.ibm.com/s390/vm/NFS/.

The publishing of long-named Web pages to VM on CMS minidisks or SFS file pools requires file name shortening. This name shortening requires later manual adjustment to previously established hyperlinks in other pages. The use of the BFS instead allows file names of up to 255 characters. Spaces and a number of other special characters should still be avoided, because they are classified as unsafe and need to be encoded to be used. The unsafe classifications are stated in the HTTP RFC1945 and the URL RFC1738. The publishing to VM from a desktop Web publishing tool is made much easier by the presence and use of the BFS. NFS is the easiest tool for the publishing user and recommended versus using FTP to place the file into BFS. The reason that NFS is the easiest publishing target is that NFS appears to the desktop as another PC network hard drive.

When publishing, or moving the file from the PC to VM, the file will either be translated from ASCII to EBCDIC or moved in binary mode. A binary mode transfer leaves the HTML page in its original ASCII character set. We recommend the binary mode transfer. There is also a slight performance boost in moving the page in binary mode, which is the result of avoiding a character translation on the file. This benefit is obtained first when publishing and second when the page is served by a Web server.

In addition, when publishing using the FTP capabilities of either Netscape Composer or FrontPage, there is no choice: The HTML page will be moved in binary mode.

Pages developed on VM, input to CGIs on VM, manually transferred files and sample pages from the Web server products on VM are in EBCDIC mode. Therefore the Web servers on VM come preconfigured with the standard Web page file types set to .HTM or HTML and also with EBCDIC to ASCII translation.

If both desktop binary pages and EBCDIC pages are to be served, then the Web server needs to know what file type is what format to serve the page in the right format. To handle the binary type format, the VM Web servers come with preconfigured file types. For VM:Webgateway Web Server the default file type is HTMLBIN and for EnterpriseWeb/VM the default file type is HTMLA.

To utilize a non-standard desktop binary file type instead requires changing the file type when you save or publish pages. This also needs attention when looking for pages while creating links. You need to change the file type filter from `HMTL Files (*.htm, *.html)` to `All Files (*.*)` to list the pages that have been saved with the .HTMLBIN or .HTMLA file types.

If you plan to use the desktop default file types, then you have to change the VM Web server default settings to reflect this. One way of doing this could be by changing file type .HTM to the binary ASCII method of serving, and using file type .HTML for the EBCDIC pages.

The following is an example:

.HTML	All EBCDIC Web pages and links
.HTM	All binary (ASCII) Web pages and links

Before changing the file type definitions, you need to make sure that all your EBCDIC pages have a file type of .HTML. If you change the VM Web server defaults you will not need to change the file type when publishing or searching for files to link on your desktop Web publishing tool.

At present, only Sterling Software′s VM:Webgateway Web Server Release 2.2 provides built-in support for serving Web pages and executing CGIs from a BFS directory.

Beyond Software′s EnterpriseWeb/VM contains support for SFS file pools, but this file structure limits the file name and file type to 8 characters. To serve pages from the BFS from either EnterpriseWeb/VM or Webshare, CGIs need to be written.

7.1 Summary of Steps Needed for Publishing

Following is a summary of the steps necessary to publish from a desktop Web publishing tool to a BFS directory. Either NFS or FTP will be used to move the file from the desktop to VM. It is not the intention of this chapter to cover the installation of NFS, FTP, BFS, VM:Webgateway or EnterpriseWeb/VM. The summary points to the appropriate chapter where we discuss what steps are needed in addition to the base installation, which we assume is already done.

- A BFS file pool is available on VM.
  - Check that the PTFs in Table 10 on page 187 are applied.
  - Sample file pool server directory entries can be found in 7.3.1, “Standard File Pools” on page 188 and in 7.3.2, “Test File Pool” on page 190.
- An NFS server is available on VM.
  - The directory entry we used can be found in 7.3.3, “VMNFS” on page 191
- Additional POSIX class directory entries are added to the directory. See 7.3.4, “Added Directory Statements” on page 192:
  - GLOBALDEFS (see Figure 71 on page 192)

  - FTPSERVE (see Figure 72 on page 192)
  - User entry (see Figure 73 on page 192)
- Enroll the NFS user into the BFS file pool.
  - See 7.5.1, "ENROLL User in a File Pool" on page 195.
- A Web server is available on VM.
  - If you have chosen to change the HTM file type then see 7.6, "Changing the HTM File Type" on page 196.
  - VM:Webgateway Web Server Release 2.2 is available.

    Set up a user path in VM:Webgateway. See 7.6.1, "VM:Webgateway BFS Access Setup" on page 196.
  - EnterpriseWeb/VM using SFS.

    Grant the EnterpriseWeb/VM server read access to user pages. See 7.6.2, "EnterpriseWeb/VM SFS Access" on page 197.
- NFS client software is available on the desktop.
  - See 7.6.3, "NFS Client" on page 198.
- A new page file type is added to DOS.
  - See 7.6.4, "Set Up htmlbin or htmla" on page 200.
- Change the setting in Windows Explorer to show DOS extensions.
  - See 7.6.5, "Show DOS Extensions" on page 200.
- A desktop Web publishing tool is available. Following are the products we used:
  - See 7.7, "NetObjects Fusion 3.0" on page 200.
  - See 7.8, "Netscape Composer 4.07" on page 203.
  - See 7.9, "Microsoft FrontPage Express" on page 206.

  Note: Free trial versions were used for our tests. These releases are known to often have limitations or problems. We would recommend purchased releases for more than a trial use.

7.2 Overview of the BFS

BFS files and directories are stored in file pools that are managed by the file servers. The file servers also support the Shared File System (SFS). There are many advantages that both the BFS and SFS have over minidisks. The primary advantage of BFS over both CMS minidisks and SFS for desktop Web publishing purposes is the long identification support (file name and file type).

Note: To utilize BFS for publishing, the following PTFs are needed for VM/ESA V2.3.0, CMS 14 and TCP/IP Function Level 3.1.0

Table 10. PTFs Required for Publishing to BFS

PTF	APAR	Description
UM29016	VM61794	BFS synchronous I/O problem, CMS 14
UQ20816	PQ14734	FTP incorrectly translated binary files, TCP/IP 310
UQ20817	PQ18558	FTP to BFS resulted in 0 length files, TCP/IP 310

SFS is the component of VM/ESA that provides a hierarchical, managed file system to CMS users, as opposed to the traditional CMS minidisk file system. If you have performed a DDR install of VM/ESA Version 2 or later, then a sample BFS structure and SFS will have been installed. Otherwise you may have to build a BFS structure or install the Open Edition Shell.

Unlike the conventional record-oriented CMS file systems, both minidisk-based and SFS, BFS treats files as nothing more than an ordered collection of bytes like any other Unix file system. BFS has semantics, file naming conventions, and file structures that are different from the conventional CMS file systems.

BFS allows files to be created and used in a Unix-style format. Like files in SFS, BFS files are stored in CMS file pools. VM/ESA also provides a set of CMS Pipeline stages and a set of OPENVM subcommands to support BFS. Some native CMS commands (for example, XEDIT) also support access to BFS files and directories, using extensions to the CMS record file system interface.

7.3 Directory Entries

Following are the directory entries or added directory statements for the servers supporting NFS, BFS, FTP and the CMS user. The directory maintenance product that was used was Directory Maintenance VM/ESA Release 5.0.

7.3.1 Standard File Pools

VM/ESA Version 2 Release 3 is shipped with a number of SFS file pools ready to be installed by the installation process. Three of these predefined file pools are:

VMSYS	SFS file pool for system resources; for example, program products
VMSYSU	SFS file pool for general user data files and directories
VMSYSR	CRR recovery server file pool for coordinated resource recovery across multiple file pools and improved SFS performance

Here is the USER DIRECT entry for the VMSYSU file pool server, VMSERVU, that we used here at the ITSO Center in Poughkeepsie for this residency.

```
USER VMSERVU XXXXXXXX 32M 32M BG 1
   ACCOUNT ITSXXXX
   IPL 190
   POSIXOPT SETIDS ALLOW
   IUCV ALLOW
   IUCV *IDENT RESANY GLOBAL
   MACH XC
   OPTION MAXCONN 2000 NOMDCFS APPLMON ACCT QUICKDSP SVMSTAT
   SHARE REL 1500
   XCONFIG ADDRSPACE MAXNUMBER 100 TOTSIZE 8192G SHARE
   XCONFIG ACCESSLIST ALSIZE 1022
   CONSOLE 0009 3215 T VMAINT
   SPOOL 000C 2540 READER *
   SPOOL 000D 2540 PUNCH A
   SPOOL 000E 1403 A
   LINK MAINT 0190 0190 RR
   LINK MAINT 0193 0193 RR
   LINK MAINT 019D 019D RR
* All but first R'ed in samples
   MDISK 0191 3390 2187 3 VMZP1P MR XXXX YYYY
   MDISK 0301 3390 1923 10 VMZP1P R XXXX YYYY
   MINIOPT NOMDC
   MDISK 0302 3390 1933 14 VMZP1P R XXXX YYYY
   MINIOPT NOMDC
   MDISK 0303 3390 1950 14 VMZP1P R XXXX YYYY
   MINIOPT NOMDC
   MDISK 0304 3390 1947 3 VMZP1P R XXXX YYYY
   MDISK 0305 3390 1964 6 VMZP1P R XXXX YYYY
```

Figure 66. VMSYSU File Pool Server Directory

For the VMSYSR file pool server, VMSERVR, we used:

```
USER VMSERVR XXXXXXXX 32M 32M BG
   ACCOUNT ITSXXXX
   IPL 190
   IUCV ALLOW
   IUCV *IDENT RESANY GLOBAL
   MACHINE XA
   OPTION MAXCONN 2000 APPLMON ACCT QUICKDSP SVMSTAT
   SHARE REL 1500
   CONSOLE 0009 3215 T OPERATOR
   SPOOL 000C 2540 READER *
   SPOOL 000D 2540 PUNCH A
   SPOOL 000E 1403 A
   LINK MAINT 0190 0190 RR
   LINK MAINT 0193 0193 RR
   LINK MAINT 019D 019D RR
* All WR'ed in samples
   MDISK 0191 3390 2190 2 VMZP1P WR XXXX YYYY
   MDISK 0301 3390 1914 2 VMZP1P WR XXXX YYYY
   MINIOPT NOMDC
   MDISK 0302 3390 1912 1 VMZP1P WR XXXX YYYY
   MINIOPT NOMDC
   MDISK 0303 3390 1913 1 VMZP1P WR XXXX YYYY
   MINIOPT NOMDC
   MDISK 0304 3390 1916 2 VMZP1P WR XXXX YYYY
   MINIOPT NOMDC
   MDISK 0305 3390 1918 1 VMZP1P WR XXXX YYYY
   MINIOPT NOMDC
   MDISK 0306 3390 1919 2 VMZP1P WR XXXX YYYY
   MINIOPT NOMDC
   MDISK 0307 3390 1921 2 VMZP1P WR XXXX YYYY
   MINIOPT NOMDC
```

Figure 67. VMSYSR File Pool Server Directory

For the VMSYS file pool server, VMSERVS, we used:

```
USER VMSERVS XXXXXXXX 64M 64M BG 1
   ACCOUNT ITSXXXX
   IPL 190
   POSIXOPT SETIDS ALLOW
   IUCV ALLOW
   IUCV *IDENT RESANY GLOBAL
   MACHINE XC
   OPTION MAXCONN 2000 APPLMON ACCT NOMDCFS QUICKDSP SVMSTAT
   SHARE REL 1500
   XCONFIG ADDRSPACE MAXNUMBER 100 TOTSIZE 8192G SHARE
   XCONFIG ACCESSLIST ALSIZE 1022
   CONSOLE 0009 3215 T VMAINT
   SPOOL 000C 2540 READER *
   SPOOL 000D 2540 PUNCH A
   SPOOL 000E 1403 A
   LINK MAINT 0190 0190 RR
   LINK MAINT 0193 0193 RR
   LINK MAINT 019D 019D RR
* ALL BUT FIRST R'ED IN SAMPLES
   MDISK 0191 3390 2858 3 VMZP1P MR
   MDISK 0301 3390 2861 2 VMZP1P MR
   MINIOPT NOMDC
   MDISK 0302 3390 2863 3 VMZP1P MR
   MINIOPT NOMDC
   MDISK 0303 3390 2866 3 VMZP1P MR
   MINIOPT NOMDC
   MDISK 0304 3390 2869 3 VMZP1P MR
   MDISK 0305 3390 2872 36 VMZP1P MR
   MDISK 0306 3390 2 100 VMZU1A MR
```

Figure 68. VMSYS File Pool Server Directory

7.3.2 Test File Pool

A test file pool was defined and used:

SFSTEST BFS and SFS file pool for test.

Following is the directory of the SFSTEST file pool server.

```
USER SFSTEST XXXXXXX 32M 32M BG 1
* NOTE: THIS IS A TEST SFS GROUP FILE POOL MACHINE.
   ACCOUNT ITS3000
   IPL 190
   IUCV ALLOW
   IUCV *IDENT RESANY GLOBAL
   MACHINE XA
   OPTION MAXCONN 2000 NOMDCFS APPLMON ACCT
   SHARE REL 1500
   CONSOLE 0009 3215 T COSTA
   SPOOL 000C 2540 READER *
   SPOOL 000D 2540 PUNCH A
   SPOOL 000E 1403 A
   LINK MAINT 0190 0190 RR
   LINK MAINT 0193 0193 RR
   LINK MAINT 019D 019D RR
* ALL WR'ED IN SAMPLES..
* 191 IS THE RUN AND BACKUP DISK
   MDISK 0191 3390 2848 100 VMZU1R MR
* 250 IS THE CONTROL DISK
   MDISK 0250 3390 2588 30 VMZU1R R
   MINIOPT NOMDC
* 405 IS LOG1 DISK
   MDISK 0405 3390 744 10 VMZU1R R
   MINIOPT NOMDC
* 406 IS LOG2 DISK
   MDISK 0406 3390 1036 10 VMZU1R R
   MINIOPT NOMDC
* 260 IS CATALOG DISK  1 MDK00001 GROUP=1
   MDISK 0260 3390 2618 10 VMZU1R R
* 310 IS A DATA DISK   1 MDK00002 GROUP=2
   MDISK 0310 3390 2628 200 VMZU1R R
```

Figure 69. SFSTEST File Pool Server Directory

7.3.3 VMNFS

Following is the directory entry for the VMNFS server we used. VMNFS provides the NFS server functions.

```
USER VMNFS XXXXXXXX 32M 64M BG
* PRIV: TCP/IP SVM.  NETWORK FILE SERVER.
* PRIV: CLASS B NEEDED FOR DIAG X'84' AND X'64'
   ACCOUNT ITSXXXX
   IPL CMS
   IUCV RACFVM PRIORITY MSGLIMIT 255
   POSIXINFO UID 0 GID 0
   POSIXOPT QUERYDB ALLOW
   MACHINE XA
   OPTION QUICKDSP SVMSTAT ACCT MAXCONN 1024
   CONSOLE 0009 3215
   SPOOL 000C 3505 A
   SPOOL 000D 3525 A
   SPOOL 000E 1403 A
   LINK MAINT 0190 0190 RR
   LINK MAINT 019E 019E RR
   LINK MAINT 019F 019F RR
* LINKS TO DISKS FOR TCP/IP V2R3 310 RSU 9804
   LINK TCPMAINT 0591 0591 RR
   LINK TCPMAINT 0592 0592 RR
   LINK TCPMAINT 0198 0198 RR
   MDISK 0195 3390 3253 10 VMZP1P MR
   MDISK 0191 3390 2272 2 VMZP1P MR
```

Figure 70. VMNFS Server Directory Entry

Note: PCNFSD YES should be specified in VMNFS CONFIG.

7.3.4 Added Directory Statements

Directory statements added to GLOBALDEFS:

```
POSIXGROUP system   0
POSIXGROUP staff    1
POSIXGROUP bin      2
POSIXGROUP sys      3
POSIXGROUP adm      4
POSIXGROUP mail     6
POSIXGROUP security 7
POSIXGROUP nobody   4294967294
```

Figure 71. POSIXGROUP Directory Statements

Directory statements added to FTPSERVE:

```
POSIXINFO UID 0 GID 0
POSIXOPT  QUERYDB ALLOW
```

Figure 72. Directory Statements Added to FTPSERVE

Directory statements added to the desktop publishing users directory entry (the GNAME parameter is optional):

```
POSIXINFO UID 28 GNAME nobody
```

Figure 73. Directory Statement Added to Publishing Userid

The following optional directory statements add a default initial directory and mount of the BFS file pool root. The statements eliminate the need to mount (OPENVM MOUNT) and set directory (OPENVM SET DIR) to set up the user′s BFS environment.

```
POSIXINFO IWDIR /home/cmsps
POSIXINFO FSROOT /../VMBFS:VMSYS:ROOT/
```

Figure 74. Optional User Directory Statements for Publishing Userid

7.4 OpenEdition and OpenEdition Shell and Utilities

The redbook *OpenEdition for VM/ESA Implementation and Administration Guide*, SG24-4747, contains an outstanding description of the whole OpenEdition environment. For your convenience, a short overview is included here.

OpenEdition for VM/ESA provides a POSIX-compliant file system called the Byte File System (BFS). OpenEdition Shell and Utilities is an optional priced feature of VM/ESA. It is recommended for an expanded set of capabilities and a Unix-style environment.

You can set up and use BFS without the OpenEdition Shell and Utilities feature. We installed the feature. You can (and we did), previous to the OpenEdition install, use the LOADBFS BFS command to build an „OpenEdition-compliant tree structure. For more information see ″1.1.11 Planning for OpenEdition for VM/ESA″ in *VM/ESA Planning and Administration V2R3.0*, SC24-5750. For compatibility with sites without the OpenEdition Shell and Utilities, only the BFS and OPENVM commands are covered.

7.4.1 POSIX Terminology

Here are the definitions of some of the more common POSIX terms that arise when using BFS:

Path name	A character string that identifies a path to a file or directory. OpenEdition supports path names up to 1023 characters long.
Relative path name	A path name that identifies the path to a file or directory from the current working directory. Relative path names do not begin with a slash (/).
Absolute path name	A path name that identifies the path to a file or directory from the file system root. Absolute path names always begin with a slash (/).
File name	A character string that names a file within a directory. OpenEdition supports file names up to 255 characters long.
Directory	A special file that contains directory entries. Directory entries must be unique within a given directory, but different directory entries can associate different names to the same file.
Current (or working) directory	The directory associated with a process (for example, the shell) that is used to resolve relative path names.

The characters used in file and path names should be drawn from the POSIX portable character set, but this is not enforced by OpenEdition for VM/ESA. The portable character set consists of:

- The uppercase letters A-Z
- The lowercase letters a-z
- The digits 0-9
- The special characters dot (.), underscore (_) and hyphen (-)

A compliant name cannot start with a hyphen. A valid POSIX file name might be:

karen.htmlbin

and its corresponding absolute path name might look like this:

/home/dave/pages/examples/karen.htmlbin

Note that everything up to and including the last slash (/) in this example is part of the directory path, and everything after the last slash is the file name itself. When the current directory is set to /home/dave, the relative path name is pages/examples/karen.htmlbin.

Because the slash (/) is the path separator character, it cannot be used in a file name. Also, because the standard OpenEdition shell interpreter assigns special meanings to the following characters, it is not a good idea to use them in file or directory names either:

```
(blank) * # / \ < > | & $ ? ( ) { }
```

Figure 75. Special Characters

Make file names easy to remember. Unlike VM/ESA, which restricts both file names and file types to eight (or less) characters, OpenEdition permits file names to be up to 255 characters long. Use the dot (.) or the underscore (_) to make file names more readable and easy to remember, for example:

will_jones_birthday_gift_list

Over the years, users of Unix-style operating systems have developed a set of conventions for arranging the initial directories off of the main or root directory. The POSIX BFS follows these conventions as implemented by OpenEdition for VM/ESA, so you are likely to see a first-level directory structure that looks something like this:

```
/          (← the root directory)
  bin      (contains executable commands)
  dev      (support for hardware devices)
  etc      (contains administration files, the toolbox)
  home     (contains user directories and files)
  lib      (symbolic link to /usr/lib libraries and shared libraries)
  opt      (contains DCE administration files)
  tmp      (contains system temporary files and work areas)
  u        (symbolic link to /home)
  usr      (contains system executable files, administration files, etc.)
  var      (contains log files, security and spool files, etc.)
```

Figure 76. Typical BFS Root Tree Structure

7.5 Some Common SFS and BFS Commands

Some readers may be new to BFS, and even to SFS. Therefore, the commands listed in Table 11 may be useful.

Table 11 (Page 1 of 2). Some SFS and BFS Tasks and Commands

Task	SFS command	BFS command	Shell
Enroll a user	ENROLL USER xyz	ENROLL USER xyz (BFS	
Create a directory	CREATE DIR dirid	OPENVM CREATE DIR path	mkdir
Erase a directory	ERASE dirid	OPENVM ERASE path	rmdir
Create an alias	CREATE ALIAS ...	OPENVM CREATE LINK ... OPENVM CREATE EXTLINK ... OPENVM CREATE SYMLINK ...	ln
Define a fault directory	ACCESS dir A	OPENVM SET DIR path	cd
Copy a file	COPYFILE ...	OPENVM GETBFS ... OPENVM PUTBFS ...	cp
Erase a file	ERASE fn ft fm	OPENVM ERASE path	rm

Table 11 (Page 2 of 2). Some SFS and BFS Tasks and Commands

Task	**SFS command**	**BFS command**	**Shell**
Move a file or directory	RELOCATE ...	OPENVM RENAME ...	mv
Xedit a file	XEDIT fn ft fm	XEDIT path (NAMET BFS	ed
Set permissions	GRANT AUTH ... REVOKE AUTH ...	OPENVM PERMIT ...	chmod
Get access to files	ACCESS dir fm	OPENVM MOUNT ...	
Drop access to files	RELEASE fm	OPENVM UNMOUNT ...	
List files and directories	LISTFILE * * fm	OPENVM LISTFILE ...	ls
Change a group or owner	-	OPENVM OWNER ...	chgrp chown
Define or display a creation mask	-	OPENVM SET MASK ... OPENVM QUERY MASK ...	umask
Execute an application	-	OPENVM RUN ...	appl. name

To get more information about SFS commands, issue HELP CMS or HELP SFSADMIN.

To get more information about BFS commands, issue HELP OPENVM or HELP OPENVM MENU.

7.5.1 ENROLL User in a File Pool

A user could operate without being enrolled in a BFS file pool if the file pool has <PUBLIC> enrollment. Generally, users will be in file pools separate from system use and OpenEdition Shell and Utilities use for performance reasons.

Enrolling users to a BFS file pool provides their own root, which can be mounted separately or on top of any directory under their control.

When sharing a file pool for both SFS and BFS, and a user is enrolling in both, the name of the file space needs to be different for the BFS enrollment. The following CMSPS2 parameter is the file space name:

```
ENROLL USER CMSPS SFSTEST:  ENROLL USER CMSPS2 SFSTEST: (BFS USER
CMSPS
```

If the user is just enrolling into a BFS file pool, the file space name can match the user ID:

```
ENROLL USER CMSPS VMSYSU: (BFS USER CMSPS
```

7.5.2 Mount a User File Space over a Directory

For users to access their BFS directory from a CMS logon, either the optional directory statements in Figure 74 on page 192 are used or the following action needs to be taken at each logon or IPL CMS:

```
OPENVM MOUNT /../VMBFS:VMSYS:ROOT/ /

OPENVM SET DIR /home/cmsps
```

OPENVM MOUNT /../VMBFS:SFSTEST:CMSPS2/ /home/cmsps/

A permanent link can be established by creating an external link. The following needs to be established by a user with write authority to the /home/ directory:

OPENVM CREATE EXTLINK /home/cmsps MOUNT /../VMBFS:SFSTEST:CMSPS2/

7.6 Changing the HTM File Type

For VM:Webgateway Web Server Release 2.2, the following shows how to list and change the .HTM file type from EBCDIC to binary:

```
vmwebsrv q filetype htm
Filetype  Characteristics
--------  ---------------------------------------------------------------
HTM       FILE TRANSLATE USENGLISH CONTENT-TYPE text/html SSI NO FILTER NONE
Ready; T=0.01/0.01 14:23:27
vmwebsrv q filetype htmlbin
Filetype  Characteristics
--------  ---------------------------------------------------------------
HTMLBIN   FILE TRANSLATE NONE CONTENT-TYPE text/html SSI NO FILTER NONE
Ready; T=0.01/0.01 14:23:36
VMWEBSRV CONFIG FILETYPE REPLACE HTM FILE CONTENT-TYPE text/html
         TRANSLATE NONE SSI NO FILTER NONE
```

Figure 77. Change the HTM Translation Setting

For EnterpriseWeb/VM, the following shows how to change the definition for the HTM file type:

- Logon to EWEBADM.
- XEDIT $EWEB MEDIAMAP D
- Find /htm/.
- Change 8-bit to ASCII:

```
htm   text/html        -    8bit  -
htm   text/html        -    ascii -
```

- To avoid a problem of no translation occurring in the automatic index function, either move the HTM record to precede the other htm... records in the MEDIAMAP file, or obtain and install WEB PTF LVL WEB00431 from Beyond Software.
- Key in `file`.
- Cycle EnterpriseWeb/VM servers.

7.6.1 VM:Webgateway BFS Access Setup

Define the appropriate BFS user path to VM:Webgateway:

VMWEBSRV SET USERROOT cmsps BFS
/../VMBFS:VMSYS:ROOT/home/cmsps/pages/

BFS read and execute access is needed for the Web server to serve pages and to run CGI applications, respectively, from the user's BFS. To establish the authority for the owning user, mount the BFS file pool. Read permission will need to be established for each directory within the directory tree structure. Before files are created, you can set the mask and new files will then obtain the mask permissions when they are created.

OPENVM PERMIT /home/cmsps/ --- --- r-x (ADD OPENVM PERMIT /home/cmsps/pages/ --- --- r-x (ADD OPENVM SET MASK rwx r-x r-x

If you plan to have the index.html file be a binary ASCII index.htm file, you need to create a VMWEBSRV.DIRMAP file in the file folder containing the record INDEX index htm:

- OPENVM MOUNT /../VMBFS:VMSYS:ROOT/ /
- OPENVM SET DIR /home/cmsps/
- XEDIT ../pages/VMWEBSRV.DIRMAP (NAMETYPE BFS
- Insert the following record:
  - INDEX index htm
- FILE

7.6.2 EnterpriseWeb/VM SFS Access

Grant the appropriate SFS authorization to the EnterpriseWeb/VM server for the user pages.

For serving user pages, grant the following:

GRANT AUTH SFSTEST:CMSPS.WEBSHARE. TO EWEB01 (READ NEWREAD

EnterpriseWeb/VM has a demo publishing CGI. To test, you will either need to have the FASTPATH option turned on in the configuration, or remove the /vm/ directory from the links in the demo EWPUBSFS HTML file on the EWEBADMs 194 minidisk. We removed the /vm/ from EWPUBSFS HTML.

You will also need to add EWPUBSYS *CGI * to the HTBIN FILELIST on the EWEBADMs 194 minidisk.

Grant write authority to the EnterpriseWeb/VM servers:

GRANT AUTH SFSTEST:CMSPS.WEBSHARE. TO EWEB01 (WRITE NEWWRITE

You are now ready to use the publishing demo:

- Open the browser and go to

  `http://your_server_domain_name/EnterpriseWeb/demos/sfspublish.html`

  Fill in the SFS directory:

  `sfstest:cmsps.webshare.`
- Select **Browse**, then find and select the previously saved .htmla file.
- Select **Publish**.
  The page will be published and then served to the browser.

Note: Do not leave this test demo executable in an “as is” state, because the demo will allow loading files to any SFS directory that the EnterpriseWeb/VM server has write authorization to without any browsing user authentication.

7.6.3 NFS Client

An NFS client will enable mounting the publishing user′s BFS directory on the publishing user′s PC desktop as a network drive. There are several NFS clients available. We used the NFS client contained in IBM eNetwork Communications Suite for Windows 3.1, Windows 95 and Windows NT Version 1.1. This suite contains FTP Software′s NFS client. An evaluation CD kit was available as of October 1998. You can register for an evaluation CD at `http://www.software.ibm.com/enetwork/hostsolution/kit/`

Check for current information about NFS clients and VM on the Web at `http://www.ibm.com/s390/vm/NFS/winnfs.html`

7.6.3.1 NFS Client Install

We installed version 1.1 from the IBM eNetwork Communication Suite CD.

The steps to install the FTP Software′s NFS client included on the IBM eNetwork Communications Suite CD are as follows:

- Insert the CD; the setup.exe should self-launch.
- Select to install **FTP Software and TCP/IP applications**.
- Select **Next** on the Network Access Suite 3.0 setup.
- Continue responding to install prompts.
- At the end of the installation the Install Wizard prompts for a reboot. We recommend not selecting a reboot at this time.
- Select **Exit** from eNetwork Communications Suite.
- Then shut down and restart the PC.

The following are the client configuration tasks:

- To configure the NFS client to be able to find the VM/ESA NFS server, check the Web for complete and current information at `http://www.ibm.com/s390/vm/NFS/winhosts.html`

  The steps we used were:

  – Make sure there is an entry for the VM/ESA host in your c:\windows\hosts file. If you do not have a hosts file, copy the sample file called hosts.sam.
  – To add the VM/ESA hostname and IP address, select **Start**, **Program**, **MS-DOS Prompt**.
  – Add the IP address (`9.12.13.65`) and hostname (`wtscvmt`) of the VM/ESA NFS server.
  – Be sure the host file has a blank line at the end of the file.
  – Select **Save** then **File**.

- Next add the VM/ESA NFS Server to the NFS client configuration, as follows:

  – Double click the **Network Neighborhood** icon.
  – Double click **Entire Network**.
  – Double click **NFS Servers I have Configured**.
  – Click your right mouse button on an empty spot in that window to obtain a list.
  – Choose **Properties**.
  – Fill in the new host name and click **Add**. Our hostname was `wtscvmt`.
  – Select **Close**.
  – Do not check Connect using TCP from the Properties panel for Drive Options. VM′s NFS server uses UDP.

- Then add authentication for the NFS server connection:
  - Right click on the host name you just added, **wtscvmt**.
  - Select **Properties**.
  - Select the **Security** tab.
  - In the Username field, enter your VM user ID.
  - In the Password field, enter the logon password for your VM user ID.
  - Select **Apply** and **Close**.
- Now you are able to mount a BFS directory as a network drive. See `http://www.ibm.com/s390/vm/NFS/winbfs.html` for current information. To mount the BFS directory:
  - Create an alias:
    - Bring up Windows Explorer.
    - Go to NFS Servers I Have Configured, as you did when you configured the NFS server.
    - Select the host, and click the right mouse button to obtain a list.
    - Select **Create Alias**.
    - Fill in the alias name that you want to use. We used `wtscvmttrano`.
    - Fill in the Path, specifying the mount operands, as follows `/../VMBFS:VMSYSU:ROOT/home/cmsps,trans=no`
    - Select **OK** to create the alias.
  - Issue the mount:
    - Go to Windows Explorer.
    - On the menu bar, select **Tools**.
    - Select **Map Network Drive**.
    - Fill in \\hostname\alias. (Note the backslash following hostname.) We used `\\wtscvmt\wtscvmttrano`.
    - Select **OK**.

Some recommendations and additional information:

- Be sure to write down aliases and parameters when the alias is created, because you will need these alias names in later steps. You cannot list existing aliases or their operands.
- Use `/../VMBFS:FPCOOL:ROOT/u/jake,trans=no`. The result of no translation is that the files are stored in ASCII only, not EBCDIC.
- Access no-translation directories only from the desktop. If you must access them from CMS, you should use the BFSLIST exec. BFSLIST can be downloaded from

  `http://www.ibm.com/s390/vm/download/packages/`
- For SFS directories, use `sfstest:cmsps.webshare,record=binary,trans=no` for the Path information.
- For CMS minidisks use, `cmsps.191,rw,record=binary` for the Path information.
- If the network drive contains a mix of ASCII and EBCDIC information, we recommend mounting the same file structure as a second network drive. The second mount would have translation turned on. Everything written or read to and from the second drive would have the ASCII to EBCDIC translation performed. This is great for viewing text README.txt files on the host and viewing host files containing text on the PC. To perform the second mount of the same drive, repeat both the Create an alias and Issue the mount steps. The path information would be:

- For BFS, use: `/../VMBFS:FPCOOL:ROOT/u/jake,trans=yes,xlat=p`
- For SFS, use: `fpcool:jake.mission,record=nl,nlvalue=0D0A`
- For CMS Mdisks, use: `jake.191,rw,record=nl,nlvalue=0D0A`

7.6.4 Set Up htmlbin or htmla

This step is not necessary if you have opened up .htm as a binary ASCII file type in VM:Webgateway. This step can also be performed in Windows Explorer.

There is a need to create a new extension type in Windows 95 because the desktop publishing performs a binary transfer of the HTML file to VM. Creating this new extension type informs the Web server that the file is binary and ASCII. The Web server does not perform an EBCDIC to ASCII translation when serving the HTML page. Which file type you need to create depends on the Web server you have. If you have VM:Webgateway Web Server, then define .htmlbin. If you have EnterpriseWeb/VM or Webshare, then define .htmla.

A list of the tasks needed to create the new extension follows:

- Open Windows Explorer.
- Select **View** and **Options**.
- Select the **File Types** tab.
- Select **New Type**.
- Fill in the following:
  - Description of Type: html to publish to BFS
  - Associated extension: `htmlbin`
  - Content Type (MIME): `text/html`

  Select **OK** to save and exit.
- Select **Close** to save and exit.

7.6.5 Show DOS Extensions

Perform this step if you choose to use htmlbin or htmla instead of changing the htm definition on the Web server.

Following are the steps to change the setting in Windows Explorer to show DOS extensions:

- Open Windows Explorer.
- Select **Folder**.
- Select **View** and **Options**.
- Deselect **Hide MS-DOS file extensions for file types that are registered**.
- Select **Apply** and **OK** to save and exit.

7.7 NetObjects Fusion 3.0

The NetObjects Fusion 3.0 Trial product is available for a 30-day trial. The product is located at

`http://www.netobjects.com/products/html/download.html`

7.7.1 Trial Version Publishing Limitations

The trial version of NetObjects Fusion has publishing disabled. We saw the effect in NFS and FTP. The FTP upload was successful in the version but NetObjects Fusion does not recognize message 250 from the FTP server as a successful upload. So NetObjects Fusion never believes that the publish was successful. See Figure 79 on page 203.

7.7.2 NetObjects Fusion NFS Setup

If you have a trial version you will have to publish to the C: drive, then copy the files to the NFS drive. If you have the full version, you can set up the NFS network drive as a local publishing site.

When publishing forms created with NetObjects Fusion, you need to fill `ACTION=“”` with the CGI name and subfolder, if any, like: `<FORM NAME=“Guestbook Form” ACTION=“” METHOD=POST>` `<FORM NAME=“Guestbook Form” ACTION=“htbin/test.svmexec” METHOD=POST>`.

To set up NetObjects Fusion 3.0 to publish using NFS:

- Bring up NetObjects Fusion.
- Select **Publish** and **Publish Setup**.
- Select **Server Locations** and **Add**.
- Decide on a Server Name and type in the chosen name.
- Select **Local** and **Browse**.
  - Select the previously mounted NFS drive and the directory you wish to publish to (full version).
  - Or, select a folder on the C: drive (trial version).
- Change **Rename the home page of each directory as** to the **Index** selection.
- If appropriate, change **Make the extension of each page** from .html to the planned file type.
- Select **Replace spaces and other special characters with underscores**.
- Select **OK** to save and close (twice).

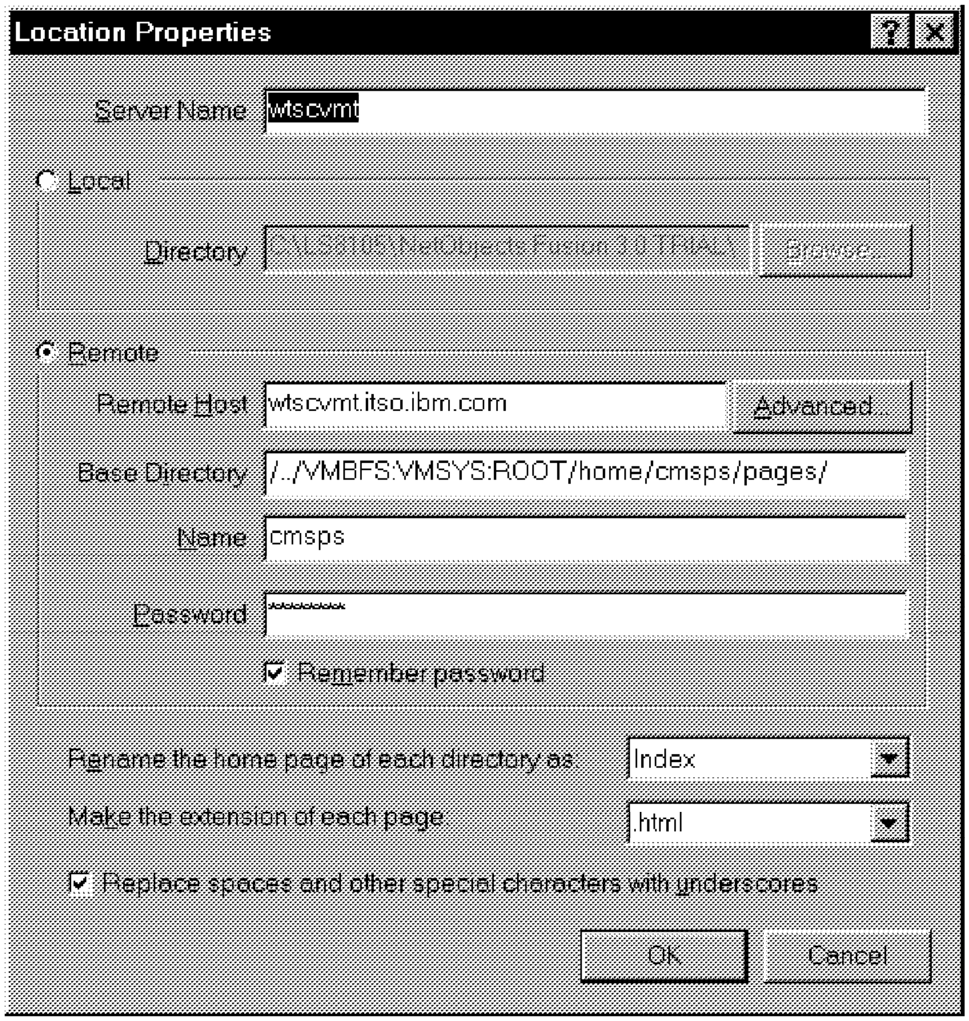
Note: We used an extension of .htm for local publishing.

7.7.3 NetObjects Fusion FTP Setup

Following are the steps to set up FTP publishing in NetObjects Fusion:

- Bring up NetObjects Fusion.
- Select **Publish** and **Publish Setup**.
- Select **Server location** and **Add**.
- Fill in or select the following fields:
  - ServerName.
  - Select **Remote**.
  - Remote Host.
  - Base Directory.
  - Name.
  - Password.
  - Select **Replace spaces with....**.

  See Figure 78 on page 202.
- Select **OK** to save and exit twice.


Location Properties

Server Name: wtscvmt

Local

Directory: C:\LS8105\NetObjects Fusion 3.0 TRIAL\

Remote

Remote Host: wtscvmt.itso.ibm.com

Base Directory: /../VMBFS:VMSYS:ROOT/home/cmsps/pages/

Name: cmsps

Password: *******

[✓] Remember password

Rename the home page of each directory as: Index

Make the extension of each page: .html

[✓] Replace spaces and other special characters with underscores

Figure 78. Set Up Publishing in NetObjects Fusion

Note: Your site security may require you to not fill in the password. Also, we actually used a subdirectory of /netf/.
NetObjects Fusion, when using FTP, will transfer the pages in text mode, which will translate the pages to EBCDIC. Therefore, the file type extensions remain .html.

7.7.4 NetObjects Fusion NFS Publishing

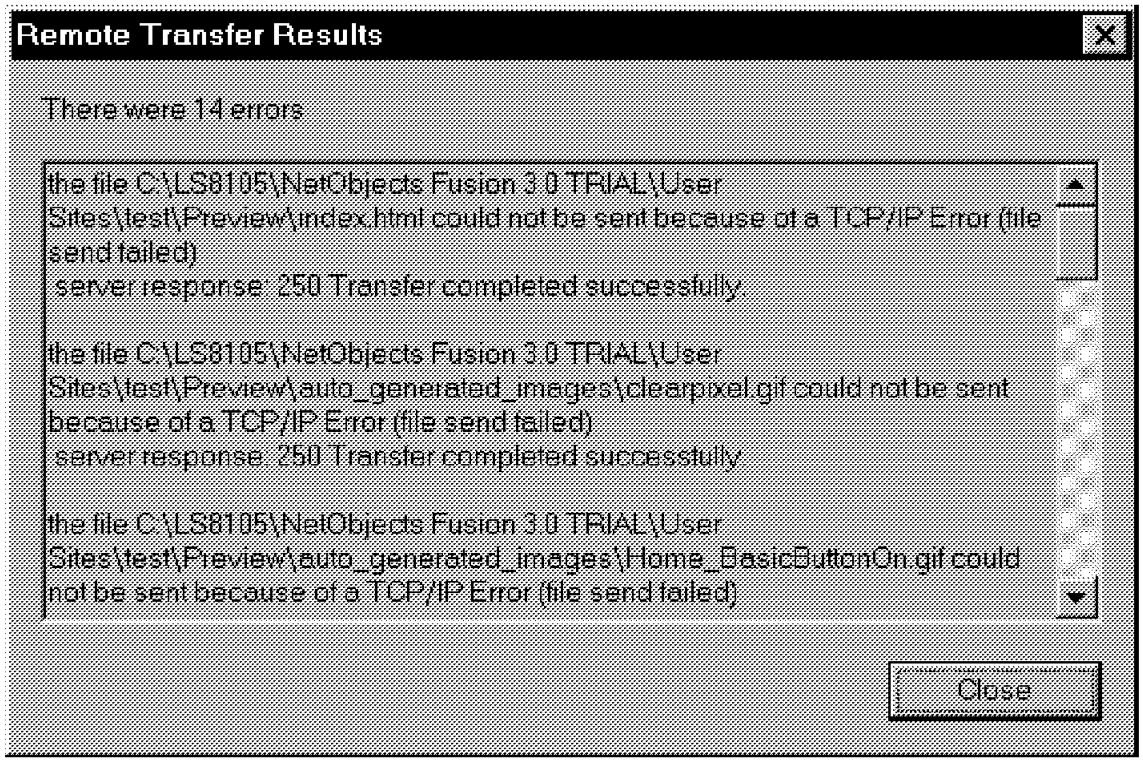
Following are instructions on how to publish to NFS:

- Select **Publish** on the tool bar.
- Select the **Publish** button on the right of the tool bar.
- Select the newly added local server from the location pulldown.
- Select **OK** to publish.
- If you have the trial version, then copy files from the local drive publish folder to the NFS network drive.

7.7.5 NetObjects Fusion FTP to BFS Publishing

Following are the steps for publishing pages with NetObjects Fusion to an FTP site:

- Select **Publish** and **Publish Site**.
- Select the new remote site you created in the previous FTP setup procedure.
- Select **OK** to publish the pages. The pages will publish successfully, even though, if you have the trial version, you will get the error messages shown in Figure 79.
- Select **Close** to exit.


Remote Transfer Results

There were 14 errors

```
the file C:\LS8105\NetObjects Fusion 3.0 TRIAL\User
Sites\test\Preview\index.html could not be sent because of a TCP/IP Error (file
send failed)
 server response: 250 Transfer completed successfully.

the file C:\LS8105\NetObjects Fusion 3.0 TRIAL\User
Sites\test\Preview\auto_generated_images\clearpixel.gif could not be sent
because of a TCP/IP Error (file send failed)
 server response: 250 Transfer completed successfully.

the file C:\LS8105\NetObjects Fusion 3.0 TRIAL\User
Sites\test\Preview\auto_generated_images\Home_BasicButtonOn.gif could
not be sent because of a TCP/IP Error (file send failed)
```

Figure 79. Error Messages When Publishing in NetObjects Fusion (Trial Version)

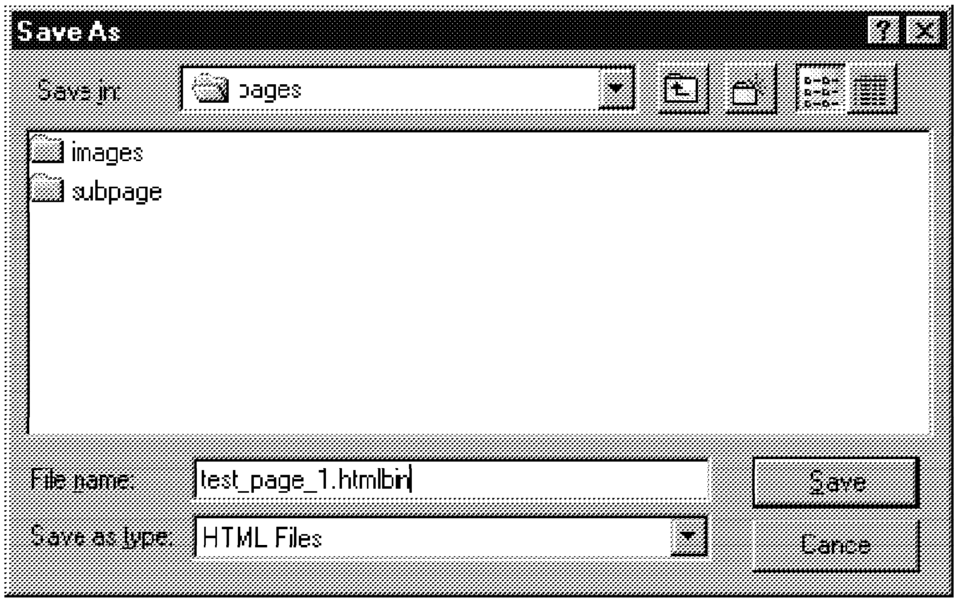
7.8 Netscape Composer 4.07

Netscape Composer is the Web page composition facility inside the Netscape Communicator product. Netscape Communicator 4.07 was the version we used.

7.8.1 Netscape Composer NFS Publishing

We used a no-translation NFS network drive. We planned for an .htm file type. We also used the .htmlbin file type. To publish using the network drive, perform the following:

- Select **File** and **Save As**.
- Select the network drive letter you mounted earlier, and the appropriate folder.
- Fill in the file name and add `.htmlbin` or `.htm` as planned. See Figure 80 on page 204
- Select **Save** to save and exit.



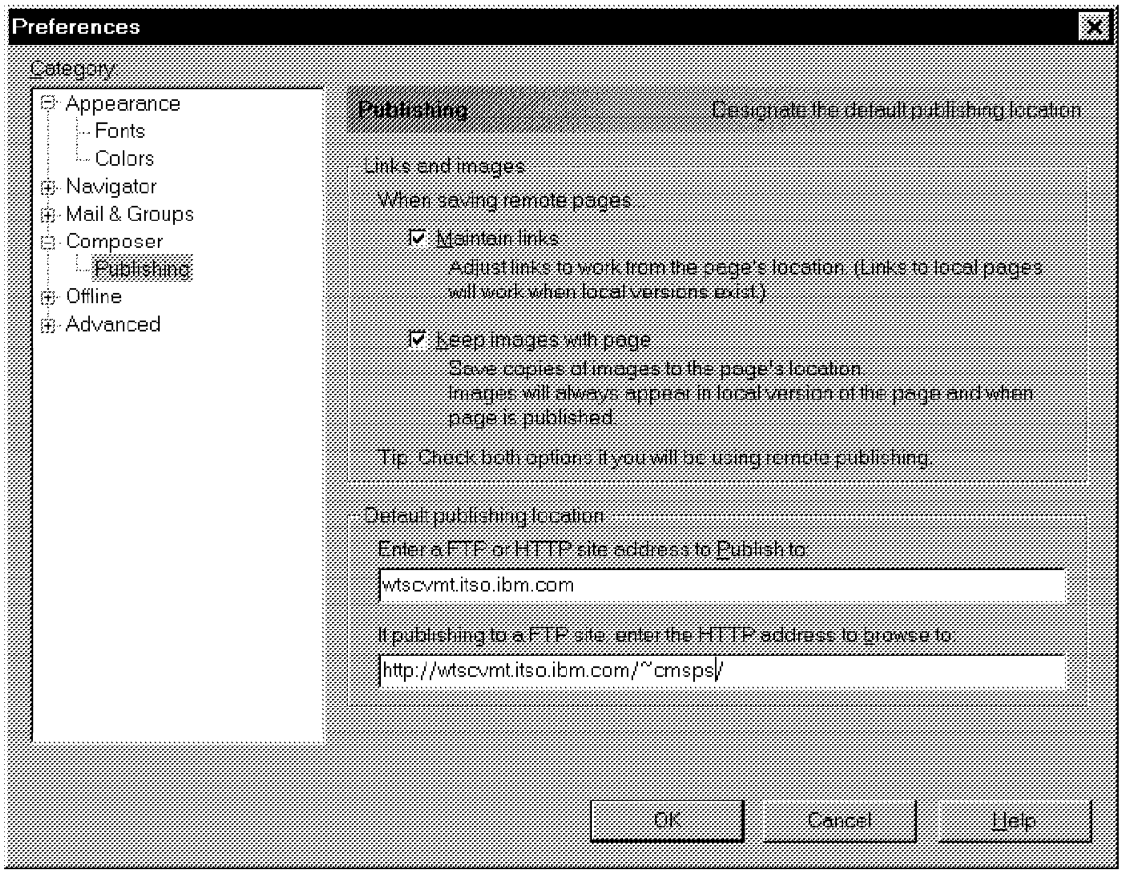
File name: test_page_1.htmlbin

Figure 80. Save File as .htmlbin

7.8.2 Netscape Composer FTP Setup

Following is how to set up FTP publishing:

- Select **Edit** and **Preferences**.
- Fill in Author Name.
- Expand Composer and select **Publishing**.
- Fill in the site address and browser address as shown in Figure 81 on page 205.
- Select **OK** to save and exit.


Preferences

Category

- Appearance
  - Fonts
  - Colors
- Navigator
- Mail & Groups
- Composer
  - Publishing
- Offline
- Advanced

Publishing — Designate the default publishing location

Links and images

When saving remote pages

☑ Maintain links

Adjust links to work from the page's location. (Links to local pages will work when local versions exist.)

☑ Keep images with page

Save copies of images to the page's location. Images will always appear in local version of the page and when page is published.

Tip: Check both options if you will be using remote publishing.

Default publishing location

Enter a FTP or HTTP site address to Publish to:

wtscvmt.itso.ibm.com

If publishing to a FTP site, enter the HTTP address to browse to:

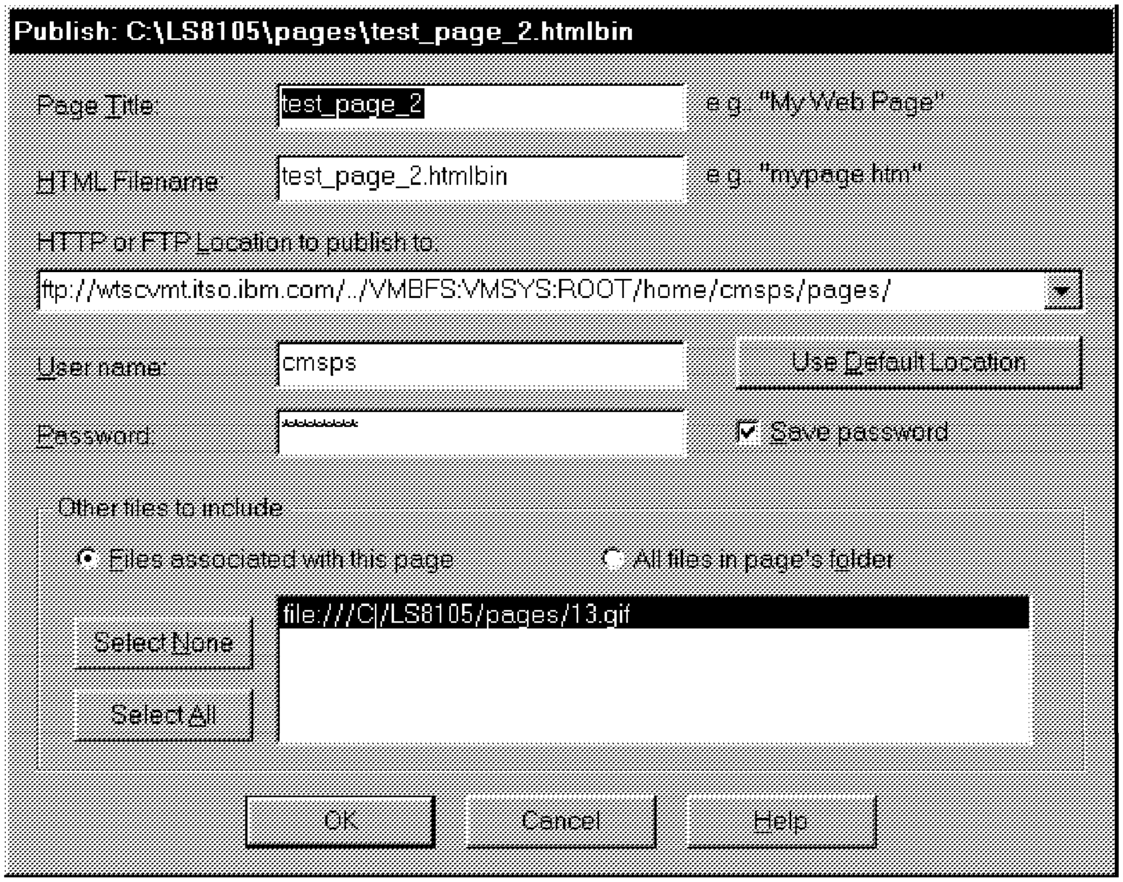
http://wtscvmt.itso.ibm.com/~cmsps/

Figure 81. Netscape Composer Publish Setup

7.8.3 Netscape Composer FTP Publishing

The publishing setup could be performed the first time you publish. Following is how to set up publishing:

- After the page is composed and saved as `htmlbin` or `.htm` as planned select **Publish**.
- Fill in Page Title, HTML Filename, FTP Location, Username and Password. See Figure 82 on page 206.
- Select **OK** to publish.
- Watch for error messages.


Publish: C:\LS8105\pages\test_page_2.htmlbin

Page Title: test_page_2 e.g. "My Web Page"

HTML Filename: test_page_2.htmlbin e.g. "mypage.htm"

HTTP or FTP Location to publish to:

ftp://wtscvmt.itso.ibm.com/../VMBFS:VMSYS:ROOT/home/cmsps/pages/

User name: cmsps

Password: ********

[✓ Save password]

Other files to include

[● Files associated with this page] ○ All files in page's folder

file:///C|/LS8105/pages/13.gif

Figure 82. Publish Page or Folder of Pages

You can select all pages in a folder, unless files associated with this page are in other folders.

7.9 Microsoft FrontPage Express

The FrontPage Express product is similar to Netscape Composer. An additional function is the form creation capability.

When developing pages with Microsoft FrontPage Express, you will publish utilizing NFS because the Web Publishing Wizard utilizing FTP does not operate properly. The purchased version of FrontPage ′98 is reported to work properly for FTP publishing. We did not test FrontPage ′98. A section on configuring Web Publishing Wizard is included. The Microsoft FrontPage Express product is part of the Internet Explorer 4.0 download. The level we used was version 2.0.2.1118. Following are instructions on how to set up and publish with the Front Page Express to an NFS network drive:

- Select **Start** and **Programs**.
- Select **Internet Explorer** and **Front Page Express**.
- Create a page.
- Select **File** and **Save As**.
- Clear out the Page Location field.
- Select **As File**.
- Select the previously created network drive and appropriate folder.

- Fill in filename `.htmlbin` or `.htm` as appropriate.
- Select **Save**.

Note: When linking together pages in FrontPage Express, do the following:

- Select **Insert** and **Hyperlink**.
- Select the **World Wide Web** tab.
- Select **http:** in HyperLink Type.
- Fill in only the filename .htmlbin in the URL field.
- Select **OK** to complete the link.

7.9.1 Web Publishing Wizard

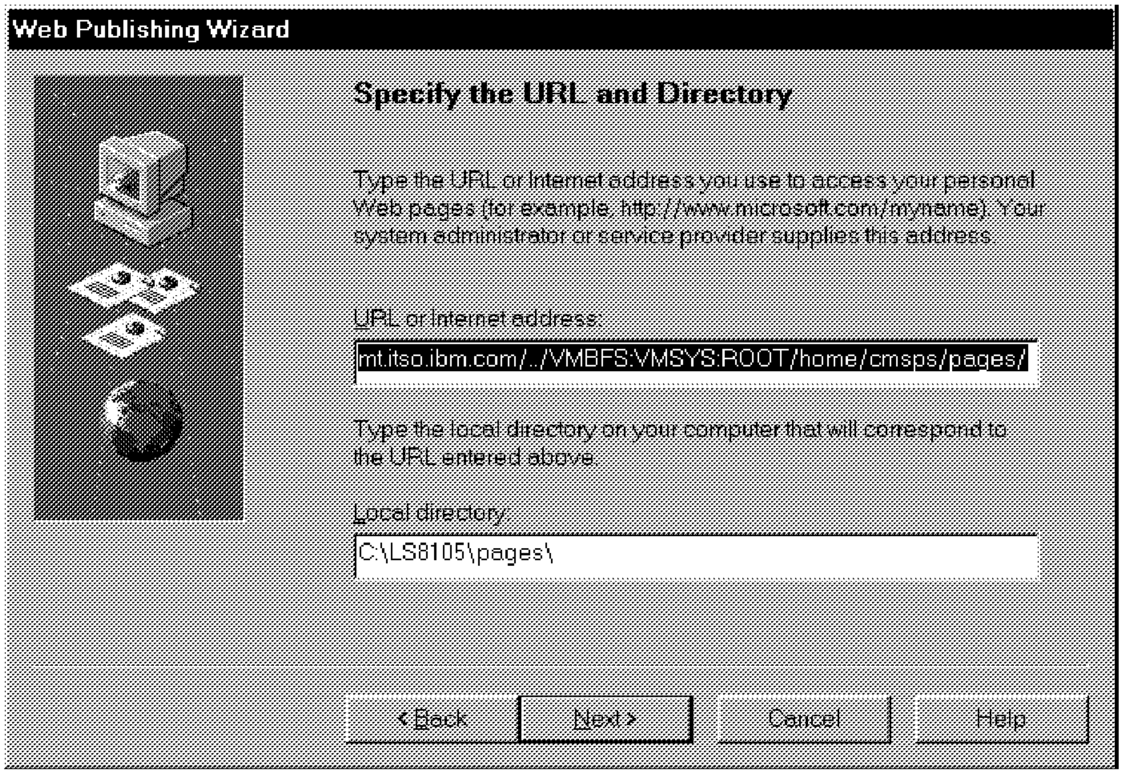
Following are the instructions on how to set up and publish with the Web Publishing Wizard:

- Select **Start** and **Programs**.
- Select **Internet Explorer** and **Web Publishing Wizard**.
- Select **Next**.
- Select **File or Folder** and **Next**.
- Select **New**.
- Fill in Descriptive Name and Select **Advanced**.
- Select **FTP** and **Next**.
- Fill in the URL with a fully qualified FTP address:

  ```
  ftp://wtscvmt.itso.ibm.com/../VMBFS:VMSYS:ROOT/home/cmsps/pages
  ```

- Select **Next**. See Figure 83 on page 208.
- Select **Finish**.
- Fill in User Name and Password.
- Select **OK** to publish.

Note: The preceding is for informational purposes only. The objective is to assist in setting up FrontPage ′98.



Specify the URL and Directory

Type the URL or Internet address you use to access your personal Web pages (for example, http://www.microsoft.com/myname). Your system administrator or service provider supplies this address.

URL or Internet address:

mt.itso.ibm.com/../VMBFS:VMSYS:ROOT/home/cmsps/pages/

Type the local directory on your computer that will correspond to the URL entered above.

Local directory:

C:\LS8105\pages\

Figure 83. Publish Page with Web Publishing Wizard

Appendix A. Java

One of the reasons Java has become so popular is its value for enhancing Internet World Wide Web content. The basic language of the WWW, HTML, is easy to write and make available to WWW clients, but it lacks sophistication for complicated tasks. For example, user interfaces such as dialog boxes can be written in HTML, but there are limited facilities for ensuring consistency and enforcing open standards. As a result, a WWW application that uses an HTML user interface must check data entered by the client at the server. This defeats some of the value of a client/server architecture; usually, data validation is delegated to the client, where the speed of the client machine can be utilized and network delays avoided.

Beyond providing smart interfaces in HTML documents, Java can perform any tasks processed by other languages. Moreover, its broad library of standardized functions permits programmers to ignore platform limitations.

A very interesting feature of integrated Java runtime environments in browsers is the ability to let Java applets communicate with the host from which they are loaded.

A.1 Presentation

Many books describe the Java development environment and the Java runtime environment. We do not intend to replicate the information here, we will just define some Java terms.

For more information about Java and VM, see *VM/ESA Network Computing with Java and NetRexx*, SG24-5148.

For a list of Java books and reviews, consult:

```
http://www.ibm.com/java/education/books
```

The essential Java vocabulary consists of:

class	The compiled Java code. Each class file represents a Java object. It contains Java byte code produced by compilation.
JVM	The Java Virtual Machine is the environment where Java code runs. It can be imbedded in a browser. It interprets Java byte code to let it run on the local platforms.
JIT	Just In Time compilers are used in the JVM to avoid interpretation performance issues. Some compile all Java byte code to hardware instructions before Java program execution, others keep already interpreted code in memory so as not to compile it again the next time it is executed.
JDK	The Java Developer Kit is the Java “libraries.” Each JVM runs a specific JDK level.
applet	Java code intended to run as part of an HTML page.
jar	A Java archive is a set of Java classes in a single compressed file. Jar files are used to send all applet code in only one network download. The JVM then extracts the needed class files.

NetRexx NetRexx is an alternative to the Java language that takes advantage of the JVM while not forcing the programmer to have to face the complexities of the Java language. With NetRexx, you can create programs and applets for the Java environment using REXX syntax. For more information, visit Java and NetRexx pages on the VM/ESA site at

`http://www.ibm.com/s390/vm/java`

The typical Java runtime environment looks like Figure 84.

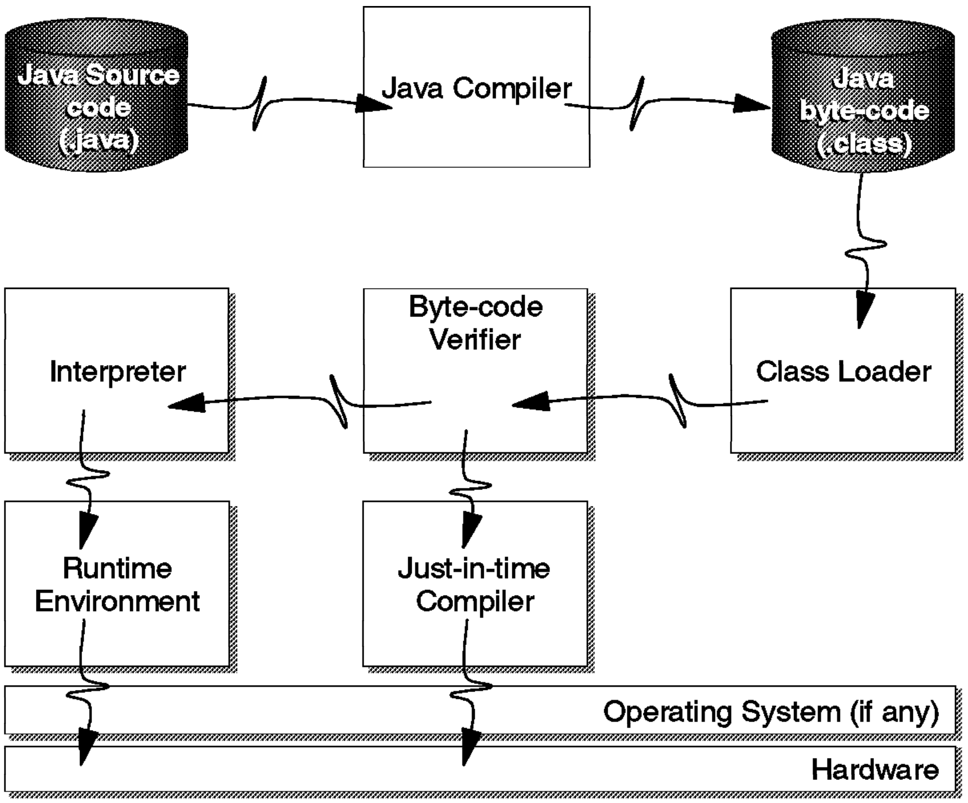


Figure 84. Java Virtual Machine Environment

The Java byte code runs in the JVM environment. The JVM interprets the byte code into local system code and manages JDK function calls. JVM and JDK local implementation are responsible for interfacing with local system calls. With such an architecture, the Java programmer does not have to care about specific (operating system or hardware) architectures. For example, the Java Abstract Windowing Toolkit (AWT), which manages graphic interfaces, contains a large library of graphic components. The programmer writes only one piece of code that will generate different graphic environment calls in various user execution environments (X-Window, Presentation Manager, Windows).

The class loader is the JVM process that automatically loads class files needed (that is referenced) by the current object. As a typical Java application is a set of classes, you start the application by indicating the ″main″ file to be run to the Java loader. The loader scans all classes needed to execute this file. All referenced objects not known by the Java virtual machine will be loaded by the class loader from the same location as the ″main″ class. In an HTTP environment, the class loader creates URL connections to download peripheral objects (to be in the same URL path as the Java start object).

A.2 Implementation Considerations

The first consideration before starting any Java implementation is to check what browsers will contact your site. While Java was first available in 1996, there are still a lot of browsers that do not support it.

A.2.1 JDK Levels

Because of Java′s constant evolution and improvement, new levels of the JDK are provided at a very high rate. Choosing the right JDK level for programming is a Java-specific issue: the latest JDK versions include new functions that are unknown by older versions, programming methods change (for example, AWT event listening), bugs are fixed. All of this is moving fast!

To choose the appropriate JDK level, you have to consider three basic environments:

- Development environment. This is where you write and test your code. It may be a basic line mode environment or an Integrated Development Environment (IDE) such as IBM VisualAge for Java.
- Compilation environment. You may compile Java code in your development environment, but you may also choose to run it somewhere else. Some considerations are as follows:
  - Your development environment includes a very recent version of the JDK and you want to verify with an older JDK compiler that your code is compatible with old JVMs.
  - You use an IDE but your code is intended to run in another system with its own JDK. The safe way is to transfer the production Jav source to your target system and to compile it there.
- Runtime environment. This is, in most cases, the user′s local JVM. In the Web implementation it is the browser-controlled JVM. To determine the version of a browser, JVM looks for the “Java Console” window of the browser.

These environments could be on the same machine, each with a different JDK level. It is common that a Java developer machine (see Figure 85 on page 212) contains an operating system Java developer kit and an IDE with its own version of the JDK and the browser JVM version.

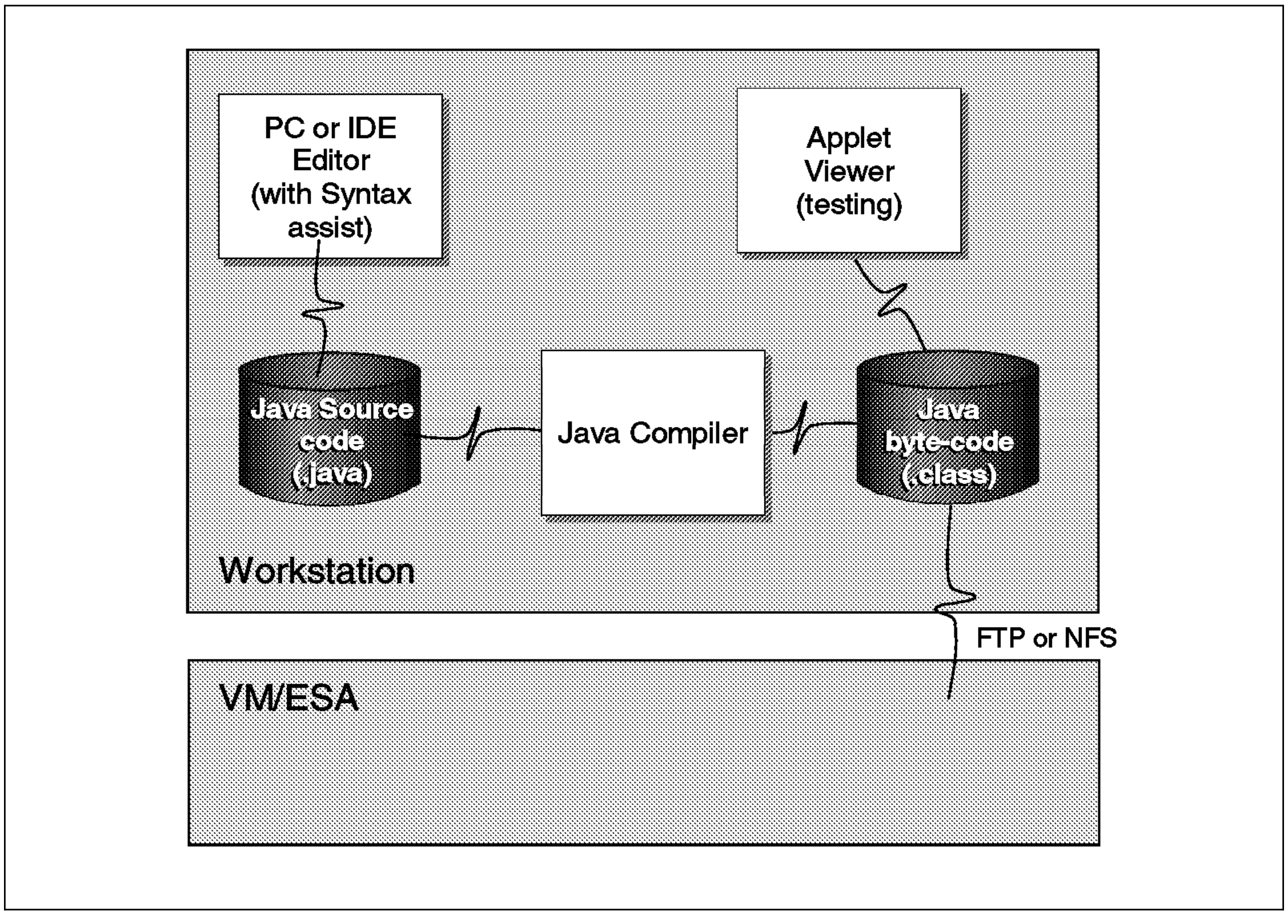


Figure 85. A Typical Java Development Environment

A safe approach could be to consider the lowest JVM level intended to run your code as the JDK level you will develop with. It is the only choice if you cannot control user browser versions.

Whatever level you use, you may indicate it clearly in your site help page with links to obtain the latest browser versions (some browsers are delivered with different JVM levels while the browser version stay the same! Do not use the browser version as an indication to determine the imbedded JVM level).

A.2.2 Serving Java Code from VM Web Servers

In order to serve Java code from your favorite VM Web server, you may need to change the server configuration.

A.2.2.1 Declaring Java Types

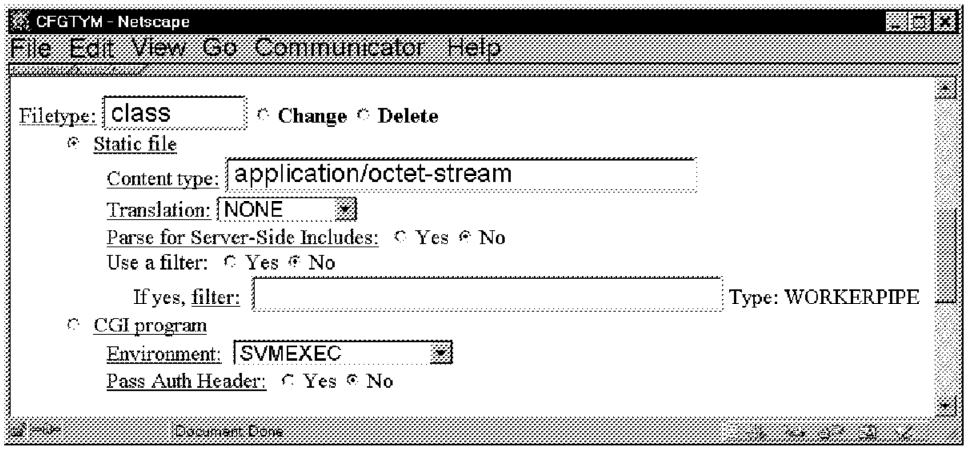
The Java byte code loaded from your VM Web server is considered binary code. If the Web server ignores this detail, it may try to apply an EBCDIC to ASCII translation that will cause "bad magic number" errors on the browser JVM when it verifies the checksum included in the Java code.

On EnterpriseWeb/VM Release 1.4, no additional configuration is needed to serve Java files.

On Webshare Release 1.2.4 you need to declare two new file types in HTTPD CONFIG:

```
type        class       0       application/octet-stream    binary
type        jar         0       application/java-archive    binary
```

On VM:Webgateway Web Server Release 2.2 we also added these file types using the administration online interface as shown in Figure 86.



Filetype: class Change Delete
[✓ Static file]
Content type: application/octet-stream
Translation: NONE
Parse for Server-Side Includes: Yes [✓ No]
Use a filter: Yes [✓ No]
If yes, filter: Type: WORKERPIPE
CGI program
Environment: SVMEXEC
Pass Auth Header: Yes [✓ No]

Figure 86. Adding Java Types to a VM:Webgateway Configuration

A.2.2.2 Naming Java Files

You also need to consider the way Java resource names are resolved on the server. Unlike minidisk and SFS files, Java file names are case sensitive. The first part of the name (before the dot) is not limited to eight characters, and the browser JVM loader will check that the Java class file name is the same as the Java class it contains. Moreover, the file type must be “class” in lower case.

The VM Web servers do not address Java class name issues similarly.

In the Webshare or EnterpriseWeb/VM server you may either use the FILELIST facility to hide the real VM name, or the Web server will truncate the class name to eight characters, translate it into uppercase and then look for a CMS file with that name. The latter technique seems easier, but you have to watch out for Java class names with the same eight first characters!

In VM:Webgateway you need either to use a DIRMAP file (which is case insensitive) or preferably serve the Java files from BFS (which is case sensitive).

To copy the Java code into the server file structure you may upload it from the development environment without translation (binary mode) or upload the Java source to the BFS environment, compile it with the IBM Java Port for VM/ESA and copy the class file into your Web server tree.

Watch the IBM Java download page to obtain the latest version of the IBM Java Port for VM/ESA:

```
http://www.ibm.com/java/jdk/download
```

A.3 Applets

Since Java is relatively new, there are still many possibilities to explore before understanding its full value. However, here are some early thoughts on the value Java applets could provide for VM/ESA applications:

- Modern user interface, without the cost of writing user interfaces for many different systems

  One of the issues that a software builder faces today is the construction of user interfaces for a wide variety of client machines. If you want to build a client/server application with a graphical user interface (GUI), you may have to write many different GUIs, depending on the types of client machines in your organization. Take, for example, Lotus Notes. Lotus has had to provide client GUIs for OS/2, Windows 3.1, Windows95 and NT, UNIX, and so on. You face the same problem in your enterprise unless you have settled on only one client operating system. If you plan to deploy an application on the Internet, you have no control over the client machine type, and so the GUI problem becomes almost insurmountable.

  Java provides a windowing class library the Abstract Windowing Toolkit (AWT)). With this, you can write one GUI that can be run on any machine with the Java Virtual Machine installed, and with the capability to display GUI windows. The implementation of the JVM on the client machine maps the AWT to native windowing controls.

- Software distribution and maintenance minimized

  Without Java, client/server application deployment requires the installation of client software. This is already a difficult problem in an enterprise. If you are planning to deploy an application on the Internet, you cannot depend on quality systems support staff to install the client software, because the client could be anywhere in the world. Furthermore, if you do make client software available, you will have to be prepared to support customers who attempt to install the software on a wide variety of client machine types, software levels, and so on.

  Java applets are dynamically downloaded applications. The download is performed automatically by the client machine when the application is started (perhaps by clicking a button on an HTML document in the client browser). The client needs to have:

  – An operating system
  – A network connection and software that enables TCP/IP protocols
  – A WWW browser that enables Java

  Since this sort of configuration is now common (popular operating systems provide all of this support now, and many client machines come preloaded with such software), you can relax and let the server do all the work of distributing up-to-date software, on demand. You do not have to worry about back-level clients.

- Extending your operational applications to the Internet (or an intranet) without recoding

  All System/390 customers have existing applications that were designed and implemented before the World Wide Web was even thought of. Many applications were designed with 3270 interfaces in mind, so some level of data validation was delegated to the 3270 device. For example, field widths and types could be checked at the data entry station.

The WWW provides very little to take the place of that data checking. HTML has limited facilities inside the form element; and JavaScript (which could do some checking) is not always enabled on WWW browsers.

Java allows you to deploy a user interface such as a dialog box, which checks the data before sending it to the server. Thus, you could leave your operational application unchanged and use the new Java applet to check and format data. Application response and usability could be improved, since basic checking and feedback to the user can be done on the client machine, which is faster than going back to the server for each interaction.

- Improved presentation

  You can use a Java applet to retrieve data from a VM/ESA server, and then provide processing within the applet that allows the user to choose the format of a presentation. You will need to be careful, however, about data integrity if you plan to update the data from the client. That is, if you download data, process it for a while, and then update it on the server, you need some technique to ensure that you are not performing updates on data that has already been updated by another application. This is a standard client/server design issue, not restricted to Java.

As shown in Figure 87 on page 216, the HTML APPLET tag is the description of imbedded Java programs.

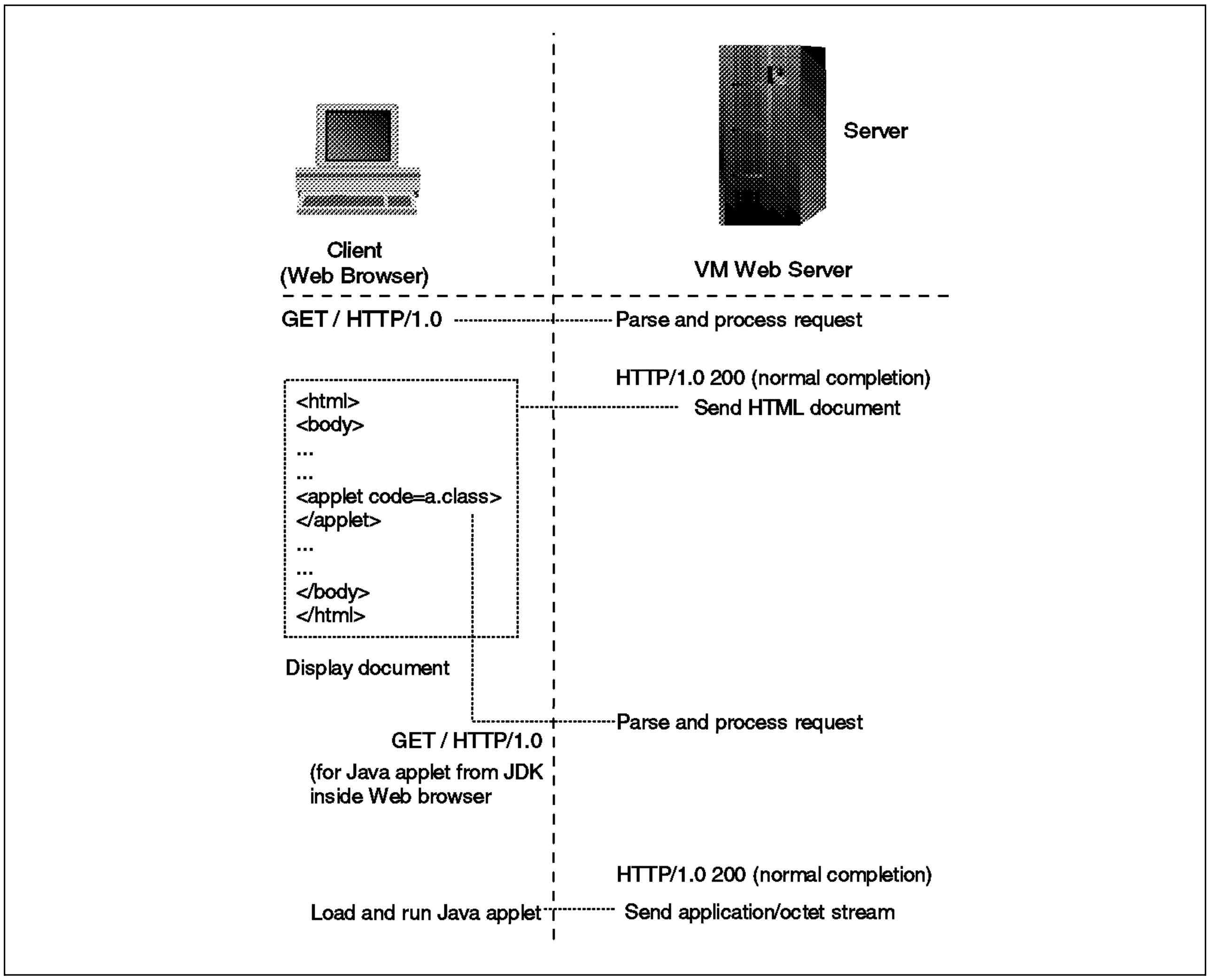


Figure 87. HTTP Processing of an HTML Document with Applet Tag

To load a Java applet into the browser JVM, you need to code into your HTML document an APPLET tag in the form:

```
<APPLET  CODE      =  appletFile.class     1
         CODEBASE  =  codebaseURL          2
         ALT       =  alternateText        3
         NAME      =  appletInstanceName   4
         WIDTH     =  pixels               5
         HEIGHT    =  pixels               6
         ALIGN     =  alignment            7
         VSPACE    =  pixels               8
         HSPACE    =  pixels               9
         ARCHIVES  =  archive.jar  >       10
   < PARAM NAME = appletAttribute1  VALUE = appletValue1 >  11
   < PARAM NAME = appletAttribute2  VALUE = appletValue2 >
     .
</APPLET>
```

- **1** This required attribute gives the name of the file that contains the applet′s compiled Applet subclass. This file is relative to the base URL of the applet and cannot be absolute.
- **2** This optional attribute specifies the directory that contains the code for the applet. If this attribute is not specified, the document′s URL is used.

- **3** This optional attribute specifies any text that should be displayed if the browser understands the APPLET tag but cannot run Java applets.
- **4** This optional attribute specifies a name for the applet instance, which makes it possible for applets on the same page to find and communicate with each other.
- **5** This required attribute specifies the initial width, in pixels, of the applet display area, not counting any windows or dialogs that the applet brings up.
- **6** This required attribute specifies the initial height, in pixels, of the applet display area, not counting any windows or dialogs that the applet brings up.
- **7** This attribute specifies the alignment of the applet. The possible values of this attribute are the same as those for the HTML IMG tag: left, right, top, texttop, middle, absmiddle, baseline, bottom, absbottom.
- **8** This attribute specifies the number of pixels above and below the applet. It is treated the same way as the VSPACE attribute on the HTML IMG tag.
- **9** This attribute specifies the number of pixels on each side of the applet. It is treated the same way as the HSPACE attribute on the HTML IMG tag.
- **10** This tag specifies that all files used by the applet are to be found in an archive file created with the Java tool jar.
- **11** This tag is the only way to specify an applet-specific attribute. Applets access their attributes with the getParameter() method.

You can, of course, use CGI to create the HTML document, thereby controlling the applet parameter number and values.

The Pp application is an example of how to convert a 3270 application to Java.

It shows you how to:

- Read a CMS file across an HTTP connection.
- Call a VM CGI, gathering its parameters like an HTML form.
- Read CGI outputs.

The original Web-enabled VM application is used to select a set of products from a list and send an order to a server machine using RSCS facilities. The 3270 interface consists of a fullscreen product list built from a CMS file. The navigation follows SAA standards; it allows users to select products, view information on the products, scroll up and down in the list, and send the final request.

The Pp Java application permits all of these. It adds a counter indicating how many 3380 cylinders are needed to install the product set. Unlike a CGI implementation, the Web server is only used at applet loading time and at final FORM completion, where a VM CGI creates the RSCS file. All application interactions are handled inside the applet, avoiding server processing

Once loaded, the applet:

- Reads APPLET tag parameters.
- Opens a URL connection to read the product list with the Web server.
- Initializes its AWT interface.
- Permits the user to:
  - Add products to an order list.

    – Remove products previously selected from the order list.
    – Open a frame to display details for a selected product
- Process the order, calling a VM CGI whose response will be displayed in a new frame.

The APPLET tag should contain two parameters:

```
<param name=SourceList value="/~ vmwebcd/java/products.txt">
<param name=TargetCGI  value="/~vmwebcd/cgi/ppord">
```

SourceList parameter indicates the URL path to the product list. The URL protocol, server and port are retrieved dynamically inside the applet. If the applet is loaded from disk (by the Appletviewer tool, for example), this variable must contain the file path. This allows the applet to be tested on a PC without any Web server connection.

TargetCGI parameter is the URL path to the VM CGI that will process the applet order. The order will appear to CGI like any POST HTML form request.

The Pp application is a set of six objects:

PpWindow The base object pointed to by the applet code parameter.

This object is responsible for:

- Retrieving the Web server protocol, address and port:

```
URL       myURL = getCodeBase();
fileProtocol    = myURL.getProtocol();
fileServer      = "://"+myURL.getHost()+":"+myURL.getPort();
```

- Reading applet tag parameters:

```
String filePath = getParameter("SourceList");
String cgiPath  = getParameter("TargetCGI");
```

- Creating an instance of the object that reads the CMS file:

```
myList     = new PpList(fileProtocol,fileServer,filePath);
```

- Building the applet frame graphic design. The frame is divided into three zones as shown in Figure 88 on page 219.

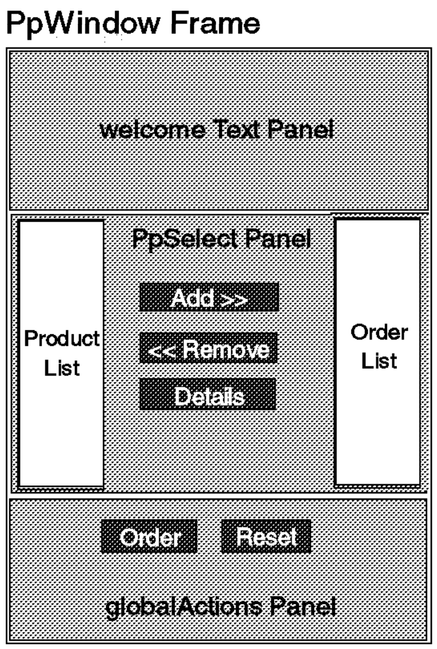


Figure 88. Pp User Interface

The upper zone (welcomeText panel) contains text information. The middle zone is occupied by a PpSelect object. The bottom zone is the globalActions panel and contains the button to run the order processing and the button that resets the order list content.

```
setLayout(new GridLayout(3,1));
welcomeText.setLayout(new FlowLayout());
welcomeText.setFont(new Font("Courrier",Font.PLAIN,12));
add(welcomeText);

selPanel     = new PpSelect(myList.getVList());
add("Cmde",selPanel);

globalActions.setLayout(new FlowLayout());
bcom.setFont(new Font("Helvetica",Font.BOLD,18));
bcom.setBackground(Color.gray);
globalActions.add(bcom);
brst.setFont(new Font("Helvetica",Font.BOLD,18));
brst.setBackground(Color.gray);
globalActions.add(brst);
add("Actions",globalActions);
```

- Listening to user actions:

```
public boolean action(Event evt, Object arg) {
  if (evt.target instanceof Button) {
 String bsel = (String)arg;
    // adds a product to order list
 if (bsel.equals(selPanel.badd.getLabel())) selPanel.addPp(myList);
    // remove product from order list
 else if (bsel.equals(selPanel.bdel.getLabel()))
        selPanel.delPp(myList);
    // get product details
 else if (bsel.equals(selPanel.bdetail.getLabel())) {
   String sel = (String)selPanel.inputList.getSelectedItem();
   PpDetail winDetail  = new PpDetail(myList.getPpack(sel));
  }
     // send the order form
  else if (bsel.equals(bcom.getLabel())) createOrder();
  else if (bsel.equals(brst.getLabel())) selPanel.resetOrder();
   }
   return true;
 }
```

This event handler is based on the JDK 1.02 model. This is a real programming choice because a new method of handling events came with JDK 1.1 that programmers are intended to use. But the JDK 1.1 method dos not run on JVM 1.02 and there are still a lot of browsers running the 1.02 JVM level. We chose to use the old event handler method because it runs on all JVM levels and our application is too small to warrant dealing with old model issues.

However, you can see in the event handler some tests on buttons not defined in the current object (badd,bdel,bdetail): they are PpSelect buttons. In 1.02 the event model, imbedded component events have to be trapped in the imbedding AWT container. Another consequence is ″deprecated API″ warning messages when you compile this code with the latest JDK compilers.

Ppack This object represents a product. Its constructor receives a line from the product list and parses it:

```
Ppack(String s) {               // s is a line from the CMS file
 desc=s.substring(0,12);        // product description
 prodNumber=s.substring(13,21); // product number
 cm=s.substring(22,65);         // comments
 id=s.substring(67,75);         // unique identifier
 cyl=s.substring(77,81).trim(); // 3380 cylinders
}
```

It also defines a function to access each individual object variable.

PpList Contains the complete collection of Ppack objects. At initializatio, PpList reads the product list from a disk or URL, and then creates a new Ppack object for each line and includes it in a vector list and a hash table:

```
String url = fileProtocol+fileServer+filePath;
URL    theURL = null ;
 // create product file URL object
try { theURL = new URL(url); }
catch ( MalformedURLException e) {System.out.println(″Bad URL: ″ + url);}
URLConnection conn = null;
DataInputStream r;
 // open URL and read thru the end
try {
 conn = theURL.openConnection();
        conn.connect();
 r = new DataInputStream(new BufferedInputStream(conn.getInputStream()));
 try {
  String line;
  while( (line = r.readLine()) |= null) {
    // create Ppack object
      Ppack pp = new Ppack(line);
      // add it in hash table
        hashPp.put(pp.getDesc(),pp);
      // add it on vector
        vectorPp.addElement(pp);
   }
  }
  catch (IOException e) {
    System.out.println("Network error reading product list:"+e);}
}
catch (IOException e) {
   System.out.println(url+" not found or unreachable: "+e);}
```

We created two product object tables (only for convenience):

- Vector tables are collections of objects that are easy to scan so we used one for sequential or index position access.
- Hash tables are more suited for indexed access. The insertion of a new object in a hash table also needs an index key (the hash code). Our hash table contains a short description of the product hash code as the index for the product object (Ppack).

PpSelect Manages the more interactive part of the applet. This is the middle frame appearing on the screen. It contains:

- The selection list of products
- Buttons to control selection
- The list of products currently selected

This object also processes its own button events, even if they are also trapped in the PpWindow object. For example, here is the routine that adds objects to the order list:

```
public void addPp(PpList pList) {
     // product decription
  String sel = (String)inputList.getSelectedItem();
     // already in order list ?
  if (sel|=null & isNotInOrderList(sel)) {
     // add in order list
   outputList.addItem(sel);
   totCylinders+=
      Integer.valueOf(pList.getPpack(sel).getCyl()).intValue();
      // update cylinder label
   labCyl.setText("3380 cylinders used:"+totCylinders+" ");
  }
}
```

PpDetail This pop-up frame displays the details about a product.

PpOrder Calls the CGI and displays the data sent back by the CGI:

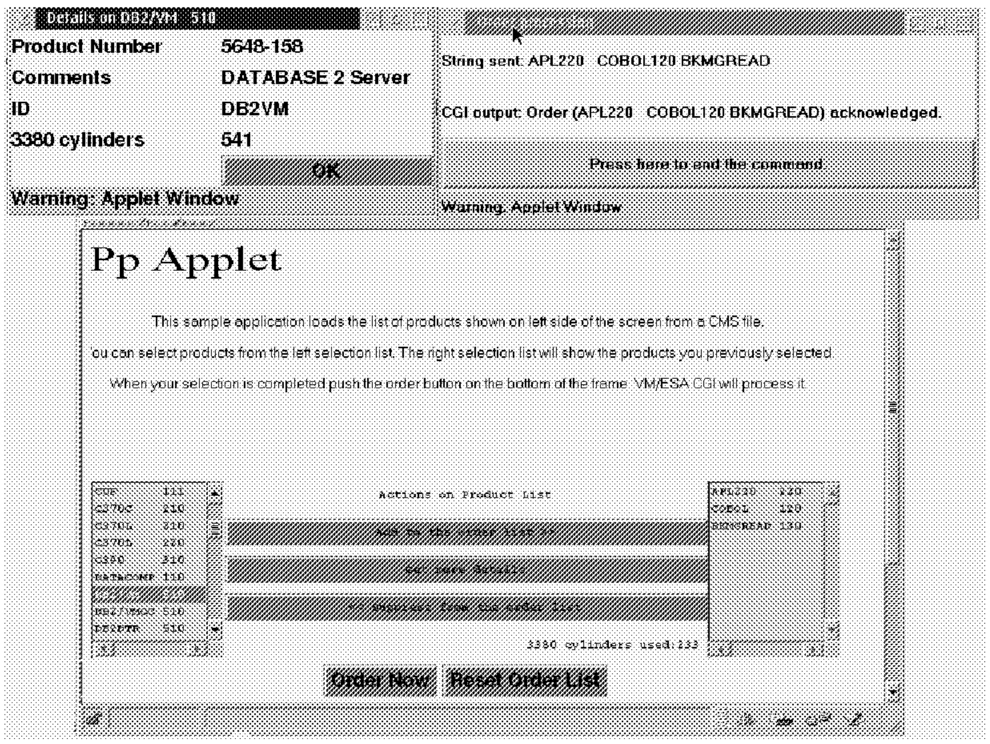
```
URL url = new URL(goURL);
URLConnection con = url.openConnection();
        // simulate a POST form request
con.setDoOutput(true);
con.setDoInput(true);
con.setUseCaches(false);
        // set HTTP descriptions
con.setRequestProperty("Content-type",
                       " application/x-www-form-urlencoded");
con.setRequestProperty("Content-length",
                       " "+data.length()+"");
        // ouput stream
PrintStream out = new PrintStream(con.getOutputStream());
        // pass FORM data
out.print("JAVAS="+data);
out.flush();
out.close();
        // listen for HTTP server response
DataInputStream in = new DataInputStream(con.getInputStream());
String s;
String vmresp = "No response from the host." ;
while ((s = in.readLine()) |= null) vmresp=s;
in.close();
       // returns
return vmresp;
```

This applet demonstrates how Java can be used to deal with user actions. CGIs are still used, but not to generate rendering tags; they are used where they are best at: accessing host resources. So the CGI is not polluted by HTML tag considerations, it just sends the needed data to the applet. In this way, the VM Web server is not called to navigate application pages. Java applets keep the CGI a single piece of code to maintain and delivered from a single point.

To run this applet we used the following URL:

`http://wtscvmt.itso.ibm.com/~ vmwebcd/java/Pp.html`

You should see a screen like Figure 89.



Details on DB2/VM 510

Product Number	5648-158
Comments	DATABASE 2 Server
ID	DB2VM
3380 cylinders	541

Warning: Applet Window

String sent: APL220 COBOL120 BKMGREAD

CGI output: Order (APL220 COBOL120 BKMGREAD) acknowledged.

Warning: Applet Window

Pp Applet

This sample application loads the list of products shown on left side of the screen from a CMS file.

'ou can select products from the left selection list. The right selection list will show the products you previously selected.

When your selection is completed push the order button on the bottom of the frame. VM/ESA CGI will process it.

```
CUF      111
C370C    210
C370L    210
C370L    220
C390     310
DATACONF 110
[illegible]
DB2/VMQC 510
DB2DTR   510
```

Actions on Product List

```
APL220   220
COBOL    120
BKMGREAD 130
```

3380 cylinders used:233

Figure 89. Pp Screen

A.4 Java Servers

In the applet example, we showed how an applet can use Web server resources using the HTTP protocol. But the applet is not restricted to communicate with the Web server virtual machine: the applet implementation allows the applet to communicate with any TCP port on the same host from which it was downloaded. This permits applets to use other server machines, giving them independence from the Web server machine. The Web server is just the distribution mechanism for the Java application.

TCP/IP networking can be done in several ways, depending on whether the host server runs Java code or another language. Even if the VM/ESA server may use any language, Java servers make it possible to use new Java interfaces and connectors that appear every month on the Web to access remote applications.

The Java server sample we present here uses socket programming to dialog with the applet code. We could also have written it with REXX socket functions.

The applet code is very simple (but powerful):

```
public class SockApplet extends Applet {
  public static final int DAYTIME_PORT = 5050;
  public String hostvalue;
  public String daytime;
  public void init() {
  hostvalue = getCodeBase().getHost();
    Socket s;
    try {
      s = new Socket(hostvalue, DAYTIME_PORT);
      BufferedReader i = new BufferedReader(
                           new InputStreamReader(s.getInputStream()));
      daytime = (i.readLine());
      i.close();
      s.close();
    }
    catch (Exception e) {
      System.err.println("Error: " + e.getMessage());
    }
  }

  public void paint(Graphics g) {
    g.drawString(daytime, 50, 25);
  }
}
```

The applet opens a socket connection to TCP port 5050, reads the incoming response from the socket server and extract the results (daytime variable). The server URL is retrieved from applet environment variables.

As the applet just displays data sent by the socket server, we need a server to send data to a socket client. We wrote a Java server to send the current date to any incoming client connections. The server runs three threads to insure good availability.

The server code is divided into three sections.

The main procedure just defines an object of its own class and sends a “go” message to the newly created “w” object.

```
public class SockServer implements Runnable {
  private ServerSocket ss;          // Define a server socket
                                    // accessible by all threads

  public static void main(String args°§) throws Exception {
    SockServer w = new SockServer();
    w.go();                         // Define instance of this class
  }
```

The “go” function creates the three parallel processes, each thread running the ″w″ object (this).

```
  public void go() throws Exception {
    ss = new ServerSocket(5050, 5);
    Thread t1 = new Thread(this, ″1″); // Thread named 1
    Thread t2 = new Thread(this, ″2″); // Thread named 2
    Thread t3 = new Thread(this, ″3″); // Thread named 3
    t1.start();                        // Start each of the threads
    t2.start();
    t3.start();
  }
```

Then, the start procedure of each object is called:

```
public void run() {
  Socket s = null;
  BufferedWriter out = null;
  String myname = Thread.currentThread().getName();
  String enc = ″8859_1″;

  for(;;) {
    try {
      System.out.println(″Thread ″ + myname + ″ about to accept..″);
      s = ss.accept();
      System.out.println(″Thread ″ + myname + ″ accepted a connection″)
      out = new BufferedWriter(
             new OutputStreamWriter(s.getOutputStream(), enc));       ;
      Date dt = new Date();
      DateFormat df = DateFormat.getDateTimeInstance(DateFormat.FULL,
                      DateFormat.DEFAULT);
      out.write(″Thread ″ + myname + ″: ″ + df.format(dt));
      Thread.sleep(10000); // we put a sleep here to force the
                           // threads to run somewhat in sequence
                           // and cycle through the set
      out.write(″çn″);
      out.close();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Each process sends a message to the Java console indicating it is ready to receive socket connections. The threads then wait for a connection when a request arrives, the thread sends back the system date on the incoming client connection.

The sleep interval is just used to confirm on the VM Java console that different threads are used. If it is omitted, the first thread would be used for every connection (being the first one available) unless a very high call rate exists. If you intend to run this code in production, you should delete this line.

This sample shows why it is easy to run multi-thread applications with Java and access TCP/IP sockets. It is as easy as REXX socket programming.

Before running the applet, insure your browser supports JDK1.1.1 and that the CMS Java level is VM61822 minimum.

To run this applet:

- Access Java sample directory:
- Copy the server code into BFS root directory.
  - Log on a user ID with root Posix authority.
  - Access VMWEBSRV.WEBSHARE.JAVA and type (case sensitive commands):

    ```
    OPENVM MOUNT /../VMBFS:yourfilepoolID:ROOT/ /
    OPENVM SET MASK r-x r-x r-x
    OPENVM PUTBFS SOCKSERV CLASS * SockServer.class (REPLACE NOTRANSLATE
    BFSLINE NONE
    ```

- Start the Java server by:

  ```
  GLOBAL LOADLIB SCEERUN
  OPENVM SHELL
  java SockServer
  ```

  The following messages should appear:

  ```
  Thread 1 about to accept..
  Thread 2 about to accept..
  Thread 3 about to accept..
  ```

- Run the applet calling the URL:

  `http://wtscvmt.itso.ibm.com/~ vmwebcd/java/SockAppl.html`

  The applet will show you the VM date and time.

To stop the Java server, type:

- `¢c`
- `rm SockServer.class`
- `exit`

A.5 A Rising New World

The samples presented in this section are just an introduction to what Java can do for you. However, they do show how easy Java programming is because the hard work is done by the JDK. You can see how compact this code is, minimizing maintenance cost.

Moreover, many IDEs are available to assist Java designers. An application like the applet sample may take an experienced Java programmer one day to create using an IDE.

Any Java code is intended to run on any Java virtual machine. For example, we ran a Java ″chat″ application (without any modification) under VM/ESA, although it was designed and tested under Windows95. This means the large Java library known as the Internet is waiting to spice up your browser pages and broaden your VM/ESA Web server′s abilities.

Appendix B. Contents of the Associated CDs

There are two CDs included with this redbook:

CD1	Sample code written during the project
CD2	Webshare 1.2.4 from Beyond Software Inc., a shareware Web server for VM/ESA.

B.1 Files on CD1

The following files are on this CD:

VMWEBCD VMA	VMARC file of the whole SFS (includes all the files)
VMWEBCD ZIP	PC file format of the sample files
5347AX2 HTM	This appendix in HTML format
VMARC MODULE	Module to unpack the VMWEBCD file and the Webshare file (CMSHTTPD)
VMARC HEL	VMARC CMS help file
VIG13TUT PDF	VM:Webgateway Tutorial in Acrobat Reader format see also B.1.9, "VM:Webgateway Tutorial" on page 234
INDEX HTM	The INDEX HTM file offers you the possibility to look at the samples with your Web browser. Simply open the INDEX HTM file on CD1 with your favorite browser and follow the instructions on the screen.
LOVEVMBG JPG	Images used in INDEX HTM
VMBSUNMV GIF	Images used in INDEX HTM
VMS390 GIF	Images used in INDEX HTM

The VMWEBCD VMARC file on CD1 contains a copy of the SFS directory tree described in this appendix. You can restore this SFS directory tree to VM by taking the following steps:

1. Log onto a user ID that can issue CREATE DIR commands for the target SFS directory where you wish to load the data. The full directory structure contained in the VMWEBCD VMARC file will be created as needed off the base directory name given later.
2. If you do not already have a copy of VMARC MODULE, obtain one from the CD or from

   `http://www.ibm.com/s390/vm/`
3. Upload the VMARC file in binary.
4. Run the file through the pipeline

   ```
   PIPE < VMARC MODULBIN A | deblock cms | > VMARC MODULE A.
   ```
5. Upload file VMWEBCD VMARC to VM in binary.
6. Run the file through the pipeline

   ```
   PIPE < VMWEBCD VMABIN A | fblock 80 00 | > VMWEBCD VMARC A F 80.
   ```
7. Issue VMARC UNPACK VMWEBCD VMARC * VMARCSFS EXEC A.

8. Issue VMARCSFS LOAD VMWEBCD TO target_sfs_directory.

   VMARCSFS expects the VMWEBCD VMARC file on the A-Disk

   The target SFS directory should be given as a fully qualified SFS directory ID. If the directory is accessed by the user ID, then it should be R/W.

The contents of the VMWEBCD ZIP file is also on CD1 for you to look through the samples.

When you have loaded the VMWEBCD file into a SFS, it has the following directories.

If not otherwise stated, the samples run on all three Web servers.

B.1.1 URL Root for the VM Web CD

- VM directory: VMWEBCD.WEBSHARE.
- VM Web CD directory: \

INDEX HTML	The page displayed when the root is fetched
LOVEVMBG JPG	Images used in INDEX HTML
VMBSUNMV GIF	Images used in INDEX HTML
VMS390 GIF	Images used in INDEX HTML

B.1.2 VM:Webgateway CGI Extension Sample

- VM directory: VMWEBCD.WEBSHARE.CGIGATEWAY
- Server URL: `http://Your_Server_Domain_Name/~ vmwebcd/CGIGATEWAY/lmcmd.vmgw`
- VM Web CD directory: \Gateway

LMCMD VMGW	Sample gateway CGI discussed on page 69
LMCMDH VHTML	HTML skeleton used by LMCMD VMGW
VMWEBSRV DIRMAP	DIRMAP authentication for LMCMD

B.1.3 Database

B.1.3.1 DB2 World Wide Web Connection Version 1 Demonstration

- VM directory: VMWEBCD.WEBSHARE.DB2WWW
- Server URL: `http://Your_Server_Domain_Name/~ vmwebcd/db2indx.html`
- VM Web CD directory: \Db2\Db2www

NOTICES HTML	DB2 WWW Connection notices
INSTALL HTML	README file for installation of DB2 World Wide Web Connection Version 1 and its demonstration programs
DB2WWW VMARC	DB2 World Wide Web Connection Version 1 code and demonstration in VMARC format
DB2WDOC HTML	DB2 World Wide Web Connection Version 1 Application Developer′s Guide
D2WHEAD1 GIF	Image referenced out of DB2WDOC HTML

B.1.3.2 DB2 World Wide Web Connection Version 1 Samples

DB2INDX HTML	Index to execute DB2 WWW macros
EMPQRY1 D2W	DB2 WWW macro, discussed on page 88
EMPQRYC1 D2W	DB2 WWW macro, discussed on page 91
EMPCHG1 D2W	DB2 WWW macro, discussed on page 91
EMPQRY2 D2W	DB2 WWW macro, discussed on page 98
EMPQRYC2 D2W	DB2 WWW macro, discussed on page 99
EMPCHG2 D2W	DB2 WWW, same as EMPCHG1 D2W
EMPQRY3 D2W	DB2 WWW macro, discussed on page 99
EMPQRYC3 D2W	DB2 WWW macro, same as EMPQRYC2 D2W
EMPCHG3 D2W	DB2 WWW macro, discussed on page 99
EMPQRY4 D2W	DB2 WWW macro, same as EMPQRY3 D2W
EMPQRYC4 D2W	DB2 WWW macro, discussed on page 100
EMPCHG4 D2W	DB2 WWW macro, same as EMPCHG3 D2W
LOGMODIF EXEC	REXX procedure, discussed on page 99

B.1.3.3 REXX SQL Samples

- VM directory: VMWEBCD.WEBSHARE.RXSQL
- Server URL: `http://Your_Server_Domain_Name/~ vmwebcd/rxsql/cgi/rxsquery`
- VM Web CD directory: \Db2\Rxsql

RXSQUERY CGI	Sample CGI executing REXX SQL commands described on page 104
CGI DIRMAP	Web server control files for URL resolution and security
CGI FILELIST	Web server control files for URL resolution and security
VMWEBSRV DIRMAP	Web server control files for URL resolution and security

B.1.4 MQSeries

- VM directory: VMWEBCD.WEBSHARE.MQSERIES
- Server URL: `http://Your_Server_Domain_Name/~ vmwebcd/mqseries/mqmain.html`
- VM Web CD directory: \Mqseries\MQINDEX.HTM

MQWEB EXEC	Sample REXX commands to insert in virtual machine PROFILE EXEC that will run MQSeries CGI (server, workers, and so on)
MQMAIN HTML	MQSeries base HTML file (it defines the frames)
MQINDEX HTML	Top frame of the MQSeries page
MQSMBOT HTML	Original bottom frame page
MQSMPUT HTML	Put sample form
MQSMPUT CGI	Sample CGI to put a message in an MQSeries queue, discussed on page 115
MQSMGET CGI	Sample CGI to get a message from an MQSeries queue, discussed on page 118
MQSMBRW HTML	Queue to browse a selection page

MQSMBRW CGI	Sample CGI to browse MQSeries queue content, discussed on page 119
MQSMCICS HTML	Input form to enter a CICS command
MQSMCICS CGI	Sample CGI to pass commands through the MQSeries/CICS bridge, discussed on page 123
MQSMPIMS HTML	Input form to enter a CICS command
MQSMPIMS CGI	Sample CGI to pass commands through the MQSeries/IMS bridge discussed in 4.9.1.11, “MQSeries/IMS Bridge” on page 126
Tools	(See 4.9.1.12, “MQSeries Bridge Tools” on page 127 for a full list of MQSeries bridge tools)
CGI DIRMAP	Web server control files for URL resolution and security
CGI FILELIST	Web server control files for URL resolution and security
VMWEBSRV DIRMAP	Web server control files for URL resolution and security

B.1.5 Java

- VM directory: VMWEBCD.UTILITIES.JAVA

PP HTML	Sample HTML file to load sample applet
PPWINDOW CLASS	Java class discussed on page 218
PPWINDOW JAVA	Java code discussed on page 218
PPSELECT CLASS	Java class discussed on page 221
PPSELECT JAVA	Java code discussed on page 221
PPORDER CLASS	Java class discussed on page 221
PPORDER JAVA	Java code discussed on page 221
PPORD CGI	CGI to process applet requests
PPLIST CLASS	Java class discussed on page 220
PPLIST JAVA	Java code discussed on page 220
PPDETAIL CLASS	Java class discussed on page 221
PPDETAIL JAVA	Java code discussed on page 221
PPACK CLASS	Java class discussed on page 220
PPACK JAVA	Java code discussed on page 220
PRODUCTS TXT	Product list used by Pp applet
SOCKAPPL HTML	Sample HTML file to load sample applet
SOCKAPPL CLASS	Java class discussed on page 223
SOCKAPPL JAVA	Java code discussed on page 223
SOCKSERV CLASS	Java class discussed on page 223
SOCKSERV JAVA	Java code discussed on page 223
CGI DIRMAP	Web server control files for URL resolution and security
CGI FILELIST	Web server control files for URL resolution and security
VMWEBSRV DIRMAP	Web server control files for URL resolution and security

B.1.6 Introductory Example

B.1.6.1 Program Materials

- VM directory: VMWEBCD.WEBSHARE.LEARN
- VM Web CD directory: \Learn

PHONET CGI	Sample PHONE application for Webshare and EnterpriseWeb/VM discussed on page 46
PHONET SVMEXEC	Sample PHONE application for VM:Webgateway discussed on page 46
PHONE1 CGI	Sample PHONE application for Webshare and EnterpriseWeb/VM discussed in section 3.1.2, “Sample CGI Program” on page 22
PHONE1 SVMEXEC	Sample PHONE application for VM:Webgateway discussed in 3.1.3, “Sample CGI Program for VM:Webgateway” on page 24
PHONE2 CGI	Sample PHONE application for Webshare discussed in 3.3.2, “Enhancing Our Sample Program” on page 31
PHONE2 SVMEXEC	Sample PHONE application for VM:Webgateway discussed in 3.3.2, “Enhancing Our Sample Program” on page 31
PHONE2 EWEBCGI	Sample PHONE application for EnterpriseWeb/VM supplied by Beyond Software Inc. (similar to PHONE2 CGI except that it does not support using an ISINDEX query to search)
PHONEBFS SVMEXEC	Sample PHONE application for VM:Webgateway with the program and the data stored in BFS discussed in 4.3, “Byte File System” on page 81
HTTPISO EXEC	Tools for conversion between HTTP and ISO format dates discussed in section 3.4.3.2, “Converting Time Stamps in REXX” on page 45
ISOHTTP EXEC	Tools for conversion between HTTP and ISO format dates discussed in section 3.4.3.2, “Converting Time Stamps in REXX” on page 45
FOOTER HTMLPART	Prototype HTML templates for CGIs in this section
PROLOG1 HTMLPART	Prototype HTML templates for CGIs in this section
PROLOG2 HTMLPART	Prototype HTML templates for CGIs in this section
UPDATEFM HTMLPART	Prototype HTML templates for CGIs in this section
HOME HTML	Sample “Home Page” HTML document
SELECT HTML	HTML file referred to in 3.3.2, “Enhancing Our Sample Program” on page 31
SRVPUSH SVMEXEC	Sample of server push for VM:Webgateway discussed in 3.4.8, “Server Push” on page 53
SRVPUSH WRKEXEC	Sample of server push for VM:Webgateway discussed in 3.4.8, “Server Push” on page 53
TEST CGI	Copy of material from B.1.8, “Files from Web Server Solutions for VM/ESA” on page 234

TEST SVMEXEC	Copy of material from B.1.8, "Files from Web Server Solutions for VM/ESA" on page 234
TIMEQRY WRKEXEC	Sample of server push for VM:Webgateway, making use of IUCV to display an RSCS response
UPDATE CGI	The CGI called by the UPDATE HTML form discussed on page 38 for Webshare
UPDATE HTML	An update form for the PHONE sample application
UPDATE SVMEXEC	The CGI called by the UPDATE HTML form discussed on page 38 for VM:Webgateway
UPDATE EWEBCGI	The CGI called by UPDATE HTML for EnterpriseWeb/VM supplied by Beyond Software Inc. (similar to UPDATE CGI)
	The CGI called by the UPDATE HTML form
UPDATECO CGI	The sample CGI from "Writing Cookies" on page 49 for Webshare
UPDATECO SVMEXEC	The sample CGI from "Writing Cookies" on page 49 for VM:Webgateway
UPDATECO EWEBCGI	Sample program for EnterpriseWeb/VM supplied by Beyond Software Inc. (similar to UPDATECO CGI)
UPDATEFM CGI	The sample CGI from "Reading Cookies" on page 48 for Webshare
UPDATEFM SVMEXEC	The sample CGI from "Reading Cookies" on page 48 for VM:Webgateway
UPDATEFM EWEBCGI	Sample program for EnterpriseWeb/VM supplied by Beyond Software Inc. (similar to UPDATEFM CGI)
UPDATEMD CGI	A sample CGI for updating minidisk files
UPDATEMD HTML	A sample CGI for updating minidisk files
CGI DIRMAP	Web server control files for URL resolution and security
CGI FILELIST	Web server control files for URL resolution and security
VMWEBSRV DIRMAP	Web server control files for URL resolution and security

B.1.6.2 PHONE Data Files

- VM directory: VMWEBCD.WEBSHARE.DATA

PHONE DATA	Data file read and updated by the sample programs
PHONE ORIGINAL	Original version of the PHONE DATA file

B.1.7 Utilities

B.1.7.1 BookMaster to HTML Conversion Tools

- VM directory: VMWEBCD.UTILITIES.B2H

B2HFILTR REXX	A VM:Webgateway filter demonstrating the use of B2H
B2H package	A conversion utility for turning BookMaster documents into HTML. It supports both real-time conversion (by use of a CGI or filter such as B2HFILTR REXX) and batch conversions (when invoked as an EXEC). It includes the following files:

- B2H ANNOUNCE
- B2H EXEC
- B2H HELPCMS
- B2H HTML
- B2H MVSCEXEC
- B2H NEWS
- B2H PACKAGE
- B2H PROFILE
- B2H SCRIPT
- B2H SEXEC
- B2H SYMBOLS
- B2H ZIPBIN
- B2HAPP SCRIPT
- B2HEXA SCRIPT
- B2HIBM FOOTER
- B2HINF SCRIPT
- B2HLINK GIFBIN
- B2HMSG SCRIPT
- B2HR2 HTML
- B2HSETUP SCRIPT
- B2HSYS SCRIPT
- B2HUSE SCRIPT
- B2HUSER EXEC
- B2HUSER SYMBOLS

B.1.7.2 BFSLIST

- VM directory: VMWEBCD.UTILITIES

BFSLIST package A FILELIST-like utility for BFS. It includes the following files:

- $BFSEXEC XEDIT
- $BFSLIST XEDIT
- BFSLIST EXEC
- BFSLIST HELPCMS
- BFSLIST PACKAGE
- BFSTREE EXEC
- BFSTREE HELPCMS

B.1.7.3 Other Utilities

- VM directory: VMWEBCD.UTILITIES

TEST REXX	Copy of material from B.1.8, “Files from Web Server Solutions for VM/ESA” on page 234
HTTPISO EXEC	Tools for conversion between HTTP and ISO format dates discussed in section 3.4.3.2, “Converting Time Stamps in REXX” on page 45
ISOHTTP EXEC	Tools for conversion between HTTP and ISO format dates discussed in section 3.4.3.2, “Converting Time Stamps in REXX” on page 45
PHONE1 REXX	A VM:Webgateway FILTER version of the PHONE application provided in B.1.6, “Introductory Example” on page 231

SYSDATA CGI	Sample tool to create a page of information about the VM system it is run on
SYSDATA REXX	Sample tool to create a page of information about the VM system it is run on
SYSDATA SVMEXEC	Sample tool to create a page of information about the VM system it is run on
TCPSNIFF EXEC	Original TCP application debugging tool described in 3.6, "Debugging Your CGI Programs" on page 57
TCPSNIFF REXX	Original TCP application debugging tool described in 3.6, "Debugging Your CGI Programs" on page 57
VMARCSFS EXEC	Drives VMARC commands against an SFS directory tree Type `VMARCSFS ?` for more information.
VMARC HELPCMS	A data compaction and file archiving utility for VM/CMS
VMARC MODULE	A data compaction and file archiving utility for VM/CMS
WEBSNIFF EXEC	Enhanced TCP application debugging tool described in 3.6, "Debugging Your CGI Programs" on page 57
WEBSNIFF REXX	Enhanced TCP application debugging tool described in 3.6, "Debugging Your CGI Programs" on page 57

B.1.8 Files from Web Server Solutions for VM/ESA

- VM directory: VMWEBCD.UTILITIES

TEST CGI A sample CGI that will operate on all three Web servers documented in *Web Server Solutions for VM/ESA*, SG24-4874. The sample program included can be used much like the POSTTEST CGI program provided with Webshare. When it is run, it displays:

- Any input values passed from the Web server
- All standard CGI environment variables
- Any additional environment variables provided by the Web server
- Any other variables set by the program

TEST SVMEXEC A copy of TEST CGI with a changed file type

B.1.9 VM:Webgateway Tutorial

The VM:Webgateway Tutorial (VIG13TUT.PDF) is on CD1 in Acrobat Reader PDF format. If you do not have Acrobat Reader installed on your PC, you can obtain a free copy at

```
http://www.adobe.com/prodindex/acrobat/readstep.html
```

B.2 Files on CD2

This second CD contains Webshare 1.2.4, a shareware Web server for VM/ESA.

B.2.1 Webshare

The following files are on CD2 (labeled Webshare 1.2.4):

CMSHTTPD VMARC	Webshare freeware Web server in VMARC format.
README LIC	Important Webshare license information see also Appendix C, “Beyond Software Inc. Webshare - IBM Disclaimer” on page 237.
BEYOND VMA	CGI samples provided by Beyond Software Inc. for EnterpriseWeb/VM in VMARC format.

This CD contains a copy of the actual Webshare code (CMSHTTPD.VMA in binary format) and a README file (README.LIC in text format) with the license conditions for installing and using this shareware program. See Appendix C, “Beyond Software Inc. Webshare - IBM Disclaimer” on page 237 for more license information. If you have not yet installed the VMARC MODULE on your system, go to B.1, “Files on CD1” on page 227 for details on how to install VMARC from CD1. To use the Webshare Web server:

1. Upload the CMSHTTPD file in binary from CD2 to your VM system.
2. Run the file through this pipeline to unblock the file:
   ```
   PIPE < CMSHTTPD VMABBIN  A | fblock 80 00 | > CMSHTTPD VMARCA F 80
   ```
3. Unpack the resulting file using VMARC:
   ```
   VMARC UNPK CMSHTTPD VMARC A
   ```

To use the Beyond Software Inc. provided CGI samples:

1. Upload the BEYOND VMA file in binary from CD2 to your VM system.
2. Run the file through this pipeline to unblock the file:
   ```
   PIPE < BEYOND VMABIN A | fblock 80 00 | > BEYOND VMARC A F 80
   ```
3. Unpack the resulting file using VMARC:
   ```
   VMARC UNPK BEYOND VMARC A
   ```

B.3 Files Available on the World Wide Web

You will also find the VMWEBCD VMARC and ZIP files on the Web site

```
http://www.ibm.com/s390/vm/download/packages/
```

In addition, you will find several of these packages on the World Wide Web. In particular, you can obtain the B2H and BFSLIST packages (and many other great tools) from

```
http://www.ibm.com/s390/vm/
```

You can obtain Webshare from

```
http://www.beyond-software.com/
```

Appendix C. Beyond Software Inc. Webshare - IBM Disclaimer

The Webshare program is owned by Beyond Software, Inc. and is distributed to you by IBM ″AS IS″ and as a convenience to you only, WITHOUT WARRANTIES OF ANY KIND.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CAN NOT BE EXCLUDED, IBM MAKES NO WARRANTIES OR CONDITIONS EITHER EXPRESSED OR IMPLIED, INCLUDED WITHOUT LIMITATION, THE WARRANTY OF NON-INFRINGEMENT AND THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING WEBSHARE. IBM MAKES NO WARRANTY REGARDING THE CAPABILITY OF WEBSHARE TO CORRECTLY PROCESS, PROVIDE AND/OR RECEIVE DATE DATA WITHIN AND BETWEEN THE 20TH AND 21ST CENTURIES.

IBM WILL NOT BE LIABLE FOR ANY DIRECT OR INDIRECT DAMAGES, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST SAVINGS, OR ANY INCIDENTAL, SPECIAL OR OTHER ECONOMIC CONSEQUENTIAL DAMAGES, IN CONNECTION WITH YOUR USE OF WEBSHARE, EVEN IF IBM IS INFORMED OF THEIR POSSIBILITIES

NOTICE

You accept this Software with the understanding that Beyond Software Incorporated makes no representations or warranties as to the suitability of the Software for your particular purpose, and that to the extent you use or implement this Software in your own setting, you do so at your own risk. In no event will Beyond Software Incorporated be liable for any direct loss and damages, or any damages whether consequential, incidental, or special, arising out of the use of or inability to use the material provided. Please read the LICENSE which follows to determine if you want to use this Software. * Copyright Beyond Software Incorporated, 1997, All rights reserved.

IF YOU DOWNLOAD OR USE THIS SOFTWARE YOU AGREE TO THESE TERMS

Beyond Software Incorporated grants you a license to use the Software only in the country where you acquired it. The Software is copyrighted and licensed (not sold). We do not transfer title to the Software to you. You obtain no rights other than those granted you under this license.

Under this license, you may:

- Use the Software on one or more machines at a time.
- Make copies of the Software for use or backup purposes within your enterprise.
- Modify the Software and merge it into another program.

You must reproduce the copyright notice and any other legend of ownership on each copy or partial copy of the Software.

You may NOT:

- Reverse assemble, reverse compile, or otherwise translate any program contained within the Software.
- Post modified versions of this material for public access.
- Sublicense, assign or transfer the license to the Software or accompanying documentation. Any attempt otherwise to sublicense, assign or transfer any of the rights, duties or obligations hereunder is void.

We do not warrant that the Software is free from claims by a third party of copyright, patent, trademark, trade secret, or any other intellectual property infringement.

Under no circumstances are we liable for any of the following:

1. Third-party claims against you for losses or damages
2. Loss of, or damage to, your records or data.
3. Economic consequential damages (including lost profits or savings) or incidental damages, even if we are informed of their possibility.

Some jurisdictions do not allow these limitations or exclusions, so they may not apply to you.

We do not warrant uninterrupted or error free operation of the Software. We have no obligation to provide service, defect correction, or any maintenance for the Software. We have no obligation to supply any Software updates or enhancements to you even if such are or later become available.

IF YOU DOWNLOAD OR USE THIS SOFTWARE YOU AGREE TO THESE TERMS.THERE ARE NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE

Some jurisdictions do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

If you make comments and suggestions (collectively called ″Feedback″) relating to your use of the Software, you grant Beyond Software Incorporated a non-exclusive, royalty-free, irrevocable, unrestricted and world-wide license to use, have used and make copies in case of documents, such Feedback in any manner as Beyond Software Incorporated determines, including use of Feedback in the development, manufacture, marketing and maintenance of products and services incorporating such feedback by Beyond Software Incorporated.

You may terminate this license at any time. We may terminate this license if you fail to comply with any of its terms. In either event, you must destroy all your copies of the Software.

You are responsible for the payment of any taxes resulting from this license.

If any provision of this Agreement is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This Agreement shall be governed by California law (except for conflict of law provisions). The application the United Nations Convention of Contracts for the International Sale of Goods is expressly excluded.

Should you have any questions concerning this Agreement, you may contact Beyond at:

Beyond Software Incorporated
1040 East Brokaw Road
San Jose, California 95131-2309
U.S.A.
Phone: 408-436-5900
Fax: 408-436-5915
E-mail: info@beyond-software.com

Appendix D. Special Notices

This publication is intended to help system programmers and application programmers to Web-enable existing data and applications and also design new applications for the Web. The information in this publication is not intended as the specification of any programming interfaces that are provided by VM/ESA or any of the Web servers.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (″vendor″) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

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.

Appendix E. Related Publications

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

E.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see "How to Get ITSO Redbooks" on page 247.

- *Web Server Solutions for VM/ESA*, SG24-4874-01
- *Exploiting Recent CMS Function: A User's Guide to CMS Application Multitasking*, SG24-5164
- *VM/ESA Network Computing with Java and NetRexx*, SG24-5148
- *OpenEdition for VM/ESA Implementation and Administration Guide*, SG24-4747

E.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year.

CD-ROM Title	**Subscription Number**	**Collection Kit Number**
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
Lotus Redbooks Collection	SBOF-6899	SK2T-8039
Tivoli Redbooks Collection	SBOF-6898	SK2T-8044
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
RS/6000 Redbooks Collection (PDF Format)	SBOF-8700	SK2T-8043
Application Development Redbooks Collection	SBOF-7290	SK2T-8037

E.3 Other IBM Publications

This publication is also relevant as a source of further information:

- *VM/ESA CMS Application Development Guide*, SC24-5761
- *MQSeries: Technical Reference*, SC33-0850
- *MQSeries: Application Programming Reference*, SC33-1673
- *MQSeries: Application Programming Guide*, SC33-0807
- *MQSeries Clients*, GC33-1632
- *MQSeries for MVS/ESA System Management Guide*, SC33-0806
- *DB2 for VM V5R1 Interactive SQL Guide and Reference*, SC09-2409
- *IMS/ESA V6 OTMA Guide and Reference*, SC26-8743
- *VM/ESA Planning and Administration V2R3.0*, SC24-5750.

E.4 External Publications

- *HTML Sourcebook, Third Edition*, ISBN: 0-471-17575-7
- *HTML: The Definitive Guide*, ISBN: 1-56592-492-4
- *HTML, JAVA, CGI, VRML, SGML Web Publishing* , ISBN: 1-56592-492-4
- *VM:Webgateway Tutorial*, which comes with VM:Webgateway and is also available on our associated CD (see B.1.9, "VM:Webgateway Tutorial" on page 234 for more information)

E.5 Web Sites

- *VM/ESA Operating System Home Page*

  `http://www.ibm.com/s390/vm/`

- *The Internet Engineering Task Force (IETF) Home Page*

  `http://www.ietf.org/`

  There you will find links to all of the Internet standards, experimental and informational RFCs and draft documents of proposed Internet standards. Documents of interest include:

  – *RFC 1945 - Hypertext Transfer Protocol -- HTTP/1.0*
  – *RFC 2068 - Hypertext Transfer Protocol -- HTTP/1.1*
  – *RFC 1866 - Hypertext Markup Language - 2.0*
  – *RFC 1867 - Form-based File Upload in HTML*
  – *RFC 1942 - HTML Tables*
  – *RFC 1980 - A Proposed Extension to HTML: Client-Side Image Maps*
  – *RFC 1738 - Uniform Resource Locators (URL)*
  – *RFC 1630 - Universal Resource Identifiers (URI)*
  – *RFC 1808 - Relative Uniform Resource Locators*

- *An Introduction to Writing Webshare CGI Scripts*

  `http://www.beyond-software.com/Products/Presentations/Webshare/WebshareCGIs.html`

- *The WWW Common Gateway Interface Version 1.1*

  `http://www.ics.uci.edu/pub/ietf/http/related/draft-robinson-www-interface-01.txt`

- Drop in for a visit at Melinda Varian′s home page at the following location. There you will find many links and information concerning REXX and Pipelines

  `http://pucc.princeton.edu/˜Melinda/`

- *Internet Software for VM* page. It is provided by the Beyond Software Corporation at URL:

  `http://www.beyond-software.com/Software/Software.html`

- Further information about Beyond Software Inc. and EnterpriseWeb/VM is found on the Web at the following address:

  `http://www.beyond-software.com`

- *VM and the Internet* page. Contains articles, transcripts, helpful links, and product information

  `http://www.vm.sterling.com/general/documents/webindex.html`

- For further information about Sterling Software, Inc., the VM Software Division, and VM:Webgateway, go to:

```
http://www.sterling.com/
http://www.vm.sterling.com/
```

Be sure to also refer to 5.10, “References” on page 154 and 6.6, “References” on page 183 for additional reference sources.

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** http://www.redbooks.ibm.com/

  Search for, view, download or order hardcopy/CD-ROMs redbooks from the redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this redbooks site.

  Redpieces are redbooks in progress; not all redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders via e-mail including information from the redbook fax order form to:

In United States:	e-mail address: usib6fpl@ibmmail.com
Outside North America:	Contact information is in the ″How to Order″ section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl/

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the ″How to Order″ section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl/

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the ″How to Order″ section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl/

This information was current at the time of publication, but is continually subject to change. The latest information for customers may be found at http://www.redbooks.ibm.com/ and for IBM employees at http://w3.itso.ibm.com/.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and redbooks by accessing the IBM Intranet Web site at http://w3.itso.ibm.com/ and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may also view redbook, residency and workshop announcements at http://inews.ibm.com/.

IBM Redbook Fax Order Form

Please send me the following:

Title	**Order Number**	**Quantity**

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

- Invoice to customer number

- Credit card number

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

List of Abbreviations

ACL	access control list
ANSI	American National Standards Institute
APA	all points addressable
ASCII	American National Standard Code for Information Interchange
AWT	Abstract Windowing Toolkit
CERN	Conseil Europeen pour la Recherche Nucleaire (European organization for nuclear research)
CGI	Common Gateway Interface (programs that provide services on the WWW)
CICS	customer information control system (software, IBM)
CMS	conversational monitor system (VM-based software, IBM)
CP	control program
CPU	central processing unit
CRLF	carriage return/line feed
DASD	direct access storage device
DNS	domain name service
DPL	distributed program link
EBCDIC	extended binary coded decimal interchange code
ESM	external security manager (VM/CMS SFS authorization exits)
FTP	file transfer protocol
GIF	graphic interchange format
GML	generalized markup language (text format language)
GMT	Greenwich mean time, also known as UTC
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
IBM	International Business Machines Corporation
ID	identification/identifier
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
I/O	input/output
IP	Internet Protocol (ISO)
IPL	initial program load
ISO	International Standards Organization
ITSO	International Technical Support Organization
IUCV	inter-user communication vehicle
JPEG	Joint Photographic Experts Group (CCITT/ISO, multimedia standards)
JDK	Java Developer Kit
JIT	Just In Time compiler
JVM	Java Virtual Machine
MIME	Multipurpose Internet Mail Extensions (RFC 1341)
MPEG	Moving Pictures Experts Group (CCITT/ISO, multimedia standards)
MQMD	MQSeries message descriptor
NCSA	The National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign
NFS	network file system (USA, Sun Microsystems Inc.)
OV	OfficeVision product
PROFS	Professional Office System
RACF	resource access control facility
REXX	restructured extended executor language
RFC	Request for Comment
RSA	Rivest-Shamir-Adleman algorithm (cryptograph, named after Ronald Rivest, Adi Shamir and Leonard Adleman)
RSCS	remote spooling communications subsystem (VM′s counterpart to MVS JES NJE)
SCIF	single console image facility
SFS	shared file system (hierarchical sharable VM/CMS file system)

SGML	standard generalized mark-up language (ISO 8879)
SHARE	an association of IBM engineering/scientific customers with large computing systems
SSI	server side include
SSL	Secure Sockets Layer
SVM	Service Virtual Machine (a server)
TCP	Transmission Control Protocol (USA, DoD)
TCP/IP	Transmission Control Protocol/Internet Protocol (USA, DoD, ARPANET; TCP=layer 4, IP=layer 3, UNIX-ish/Ethernet-based system-interconnect protocol)
TCPIP	Transmission Control Protocol Internet Protocol
URL	Uniform Resource Locator
UTC	coordinated universal time, also know as GMT
VM	virtual machine (IBM System 370 & 390)
VM/CMS	virtual machine/conversational monitor system (IBM)
VM/ESA	virtual machine/enterprise systems architecture (IBM)
VMNFS	virtual machine network file system (see NFS)
VMTOOLS	VM programs and tools library (internal to IBM)
WAIS	wide area information servers
WWW	World Wide Web
W3C	World Wide Web (WWW) Consortium

Index

ITSO Redbook Evaluation

Web-Enabling VM Resources
SG24-5347-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and Fax it to: USA International Access Code + 1 914 432 8264 or:**

- Use the online evaluation form found at http://www.redbooks.ibm.com
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?
__Customer __Business Partner __Solution Developer __IBM employee
__None of the above

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction ____________

Please answer the following questions:

Was this redbook published in time for your needs? Yes____ No____

If no, please explain:
____

____

____

____

What other redbooks would you like to see published?
____

____

____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)
____

____

____

____

____

SG24-5347-00
Printed in the U.S.A.

Web-Enabling VM Resources

SG24-5347-00